Package ‘piqp’

August 14, 2023

Title R Interface to Proximal Interior Point Quadratic Programming Solver

Version 0.2.2

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 7.2.3

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

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

LinkingTo Rcpp, RcppEigen

Suggests knitr, rmarkdown, slam, tinytest

VignetteBuilder knitr

Imports Matrix, methods, R6, Rcpp

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 2023-08-14 09:30:02 UTC

R topics documented:

`piqp-package` *R Interface to PIQP Solver*

Description

PIQP is an Proximal Interior Point Quadratic Programming solver, which can solve dense and sparse quadratic programs described in described in Schwan, Jiang, Kuhn, and Jones (2023) (https://arxiv.org/abs/2304.00290). Combining an infeasible interior point method with the proximal method of multipliers, the algorithm can handle ill-conditioned convex QP 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.

Author(s)

Balasubramanian Narasimhan, Roland Schwan (C), Yuning Jiang, Daniel Kuhn, Colin N. Jones

---

`piqp` *PIQP Solver object*

Description

PIQP Solver object

Usage

```
piqp(
  P = NULL,
  c = NULL,
  A = NULL,
  b = NULL,
  G = NULL,
  h = NULL,
  x_lb = NULL,
```

```
  x_ub = 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`	numeric vector
`x_lb`	a numeric vector of lower bounds, default `NULL` indicating `-Inf` for all variables, otherwise should be number of variables long
`x_ub`	a numeric vector of upper bounds, default `NULL` indicating `Inf` for all variables, otherwise should be number of variables long
`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 R6-object of class "piqp_model" with methods defined which can be further used to solve the problem with updated settings / parameters.

Usage

```
model = piqp(P = NULL, c = NULL, A = NULL, b = NULL, G = NULL, h = NULL, x_lb = NULL, x_ub = NULL, settings

model$solve()
model$update(P = NULL, c = NULL, A = NULL, b = NULL, G = NULL, h = NULL, x_lb = NULL, x_ub = NULL)
model$get_settings()
model$get_dims()
model$update_settings(new_settings = piqp_settings())

print(model)
```

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 <- c(0.2, -1.)
x_lb <- c(-1., -Inf)
x_ub <- c(1., Inf)

settings <- list(verbose = TRUE)

model <- piqp(P, c, A, b, G, h, x_lb, x_ub, settings)

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

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

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

---

piqp_model — *The PIQP Solver Model Class*

Description

This class wraps around the PIQP C++ Solver and exposes methods and fields of the C++ object. Users will never need to directly create instances this class and should use the more user-friendly functions piqp() and solve_piqp().

Methods

Public methods:

- piqp_model$new()
- piqp_model$solve()
- piqp_model$update()
- piqp_model$get_settings()
- piqp_model$get_dims()
- piqp_model$update_settings()
- piqp_model$clone()

Method `new()`**:** Create a new piqp_model object

Usage:

```
piqp_model$new(
  P,
  c,
  A,
  b,
  G,
  h,
  x_lb,
  x_ub,
  settings = list(),
  dense_backend,
  dims
)
```

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` numeric vector

`x_lb` a numeric vector of lower bounds

`x_ub` a numeric vector of upper bounds

`settings` list with optimization parameters

`dense_backend` a flag indicating if the dense solver is to be used

`dims` the dimensions of the problem, a named list containing `n`, `p` and `m`.

Returns: a piqp_model object that can be used to solve the QP

Method `solve()`**:** Solve the QP model

Usage:

```
piqp_model$solve()
```

Returns: a list containing the solution

Method `update()`: Update the current piqp_model with new data

Usage:

```
piqp_model$update(
  P = NULL,
  c = NULL,
  A = NULL,
  b = NULL,
  G = NULL,
  h = NULL,
  x_lb = NULL,
  x_ub = NULL
)
```

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` numeric vector

`x_lb` a numeric vector of lower bounds

`x_ub` a numeric vector of upper bounds

`settings` list with optimization parameters

`dense_backend` a flag indicating if the dense solver is to be used

`dims` the dimensions of the problem, a named list containing `n`, `p` and `m`.

Method `get_settings()`: Obtain the current settings for this model

Usage:

```
piqp_model$get_settings()
```

Method `get_dims()`: Obtain the dimensions of this model

Usage:

```
piqp_model$get_dims()
```

Method `update_settings()`: Update the current settings with new values for this model

Usage:

```
piqp_model$update_settings(new_settings = list())
```

Arguments:

`new_settings` a list of named values for settings, default empty list; see `piqp_settings()` for a comprehensive list of defaults

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
piqp_model$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

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,
  reg_lower_limit = 1e-10,
  reg_finetune_lower_limit = 1e-13,
  reg_finetune_primal_update_threshold = 7L,
  reg_finetune_dual_update_threshold = 5L,
  max_iter = 250L,
  max_factor_retires = 10L,
  preconditioner_scale_cost = 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-07,
  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)

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 = 5)

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_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-7)

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.

$$Ax = b$$

$$Gx \leq h$$

$$x_{lb} \leq x \leq x_{ub}$$

for real matrices P (nxn, positive semidefinite), A (pxn) with m 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 = NULL,
  x_lb = NULL,
  x_ub = 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`	numeric vector
`x_lb`	a numeric vector of lower bounds, default `NULL` indicating `-Inf` for all variables, otherwise should be number of variables long

x_ub — a numeric vector of upper bounds, default NULL indicating Inf for all variables, otherwise should be number of variables long

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 <- c(0.2, -1.)
x_lb <- c(-1., -Inf)
x_ub <- c(1., Inf)

settings <- list(verbose = TRUE)

# Solve with PIQP
res <- solve_piqp(P, c, A, b, G, h, x_lb, x_ub, 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
```

Index