

# Package ‘Rcsdp’

April 28, 2009

**Version** 0.1-4

**Title** R interface to the CSDP semidefinite programming library

**Author** Hector Corrada Bravo (CSDP by Brian Borchers)

**Maintainer** Hector Corrada Bravo <hcorrada@gmail.com>

**Description** R interface to the CSDP semidefinite programming library. Installs version 6.0.1 of CSDP from the COIN-OR website if required. An existing installation of CSDP may be used by passing the proper configure arguments to the installation command. See the INSTALL file for further details.

**LazyLoad** yes

**Suggests** Matrix

**Enhances** Matrix

**License** file LICENSE

**URL** <https://projects.coin-or.org/Csdp/>

**Repository** CRAN

**Repository/R-Forge/Project** rcsdp

**Repository/R-Forge/Revision** 16

**Date/Publication** 2009-04-28 07:56:04

## R topics documented:

csdp . . . . .	2
csdp-sparse . . . . .	5
readsdp . . . . .	6
<b>Index</b>	<b>8</b>

---

 csdp

 Solve semidefinite program with CSDP
 

---

### Description

Interface to CSDP semidefinite programming library. The general statement of the primal problem is

$$\begin{aligned} \max \quad & \text{tr}(CX) \\ \text{s.t.} \quad & A(X) = b \\ & X \succeq 0 \end{aligned}$$

with  $A(X)_i = \text{tr}(A_i X)$  where  $X \succeq 0$  means  $X$  is positive semidefinite,  $C$  and all  $A_i$  are symmetric matrices of the same size and  $b$  is a vector of length  $m$ .

The dual of the problem is

$$\begin{aligned} \min \quad & b'y \\ \text{s.t.} \quad & A'(y) - C = Z \\ & Z \succeq 0 \end{aligned}$$

where  $A'(y) = \sum_{i=1}^m y_i A_i$ .

Matrices  $C$  and  $A_i$  are assumed to be block diagonal structured, and must be specified that way (see Details).

### Usage

```
csdp(C, A, b, K, control=csdp.control())
```

### Arguments

C	A list defining the block diagonal cost matrix $C$ .
A	A list of length $m$ containing block diagonal constraint matrices $A_i$ . Each constraint matrix $A_i$ is specified by a list of blocks as explained in the Details section.
b	A numeric vector of length $m$ containing the right hand side of the constraints.
K	Describes the domain of each block of the sdp problem. It is a list with the following elements: <b>type:</b> A character vector with entries "s" or "l" indicating the type of each block. If the $j$ th entry is "s", then the $j$ th block is a positive semidefinite matrix. otherwise, it is a vector with non-negative entries. <b>size:</b> A vector of integers indicating the dimension of each block.
control	Control parameters passed to csdp. See CSDP documentation.

## Details

All problem matrices are assumed to be of block diagonal structure, and must be specified as follows:

1. If there are `nblocks` blocks specified by `K`, then the matrix must be a list with `nblocks` components.
2. If `K$type == "s"` then the `j`th element of the list must define a symmetric matrix of size `K$size`. It can be an object of class `"matrix"`, `"simple_triplet_sym_matrix"`, or a valid class from the class hierarchy in the `"Matrix"` package.
3. If `K$type == "l"` then the `j`th element of the list must be a numeric vector of length `K$size`.

This function checks that the blocks in arguments `C` and `A` agree with the sizes given in argument `K`. It also checks that the lengths of arguments `b` and `A` are equal. It does not check for symmetry in the problem data.

## Value

<code>X</code>	Optimal primal solution $X$ . A list containing blocks in the same structure as explained above. Each element is of class <code>"matrix"</code> or a numeric vector as appropriate.
<code>Z</code>	Optimal dual solution $Z$ . A list containing blocks in the same structure as explained above. Each element is of class <code>"matrix"</code> or a numeric vector as appropriate.
<code>y</code>	Optimal dual solution $y$ . A vector of the same length as argument <code>b</code>
<code>pobj</code>	Optimal primal objective value
<code>dobj</code>	Optimal dual objective value
<code>status</code>	Status of returned solution. <ul style="list-style-type: none"> <li><b>0:</b> Success. Problem solved to full accuracy</li> <li><b>1:</b> Success. Problem is primal infeasible</li> <li><b>2:</b> Success. Problem is dual infeasible</li> <li><b>3:</b> Partial Success. Solution found but full accuracy was not achieved</li> <li><b>4:</b> Failure. Maximum number of iterations reached</li> <li><b>5:</b> Failure. Stuck at edge of primal feasibility</li> <li><b>6:</b> Failure. Stuch at edge of dual infeasibility</li> <li><b>7:</b> Failure. Lack of progress</li> <li><b>8:</b> Failure. <math>X</math> or <math>Z</math> (or Newton system <math>O</math>) is singular</li> <li><b>9:</b> Failure. Detected NaN or Inf values</li> </ul>

## Author(s)

Hector Corrada Bravo. CSDP written by Brian Borchers.

## References

- <https://projects.coin-or.org/Csdp/>
- Borchers, B.:  
*CSDP, A C Library for Semidefinite Programming Optimization Methods and Software* 11(1):613-623, 1999  
<http://euler.nmt.edu/~brian/csdppaper.pdf>
- Lu, F., Lin, Y., and Wahba, G.:  
*Robust Manifold Unfolding with Kernel Regularization* TR 1108, October, 2005.  
<http://www.stat.wisc.edu/~wahba/ftp1/tr1108rr.pdf>

## Examples

```

C <- list(matrix(c(2,1,
                  1,2), 2, 2, byrow=TRUE),
          matrix(c(3,0,1,
                  0,2,0,
                  1,0,3), 3, 3, byrow=TRUE),
          c(0,0))
A <- list(list(matrix(c(3,1,
                      1,3), 2, 2, byrow=TRUE),
              matrix(0, 3, 3),
              c(1,0)),
          list(matrix(0, 2, 2),
              matrix(c(3,0,1,
                      0,4,0,
                      1,0,5), 3, 3, byrow=TRUE),
              c(0,1)))

b <- c(1,2)
K <- list(type=c("s", "s", "1"), size=c(2, 3, 2))
csdp(C, A, b, K)

# Manifold Unrolling broken stick example
# using simple triplet symmetric matrices
X <- matrix(c(-1,-1,
              0,0,
              1,-1), nc=2, byrow=TRUE);
d <- as.vector(dist(X)^2);
d <- d[-2]

C <- list(.simple_triplet_diag_sym_matrix(1, 3))
A <- list(list(simple_triplet_sym_matrix(i=c(1, 2, 2), j=c(1, 1, 2), v=c(1, -1, 1), n=3)),
          list(simple_triplet_sym_matrix(i=c(2, 3, 3), j=c(2, 2, 3), v=c(1, -1, 1), n=3)),
          list(matrix(1, 3, 3)))

K <- list(type="s", size=3)
csdp(C, A, c(d, 0), K)

```

## Description

Support for sparse matrices in package `Rcsdp`. The class `simple_triplet_sym_matrix` is defined to provide support for symmetric sparse matrices. Its definition is copied from the package `relations` by Kurt Hornik. Coercion functions from objects of class `matrix` and classes in the `Matrix` hierarchy are provided.

## Usage

```
simple_triplet_sym_matrix(i, j, v, n=max(c(i, j)), check.ind=FALSE)
## S3 method for class 'matrix':
as.simple_triplet_sym_matrix(x, check.sym=FALSE)
## S3 method for class 'simple_triplet_sym_matrix':
as.matrix(x, ...)
## S3 method for class 'simple_triplet_sym_matrix':
as.vector(x, ...)
.simple_triplet_zero_sym_matrix(n, mode="double")
.simple_triplet_diag_sym_matrix(x, n)
.simple_triplet_random_sym_matrix(n, occ=.1, nnz=occ*n*(n+1)/2, rfun=rnorm, seed=NULL)
```

## Arguments

<code>i</code>	Row indices of non-zero entries.
<code>j</code>	Column indices of non-zero entries.
<code>v</code>	Non-zero entries.
<code>n</code>	Size of matrix.
<code>check.ind</code>	Checks that arguments <code>i</code> and <code>j</code> indicate entries in the lower triangular part of the matrix. Default <code>FALSE</code> .
<code>check.sym</code>	Checks if matrix object is symmetric. Default <code>FALSE</code> .
<code>x</code>	Object of class <code>matrix</code> or <code>simple_triplet_sym_matrix</code> .
<code>mode</code>	Type of zero matrix to create. Default <code>double</code> .
<code>occ</code>	Ratio of occupancy of random sparse matrix. Default <code>.1</code> .
<code>nnz</code>	Number of non-zero entries in random sparse matrix. Default corresponds to <code>occ=.1</code> .
<code>rfun</code>	Function to generate random entries in sparse matrix. Default <code>rnorm</code> .
<code>seed</code>	Random number generator seed. Set by function <code>set.seed</code> before generating random sparse matrix. Default <code>NULL</code> .
<code>...</code>	Arguments passed on to casting functions.

**Details**

TO DO

**Value**

TO DO

**Author(s)**

Hector Corrada Bravo

**References**

TO DO

**See Also**

[csdp](#)

**Examples**

```
# TO DO
```

---

readsdp

*Reading and writing semidefinite programs for SDPA format files.*

---

**Description**

Functions to read and write semidefinite program data and solutions in SDPA format.

**Usage**

```
readsdp(file="", verbose=FALSE)
writesdp(C, A, b, K, file="")
readsdp.sol(K, C, m, file="")
writesdp.sol(X, Z, y, K, file="")
```

**Arguments**

<code>file</code>	The name of the file to read from or write to.
<code>C</code>	Block structured cost matrix
<code>A</code>	List of block structured constraint matrices
<code>b</code>	RHS vector
<code>K</code>	Cone specification, as used in <a href="#">csdp</a>
<code>X</code>	Block structured primal optimal solution matrix
<code>Z</code>	Block structured dual optimal solution matrix

<code>y</code>	Dual optimal solution vector
<code>verbose</code>	Printout information as problem is read. Passed to CSDP's readsdp function. Default FALSE
<code>m</code>	Number of constraints in problem.

### Details

Block structured matrices must be specified as described in [csdp](#). Files read must be in SDPA format (see <http://infohost.nmt.edu/~sdplib/FORMAT>). However, these functions don't support comments or grouping characters (e.g. braces, parentheses) in the block sizes specification.

### Value

Function `readsdp` returns a list with elements  $C, A, b, K$ . Function `readsdp.sol` returns a list with elements  $X, Z, y$ . All returned matrices are lists of objects of class `simple_triplet_sym_matrix`.

### Author(s)

Hector Corrada Bravo

### References

<http://infohost.nmt.edu/~sdplib/FORMAT>

### See Also

[csdp](#)

### Examples

```
# TO DO
```

# Index

## \*Topic **optimize**

`csdp`, 2

## \*Topic **utilities**

`csdp-sparse`, 5

`.simple_triplet_diag_sym_matrix`  
(*csdp-sparse*), 5

`.simple_triplet_random_sym_matrix`  
(*csdp-sparse*), 5

`.simple_triplet_zero_sym_matrix`  
(*csdp-sparse*), 5

`as.matrix.simple_triplet_sym_matrix`  
(*csdp-sparse*), 5

`as.simple_triplet_sym_matrix.matrix`  
(*csdp-sparse*), 5

`as.vector.simple_triplet_sym_matrix`  
(*csdp-sparse*), 5

`csdp`, 2, 6, 7

`csdp-sparse`, 5

`readsdp`, 6

`simple_triplet_sym_matrix`  
(*csdp-sparse*), 5

`simple_triplet_sym_matrix-class`  
(*csdp-sparse*), 5

`writesdp` (*readsdp*), 6