Package 'convexjlr'

December 16, 2018

Type Package

Title Disciplined Convex Programming in R using 'Convex.jl'

Version 0.8.1

Date 2018-12-16

Description Provides a simple high-level wrapper for
'Julia' package 'Convex.jl' (see <https://github.com/JuliaOpt/Convex.jl> for
more information),
which makes it easy to describe and solve convex optimization problems in R.
The problems can be dealt with include:
linear programs,
second-order cone programs,
semidefinite programs,
exponential cone programs.

Depends R (>= 3.4.0)

License Apache License | file LICENSE

Encoding UTF-8

LazyData true

Imports JuliaCall (>= 0.12.1), magrittr

RoxygenNote 6.1.1

Suggests testthat, knitr, rmarkdown, plotrix

VignetteBuilder knitr

SystemRequirements Julia (>= 0.6.0), Convex.jl, SCS.jl, ECOS.jl

URL https://github.com/Non-Contradiction/convexjlr

NeedsCompilation no

Author Changcheng Li [aut, cre]

Maintainer Changcheng Li <cxl508@psu.edu>

Repository CRAN

Date/Publication 2018-12-16 21:30:03 UTC

R topics documented:

addConstraint *Add constraints to optimization problem*

Description

addConstraint add additional constraints to optimization problem.

Usage

```
addConstraint(p, ...)
```

Arguments

p	optimization problem to add constraints.
...	additional constraints.

Value

the optimization problem with the additional constraints.

Examples

```
## Not run:
    convex_setup()
    x <- Variable(4)
    b <- J(c(1:4))
    p <- minimize(sum((x - b) ^ 2))
    p <- addConstraint(p, x >= 0, x <= 3)

## End(Not run)
```

convex_setup *Doing the setup for the package convexjlr*

Description

This function does the setup for the package convexjlr. Firstly it will try to establish the connect to Julia via the XRJulia interface, or try to embed julia in R through JuliaCall. Secondly it will check for the Julia packages Convex and SCS, if the packages are not found, it tries to install them into Julia. Finally, it will try to load the Julia packages and do the necessary initial setup.

Usage

```
convex_setup(backend = c("JuliaCall"), JULIA_HOME = NULL)
```

Arguments

backend	the backend to use, only JuliaCall is supported currently.
JULIA_HOME	the path to julia binary, if not set, convexjlr will try to use the julia in path.

Examples

```
## Not run:
convex_setup()

## End(Not run)
```

---

cvx_optim *Solve optimization problem*

Description

cvx_optim solves optimization problem using Convex.jl.

Usage

```
cvx_optim(p, solver = c("SCS", "ECOS"), ...)
```

Arguments

p	optimization problem to be solved.
solver	convex problem solver to be used. Currently convexjlr supports SCS and ECOS, with SCS solver as the default.
...	the optional solver options, like the maximal iteration times. For the solver options, you can see http://www.cvxpy.org/tutorial/advanced/index.html#setting-solver-options for reference.

Value

status of optimized problem.

Examples

```
## Not run:
    convex_setup()
    x <- Variable()
    b <- 1
    p <- minimize(sum((x - b) ^ 2))
    cvx_optim(p)

## End(Not run)
```

dot *Inner product*

Description

Inner product of two input vectors.

Usage

```
dot(x, y)
```

Arguments

x	input vector, one input vector need to be constant.
y	input vector, one input vector need to be constant.

dotsort *Inner product of two vectors after sorted*

Description

Inner product of two input vectors after sorted.

Usage

```
dotsort(x, y)
```

Arguments

x	input vector, one input vector needs to be constant.
y	input vector, one input vector needs to be constant.

entropy *sum(-x * log(x))*

Description

sum(-x * log(x)).

Usage

```
entropy(x)
```

Arguments

x	input vector or matrix, x > 0.

Expr *Create expressions to be used for optimization problem creation*

Description

Expr create expressions, which can be used later for problem creation.

Usage

```
Expr(x)
```

Arguments

x	expression to be created.

Examples

```
## Not run:
    convex_setup()
    x <- Variable(2)
    x1 <- Expr(x + 1)

## End(Not run)
```

geomean *Geometric mean of x and y*

Description

Geometric mean of x and y.

Usage

```
geomean(x, y)
```

Arguments

x	input vector, x > 0.
y	input vector, y > 0.

huber *Huber loss*

Description

Huber loss.

Usage

```
huber(x, M = 1)
```

Arguments

x	input vector.
M	M >= 1.

J *Make a variable to be of Julia's awareness*

Description

Make a variable to be of Julia's awareness, so it can be further used in the definition of optimization problem.

Usage

```
J(x)
```

Arguments

x	the R object sent to Julia

Examples

```
## Not run:
    convex_setup()
    b <- J(c(1:2))

## End(Not run)
```

lambdamax *Largest eigenvalues of x*

Description

Largest eigenvalues of x.

Usage

```
lambdamax(x)
```

Arguments

x	input matrix.

lambdamin *Smallest eigenvalues of x*

Description

Smallest eigenvalues of x.

Usage

```
lambdamin(x)
```

Arguments

x	input matrix.

logdet *Log of determinant of x*

Description

Log of determinant of x.

Usage

```
logdet(x)
```

Arguments

x	input matrix, needs to be positive semidefinite.

logisticloss *log(1 + exp(x))*

Description

log(1 + exp(x)).

Usage

```
logisticloss(x)
```

Arguments

x	input vector.

logsumexp *log(sum(exp(x)))*

Description

log(sum(exp(x))).

Usage

```
logsumexp(x)
```

Arguments

x	input vector.

matrixfrac *x^T P^-1 x*

Description

x^T P^-1 x.

Usage

```
matrixfrac(x, P)
```

Arguments

x	input vector.
P	input matrix, needs to be positive semidefinite.

maximum *Largest elements*

Description

Largest elements of input vector x.

Usage

```
maximum(x)
```

Arguments

x	input vector.

minimum *Smallest elements*

Description

Smallest elements of input vector x.

Usage

```
minimum(x)
```

Arguments

x	input vector.

neg *Negative parts*

Description

Negative parts of input vector x.

Usage

```
neg(x)
```

Arguments

x	input vector.

norm — *p-norm of x*

Description

p-norm of x.

Usage

```
norm(x, p = 2)
```

Arguments

x	input vector.
p	a number greater than 1.

nuclearnorm — *Sum of singular values of x*

Description

Sum of singular values of x.

Usage

```
nuclearnorm(x)
```

Arguments

x	input matrix.

operatornorm — *Largest singular value of x*

Description

Largest singular value of x.

Usage

```
operatornorm(x)
```

Arguments

x	input matrix.

pos *Positive parts*

Description

Positive parts of input vector x.

Usage

```
pos(x)
```

Arguments

x	input vector.

problem_creating *Create optimization problem*

Description

Create different kinds of optimization problems with targets and constraints.

Usage

```
minimize(...)

maximize(...)

satisfy(...)
```

Arguments

...	optimization targets and constraints.

Examples

```
## Not run:
    convex_setup()
    x <- Variable(4)
    b <- J(c(1:4))
    p <- minimize(sum((x - b) ^ 2), x >= 0, x <= 3)
    p <- maximize(-sum((x - b) ^ 2), x >= 0, x <= 3)
    p <- satisfy(sum((x - b) ^ 2) <= 1, x >= 0, x <= 3)

## End(Not run)
```

property *Get properties of optimization problem*

Description

Get properties of solved optimization problem, like the status of problem (optimal, infeasible and etc.), or the optimal value of the solved optimization problem.

Usage

```
status(p)

optval(p)
```

Arguments

p	optimization problem.

Examples

```
## Not run:
    convex_setup()
    x <- Variable(2)
    b <- J(c(1:2))
    p <- minimize(sum((x - b) ^ 2))
    cvx_optim(p)
    status(p)
    optval(p)

## End(Not run)
```

quadform *x^T P x*

Description

x^T P x.

Usage

```
quadform(x, P)
```

Arguments

x	input vector, either x or P must be constant.
P	input matrix, either x or P must be constant, P needs to be semidefinte if x is not constant.

square *Square of x*

Description

Square of x.

Usage

```
square(x)
```

Arguments

x	input vector.

sumlargest *Sum of the largest elements*

Description

Sum of k largest elements of input vector x.

Usage

```
sumlargest(x, k)
```

Arguments

x	input vector.
k	a positive integer.

sumsmallest *Sum of the smallest elements*

Description

Sum of k smallest elements of input vector x.

Usage

```
sumsmallest(x, k)
```

Arguments

x	input vector.
k	a positive integer.

sumsquares *Sum of squares of x*

Description

Sum of squares of x.

Usage

```
sumsquares(x)
```

Arguments

x	input vector.

tr *Trace of matrix*

Description

Trace of input matrix x.

Usage

```
tr(x)
```

Arguments

x	input matrix.

value *Get values of expressions at optimizer*

Description

Value returns the values of expressions at optimizer (minimizer, maximizer and etc.).

Usage

```
value(...)
```

Arguments

...	expressions needed to evaluate.

Examples

```
## Not run:
    convex_setup()
    x <- Variable(4)
    b <- J(c(1:4))
    p <- minimize(sum((x - b) ^ 2))
    cvx_optim(p)
    value(x[1] + x[2], x[3] + x[4])

## End(Not run)
```

`variable_creating` *Create variable for optimization problem*

Description

Create variable (vector, matrix, semidefinite matrix and etc.) for optimization problem.

Usage

```
Variable(size = 1, sign = c("None", "Positive", "Negative"))

Semidefinite(size = 1, sign = c("None", "Positive", "Negative"))
```

Arguments

`size`	variable size.
`sign`	whether variable is element-wise positive, element-wise negative or neither.

Examples

```
## Not run:
    convex_setup()
    x <- Variable(4)
    X <- Variable(c(4, 4), sign = "Positive")
    S <- Semidefinite(4)

## End(Not run)
```

vec — *Vector representation*

Description

Vector representation of input matrix x.

Usage

```
vec(x)
```

Arguments

x	input matrix.

vecdot — *Inner product of vector representation of two matrices*

Description

Inner product of vector representation of two input matrices.

Usage

```
vecdot(x, y)
```

Arguments

x	input matrix, one input matrices need to be constant.
y	input matrix, one input matrices need to be constant.

vecnorm — *p-norm of vector representation of x*

Description

p-norm of vector representation of x, which is deprecated, use `norm(vec(x), p=2)` instead.

Usage

```
vecnorm(x, p = 2)
```

Arguments

x	input matrix.
p	a number greater than 1.

Index