Package: scs (via r-universe)

November 1, 2024

Version 3.2.4

Title Splitting Conic Solver

Description Solves convex cone programs via operator splitting. Can solve: linear programs ('LPs'), second-order cone programs ('SOCPs'), semidefinite programs ('SDPs'), exponential cone programs ('ECPs'), and power cone programs ('PCPs'), or problems with any combination of those cones. 'SCS' uses 'AMD' (a set of routines for permuting sparse matrices prior to factorization) and 'LDL' (a sparse 'LDL' factorization and solve package) from 'SuiteSparse' (<https://people.engr.tamu.edu/davis/suitesparse.html>).

Depends R (>= 3.5.0)

SystemRequirements GNU Make

Suggests Matrix, slam, tinytest

Encoding UTF-8

License GPL-3

URL https://github.com/FlorianSchwendinger/scs

RoxygenNote 7.2.0

Repository https://florianschwendinger.r-universe.dev

RemoteUrl https://github.com/florianschwendinger/scs

RemoteRef HEAD

RemoteSha 8848b0df71fdbb96d9c1b937ea11b341416a5a57

Contents

scs *SCS - Splitting Conic Solver*

Description

Solves convex cone programs via operator splitting.

Usage

```
scs(A, b, obj, P = NULL, cone, initial = NULL, control = scs_control())
```

Arguments

A	a matrix of constraint coefficients. **NOTE:** The rows of matrix A have to be ordered according to the order given in subsection "Allowed cone parameters". For more information see **README**.
b	a numeric vector giving the primal constraints
obj	a numeric vector giving the primal objective
P	a symmetric positive semidefinite matrix, default NULL
cone	a list giving the cone sizes
initial	a named list (warm start solution) of three elements: x (length = length(obj)), y (length = nrow(A)), and s (length = nrow(A)), default NULL indicating no warm start.
control	a list giving the control parameters. For more information see **README**.

Details

Important Note:
The order of the rows in matrix A has to correspond to the order given in the table "Cone Arguments", which means means rows corresponding to *primal zero cones* should be first, rows corresponding to *non-negative cones* second, rows corresponding to *second-order cone* third, rows corresponding to *positive semidefinite cones* fourth, rows corresponding to *exponential cones* fifth and rows corresponding to *power cones* at last.

SCS can solve:

1. linear programs (LPs)
2. second-order cone programs (SOCPs)
3. semidefinite programs (SDPs)
4. exponential cone programs (ECPs)
5. power cone programs (PCPs)
6. problems with any combination of cones, which can be defined by the parameters listed in the subsection "Allowed cone parameters"

Allowed *cone* parameters are:

Parameter	Type	Length	Description
z	integer	1	number of primal zero cones (dual free cones), which corresponds to the primal equality constraints
l	integer	1	number of linear cones (non-negative cones)
bsize	integer	1	size of box cone
bl	numeric	$bsize - 1$	lower limit for box cone
bu	numeric	$bsize - 1$	upper limit for box cone
q	integer	≥ 1	vector of second-order cone sizes
s	integer	≥ 1	vector of positive semidefinite cone sizes
ep	integer	1	number of primal exponential cones
ed	integer	1	number of dual exponential cones
p	numeric	≥ 1	vector of primal/dual power cone parameters

Value

list of solution vectors x, y, s and information about run

Examples

```
A <- matrix(c(1, 1), ncol=1)
b <- c(1, 1)
obj <- 1
cone <- list(z = 2)
control <- list(eps_rel = 1e-3, eps_abs = 1e-3, max_iters = 50)
sol <- scs(A = A, b = b, obj = obj, cone = cone, control = control)
sol
```

`scs_control` *SCS Control Arguments*

Description

Details to the *control* parameters.

Usage

```
scs_control(
  max_iters = 100000L,
  eps_rel = 1e-04,
  eps_abs = 1e-04,
  eps_infeas = 1e-07,
  alpha = 1.5,
  rho_x = 1e-06,
  scale = 0.1,
  verbose = FALSE,
  normalize = TRUE,
```

```
  warm_start = FALSE,
  acceleration_lookback = 0L,
  acceleration_interval = 1L,
  adaptive_scale = TRUE,
  write_data_filename = NULL,
  log_csv_filename = NULL,
  time_limit_secs = 0
)
```

Arguments

max_iters	an integer giving the maximum number of iterations (default is 100000L).
eps_rel	a double specifying relative feasibility tolerance (default 1e-4).
eps_abs	a double specifying absolute feasibility tolerance (default 1e-4).
eps_infeas	a double specifying infeasibility tolerance (primal and dual) (default 1e-7).
alpha	a double giving the (Douglas-Rachford) over-relaxation parameter, allowed values are in (0, 2) (default 1.5).
rho_x	a double giving the momentum of x term (default os 1e-6).
scale	a double giving the factor (default is 1.0) by which the data is rescaled (only used if normalize is TRUE).
verbose	a logical giving if the progress should be printed (default is FALSE).
normalize	a logical giving if heuristic data rescaling should be used (default is TRUE).
warm_start	a logical indicating if a warm_start is provided (default FALSE, but a call to scs with a non-null initial argument overrides it to be effectively TRUE)
acceleration_lookback	an integer indicating How much memory to use for Anderson acceleration. More memory requires more time to compute but can give more reliable steps (default 0L, disabling it).
acceleration_interval	an integer specifying the number of iterations for which Anderson acceleration is run (default 1L).
adaptive_scale	a logical indicating whether to heuristically adapt dual through the solve (default TRUE).
write_data_filename	a string indicating filename to write problem data to (default NULL indicating no write).
log_csv_filename	a string indicating filename where SCS will write csv logs of various quantities through the solver (default NULL indicating no logging, as it makes the solver much slower).
time_limit_secs	a double indicating time limit for solve run in seconds; can be fractional (default 0.0 indicating no limit).

Value

a list containing the control parameters.

Index