Package 'piqp'

May 9, 2026

Title R Interface to Proximal Interior Point Quadratic Programming Solver

Version 0.6.2

Description An embedded proximal interior point quadratic programming solver, which can solve dense and sparse quadratic programs, described in Schwan, Jiang, Kuhn, and Jones (2023) <doi:10.48550/arXiv.2304.00290>. Combining an infeasible interior point method with the proximal method of multipliers, the algorithm can handle ill-conditioned convex quadratic programming problems without the need for linear independence of the constraints. The solver is written in header only 'C++ 14' leveraging the 'Eigen' library for vectorized linear algebra. For small dense problems, vectorized instructions and cache locality can be exploited more efficiently. Allocation free problem updates and re-solves are also provided.

License BSD_2_clause + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL https://predict-epfl.github.io/piqp-r/

BugReports https://github.com/PREDICT-EPFL/piqp-r/issues

SystemRequirements C++17

LinkingTo Rcpp, RcppEigen

Suggests knitr, rmarkdown, slam, tinytest

VignetteBuilder knitr

Imports Matrix, methods, Rcpp, S7

NeedsCompilation yes

Author Balasubramanian Narasimhan [aut, cre],
Roland Schwan [aut, cph],
Yuning Jiang [aut],
Daniel Kuhn [aut],
Colin N. Jones [aut]

Maintainer Balasubramanian Narasimhan <naras@stanford.edu>

Repository CRAN

Date/Publication 2026-02-18 20:20:03 UTC

Contents

`get_dims` *Get dimensions of a PIQP model*

Description

Get dimensions of a PIQP model

Usage

```
get_dims(model, ...)
```

Arguments

`model`	A `piqp_model` object
`...`	not used

Value

a list with named elements `n`, `p`, and `m`

`get_settings` *Get settings of a PIQP model*

Description

Get settings of a PIQP model

Usage

```
get_settings(model, ...)
```

Arguments

model	A piqp_model object
...	not used

Value

a list of current settings

---

make_csc_matrix *Convert a plain matrix or simple triplet form matrix to a Matrix::dgCMatrix (implicit) form*

Description

Convert a plain matrix or simple triplet form matrix to a Matrix::dgCMatrix (implicit) form

Usage

```
make_csc_matrix(x)
```

Arguments

x	a matrix or a simple triplet form matrix

Value

a list of row pointer, column pointer, and values corresponding to a Matrix::dgCMatrix object

---

make_csc_symm_matrix *Convert a plain matrix or simple triplet form matrix to a Matrix::dsCMatrix (implicit, upper) form*

Description

Convert a plain matrix or simple triplet form matrix to a Matrix::dsCMatrix (implicit, upper) form

Usage

```
make_csc_symm_matrix(m)
```

Arguments

m	a matrix or a simple triplet form matrix

Value

a list of row pointer, column pointer, and values corresponding to a Matrix::dsCMatrix object

piqp *PIQP Solver object*

Description

PIQP Solver object

Usage

```
piqp(
  P = NULL,
  c = NULL,
  A = NULL,
  b = NULL,
  G = NULL,
  h_l = NULL,
  h_u = NULL,
  x_l = NULL,
  x_u = NULL,
  settings = list(),
  backend = c("auto", "sparse", "dense")
)
```

Arguments

P	dense or sparse matrix of class dgCMatrix or coercible into such, must be positive semidefinite
c	numeric vector
A	dense or sparse matrix of class dgCMatrix or coercible into such
b	numeric vector
G	dense or sparse matrix of class dgCMatrix or coercible into such
h_l	numeric vector of lower inequality bounds, default NULL indicating -Inf for all inequality constraints
h_u	numeric vector of upper inequality bounds, default NULL indicating Inf for all inequality constraints
x_l	a numeric vector of lower variable bounds, default NULL indicating -Inf for all variables
x_u	a numeric vector of upper variable bounds, default NULL indicating Inf for all variables
settings	list with optimization parameters, empty by default; see piqp_settings() for a comprehensive list of parameters that may be used
backend	which backend to use, if auto and P, A or G are sparse then sparse backend is used ("auto", "sparse" or "dense") ("auto")

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 `piqp` 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 S7 object of class "piqp_model" with methods `solve()`, `update()`, `get_settings()`, `get_dims()`, `update_settings()` which can be used to solve the problem with updated settings / parameters.

Usage

```
model = piqp(P = NULL, c = NULL, A = NULL, b = NULL, G = NULL,
             h_l = NULL, h_u = NULL, x_l = NULL, x_u = NULL,
             settings = piqp_settings(),
             backend = c("auto", "sparse", "dense"))

solve(model)
update(model, P = NULL, c = NULL, A = NULL, b = NULL, G = NULL,
       h_l = NULL, h_u = NULL, x_l = NULL, x_u = NULL)
get_settings(model)
get_dims(model)
update_settings(model, new_settings = piqp_settings())
```

See Also

`solve_piqp()`, `piqp_settings()`

Examples

```
## example, adapted from PIQP documentation
library(piqp)
library(Matrix)

P <- Matrix(c(6., 0.,
              0., 4.), 2, 2, sparse = TRUE)
c <- c(-1., -4.)
A <- Matrix(c(1., -2.), 1, 2, sparse = TRUE)
b <- c(1.)
G <- Matrix(c(1., 2., -1., 0.), 2, 2, sparse = TRUE)
h_u <- c(0.2, -1.)
x_l <- c(-1., -Inf)
x_u <- c(1., Inf)

settings <- list(verbose = TRUE)

model <- piqp(P, c, A, b, G, h_u = h_u, x_l = x_l, x_u = x_u, settings = settings)
```

```
# Solve
res <- solve(model)
res$x

# Define new data
A_new <- Matrix(c(1., -3.), 1, 2, sparse = TRUE)
h_u_new <- c(2., 1.)

# Update model and solve again
update(model, A = A_new, h_u = h_u_new)
res <- solve(model)
res$x
```

`piqp_model` *The PIQP Solver Model Class*

Description

An S7 class wrapping the PIQP C++ Solver. Users will never need to directly create instances of this class and should use the more user-friendly functions `piqp()` and `solve_piqp()`.

Usage

```
piqp_model(solver_ptr = NULL, dims = list(), dense_backend = logical(0))
```

Arguments

`solver_ptr`	external pointer to the C++ solver object
`dims`	a named list with elements `n`, `p`, and `m`
`dense_backend`	logical flag indicating if the dense solver is used

`piqp_settings` *Settings parameters with default values and types in parenthesis*

Description

Settings parameters with default values and types in parenthesis

Usage

```
piqp_settings(
  rho_init = 1e-06,
  delta_init = 1e-04,
  eps_abs = 1e-08,
  eps_rel = 1e-09,
  check_duality_gap = TRUE,
  eps_duality_gap_abs = 1e-08,
  eps_duality_gap_rel = 1e-09,
  infeasibility_threshold = 0.9,
  reg_lower_limit = 1e-10,
  reg_finetune_lower_limit = 1e-13,
  reg_finetune_primal_update_threshold = 7L,
  reg_finetune_dual_update_threshold = 7L,
  max_iter = 250L,
  max_factor_retires = 10L,
  preconditioner_scale_cost = FALSE,
  preconditioner_reuse_on_update = FALSE,
  preconditioner_iter = 10L,
  tau = 0.99,
  iterative_refinement_always_enabled = FALSE,
  iterative_refinement_eps_abs = 1e-12,
  iterative_refinement_eps_rel = 1e-12,
  iterative_refinement_max_iter = 10L,
  iterative_refinement_min_improvement_rate = 5,
  iterative_refinement_static_regularization_eps = 1e-08,
  iterative_refinement_static_regularization_rel = .Machine$double.eps^2,
  verbose = FALSE,
  compute_timings = FALSE
)
```

Arguments

`rho_init`	Initial value for the primal proximal penalty parameter rho (default = 1e-6)
`delta_init`	Initial value for the augmented lagrangian penalty parameter delta (default = 1e-4)
`eps_abs`	Absolute tolerance (default = 1e-8)
`eps_rel`	Relative tolerance (default = 1e-9)
`check_duality_gap`	Check terminal criterion on duality gap (default = TRUE)
`eps_duality_gap_abs`	Absolute tolerance on duality gap (default = 1e-8)
`eps_duality_gap_rel`	Relative tolerance on duality gap (default = 1e-9)
`infeasibility_threshold`	Threshold for infeasibility detection (default = 0.9)

`reg_lower_limit`
: Lower limit for regularization (default = 1e-10)

`reg_finetune_lower_limit`
: Fine tune lower limit regularization (default = 1e-13)

`reg_finetune_primal_update_threshold`
: Threshold of number of no primal updates to transition to fine tune mode (default = 7)

`reg_finetune_dual_update_threshold`
: Threshold of number of no dual updates to transition to fine tune mode (default = 7)

`max_iter`
: Maximum number of iterations (default = 250)

`max_factor_retires`
: Maximum number of factorization retires before failure (default = 10)

`preconditioner_scale_cost`
: Scale cost in Ruiz preconditioner (default = FALSE)

`preconditioner_reuse_on_update`
: Reuse preconditioner on problem update (default = FALSE)

`preconditioner_iter`
: Maximum of preconditioner iterations (default = 10)

`tau`
: Maximum interior point step length (default = 0.99)

`iterative_refinement_always_enabled`
: Always run iterative refinement and not only on factorization failure (default = FALSE)

`iterative_refinement_eps_abs`
: Iterative refinement absolute tolerance (default = 1e-12)

`iterative_refinement_eps_rel`
: Iterative refinement relative tolerance (default = 1e-12)

`iterative_refinement_max_iter`
: Maximum number of iterations for iterative refinement (default = 10)

`iterative_refinement_min_improvement_rate`
: Minimum improvement rate for iterative refinement (default = 5.0)

`iterative_refinement_static_regularization_eps`
: Static regularization for KKT system for iterative refinement (default = 1e-8)

`iterative_refinement_static_regularization_rel`
: Static regularization w.r.t. the maximum abs diagonal term of KKT system. (default = .Machine$double.eps^2)

`verbose`
: Verbose printing (default = FALSE)

`compute_timings`
: Measure timing information internally (default = FALSE)

Value

a list containing the settings parameters.

`solve_piqp` *PIQP Solver*

Description

Solves

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

s.t.

$$Ax = b$$

$$h_l \leq Gx \leq h_u$$

$$x_l \leq x \leq x_u$$

for real matrices P (nxn, positive semidefinite), A (pxn) with p number of equality constraints, and G (mxn) with m number of inequality constraints

Usage

```
solve_piqp(
  P = NULL,
  c = NULL,
  A = NULL,
  b = NULL,
  G = NULL,
  h_l = NULL,
  h_u = NULL,
  x_l = NULL,
  x_u = NULL,
  settings = list(),
  backend = c("auto", "sparse", "dense")
)
```

Arguments

`P`	dense or sparse matrix of class dgCMatrix or coercible into such, must be positive semidefinite
`c`	numeric vector
`A`	dense or sparse matrix of class dgCMatrix or coercible into such
`b`	numeric vector
`G`	dense or sparse matrix of class dgCMatrix or coercible into such
`h_l`	numeric vector of lower inequality bounds, default `NULL` indicating `-Inf` for all inequality constraints
`h_u`	numeric vector of upper inequality bounds, default `NULL` indicating `Inf` for all inequality constraints

x_l	a numeric vector of lower variable bounds, default NULL indicating -Inf for all variables
x_u	a numeric vector of upper variable bounds, default NULL indicating Inf for all variables
settings	list with optimization parameters, empty by default; see piqp_settings() for a comprehensive list of parameters that may be used
backend	which backend to use, if auto and P, A or G are sparse then sparse backend is used ("auto", "sparse" or "dense") ("auto")

Value

A list with elements solution elements

References

Schwan, R., Jiang, Y., Kuhn, D., Jones, C.N. (2023). "PIQP: A Proximal Interior-Point Quadratic Programming Solver." doi:10.48550/arXiv.2304.00290

See Also

piqp(), piqp_settings() and the underlying PIQP documentation: https://predict-epfl.github.io/piqp/

Examples

```
## example, adapted from PIQP documentation
library(piqp)
library(Matrix)

P <- Matrix(c(6., 0.,
              0., 4.), 2, 2, sparse = TRUE)
c <- c(-1., -4.)
A <- Matrix(c(1., -2.), 1, 2, sparse = TRUE)
b <- c(1.)
G <- Matrix(c(1., 2., -1., 0.), 2, 2, sparse = TRUE)
h_u <- c(0.2, -1.)
x_l <- c(-1., -Inf)
x_u <- c(1., Inf)

settings <- list(verbose = TRUE)

# Solve with PIQP
res <- solve_piqp(P, c, A, b, G, h_u = h_u, x_l = x_l, x_u = x_u, settings = settings)
res$x
```

status_description *Return the solver status description string*

Description

Return the solver status description string

Usage

```
status_description(code)
```

Arguments

code	a valid solver return code

Value

a status description string

Examples

```
status_description(1) ## for solved problem
status_description(-1) ## for max iterations limit reached
```

update_settings *Update settings of a PIQP model*

Description

Update settings of a PIQP model

Usage

```
update_settings(model, ...)
```

Arguments

model	A piqp_model object
...	not used

Value

invisible NULL

Index