Package 'scs'

October 31, 2026

Version 3.3.1

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

Config/roxygen2/version 8.1.1

NeedsCompilation yes

Author Florian Schwendinger [aut, cre],
Brendan O'Donoghue [aut, cph],
Balasubramanian Narasimhan [aut],
Timothy A. Davis [cph],
Patrick R. Amestoy [cph],
Iain S. Duff [cph]

Maintainer Florian Schwendinger <FlorianSchwendinger@gmx.at>

Repository CRAN

Date/Publication 2026-10-31 08:20:07 UTC

Contents

scs 2
scs_control 4
scs_version 6

Index 7

`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: a base `matrix`, a `dgCMatrix` from package **Matrix**, or a `simple_triplet_matrix` from package **slam**. Other **Matrix** classes must be converted by the caller. **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: a base `matrix`, a `dsCMatrix` (either triangle stored) or `dgCMatrix` from package **Matrix**, or a `simple_triplet_matrix`. Only the upper triangle is used. 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 *box cones* third, rows corresponding to *second-order cones* fourth, rows corresponding to *positive semidefinite cones* fifth, rows corresponding to *complex positive semidefinite cones* sixth, rows corresponding to *exponential cones* seventh 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), real and complex
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 (number of bounds plus one); defaults to length(bl) + 1 when omitted
bl	numeric	$bsize - 1$	lower limits for box cone (may be -Inf)
bu	numeric	$bsize - 1$	upper limits for box cone (may be Inf)
q	integer	≥ 1	vector of second-order cone sizes
s	integer	≥ 1	vector of positive semidefinite cone sizes
cs	integer	≥ 1	vector of complex 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

A real positive semidefinite cone of order k occupies $k(k+1)/2$ rows (lower triangle, column-wise, off-diagonal entries scaled by $\sqrt{2}$); a complex positive semidefinite cone of order k occupies k^2 rows (real and imaginary parts of the lower triangle, same scaling). The experimental spectral cones of the SCS C library (log-determinant, nuclear norm, ℓ_1, sum of largest eigenvalues) are not compiled into this package. The redundant length fields qsize, ssize, cssize and psize are accepted for backward compatibility and must equal the length of the vector they describe.

Return information: info is a list with the elements iter, status, lin_sys_solver, status_val, scale_updates, pobj, dobj, res_pri, res_dual, gap, res_infeas, res_unbdd_a, res_unbdd_p, setup_time, solve_time (milliseconds), scale, comp_slack, rejected_accel_steps, accepted_accel_steps, aa_stats, lin_sys_time, cone_time and accel_time (milliseconds). status_val is the SCS exit flag: 1 solved, 2 solved inaccurately, -1 unbounded, -2 infeasible, -4 failed, -5 interrupted, -6 unbounded inaccurately, -7 infeasible inaccurately. aa_stats is a list of Anderson acceleration diagnostics: iter, n_accept, n_reject_lapack, n_reject_rank0, n_reject_nonfinite, n_reject_weight_cap, n_safeguard_reject, last_rank, last_aa_norm (NaN if no acceleration step was attempted) and last_regularization.

Value

A list with the solution vectors x, y, s and a list info describing the run, see section "Return information".

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. The defaults are those of the bundled SCS C library (see `scs_version`), except `verbose`, which is `FALSE` here.

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 = 10L,
  acceleration_interval = 5L,
  adaptive_scale = TRUE,
  write_data_filename = NULL,
  log_csv_filename = NULL,
  time_limit_secs = 0,
  adaptive_diag_scale = TRUE,
  acceleration_type_1 = TRUE,
  acceleration_regularization = 1e-08,
  acceleration_relaxation = 1
)
```

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 primal scale factor (default `1e-6`).

`scale` a double giving the initial dual scale factor (default `0.1`), updated during the solve if `adaptive_scale` 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`; a call to `scs` with a non-null `initial` argument overrides it to be effectively `TRUE`).

`acceleration_lookback`
a non-negative integer giving how much memory to use for Anderson acceleration (AA). More memory requires more time to compute but can give more reliable steps; `0L` disables acceleration (default `10L`). The AA type is chosen by `acceleration_type_1`; a negative value, which earlier SCS versions read as a request for type-II AA, is an error.

`acceleration_interval`
an integer; AA is applied every `acceleration_interval` iterations (default `5L`).

`adaptive_scale` a logical indicating whether to heuristically adapt the dual scale 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).

`adaptive_diag_scale`
a logical indicating whether to refine the scaling metric per row from the primal residual profile; requires `adaptive_scale` and is silently disabled without it (default `TRUE`).

`acceleration_type_1`
a logical; `TRUE` selects type-I AA, `FALSE` type-II, which is more stable but typically slower and is usually run with a smaller `acceleration_regularization` such as `1e-12` (default `TRUE`).

`acceleration_regularization`
a double giving the Tikhonov regularization of the AA least-squares solve. A positive value is scaled by the Frobenius norms of the AA matrices, a negative value pins the absolute regularization at `-acceleration_regularization`, and `0` disables regularization (default `1e-8`).

`acceleration_relaxation`
a double in $[0, 2]$ giving the AA relaxation factor; `1.0` is vanilla AA (default `1.0`).

Value

a list containing the control parameters.

`scs_version` *Version of the Bundled SCS Library*

Description

Returns the version string of the SCS C library compiled into this package.

Usage

```
scs_version()
```

Value

a character string, for example `"3.3.1"`.

Examples

```
scs_version()
```

Index

scs, 2
scs_control, 4
scs_version, *4*, 6