**osqp**

*OSQP Solver object*

**Description**

OSQP Solver object

**Usage**

```r
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 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 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 "osqp_model" with methods defined which can be further used to solve the problem with updated settings / parameters.

**Usage**

```r
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)
print(model)
```
osqpSettings

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

```r
## 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/](https://osqp.org/)

Usage

```r
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(QDLDL_SOLVER =
osqpSettings

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=QDLDL, 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 progress
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 boolean, 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

\[
\arg \min_x 0.5x'Px + q'x \\
\text{s.t.} \\
l_i < (Ax)_i < u_i
\]

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

Usage

\[
solve_osqp(P = \text{NULL}, q = \text{NULL}, A = \text{NULL}, l = \text{NULL}, u = \text{NULL}, \text{pars} = \text{osqpSettings}())
\]

Arguments

- \(P, A\): sparse matrices of class \text{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
- \(\text{pars}\): list with optimization parameters, conveniently set with the function \text{osqpSettings}()

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

References

See Also

osqp. The underlying OSQP documentation: \text{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., 0., 0., 0., 0.))
```r
0., -1., -3., 5., 4.)
      5, 2, sparse = TRUE)
u <- c(0., 0., -15., 100., 80)
l <- rep_len(-Inf, 5)

settings <- osqpSettings(verbos = TRUE)

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

osqp, 2, 5
osqpSettings, 2, 3
solve_osqp, 3, 5