Package 'rosqp'

February 7, 2018

Type Package

Title Quadratic Programming Solver using the 'OSQP' Library

Version 0.1.0

Date 2018-01-19

Copyright file COPYRIGHT

Description Provides bindings to the 'OSQP' solver, which can solve sparse convex quadratic programming problems with optional equality and inequality constraints.

License Apache License 2.0

Imports Rcpp (>= 0.12.14), methods, Matrix, R6

LinkingTo Rcpp

RoxygenNote 6.0.1

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

NeedsCompilation yes

Author Eric Anderson [aut, cre],
Bartolomeo Stellato [ctb, cph] (OSQP),
Goran Banjac [ctb, cph] (OSQP),
Paul Goulart [ctb, cph] (OSQP),
Stephen Boyd [ctb, cph] (OSQP),
Patrick R. Amestoy [ctb, cph] (SuiteSparse AMD),
Iain S. Duff [ctb, cph] (SuiteSparse AMD),
Timothy A. Davis [ctb, cph] (SuiteSparse LDL, SuiteSparse AMD),
John K. Reid [ctb, cph] (SuiteSparse AMD)

Maintainer Eric Anderson <anderic1@gmx.com>

Repository CRAN

Date/Publication 2018-02-07 13:10:50 UTC

R topics documented:

`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.
`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 `osqpObject$UpdateSettings(newPars)` only a subset of the settings can be updated once the problem has been initnialized.

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 methods which can be used to either update/get details of the problem, modify the optimization settings or attempt to solve the problem.

Value

An R6-object of class "rosqp_model" with methods defined which can be further used to solve the problem with updated settings / parameters.

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)
model$GetParams()
model$GetDims()
model$UpdateSettings(newPars = list())

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

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 the osqp documentation
## Not run:
library(rosqp)
library(Matrix)
set.seed(1)
n = 10
m = 1000
Ad = matrix(0, m, n)
Ad[sample(n*m, n*m/2, FALSE)] = runif(n*m/2)
x_true = (runif(n) > 0.8) * runif(n) / sqrt(n)
b = drop(Ad %*% x_true) + 0.5 * runif(m)
gammas = seq(1, 10, length.out = 11)

# % OSQP data
P = .sparseDiagonal(2*n+m, c(numeric(n), rep_len(1, m), numeric(n)))
q = numeric(2*n+m);
A = rbind(cbind(Ad,
                -Diagonal(m),
                sparseMatrix(numeric(), numeric(), x=numeric(), dims=c(m, n))),
          cbind(Diagonal(n),
                sparseMatrix(numeric(), numeric(), x=numeric(), dims=c(n, m)),
                -Diagonal(n)),
          cbind(Diagonal(n),
                sparseMatrix(numeric(), numeric(), x=numeric(), dims=c(n, m)),
                Diagonal(n))
          )
l = c(b, rep_len(-Inf, n), numeric(n))
u = c(b, numeric(n), rep_len(Inf, n))

model = osqp(P, q, A, l, u, osqpSettings(verbose = FALSE))

res = sapply(gammas, function(gamma) {
  q_new = c(numeric(n+m), rep_len(gamma, n))
  model$Update(q=q_new)
  res = model$Solve()
  res$x
})

## End(Not run)
```

`osqpSettings` *Settings for OSQP*

Description

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

Usage

```
osqpSettings(rho = 0.1, 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(SUITESPARSE_LDL_SOLVER = 0L), delta = 1e-06,
  polish = FALSE, polish_refine_iter = 3L, verbose = TRUE,
  scaled_termination = FALSE, check_termination = 25L, warm_start = TRUE,
  scaling = 10L, adaptive_rho = 1L, adaptive_rho_interval = 0L,
  adaptive_rho_tolerance = 5, adaptive_rho_fraction = 0.4)
```

Arguments

`rho`	ADMM step rho
`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, 0=Suitesparse LDL, 1=MKL Pardiso
`delta`	regularization parameter for polish
`polish`	boolean, polish ADMM solution
`polish_refine_iter`	iterative refinement steps in polish
`verbose`	boolean, write out progres
`scaled_termination`	boolean, use scaled termination criteria
`check_termination`	integer, check termination interval. If 0, termination checking is disabled
`warm_start`	boolean, warm start
`scaling`	heuristic data scaling iterations. If 0, scaling disabled
`adaptive_rho`	cboolean, is rho step size adaptive?

`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)

`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.
`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. (2017). "OSQP: An Operator Splitting Solver for Quadratic Programs." *ArXiv e-prints*. 1711.08013.

See Also

`osqp`. The underlying OSQP documentation: `http://osqp.readthedocs.io/`

Examples

```
library(rosqp)
## example, adapted from ?quadprog::solve.QP
Dmat       <- diag(3)
dvec       <- c(0,-5,0)
Amat       <- matrix(c(-4, 2, 0, -3, 1, -2, 0, 0, 1),3,3)
bvec       <- c(-8,2,0)
res = solve_osqp(Dmat, dvec, Amat, bvec)
print(res$x)
```

Index