Package 'osqp'

May 9, 2026

Title Quadratic Programming Solver using the 'OSQP' Library

Version 1.0.0

Date 2026-02-27

Copyright file COPYRIGHT

Description Provides bindings to the 'OSQP' solver. The 'OSQP' solver is a numerical optimization package for solving convex quadratic programs written in 'C' and based on the alternating direction method of multipliers. See <doi:10.48550/arXiv.1711.08013> for details.

License Apache License 2.0 | file LICENSE

SystemRequirements C++17, GNU make

Imports Rcpp (>= 0.12.14), methods, Matrix (>= 1.6.1), S7, cli

LinkingTo Rcpp

RoxygenNote 7.3.3

Collate 'RcppExports.R' 'osqp-package.R' 'sparse.R' 'solve.R' 'osqp.R' 'params.R'

NeedsCompilation yes

VignetteBuilder knitr

Suggests knitr, rmarkdown, slam, testthat

Encoding UTF-8

BugReports https://github.com/osqp/osqp-r/issues

URL https://osqp.org

Author Bartolomeo Stellato [aut, ctb, cph],
Goran Banjac [aut, ctb, cph],
Paul Goulart [aut, ctb, cph],
Stephen Boyd [aut, ctb, cph],
Eric Anderson [ctb],
Vineet Bansal [aut, ctb],
Balasubramanian Narasimhan [cre, aut]

Maintainer Balasubramanian Narasimhan <naras@stanford.edu>

Repository CRAN

Date/Publication 2026-03-01 10:00:02 UTC

Contents

`osqp` *OSQP Solver object*

Description

OSQP Solver object

Usage

```
osqp(P = NULL, q = NULL, A = NULL, l = NULL, u = NULL, pars = osqpSettings())
```

Arguments

`P`, `A`	sparse matrices of class dgCMatrix or coercible into such, with P positive semidefinite. (In the interest of efficiency, only the upper triangular part of P is used)
`q`, `l`, `u`	Numeric vectors, with possibly infinite elements in l and u
`pars`	list with optimization parameters, conveniently set with the function `osqpSettings`. For `model@UpdateSettings(newPars)` only a subset of the settings can be updated once the problem has been initialized.

Details

Allows one to solve a parametric problem with for example warm starts between updates of the parameter, c.f. the examples. The object returned by `osqp` contains several computed properties (accessed via `@`) which can be used to either update/get details of the problem, modify the optimization settings or attempt to solve the problem.

Value

An S7 object of class "OSQP_Model" with computed properties that return methods.

Usage

```
model = osqp(P=NULL, q=NULL, A=NULL, l=NULL, u=NULL, pars=osqpSettings())

model@Solve()
model@Update(q = NULL, l = NULL, u = NULL, Px = NULL, Px_idx = NULL, Ax = NULL, Ax_idx = NULL)
model@GetParams()
model@GetDims()
model@UpdateSettings(newPars = list())
```

```
model@GetData(element = c("P", "q", "A", "l", "u"))
model@WarmStart(x=NULL, y=NULL)
model@ColdStart()

print(model)
```

Method Arguments

element a string with the name of one of the matrices / vectors of the problem

newPars list with optimization parameters

See Also

solve_osqp

Examples

```
## example, adapted from OSQP documentation
library(Matrix)

P <- Matrix(c(11., 0.,
              0., 0.), 2, 2, sparse = TRUE)
q <- c(3., 4.)
A <- Matrix(c(-1., 0., -1., 2., 3.,
              0., -1., -3., 5., 4.)
              , 5, 2, sparse = TRUE)
u <- c(0., 0., -15., 100., 80)
l <- rep_len(-Inf, 5)

settings <- osqpSettings(verbose = FALSE)

model <- osqp(P, q, A, l, u, settings)

# Solve
res <- model@Solve()

# Define new vector
q_new <- c(10., 20.)

# Update model and solve again
model@Update(q = q_new)
res <- model@Solve()
```

---

osqpSettings *Settings for OSQP*

Description

For further details please consult the OSQP documentation: https://osqp.org/

Usage

```
osqpSettings(
  rho = 0.1,
  rho_is_vec = TRUE,
  sigma = 1e-06,
  max_iter = 4000L,
  eps_abs = 0.001,
  eps_rel = 0.001,
  eps_prim_inf = 1e-04,
  eps_dual_inf = 1e-04,
  alpha = 1.6,
  linsys_solver = c(OSQP_DIRECT_SOLVER = 1L),
  delta = 1e-06,
  polishing = FALSE,
  polish_refine_iter = 3L,
  verbose = TRUE,
  scaled_termination = FALSE,
  check_termination = 25L,
  check_dualgap = TRUE,
  warm_starting = TRUE,
  scaling = 10L,
  adaptive_rho = 1L,
  adaptive_rho_interval = 50L,
  adaptive_rho_tolerance = 5,
  adaptive_rho_fraction = 0.4,
  cg_max_iter = 20L,
  cg_tol_reduction = 10L,
  cg_tol_fraction = 0.15,
  cg_precond = c(OSQP_DIAGONAL_PRECONDITIONER = 1L),
  profiler_level = 0L,
  time_limit = 1e+10,
  polish = NULL,
  warm_start = NULL
)
```

Arguments

rho	ADMM step rho
rho_is_vec	boolean, whether rho is treated as a vector (per-constraint) or scalar
sigma	ADMM step sigma
max_iter	maximum iterations
eps_abs	absolute convergence tolerance
eps_rel	relative convergence tolerance
eps_prim_inf	primal infeasibility tolerance
eps_dual_inf	dual infeasibility tolerance
alpha	relaxation parameter

`linsys_solver` which linear systems solver to use, 1=OSQP_DIRECT_SOLVER (QDLDL), 2=OSQP_INDIRECT_SOLVER

`delta` regularization parameter for polishing

`polishing` boolean, polish ADMM solution

`polish_refine_iter` iterative refinement steps in polishing

`verbose` boolean, write out progress

`scaled_termination` boolean, use scaled termination criteria

`check_termination` integer, check termination interval. If 0, termination checking is disabled

`check_dualgap` boolean, check duality gap termination criteria

`warm_starting` boolean, warm start

`scaling` heuristic data scaling iterations. If 0, scaling disabled

`adaptive_rho` integer, rho adaptation strategy: 0=disabled, 1=iterations, 2=time, 3=KKT error

`adaptive_rho_interval` Number of iterations between rho adaptations rho. If 0, it is automatic

`adaptive_rho_tolerance` Tolerance X for adapting rho. The new rho has to be X times larger or 1/X times smaller than the current one to trigger a new factorization

`adaptive_rho_fraction` Interval for adapting rho (fraction of the setup time)

`cg_max_iter` maximum number of CG iterations (indirect solver only)

`cg_tol_reduction` integer, number of consecutive zero CG iterations before the tolerance gets halved (indirect solver only)

`cg_tol_fraction` CG tolerance fraction (indirect solver only)

`cg_precond` preconditioner for CG method (indirect solver only): 0=none, 1=diagonal (Jacobi)

`profiler_level` integer, level of detail for profiler annotations (0=off)

`time_limit` run time limit in seconds (1e10 effectively disables)

`polish` Deprecated. Use `polishing` instead.

`warm_start` Deprecated. Use `warm_starting` instead.

`solve_osqp` *Sparse Quadratic Programming Solver*

Description

Solves

$$arg\min_x 0.5x'Px + q'x$$

s.t.

$$l_i < (Ax)_i < u_i$$

for real matrices P (nxn, positive semidefinite) and A (mxn) with m number of constraints

Usage

```
solve_osqp(
  P = NULL,
  q = NULL,
  A = NULL,
  l = NULL,
  u = NULL,
  pars = osqpSettings()
)
```

Arguments

`P`, `A`	sparse matrices of class dgCMatrix or coercible into such, with P positive semidefinite. Only the upper triangular part of P will be used.
`q`, `l`, `u`	Numeric vectors, with possibly infinite elements in l and u
`pars`	list with optimization parameters, conveniently set with the function `osqpSettings`

Value

A list with elements x (the primal solution), y (the dual solution), prim_inf_cert, dual_inf_cert, and info.

References

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd and S. (2018). "OSQP: An Operator Splitting Solver for Quadratic Programs." *ArXiv e-prints*. 1711.08013.

See Also

`osqp`. The underlying OSQP documentation: `https://osqp.org/`

Examples

```
library(osqp)
## example, adapted from OSQP documentation
library(Matrix)

P <- Matrix(c(11., 0.,
              0., 0.), 2, 2, sparse = TRUE)
q <- c(3., 4.)
A <- Matrix(c(-1., 0., -1., 2., 3.,
              0., -1., -3., 5., 4.)
               , 5, 2, sparse = TRUE)
u <- c(0., 0., -15., 100., 80)
l <- rep_len(-Inf, 5)

settings <- osqpSettings(verbose = TRUE)

# Solve with OSQP
res <- solve_osqp(P, q, A, l, u, settings)
res$x
```

Index