Package ‘Rcsdp’

April 10, 2023

Version 0.1.57.5
Title R Interface to the CSDP Semidefinite Programming Library
Description R interface to the CSDP semidefinite programming library. Installs version 6.1.1 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
Imports methods
Enhances Matrix
License CPL-1.0
URL https://github.com/coin-or/Csdp/
RoxygenNote 7.0.2
BugReports https://github.com/hcorrada/rcsdp/issues
NeedsCompilation yes
Author Hector Corrada Bravo [aut, cre],
    Florian Schwendinger [ctb],
    Brian Borchers [aut],
    Don van den Bergh [ctb]
Maintainer Hector Corrada Bravo <hcorrada@gmail.com>
Repository CRAN
Date/Publication 2023-04-10 09:39:32 UTC

R topics documented:

  csdp .................................................. 2
  csdp-sparse ...................................... 5
  csdp.control ..................................... 6
  readsdp ......................................... 8

Index 10
csdp

Solve semidefinite program with CSDP

Description

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

\[ \max \text{ tr}(CX) \]
\[ \text{s.t. } A(X) = b \]
\[ X \succeq 0 \]

with \( A(X)_i = \text{tr}(A_iX) \) 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

\[ \min b'y \]
\[ \text{s.t. } A'(y) - C = Z \]
\[ Z \succeq 0 \]

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

\[
\text{csdp(C, A, b, K, control=csdp.control())}
\]

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:
  - **type**: A character vector with entries "s" or "l" indicating the type of each block. If the jth entry is "s", then the jth block is a positive semidefinite matrix. otherwise, it is a vector with non-negative entries.
  - **size**: 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$type == "s"` then the jth element of the list must define a symmetric matrix of size `K$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$type == "l"` then the jth element of the list must be a numeric vector of length `K$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.

csdp_minimal is a minimal wrapper to the C code underlying csdp. It assumes that the arguments `sum.block.sizes`, `nconstraints`, `nblocks`, `block.types`, and `block.sizes` are provided as if they were created by `Rcsdp:::prob.info` and that the arguments `C`, `A`, and `b` are provided as if they were created by `Rcsdp:::prepare.data`. This function may be useful when calling the csdp functionality iteratively and most of the optimization details stays the same. For example, when the control file created by `Rcsdp:::write.control.file` stays the same across iterations, but it would be recreated on each iteration by csdp.

Value

- `X` Optimal primal solution. A list containing blocks in the same structure as explained above. Each element is of class "matrix" or a numeric vector as appropriate.
- `Z` Optimal dual solution. A list containing blocks in the same structure as explained above. Each element is of class "matrix" or a numeric vector as appropriate.
- `y` Optimal dual solution. A vector of the same length as argument `b`.
- `pobj` Optimal primal objective value
- `dobj` Optimal dual objective value
- `status` Status of returned solution.

0: Success. Problem solved to full accuracy
1: Success. Problem is primal infeasible
2: Success. Problem is dual infeasible
3: Partial Success. Solution found but full accuracy was not achieved
4: Failure. Maximum number of iterations reached
5: Failure. Stuck at edge of primal feasibility
6: Failure. Stuck at edge of dual infeasibility
7: Failure. Lack of progress
8: Failure. X or Z (or Newton system O) is singular
9: Failure. Detected NaN or Inf values
Author(s)
Hector Corrada Bravo. CSDP written by Brian Borchers.

References
- https://github.com/coin-or/Csdp/

Examples
```r
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","l"),size=c(2,3,2))
csdp(C,A,b,K)
```

# Manifold Unrolling broken stick example

# using simple triplet symmetric matrices
```r
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)
```
Simple support for sparse matrices

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

```r
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

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

Value
TO DO

Author(s)
Hector Corrada Bravo

References
TO DO

See Also
csdp

tocsdp.control

Examples
# TO DO

---

## Description
Utility function to pass control parameters to csdp solver.

## Usage
```r
csdp.control(axtol = 1e-08,
             atytol = 1e-08,
             objtol = 1e-08,
             pinftol = 1e+08,
             dinftol = 1e+08,
             maxiter = 100,
             minstepfrac = 0.9,
             maxstepfrac = 0.97,
             minstepp = 1e-08,
             minstepd = 1e-08,
             usexzgap = 1,
             tweakgap = 0,
             affine = 0,
             printlevel = 1,
             perturbobj = 1,
             fastmode = 0)
```
**Arguments**

- **axtol**: Tolerance for primal feasibility.
- **atytol**: Tolerance for dual feasibility.
- **objtol**: Tolerance for relative duality gap.
- **pinftol**: Tolerance for primal infeasibility.
- **dinfotol**: Tolerance for dual infeasibility.
- **maxiter**: Maximum number of iterations used.
- **minstepfrac**: Minimum distance to edge of feasibility region for step.
- **maxstepfrac**: Maximum distance to edge of feasibility region for step.
- **minstepp**: Failure is declared if primal line search step size is shorter than this parameter.
- **minstepp**: Failure is declared if dual line search step size is shorter than this parameter.
- **usezgap**: If 0, then use objective function duality gap.
- **tweakgap**: If 1 (and **usezgap**=0) then "fix" negative duality gaps.
- **affine**: If 1, only use affine primal-dual steps and do not use barrier function.
- **printlevel**: If 0, no printing, 1 normal printing, higher values result in more debug printing.
- **perturbobj**: Amount of objective permutation used.
- **fastmode**: If 1, csdp will be faster but also less accurate.

**Details**

Parameters are fully described in CSDP user guide. [https://github.com/coin-or/Csdp/](https://github.com/coin-or/Csdp/)

**Value**

A list with values for all parameters. Any parameters not passed to function are set to default.

**Author(s)**

Hector Corrada Bravo, CSDP by Brian Borchers

**References**

[https://github.com/coin-or/Csdp/](https://github.com/coin-or/Csdp/)

**Examples**

```r
params <- csdp.control(axtol=1e-6)
```
readsdpa  

Reading and writing semidefinite programs for SDPA format files.

Description

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

Usage

```
readsdpa(file='', verbose=FALSE)
writesdpa(C,A,b,K,file='')
readsdpa.sol(K,C,m,file='')
writesdpa.sol(X,Z,y,K,file='')
```

Arguments

- **file**: The name of the file to read from or write to.
- **C**: Block structured cost matrix
- **A**: List of block structured constraint matrices
- **b**: RHS vector
- **K**: Cone specification, as used in csdp
- **X**: Block structured primal optimal solution matrix
- **Z**: Block structured dual optimal solution matrix
- **y**: Dual optimal solution vector
- **verbose**: Printout information as problem is read. Passed to CSDP's readsdpa function. Default FALSE
- **m**: 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://euler.nmt.edu/~brian/sdplib/FORMAT). However, these functions don't support comments or grouping characters (e.g. braces, parentheses) in the block sizes specification.

Value

Function readsdpa returns a list with elements \( C, A, b, K \). Function readsdpa.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://euler.nmt.edu/~brian/sdplib/FORMAT
readsdp

See Also

csdp

Examples

# TO DO
Index

* optimize
  cspd, 2
* utilities
  cspdp-sparse, 5
  .simple_triplet_diag_sym_matrix
    (cspdp-sparse), 5
  .simple_triplet_random_sym_matrix
    (cspdp-sparse), 5
  .simple_triplet_zero_sym_matrix
    (cspdp-sparse), 5
  as.matrix.simple_triplet_sym_matrix
    (cspdp-sparse), 5
  as.simple_triplet_sym_matrix.matrix
    (cspdp-sparse), 5
  as.vector.simple_triplet_sym_matrix
    (cspdp-sparse), 5
  cspdp, 2, 6, 8, 9
  cspdp-sparse, 5
  cspdp.control, 6
  cspdpMinimal(cspdp), 2
  readsdp, 8
  simple_triplet_sym_matrix
    (cspdp-sparse), 5
  simple_triplet_sym_matrix-class
    (cspdp-sparse), 5
  writedsdp(readsdp), 8