Package 'CVXR'

March 13, 2019

Type Package

Title Disciplined Convex Optimization

Version 0.99-4

VignetteBuilder knitr

URL https://cvxr.rbind.io, https://anqif.github.io/CVXR

BugReports https://github.com/anqif/CVXR/issues

Description An object-oriented modeling language for disciplined convex programming (DCP). It allows the user to formulate convex optimization problems in a natural way following mathematical convention and DCP rules. The system analyzes the problem, verifies its convexity, converts it into a canonical form, and hands it off to an appropriate solver to obtain the solution.

Depends R (>= 3.4.0)

Imports methods, R6, Matrix, Rcpp (>= 0.12.12), bit64, gmp, Rmpfr, R.utils, ECOSolveR (>= 0.4), scs (>= 1.2.3), stats

Suggests knitr, rmarkdown, testthat, nnls, reticulate, lpSolveAPI, Rglpk, slam

LinkingTo Rcpp, RcppEigen

License Apache License 2.0 | file LICENSE

Encoding UTF-8

LazyData true

Collate 'CVXR.R' 'data.R' 'globals.R' 'generics.R' 'utilities.R' 'interface.R' 'expressions.R' 'constant.R' 'variable.R' 'lin_ops.R' 'atoms.R' 'affine.R' 'problem.R' 'constraints.R' 'elementwise.R' 'solver.R' 'mosek-solver.R' 'gurobi-solver.R' 'lpsolve_solver.R' 'glpk_solver.R' 'solver_utilities.R' 'problem_data.R' 'transforms.R' 'exports.R' 'rcppUtils.R' 'R6List.R' 'ProblemData-R6.R' 'LinOp-R6.R' 'LinOpVector-R6.R' 'RcppExports.R' 'CVXcanon-R6.R' 'Deque.R' 'canonInterface.R'

RoxygenNote 6.1.1

NeedsCompilation yes

Author Anqi Fu [aut, cre],
Balasubramanian Narasimhan [aut],
Steven Diamond [aut],
John Miller [aut],
Stephen Boyd [ctb],
Paul Kunsberg Rosenfield [ctb]

Maintainer Anqi Fu <anqif@stanford.edu>

Repository CRAN

Date/Publication 2019-03-13 06:26:58 UTC

R topics documented:

`CVXR-package` *CVXR: Disciplined Convex Optimization in R*

Description

CVXR is an R package that provides an object-oriented modeling language for convex optimization, similar to CVX, CVXPY, YALMIP, and Convex.jl. This domain specific language (DSL) allows the user to formulate convex optimization problems in a natural mathematical syntax rather than the restrictive standard form required by most solvers. The user specifies an objective and set of constraints by combining constants, variables, and parameters using a library of functions with known mathematical properties. CVXR then applies signed disciplined convex programming (DCP) to verify the problem's convexity. Once verified, the problem is converted into standard conic form using graph implementations and passed to a cone solver such as ECOS or SCS.

Author(s)

Anqi Fu, Balasubramanian Narasimhan, John Miller, Steven Diamond, Stephen Boyd

Maintainer: Anqi Fu<anqif@stanford.edu>

`*,Expression,Expression-method`
Elementwise Multiplication

Description

The elementwise product of two expressions. The first expression must be constant.

Usage

```
## S4 method for signature 'Expression,Expression'
e1 * e2

## S4 method for signature 'Expression,ConstVal'
e1 * e2

## S4 method for signature 'ConstVal,Expression'
e1 * e2

mul_elemwise(lh_const, rh_exp)
```

Arguments

e1, e2	The Expression objects or numeric constants to multiply elementwise.
lh_const	A constant Expression, vector, or matrix representing the left-hand value.
rh_exp	An Expression, vector, or matrix representing the right-hand value.

Value

An Expression representing the elementwise product of the inputs.

Examples

```
A <- Variable(2,2)
c <- cbind(c(1,-1), c(2,-2))
expr <- mul_elemwise(c, A)
obj <- Minimize(norm_inf(expr))
prob <- Problem(obj, list(A == 5))
result <- solve(prob)
result$value
result$getValue(expr)
```

+,Expression,missing-method

The AddExpression class.

Description

This class represents the sum of any number of expressions.

Usage

```
## S4 method for signature 'Expression,missing'
e1 + e2

## S4 method for signature 'Expression,Expression'
```

```
e1 + e2

## S4 method for signature 'Expression,ConstVal'
e1 + e2

## S4 method for signature 'ConstVal,Expression'
e1 + e2

## S4 method for signature 'AddExpression'
to_numeric(object, values)

## S4 method for signature 'AddExpression'
size_from_args(object)

## S4 method for signature 'AddExpression'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

`e1, e2`	The Expression objects or numeric constants to add.
`object`	An AddExpression object.
`values`	A list of arguments to the atom.
`arg_objs`	A list of linear expressions for each argument.
`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Methods (by generic)

- `to_numeric`: Sum all the values.
- `size_from_args`: The size of the expression.
- `graph_implementation`: The graph implementation of the expression.

Slots

`arg_groups` A `list` of Expressions and numeric data.frame, matrix, or vector objects.

`-,Expression,missing-method`

The NegExpression class.

Description

This class represents the negation of an affine expression.

Usage

```
## S4 method for signature 'Expression,missing'
e1 - e2

## S4 method for signature 'Expression,Expression'
e1 - e2

## S4 method for signature 'Expression,ConstVal'
e1 - e2

## S4 method for signature 'ConstVal,Expression'
e1 - e2

## S4 method for signature 'NegExpression'
to_numeric(object, values)

## S4 method for signature 'NegExpression'
size_from_args(object)

## S4 method for signature 'NegExpression'
sign_from_args(object)

## S4 method for signature 'NegExpression'
is_incr(object, idx)

## S4 method for signature 'NegExpression'
is_decr(object, idx)

## S4 method for signature 'NegExpression'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

e1, e2	The Expression objects or numeric constants to subtract.
object	A NegExpression object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: Negate the value.
- size_from_args: The size of the expression.

- `sign_from_args`: The sign of the expression.
- `is_incr`: The expression is not weakly increasing in any argument.
- `is_decr`: The expression is weakly decreasing in every argument.
- `graph_implementation`: The graph implementation of the expression.

`.build_matrix_0` *Get the* `sparse` *flag field for the LinOp object*

Description

Get the `sparse` flag field for the LinOp object

Usage

```
.build_matrix_0(xp, v)
```

Arguments

`xp`	the LinOpVector Object XPtr
`v`	the `id_to_col` named int vector in R with integer names

Value

a XPtr to ProblemData Object

`.build_matrix_1` *Get the* `sparse` *flag field for the LinOp object*

Description

Get the `sparse` flag field for the LinOp object

Usage

```
.build_matrix_1(xp, v1, v2)
```

Arguments

`xp`	the LinOpVector Object XPtr
`v1`	the `id_to_col` named int vector in R with integer names
`v2`	the `constr_offsets` vector of offsets (an int vector in R)

Value

a XPtr to ProblemData Object

.LinOpVector__new *Create a new LinOpVector object.*

Description

Create a new LinOpVector object.

Usage

```
.LinOpVector__new()
```

Value

an external ptr (Rcpp::XPtr) to a LinOp object instance.

.LinOpVector__push_back *Perform a push back operation on the* args *field of LinOp*

Description

Perform a push back operation on the args field of LinOp

Usage

```
.LinOpVector__push_back(xp, yp)
```

Arguments

xp	the LinOpVector Object XPtr
yp	the LinOp Object XPtr to push

.LinOp_at_index *Return the LinOp element at index i (0-based)*

Description

Return the LinOp element at index i (0-based)

Usage

```
.LinOp_at_index(lvec, i)
```

Arguments

lvec	the LinOpVector Object XPtr
i	the index

.LinOp__args_push_back

Perform a push back operation on the `args` *field of LinOp*

Description

Perform a push back operation on the `args` field of LinOp

Usage

```
.LinOp__args_push_back(xp, yp)
```

Arguments

`xp`	the LinOp Object XPtr
`yp`	the LinOp Object XPtr to push

.LinOp__get_dense_data

Get the field `dense_data` *for the LinOp object*

Description

Get the field `dense_data` for the LinOp object

Usage

```
.LinOp__get_dense_data(xp)
```

Arguments

`xp`	the LinOp Object XPtr

Value

a MatrixXd object

.LinOp__get_id *Get the id field of the LinOp Object*

Description

Get the id field of the LinOp Object

Usage

```
.LinOp__get_id(xp)
```

Arguments

xp	the LinOp Object XPtr

Value

the value of the id field of the LinOp Object

.LinOp__get_size *Get the field* size *for the LinOp object*

Description

Get the field `size` for the LinOp object

Usage

```
.LinOp__get_size(xp)
```

Arguments

xp	the LinOp Object XPtr

Value

an integer vector

.LinOp__get_slice *Get the slice field of the LinOp Object*

Description

Get the slice field of the LinOp Object

Usage

```
.LinOp__get_slice(xp)
```

Arguments

xp	the LinOp Object XPtr

Value

the value of the slice field of the LinOp Object

.LinOp__get_sparse *Get the* sparse *flag field for the LinOp object*

Description

Get the sparse flag field for the LinOp object

Usage

```
.LinOp__get_sparse(xp)
```

Arguments

xp	the LinOp Object XPtr

Value

TRUE or FALSE

.LinOp__get_sparse_data *Get the field named* sparse_data *from the LinOp object*

Description

Get the field named sparse_data from the LinOp object

Usage

```
.LinOp__get_sparse_data(xp)
```

Arguments

xp	the LinOp Object XPtr

Value

a dgCMatrix-class object

.LinOp__get_type *Get the field named* type *for the LinOp object*

Description

Get the field named type for the LinOp object

Usage

```
.LinOp__get_type(xp)
```

Arguments

xp	the LinOp Object XPtr

Value

an integer value for type

.LinOp__new *Create a new LinOp object.*

Description

Create a new LinOp object.

Usage

```
.LinOp__new()
```

Value

an external ptr (Rcpp::XPtr) to a LinOp object instance.

.LinOp__set_dense_data *Set the field* dense_data *of the LinOp object*

Description

Set the field dense_data of the LinOp object

Usage

```
.LinOp__set_dense_data(xp, denseMat)
```

Arguments

xp	the LinOp Object XPtr
denseMat	a standard matrix object in R

.LinOp__set_size *Set the field* size *of the LinOp object*

Description

Set the field size of the LinOp object

Usage

```
.LinOp__set_size(xp, value)
```

Arguments

xp	the LinOp Object XPtr
value	an integer vector object in R

.LinOp__set_slice *Set the slice field of the LinOp Object*

Description

Set the slice field of the LinOp Object

Usage

```
.LinOp__set_slice(xp, value)
```

Arguments

xp	the LinOp Object XPtr
value	a list of integer vectors, e.g. `list(1:10, 2L, 11:15)`

Value

the value of the slice field of the LinOp Object

.LinOp__set_sparse *Set the flag* sparse *of the LinOp object*

Description

Set the flag sparse of the LinOp object

Usage

```
.LinOp__set_sparse(xp, sparseSEXP)
```

Arguments

xp	the LinOp Object XPtr
sparseSEXP	an R boolean

.LinOp__set_sparse_data *Set the field named* sparse_data *of the LinOp object*

Description

Set the field named sparse_data of the LinOp object

Usage

```
.LinOp__set_sparse_data(xp, sparseMat)
```

Arguments

xp	the LinOp Object XPtr
sparseMat	a dgCMatrix-class object

.LinOp__set_type *Set the field named* type *for the LinOp object*

Description

Set the field named type for the LinOp object

Usage

```
.LinOp__set_type(xp, typeValue)
```

Arguments

xp	the LinOp Object XPtr
typeValue	an integer value

`.LinOp__size_push_back`

Perform a push back operation on the `size` *field of LinOp*

Description

Perform a push back operation on the `size` field of LinOp

Usage

```
.LinOp__size_push_back(xp, intVal)
```

Arguments

`xp`	the LinOp Object XPtr
`intVal`	the integer value to push back

`.LinOp__slice_push_back`

Perform a push back operation on the `slice` *field of LinOp*

Description

Perform a push back operation on the `slice` field of LinOp

Usage

```
.LinOp__slice_push_back(xp, intVec)
```

Arguments

`xp`	the LinOp Object XPtr
`intVec`	an integer vector to push back

.ProblemData__get_const_to_row

Get the const_to_row field of the ProblemData Object

Description

Get the const_to_row field of the ProblemData Object

Usage

```
.ProblemData__get_const_to_row(xp)
```

Arguments

xp	the ProblemData Object XPtr

Value

the const_to_row field as a named integer vector where the names are integers converted to characters

.ProblemData__get_const_vec

Get the const_vec field from the ProblemData Object

Description

Get the const_vec field from the ProblemData Object

Usage

```
.ProblemData__get_const_vec(xp)
```

Arguments

xp	the ProblemData Object XPtr

Value

a numeric vector of the field const_vec from the ProblemData Object

.ProblemData__get_I *Get the I field of the ProblemData Object*

Description

Get the I field of the ProblemData Object

Usage

```
.ProblemData__get_I(xp)
```

Arguments

xp	the ProblemData Object XPtr

Value

an integer vector of the field I from the ProblemData Object

.ProblemData__get_id_to_col *Get the id_to_col field of the ProblemData Object*

Description

Get the id_to_col field of the ProblemData Object

Usage

```
.ProblemData__get_id_to_col(xp)
```

Arguments

xp	the ProblemData Object XPtr

Value

the id_to_col field as a named integer vector where the names are integers converted to characters

.ProblemData__get_J *Get the J field of the ProblemData Object*

Description

Get the J field of the ProblemData Object

Usage

```
.ProblemData__get_J(xp)
```

Arguments

xp	the ProblemData Object XPtr

Value

an integer vector of the field J from the ProblemData Object

.ProblemData__get_V *Get the V field of the ProblemData Object*

Description

Get the V field of the ProblemData Object

Usage

```
.ProblemData__get_V(xp)
```

Arguments

xp	the ProblemData Object XPtr

Value

a numeric vector of doubles (the field V) from the ProblemData Object

`.ProblemData__new` *Create a new ProblemData object.*

Description

Create a new ProblemData object.

Usage

```
.ProblemData__new()
```

Value

an external ptr (Rcpp::XPtr) to a ProblemData object instance.

`.ProblemData__set_const_to_row` *Set the const_to_row map of the ProblemData Object*

Description

Set the const_to_row map of the ProblemData Object

Usage

```
.ProblemData__set_const_to_row(xp, iv)
```

Arguments

`xp`	the ProblemData Object XPtr
`iv`	a named integer vector with names being integers converted to characters

.ProblemData__set_const_vec *Set the const_vec field in the ProblemData Object*

Description

Set the const_vec field in the ProblemData Object

Usage

```
.ProblemData__set_const_vec(xp, cvp)
```

Arguments

xp	the ProblemData Object XPtr
cvp	a numeric vector of values for const_vec field of the ProblemData object

.ProblemData__set_I *Set the I field in the ProblemData Object*

Description

Set the I field in the ProblemData Object

Usage

```
.ProblemData__set_I(xp, ip)
```

Arguments

xp	the ProblemData Object XPtr
ip	an integer vector of values for field I of the ProblemData object

.ProblemData__set_id_to_col *Set the id_to_col field of the ProblemData Object*

Description

Set the id_to_col field of the ProblemData Object

Usage

```
.ProblemData__set_id_to_col(xp, iv)
```

Arguments

xp	the ProblemData Object XPtr
iv	a named integer vector with names being integers converted to characters

.ProblemData__set_J *Set the J field in the ProblemData Object*

Description

Set the J field in the ProblemData Object

Usage

```
.ProblemData__set_J(xp, jp)
```

Arguments

xp	the ProblemData Object XPtr
jp	an integer vector of the values for field J of the ProblemData object

.ProblemData__set_V *Set the V field in the ProblemData Object*

Description

Set the V field in the ProblemData Object

Usage

```
.ProblemData__set_V(xp, vp)
```

Arguments

xp	the ProblemData Object XPtr
vp	a numeric vector of values for field V

/,Expression,Expression-method *The DivExpression class.*

Description

This class represents one expression divided by another expression.

Usage

```
## S4 method for signature 'Expression,Expression'
e1 / e2

## S4 method for signature 'Expression,ConstVal'
e1 / e2

## S4 method for signature 'ConstVal,Expression'
e1 / e2

## S4 method for signature 'DivExpression'
is_quadratic(object)

## S4 method for signature 'DivExpression'
size_from_args(object)

## S4 method for signature 'DivExpression'
is_incr(object, idx)

## S4 method for signature 'DivExpression'
```

```
is_decr(object, idx)

## S4 method for signature 'DivExpression'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

e1, e2	The Expression objects or numeric constants to divide. The denominator, e2, must be a scalar constant.
object	A DivExpression object.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- is_quadratic: Is the left-hand expression quadratic and the right-hand expression constant?
- size_from_args: The size of the left-hand expression.
- is_incr: Is the right-hand expression positive?
- is_decr: Is the right-hand expression negative?
- graph_implementation: The graph implementation of the expression.

<=,Expression,Expression-method

The LeqConstraint class.

Description

This class represents a $\leq$ inequality constraint.

Usage

```
## S4 method for signature 'Expression,Expression'
e1 <= e2

## S4 method for signature 'Expression,ConstVal'
e1 <= e2

## S4 method for signature 'ConstVal,Expression'
e1 <= e2

## S4 method for signature 'Expression,Expression'
e1 < e2
```

```
## S4 method for signature 'Expression,ConstVal'
e1 < e2

## S4 method for signature 'ConstVal,Expression'
e1 < e2

## S4 method for signature 'Expression,Expression'
e1 >= e2

## S4 method for signature 'Expression,ConstVal'
e1 >= e2

## S4 method for signature 'ConstVal,Expression'
e1 >= e2

## S4 method for signature 'Expression,Expression'
e1 > e2

## S4 method for signature 'Expression,ConstVal'
e1 > e2

## S4 method for signature 'ConstVal,Expression'
e1 > e2

LeqConstraint(lh_exp, rh_exp)

## S4 method for signature 'LeqConstraint'
as.character(x)

## S4 method for signature 'LeqConstraint'
id(object)

## S4 method for signature 'LeqConstraint'
size(object)

## S4 method for signature 'LeqConstraint'
is_dcp(object)

## S4 method for signature 'LeqConstraint'
canonicalize(object)

## S4 method for signature 'LeqConstraint'
variables(object)

## S4 method for signature 'LeqConstraint'
parameters(object)
```

```
## S4 method for signature 'LeqConstraint'
constants(object)

## S4 method for signature 'LeqConstraint'
residual(object)

## S4 method for signature 'LeqConstraint'
value(object)

## S4 method for signature 'LeqConstraint'
violation(object)
```

Arguments

`e1, e2`	The Expression objects or numeric constants to compare.
`lh_exp`	An Expression, numeric element, vector, or matrix representing the left-hand side of the inequality.
`rh_exp`	An Expression, numeric element, vector, or matrix representing the right-hand side of the inequality.
`x, object`	A LeqConstraint object.

Methods (by generic)

- `id`: The `constr_id` of the constraint.
- `size`: The size of the left-hand expression minus the right-hand expression.
- `is_dcp`: The constraint is DCP if the left-hand expression is convex and the right-hand expression is concave.
- `canonicalize`: The graph implementation of the object. Marks the top level constraint as the `dual_holder` so the dual value will be saved to the LeqConstraint.
- `variables`: List of Variable objects in the constraint.
- `parameters`: List of Parameter objects in the constraint.
- `constants`: List of Constant objects in the constraint.
- `residual`: The elementwise maximum of the left-hand expression minus the right-hand expression, i.e. `max_elemwise(lh_exp - rh_exp, 0)`.
- `value`: A logical value indicating whether the constraint holds. Tolerance is currently set at `1e-4`.
- `violation`: A matrix representing the amount by which the constraint is off, i.e. the numeric value of the residual expression.

Slots

`constr_id` (Internal) A unique integer identification number used internally.

`lh_exp` An Expression, numeric element, vector, or matrix representing the left-hand side of the inequality.

`rh_exp` An Expression, numeric element, vector, or matrix representing the right-hand side of the inequality.

`args` (Internal) A list that holds `lh_exp` and `rh_exp` for internal use.

`.expr` (Internal) An Expression representing `lh_exp - rh_exp` for internal use.

`dual_variable` (Internal) A Variable representing the dual variable associated with the constraint.

`==,Expression,Expression-method`
The EqConstraint class.

Description

This class represents a equality constraint.

Usage

```
## S4 method for signature 'Expression,Expression'
e1 == e2

## S4 method for signature 'Expression,ConstVal'
e1 == e2

## S4 method for signature 'ConstVal,Expression'
e1 == e2

EqConstraint(lh_exp, rh_exp)

## S4 method for signature 'EqConstraint'
is_dcp(object)

## S4 method for signature 'EqConstraint'
residual(object)

## S4 method for signature 'EqConstraint'
canonicalize(object)
```

Arguments

`e1, e2`	The Expression objects or numeric constants to compare.
`lh_exp`	An Expression, numeric element, vector, or matrix representing the left-hand side of the inequality.
`rh_exp`	An Expression, numeric element, vector, or matrix representing the right-hand side of the inequality.
`object`	An EqConstraint object.

Methods (by generic)

- `is_dcp`: The constraint is DCP if the left-hand and right-hand expressions are affine.
- `residual`: The absolute value of the left-hand minus the right-hand expression, i.e. `abs(lh_exp - rh_exp)`.
- `canonicalize`: The graph implementation of the object. Marks the top level constraint as the `dual_holder` so the dual value will be saved to the EqConstraint.

Slots

`constr_id` (Internal) A unique integer identification number used internally.

`lh_exp` An Expression, numeric element, vector, or matrix representing the left-hand side of the inequality.

`rh_exp` An Expression, numeric element, vector, or matrix representing the right-hand side of the inequality.

`args` (Internal) A list that holds `lh_exp` and `rh_exp` for internal use.

`.expr` (Internal) An Expression representing `lh_exp - rh_exp` for internal use.

`dual_variable` (Internal) A Variable representing the dual variable associated with the constraint.

`abs,Expression-method` *Absolute Value*

Description

The elementwise absolute value.

Usage

```
## S4 method for signature 'Expression'
abs(x)
```

Arguments

`x`	An Expression.

Value

An Expression representing the absolute value of the input.

Examples

```
A <- Variable(2,2)
prob <- Problem(Minimize(sum(abs(A))), list(A <= -2))
result <- solve(prob)
result$value
result$getValue(A)
```

Abs-class *The Abs class.*

Description

This class represents the elementwise absolute value.

Usage

```
Abs(x)

## S4 method for signature 'Abs'
to_numeric(object, values)

## S4 method for signature 'Abs'
sign_from_args(object)

## S4 method for signature 'Abs'
is_atom_convex(object)

## S4 method for signature 'Abs'
is_atom_concave(object)

## S4 method for signature 'Abs'
is_incr(object, idx)

## S4 method for signature 'Abs'
is_decr(object, idx)

## S4 method for signature 'Abs'
is_pwl(object)

## S4 method for signature 'Abs'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression object.
object	An Abs object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- `to_numeric`: The elementwise absolute value of the input.
- `sign_from_args`: The atom is positive.
- `is_atom_convex`: The atom is convex.
- `is_atom_concave`: The atom is not concave.
- `is_incr`: A logical value indicating whether the atom is weakly increasing.
- `is_decr`: A logical value indicating whether the atom is weakly decreasing.
- `is_pwl`: Is `x` piecewise linear?
- `graph_implementation`: The graph implementation of the atom.

Slots

`x` An Expression object.

`AffAtom-class` *The AffAtom class.*

Description

This virtual class represents an affine atomic expression.

Usage

```
## S4 method for signature 'AffAtom'
sign_from_args(object)

## S4 method for signature 'AffAtom'
is_atom_convex(object)

## S4 method for signature 'AffAtom'
is_atom_concave(object)

## S4 method for signature 'AffAtom'
is_incr(object, idx)

## S4 method for signature 'AffAtom'
is_decr(object, idx)

## S4 method for signature 'AffAtom'
is_quadratic(object)

## S4 method for signature 'AffAtom'
is_pwl(object)
```

Arguments

object	An AffAtom object.
idx	An index into the atom.

Methods (by generic)

- sign_from_args: The sign of the atom.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is concave.
- is_incr: The atom is weakly increasing in every argument.
- is_decr: The atom is not weakly decreasing in any argument.
- is_quadratic: Is every argument quadratic?
- is_pwl: Is every argument piecewise linear?

AffineProd-class *The AffineProd class.*

Description

This class represents the product of two affine expressions.

Usage

```
AffineProd(x, y)

## S4 method for signature 'AffineProd'
validate_args(object)

## S4 method for signature 'AffineProd'
to_numeric(object, values)

## S4 method for signature 'AffineProd'
size_from_args(object)

## S4 method for signature 'AffineProd'
sign_from_args(object)

## S4 method for signature 'AffineProd'
is_atom_convex(object)

## S4 method for signature 'AffineProd'
is_atom_concave(object)

## S4 method for signature 'AffineProd'
```

```
is_incr(object, idx)

## S4 method for signature 'AffineProd'
is_decr(object, idx)

## S4 method for signature 'AffineProd'
is_quadratic(object)
```

Arguments

x	An Expression or numeric constant representing the left-hand value.
y	An Expression or numeric constant representing the right-hand value.
object	An AffineProd object.
values	A list of arguments to the atom.
idx	An index into the atom.

Methods (by generic)

- validate_args: Check dimensions of arguments and linearity.
- to_numeric: The product of two affine expressions.
- size_from_args: The size of the atom.
- sign_from_args: Default to rules for times.
- is_atom_convex: Affine times affine is not convex.
- is_atom_concave: Affine times affine is not concave.
- is_incr: A logical value indicating whether the atom is weakly increasing in idx.
- is_decr: A logical value indicating whether the atom is weakly decreasing in idx.
- is_quadratic: The affine product is always quadratic.

Slots

x An Expression or numeric constant representing the left-hand value.

y An Expression or numeric constant representing the right-hand value.

---

affine_prod *Affine Product*

Description

The product of two affine expressions.

Usage

```
affine_prod(x, y)
```

Arguments

x	An Expression or numeric constant representing the left-hand value.
y	An Expression or numeric constant representing the right-hand value.

Value

An Expression representing the product of x and y.

Atom-class *The Atom class.*

Description

This virtual class represents atomic expressions in CVXR.

Usage

```
## S4 method for signature 'Atom'
validate_args(object)

## S4 method for signature 'Atom'
size(object)

## S4 method for signature 'Atom'
dim(x)

## S4 method for signature 'Atom'
nrow(x)

## S4 method for signature 'Atom'
ncol(x)

## S4 method for signature 'Atom'
is_positive(object)

## S4 method for signature 'Atom'
is_negative(object)

## S4 method for signature 'Atom'
is_convex(object)

## S4 method for signature 'Atom'
is_concave(object)

## S4 method for signature 'Atom'
canonicalize(object)
```

```
## S4 method for signature 'Atom'
graph_implementation(object, arg_objs, size,
  data = NA_real_)

## S4 method for signature 'Atom'
variables(object)

## S4 method for signature 'Atom'
parameters(object)

## S4 method for signature 'Atom'
constants(object)

## S4 method for signature 'Atom'
value(object)

## S4 method for signature 'Atom'
grad(object)

## S4 method for signature 'Atom'
domain(object)
```

Arguments

`x, object`	An Atom object.
`arg_objs`	A list of linear expressions for each argument.
`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Methods (by generic)

- `validate_args`: Raises an error if the arguments are invalid.
- `size`: The `c(row, col)` dimensions of the atom.
- `dim`: The `c(row, col)` dimensions of the atom.
- `nrow`: The number of rows in the atom.
- `ncol`: The number of columns in the atom.
- `is_positive`: A logical value indicating whether the atom is positive.
- `is_negative`: A logical value indicating whether the atom is negative.
- `is_convex`: A logical value indicating whether the atom is convex.
- `is_concave`: A logical value indicating whether the atom is concave.
- `canonicalize`: Represent the atom as an affine objective and conic constraints.
- `graph_implementation`: The graph implementation of the atom.
- `variables`: List of Variable objects in the atom.

- `parameters`: List of Parameter objects in the atom.
- `constants`: List of Constant objects in the atom.
- `value`: The value of the atom.
- `grad`: The (sub/super)-gradient of the atom with respect to each variable.
- `domain`: A list of constraints describing the closure of the region where the expression is finite.

`AxisAtom-class` *The AxisAtom class.*

Description

This virtual class represents atomic expressions that can be applied along an axis in CVXR.

Usage

```
## S4 method for signature 'AxisAtom'
size_from_args(object)

## S4 method for signature 'AxisAtom'
get_data(object)

## S4 method for signature 'AxisAtom'
validate_args(object)
```

Arguments

`object`	An Atom object.

Methods (by generic)

- `size_from_args`: The size of the atom deteremined from its arguments.
- `get_data`: A list containing `axis`.
- `validate_args`: Check that the new shape has the same number of entries as the old.

Slots

`expr`	A numeric element, data.frame, matrix, vector, or Expression.
`axis`	(Optional) The dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `NA`.

BinaryOperator-class *The BinaryOperator class.*

Description

This base class represents expressions involving binary operators.

Usage

```
## S4 method for signature 'BinaryOperator'
to_numeric(object, values)

## S4 method for signature 'BinaryOperator'
sign_from_args(object)
```

Arguments

object	A BinaryOperator object.
values	A list of arguments to the atom.

Methods (by generic)

- to_numeric: Apply the binary operator to the values.
- sign_from_args: The sign of the expression.

Slots

lh_exp The Expression on the left-hand side of the operator.

rh_exp The Expression on the right-hand side of the operator.

op_name A character string indicating the binary operation.

bmat *Block Matrix*

Description

Constructs a block matrix from a list of lists. Each internal list is stacked horizontally, and the internal lists are stacked vertically.

Usage

```
bmat(block_lists)
```

Arguments

block_lists	A list of lists containing Expression objects, matrices, or vectors, which represent the blocks of the block matrix.

Value

An Expression representing the block matrix.

Examples

```
x <- Variable()
expr <- bmat(list(list(matrix(1, nrow = 3, ncol = 1), matrix(2, nrow = 3, ncol = 2)),
                  list(matrix(3, nrow = 1, ncol = 2), x)
             ))
prob <- Problem(Minimize(sum_entries(expr)), list(x >= 0))
result <- solve(prob)
result$value
```

Bool-class *The Bool class.*

Description

This class represents a boolean variable.

Usage

```
Bool(rows = 1, cols = 1, name = NA_character_)

## S4 method for signature 'Bool'
as.character(x)

## S4 method for signature 'Bool'
canonicalize(object)

## S4 method for signature 'Bool'
is_positive(object)

## S4 method for signature 'Bool'
is_negative(object)
```

Arguments

rows	The number of rows in the variable.
cols	The number of columns in the variable.
name	(Optional) A character string representing the name of the variable.
x, object	A Bool object.

Methods (by generic)

- canonicalize: Enforce that the variable be boolean.
- is_positive: A boolean variable is always positive or zero.
- is_negative: A boolean variable is never negative.

Slots

id (Internal) A unique identification number used internally.

rows The number of rows in the variable.

cols The number of columns in the variable.

name (Optional) A character string representing the name of the variable.

primal_value (Internal) The primal value of the variable stored internally.

Examples

```
x <- Bool(3, name = "indicator") ## Boolean 3-vector
y <- Bool(3, 3) ## Matrix boolean
name(x)
as.character(x)
canonicalize(y)
is_positive(x)
is_negative(y)
```

BoolConstr-class *The BoolConstr class.*

Description

This class represents a boolean constraint, $X_{ij} \in \{0, 1\}$ for all i, j.

Usage

```
BoolConstr(lin_op)

## S4 method for signature 'BoolConstr'
format_constr(object, eq_constr, leq_constr, dims,
  solver)

## S4 method for signature 'BoolConstr'
size(object)
```

Arguments

lin_op	A list representing the linear operator equal to the .noncvx_var.
object	A BoolConstr object.
eq_constr	A list of the equality constraints in the canonical problem.
leq_constr	A list of the inequality constraints in the canonical problem.
dims	A list with the dimensions of the conic constraints.
solver	A string representing the solver to be called.

Methods (by generic)

- format_constr: Format SDP constraints as inequalities for the solver.
- size: The dimensions of the semidefinite cone.

Slots

constr_id (Internal) A unique integer identification number used internally.

lin_op A list representing the linear operator equal to the .noncvx_var.

.noncvx_var (Internal) A list representing the variable constrained to be elementwise boolean.

CallbackParam-class *The CallbackParam class.*

Description

This class represents a parameter whose value is obtained by evaluating a function.

Usage

```
CallbackParam(callback, rows = 1, cols = 1, name = NA_character_,
  sign = UNKNOWN)

## S4 method for signature 'CallbackParam'
value(object)

## S4 method for signature 'CallbackParam'
get_data(object)
```

Arguments

callback	A numeric element, vector, matrix, or data.frame
rows	The number of rows in the parameter.
cols	The number of columns in the parameter.
name	(Optional) A character string representing the name of the parameter.
sign	A character string indicating the sign of the parameter. Must be "ZERO", "POSITIVE", "NEGATIVE", or "UNKNOWN".
object	A CallbackParam object.

Methods (by generic)

- `get_data`: Returns `list(callback, rows, cols, name, sign string)`.

Slots

`callback` A numeric element, vector, matrix, or data.frame.

Examples

```
x <- Variable(2)
dim <- size(x)
y <- CallbackParam(value(x), dim[1], dim[2], sign = "POSITIVE")
get_data(y)
```

`Canonical-class` *The Canonical class.*

Description

This virtual class represents a canonical expression.

Usage

```
## S4 method for signature 'Canonical'
canonicalize(object)

## S4 method for signature 'Canonical'
variables(object)

## S4 method for signature 'Canonical'
parameters(object)

## S4 method for signature 'Canonical'
constants(object)

## S4 method for signature 'Canonical'
get_data(object)
```

Arguments

`object` A Canonical object.

Methods (by generic)

- `canonicalize`: The graph implementation of the input.
- `variables`: List of Variable objects in the expression.
- `parameters`: List of Parameter objects in the expression.
- `constants`: List of Constant objects in the expression.
- `get_data`: Information needed to reconstruct the expression aside from its arguments.

`canonicalize` *Canonicalize*

Description

Computes the graph implementation of a canonical expression.

Usage

```
canonicalize(object)

canonical_form(object)
```

Arguments

`object`	A Canonical object.

Value

A list of `list(affine expression, list(constraints))`.

`cdiac` *Global Monthly and Annual Temperature Anomalies (degrees C), 1850-2015 (Relative to the 1961-1990 Mean) (May 2016)*

Description

Global Monthly and Annual Temperature Anomalies (degrees C), 1850-2015 (Relative to the 1961-1990 Mean) (May 2016)

Usage

```
cdiac
```

Format

A data frame with 166 rows and 14 variables:

year Year
jan Anomaly for month of January
feb Anomaly for month of February
mar Anomaly for month of March
apr Anomaly for month of April
may Anomaly for month of May
jun Anomaly for month of June
jul Anomaly for month of July
aug Anomaly for month of August
sep Anomaly for month of September
oct Anomaly for month of October
nov Anomaly for month of November
dec Anomaly for month of December
annual Annual anomaly for the year

Source

https://ess-dive.lbl.gov/

References

https://ess-dive.lbl.gov/

---

cone-methods *Second-Order Cone Methods*

Description

The number of elementwise cones or the size of a single cone in a second-order cone constraint.

Usage

```
num_cones(object)

cone_size(object)
```

Arguments

object	An SOCAxis object.

Value

The number of cones, or the size of a cone.

`Constant-class` *The Constant class.*

Description

This class represents a constant.

Coerce an R object or expression into the Constant class.

Usage

```
Constant(value)

## S4 method for signature 'Constant'
as.character(x)

## S4 method for signature 'Constant'
constants(object)

## S4 method for signature 'Constant'
get_data(object)

## S4 method for signature 'Constant'
value(object)

## S4 method for signature 'Constant'
grad(object)

## S4 method for signature 'Constant'
size(object)

## S4 method for signature 'Constant'
is_positive(object)

## S4 method for signature 'Constant'
is_negative(object)

## S4 method for signature 'Constant'
canonicalize(object)

as.Constant(expr)
```

Arguments

`value`	A numeric element, vector, matrix, or data.frame. Vectors are automatically cast into a matrix column.
`x, object`	A Constant object.
`expr`	An Expression, numeric element, vector, matrix, or data.frame.

Value

A Constant representing the input as a constant.

Methods (by generic)

- `constants`: Returns itself as a constant.
- `get_data`: A list with the value of the constant.
- `value`: The value of the constant.
- `grad`: An empty list since the gradient of a constant is zero.
- `size`: The `c(row, col)` dimensions of the constant.
- `is_positive`: A logical value indicating whether all elemenets of the constant are non-negative.
- `is_negative`: A logical value indicating whether all elemenets of the constant are non-positive.
- `canonicalize`: The canonical form of the constant.

Slots

`value` A numeric element, vector, matrix, or data.frame. Vectors are automatically cast into a matrix column.

`is_1D_array` (Internal) A logical value indicating whether the value is a vector or 1-D matrix.

`sparse` (Internal) A logical value indicating whether the value is a sparse matrix.

`size` (Internal) A vector of containing the number of rows and columns.

`is_pos` (Internal) A logical value indicating whether all elements are non-negative.

`is_neg` (Internal) A logical value indicating whether all elements are non-positive.

Examples

```
x <- Constant(5)
y <- Constant(diag(3))
get_data(y)
value(y)
is_positive(y)
size(y)
as.Constant(y)
```

`Constraint-class` *The Constraint class.*

Description

This virtual class represents a mathematical constraint.

Slots

`constr_id` (Internal) A unique integer identification number used internally.

conv *Discrete Convolution*

Description

The 1-D discrete convolution of two vectors.

Usage

```
conv(lh_exp, rh_exp)
```

Arguments

lh_exp	An Expression or vector representing the left-hand value.
rh_exp	An Expression or vector representing the right-hand value.

Value

An Expression representing the convolution of the input.

Examples

```
x <- Variable(5)
h <- matrix(stats::rnorm(2), nrow = 2, ncol = 1)
prob <- Problem(Minimize(sum(conv(h, x))))
result <- solve(prob)
result$value
result$getValue(x)
```

Conv-class *The Conv class.*

Description

This class represents the 1-D discrete convolution of two vectors.

Usage

```
Conv(lh_exp, rh_exp)

## S4 method for signature 'Conv'
validate_args(object)

## S4 method for signature 'Conv'
to_numeric(object, values)
```

```
## S4 method for signature 'Conv'
size_from_args(object)

## S4 method for signature 'Conv'
sign_from_args(object)

## S4 method for signature 'Conv'
is_incr(object, idx)

## S4 method for signature 'Conv'
is_decr(object, idx)

## S4 method for signature 'Conv'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

lh_exp	An Expression or R numeric data representing the left-hand vector.
rh_exp	An Expression or R numeric data representing the right-hand vector.
object	A Conv object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check both arguments are vectors and the first is a constant.
- to_numeric: The convolution of the two values.
- size_from_args: The size of the atom.
- sign_from_args: The sign of the atom.
- is_incr: Is the left-hand expression positive?
- is_decr: Is the left-hand expression negative?
- graph_implementation: The graph implementation of the atom.

Slots

lh_exp An Expression or R numeric data representing the left-hand vector.

rh_exp An Expression or R numeric data representing the right-hand vector.

CumSum-class *The CumSum class.*

Description

This class represents the cumulative sum.

Usage

```
CumSum(expr, axis = 2)

## S4 method for signature 'CumSum'
to_numeric(object, values)

## S4 method for signature 'CumSum'
size_from_args(object)

## S4 method for signature 'CumSum'
validate_args(object)

## S4 method for signature 'CumSum'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

expr	An Expression to be summed.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, and 2 indicates columns. The default is 2.
object	A CumSum object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The cumulative sum of the values along the specified axis.
- size_from_args: The size of the atom.
- validate_args: Check that axis is either 1 or 2.
- graph_implementation: The graph implementation of the atom.

Slots

`expr` An Expression to be summed.

`axis` (Optional) The dimension across which to apply the function: `1` indicates rows, and `2` indicates columns. The default is `2`.

`cumsum_axis` *Cumulative Sum*

Description

The cumulative sum, $\sum_{i=1}^{k} x_i$ for $k = 1, \ldots, n$. When calling `cumsum`, matrices are automatically flattened into column-major order before the sum is taken.

Usage

```
cumsum_axis(expr, axis = 2)

## S4 method for signature 'Expression'
cumsum(x)
```

Arguments

`axis`	(Optional) The dimension across which to apply the function: `1` indicates rows, and `2` indicates columns. The default is `2`.
`x, expr`	An Expression, vector, or matrix.

Examples

```
val <- cbind(c(1,2), c(3,4))
value(cumsum(Constant(val)))
value(cumsum_axis(Constant(val)))

x <- Variable(2,2)
prob <- Problem(Minimize(cumsum(x)[4]), list(x == val))
result <- solve(prob)
result$value
result$getValue(cumsum(x))
```

curvature *Curvature of Expression*

Description

The curvature of an expression.

Usage

```
curvature(object)
```

Arguments

object	An Expression object.

Value

A string indicating the curvature of the expression, either "CONSTANT", "AFFINE", "CONVEX, "CONCAVE", or "UNKNOWN".

Examples

```
x <- Variable()
c <- Constant(5)

curvature(c)
curvature(x)
curvature(x^2)
curvature(sqrt(x))
curvature(log(x^3) + sqrt(x))
```

curvature-atom *Curvature of an Atom*

Description

Determine if an atom is convex, concave, or affine.

Usage

```
is_atom_convex(object)

is_atom_concave(object)

is_atom_affine(object)
```

```
## S4 method for signature 'Atom'
is_atom_convex(object)

## S4 method for signature 'Atom'
is_atom_concave(object)

## S4 method for signature 'Atom'
is_atom_affine(object)
```

Arguments

object	A Atom object.

Value

A logical value.

Examples

```
x <- Variable()

is_atom_convex(x^2)
is_atom_convex(sqrt(x))
is_atom_convex(log(x))

is_atom_concave(-abs(x))
is_atom_concave(x^2)
is_atom_concave(sqrt(x))

is_atom_affine(2*x)
is_atom_affine(x^2)
```

---

curvature-comp *Curvature of Composition*

Description

Determine whether a composition is non-decreasing or non-increasing in an index.

Usage

```
is_incr(object, idx)

is_decr(object, idx)

## S4 method for signature 'Atom'
is_incr(object, idx)

## S4 method for signature 'Atom'
is_decr(object, idx)
```

Arguments

object	A Atom object.
idx	An index into the atom.

Value

A logical value.

Examples

```
x <- Variable()
is_incr(log(x), 1)
is_incr(x^2, 1)
is_decr(min(x), 1)
is_decr(abs(x), 1)
```

---

curvature-methods *Curvature Properties*

Description

Determine if an expression is constant, affine, convex, concave, quadratic, or piecewise linear (pwl).

Usage

```
is_constant(object)

is_affine(object)

is_convex(object)

is_concave(object)

is_quadratic(object)

is_pwl(object)
```

Arguments

object	An Expression object.

Value

A logical value.

Examples

```
x <- Variable()
c <- Constant(5)

is_constant(c)
is_constant(x)

is_affine(c)
is_affine(x)
is_affine(x^2)

is_convex(c)
is_convex(x)
is_convex(x^2)
is_convex(sqrt(x))

is_concave(c)
is_concave(x)
is_concave(x^2)
is_concave(sqrt(x))

is_quadratic(x^2)
is_quadratic(sqrt(x))

is_pwl(c)
is_pwl(x)
is_pwl(x^2)
```

---

cvxr_norm *Matrix Norm (Alternative)*

Description

A wrapper on the different norm atoms. This is different from the standard "norm" method in the R base package. If `p = 2`, `axis = NA`, and `x` is a matrix, this returns the maximium singular value.

Usage

```
cvxr_norm(x, p = 2, axis = NA_real_)
```

Arguments

`x`	An Expression or numeric constant representing a vector or matrix.
`p`	The type of norm. May be a number (p-norm), "inf" (infinity-norm), "nuc" (nuclear norm), or "fro" (Frobenius norm). The default is `p = 2`.
`axis`	(Optional) The dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `NA`.

Value

An Expression representing the norm.

See Also

norm

`diag,Expression-method`
Matrix Diagonal

Description

Extracts the diagonal from a matrix or makes a vector into a diagonal matrix.

Usage

```
## S4 method for signature 'Expression'
diag(x = 1, nrow, ncol)
```

Arguments

`x`	An Expression, vector, or square matrix.
`nrow, ncol`	(Optional) Dimensions for the result when `x` is not a matrix.

Value

An Expression representing the diagonal vector or matrix.

Examples

```
C <- Variable(3,3)
obj <- Maximize(C[1,3])
constraints <- list(diag(C) == 1, C[1,2] == 0.6, C[2,3] == -0.3, C == Semidef(3))
prob <- Problem(obj, constraints)
result <- solve(prob)
result$value
result$getValue(C)
```

DiagMat-class *The DiagMat class.*

Description

This class represents the extraction of the diagonal from a square matrix.

Usage

```
DiagMat(expr)

## S4 method for signature 'DiagMat'
to_numeric(object, values)

## S4 method for signature 'DiagMat'
size_from_args(object)

## S4 method for signature 'DiagMat'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

expr	An Expression representing the matrix whose diagonal we are interested in.
object	A DiagMat object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: Extract the diagonal from a square matrix constant.
- size_from_args: The size of the atom.
- graph_implementation: The graph implementation of the atom.

Slots

expr An Expression representing the matrix whose diagonal we are interested in.

DiagVec-class *The DiagVec class.*

Description

This class represents the conversion of a vector into a diagonal matrix.

Usage

```
DiagVec(expr)

## S4 method for signature 'DiagVec'
to_numeric(object, values)

## S4 method for signature 'DiagVec'
size_from_args(object)

## S4 method for signature 'DiagVec'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

expr	An Expression representing the vector to convert.
object	A DiagVec object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: Convert the vector constant into a diagonal matrix.
- size_from_args: The size of the atom.
- graph_implementation: The graph implementation of the atom.

Slots

expr An Expression representing the vector to convert.

`diff,Expression-method`

Lagged and Iterated Differences

Description

The lagged and iterated differences of a vector. If `x` is length `n`, this function returns a length $n - k$ vector of the kth order difference between the lagged terms. `diff(x)` returns the vector of differences between adjacent elements in the vector, i.e. [x[2] - x[1], x[3] - x[2], ...]. `diff(x,1,2)` is the second-order differences vector, equivalently diff(diff(x)). `diff(x,1,0)` returns the vector x unchanged. `diff(x,2)` returns the vector of differences [x[3] - x[1], x[4] - x[2], ...], equivalent to `x[(1+lag):n] - x[1:(n-lag)]`.

Usage

```
## S4 method for signature 'Expression'
diff(x, lag = 1, differences = 1, ...)
```

Arguments

`x`	An Expression.
`lag`	An integer indicating which lag to use.
`differences`	An integer indicating the order of the difference.
`...`	(Optional) Addition `axis` argument, specifying the dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `axis = 1`.

Value

An Expression representing the kth order difference.

Examples

```
## Problem data
m <- 101
L <- 2
h <- L/(m-1)

## Form objective and constraints
x <- Variable(m)
y <- Variable(m)
obj <- sum(y)
constr <- list(x[1] == 0, y[1] == 1, x[m] == 1, y[m] == 1, diff(x)^2 + diff(y)^2 <= h^2)

## Solve the catenary problem
prob <- Problem(Minimize(obj), constr)
result <- solve(prob)
```

```
## Plot and compare with ideal catenary
xs <- result$getValue(x)
ys <- result$getValue(y)
plot(c(0, 1), c(0, 1), type = 'n', xlab = "x", ylab = "y")
lines(xs, ys, col = "blue", lwd = 2)
grid()
```

`domain` *Domain*

Description

A list of constraints describing the closure of the region where the expression is finite.

Usage

```
domain(object)
```

Arguments

`object`	An Expression object.

Value

A list of Constraint objects.

Examples

```
a <- Variable(name = "a")
dom <- domain(p_norm(a, -0.5))
prob <- Problem(Minimize(a), dom)
result <- solve(prob)
result$value

b <- Variable()
dom <- domain(kl_div(a, b))
result <- solve(Problem(Minimize(a + b), dom))
result$getValue(a)
result$getValue(b)

A <- Variable(2, 2, name = "A")
dom <- domain(lambda_max(A))
A0 <- rbind(c(1,2), c(3,4))
result <- solve(Problem(Minimize(norm2(A - A0)), dom))
result$getValue(A)

dom <- domain(log_det(A + diag(rep(1,2))))
prob <- Problem(Minimize(sum(diag(A))), dom)
result <- solve(prob, solver = "SCS")
result$value
```

---

dspop *Direct Standardization: Population*

Description

Randomly generated data for direct standardization example. Sex was drawn from a Bernoulli distribution, and age was drawn from a uniform distribution on 10,...,60. The response was drawn from a normal distribution with a mean that depends on sex and age, and a variance of 1.

Usage

```
dspop
```

Format

A data frame with 1000 rows and 3 variables:

y Response variable

sex Sex of individual, coded male (0) and female (1)

age Age of individual

See Also

dssamp

---

dssamp *Direct Standardization: Sample*

Description

A sample of dspop for direct standardization example. The sample is skewed such that young males are overrepresented in comparison to the population.

Usage

```
dssamp
```

Format

A data frame with 100 rows and 3 variables:

y Response variable

sex Sex of individual, coded male (0) and female (1)

age Age of individual

See Also

dspop

ECOS-class *The ECOS class.*

Description

This class is an interface for the ECOS solver.

Usage

```
ECOS()

## S4 method for signature 'ECOS'
lp_capable(solver)

## S4 method for signature 'ECOS'
socp_capable(solver)

## S4 method for signature 'ECOS'
sdp_capable(solver)

## S4 method for signature 'ECOS'
exp_capable(solver)

## S4 method for signature 'ECOS'
mip_capable(solver)

## S4 method for signature 'ECOS'
name(object)

## S4 method for signature 'ECOS'
import_solver(solver)

## S4 method for signature 'ECOS'
Solver.solve(solver, objective, constraints, cached_data,
  warm_start, verbose, ...)

## S4 method for signature 'ECOS'
format_results(solver, results_dict, data, cached_data)
```

Arguments

object, solver	An ECOS object.
objective	A list representing the canonicalized objective.
constraints	A list of canonicalized constraints.
cached_data	A list mapping solver name to cached problem data.

warm_start	A logical value indicating whether the previous solver result should be used to warm start.
verbose	A logical value indicating whether to print solver output.
...	Additional arguments to the solver.
results_dict	A list containing the solver output.
data	A list containing information about the problem.

Methods (by generic)

- lp_capable: ECOS can handle linear programs.
- socp_capable: ECOS can handle second-order cone programs.
- sdp_capable: ECOS cannot handle semidefinite programs.
- exp_capable: ECOS can handle exponential cone programs.
- mip_capable: ECOS cannot handle mixed-integer programs.
- name: The name of the solver.
- import_solver: Imports the ECOSolveR library.
- Solver.solve: Call the solver on the canonicalized problem.
- format_results: Convert raw solver output into standard list of results.

References

A. Domahidi, E. Chu, and S. Boyd. "ECOS: An SOCP solver for Embedded Systems." *Proceedings of the European Control Conference*, pp. 3071-3076, 2013.

See Also

ECOS_csolve and the ECOS Official Site.

Examples

```
ecos <- ECOS()
lp_capable(ecos)
sdp_capable(ecos)
socp_capable(ecos)
exp_capable(ecos)
mip_capable(ecos)
```

ECOS_BB-class *The ECOS_BB class.*

Description

This class is an interface for the ECOS BB (branch-and-bound) solver.

Usage

```
ECOS_BB()

## S4 method for signature 'ECOS_BB'
lp_capable(solver)

## S4 method for signature 'ECOS_BB'
socp_capable(solver)

## S4 method for signature 'ECOS_BB'
sdp_capable(solver)

## S4 method for signature 'ECOS_BB'
exp_capable(solver)

## S4 method for signature 'ECOS_BB'
mip_capable(solver)

## S4 method for signature 'ECOS_BB'
name(object)

## S4 method for signature 'ECOS_BB'
Solver.solve(solver, objective, constraints,
  cached_data, warm_start, verbose, ...)
```

Arguments

object, solver	A ECOS_BB object.
objective	A list representing the canonicalized objective.
constraints	A list of canonicalized constraints.
cached_data	A list mapping solver name to cached problem data.
warm_start	A logical value indicating whether the previous solver result should be used to warm start.
verbose	A logical value indicating whether to print solver output.
...	Additional arguments to the solver.

Methods (by generic)

- `lp_capable`: ECOS_BB can handle linear programs.
- `socp_capable`: ECOS_BB can handle second-order cone programs.
- `sdp_capable`: ECOS_BB cannot handle semidefinite programs.
- `exp_capable`: ECOS_BB cannot handle exponential cone programs.
- `mip_capable`: ECOS_BB can handle mixed-integer programs.
- `name`: The name of the solver.
- `Solver.solve`: Call the solver on the canonicalized problem.

References

A. Domahidi, E. Chu, and S. Boyd. "ECOS: An SOCP solver for Embedded Systems." *Proceedings of the European Control Conference*, pp. 3071-3076, 2013.

See Also

`ECOS_csolve` and the ECOS Official Site.

Examples

```
ecos_bb <- ECOS_BB()
lp_capable(ecos_bb)
sdp_capable(ecos_bb)
socp_capable(ecos_bb)
exp_capable(ecos_bb)
mip_capable(ecos_bb)
```

`Elementwise-class` *The Elementwise class.*

Description

This virtual class represents an elementwise atom.

Usage

```
## S4 method for signature 'Elementwise'
validate_args(object)

## S4 method for signature 'Elementwise'
size_from_args(object)
```

Arguments

`object`	An Elementwise object.

Methods (by generic)

- `validate_args`: Check all the shapes are the same or can be promoted.
- `size_from_args`: Size is the same as the sum of the arguments' sizes.

`entr` *Entropy Function*

Description

The elementwise entropy function, $-xlog(x)$.

Usage

```
entr(x)
```

Arguments

`x`	An Expression, vector, or matrix.

Value

An Expression representing the entropy of the input.

Examples

```
x <- Variable(5)
obj <- Maximize(sum(entr(x)))
prob <- Problem(obj, list(sum(x) == 1))
result <- solve(prob)
result$getValue(x)
```

`Entr-class` *The Entr class.*

Description

This class represents the elementwise operation $-xlog(x)$.

Usage

```
Entr(x)

## S4 method for signature 'Entr'
to_numeric(object, values)

## S4 method for signature 'Entr'
sign_from_args(object)

## S4 method for signature 'Entr'
is_atom_convex(object)

## S4 method for signature 'Entr'
is_atom_concave(object)

## S4 method for signature 'Entr'
is_incr(object, idx)

## S4 method for signature 'Entr'
is_decr(object, idx)

## S4 method for signature 'Entr'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric constant.
object	An Entr object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The elementwise entropy function evaluated at the value.
- sign_from_args: The sign of the atom is unknown.
- is_atom_convex: The atom is not convex.
- is_atom_concave: The atom is concave.
- is_incr: The atom is weakly increasing.
- is_decr: The atom is weakly decreasing.
- graph_implementation: The graph implementation of the atom.

Slots

x An Expression or numeric constant.

---

exp,Expression-method *Natural Exponential*

Description

The elementwise natural exponential.

Usage

```
## S4 method for signature 'Expression'
exp(x)
```

Arguments

x	An Expression.

Value

An Expression representing the natural exponential of the input.

Examples

```
x <- Variable(5)
obj <- Minimize(sum(exp(x)))
prob <- Problem(obj, list(sum(x) == 1))
result <- solve(prob)
result$getValue(x)
```

Exp-class *The Exp class.*

Description

This class represents the elementwise natural exponential e^x.

Usage

```
Exp(x)

## S4 method for signature 'Exp'
to_numeric(object, values)

## S4 method for signature 'Exp'
sign_from_args(object)

## S4 method for signature 'Exp'
is_atom_convex(object)

## S4 method for signature 'Exp'
is_atom_concave(object)

## S4 method for signature 'Exp'
is_incr(object, idx)

## S4 method for signature 'Exp'
is_decr(object, idx)

## S4 method for signature 'Exp'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression object.
object	An Exp object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- `to_numeric`: The matrix with each element exponentiated.
- `sign_from_args`: The atom is positive.
- `is_atom_convex`: The atom is convex.
- `is_atom_concave`: The atom is not concave.
- `is_incr`: The atom is weakly increasing.
- `is_decr`: The atom is not weakly decreasing.
- `graph_implementation`: The graph implementation of the atom.

Slots

x An Expression object.

ExpCone-class *The ExpCone class.*

Description

This class represents a reformulated exponential cone constraint operating elementwise on a, b, c.

Usage

```
ExpCone(a, b, c)

## S4 method for signature 'ExpCone'
size(object)

## S4 method for signature 'ExpCone'
as.character(x)

## S4 method for signature 'ExpCone'
variables(object)

## S4 method for signature 'ExpCone'
format_constr(object, eq_constr, leq_constr, dims,
  solver)
```

Arguments

a	The variable a in the exponential cone.
b	The variable b in the exponential cone.
c	The variable c in the exponential cone.
x, object	A ExpCone object.
eq_constr	A list of the equality constraints in the canonical problem.
leq_constr	A list of the inequality constraints in the canonical problem.
dims	A list with the dimensions of the conic constraints.
solver	A string representing the solver to be called.

Details

Original cone:

$$K = \{(a, b, c)|b > 0, be^{a/b} \leq c\} \cup \{(a, b, c)|a \leq 0, b = 0, c \geq 0\}$$

Reformulated cone:

$$K = \{(a, b, c)|b, c > 0, b\log(b) + a \leq b\log(c)\} \cup \{(a, b, c)|a \leq 0, b = 0, c \geq 0\}$$

Methods (by generic)

- `size`: The size of the `x` argument.
- `variables`: List of Variable objects in the exponential cone.
- `format_constr`: Format exponential cone constraints for the solver.

Slots

`constr_id` (Internal) A unique integer identification number used internally.

`a` The variable a in the exponential cone.

`b` The variable b in the exponential cone.

`c` The variable c in the exponential cone.

`Expression-class` *The Expression class.*

Description

This class represents a mathematical expression.

Usage

```
## S4 method for signature 'Expression'
value(object)

## S4 method for signature 'Expression'
grad(object)

## S4 method for signature 'Expression'
domain(object)

## S4 method for signature 'Expression'
as.character(x)

## S4 method for signature 'Expression'
name(object)

## S4 method for signature 'Expression'
curvature(object)

## S4 method for signature 'Expression'
is_constant(object)

## S4 method for signature 'Expression'
is_affine(object)
```

```
## S4 method for signature 'Expression'
is_convex(object)

## S4 method for signature 'Expression'
is_concave(object)

## S4 method for signature 'Expression'
is_dcp(object)

## S4 method for signature 'Expression'
is_quadratic(object)

## S4 method for signature 'Expression'
is_pwl(object)

## S4 method for signature 'Expression'
is_zero(object)

## S4 method for signature 'Expression'
is_positive(object)

## S4 method for signature 'Expression'
is_negative(object)

## S4 method for signature 'Expression'
size(object)

## S4 method for signature 'Expression'
is_scalar(object)

## S4 method for signature 'Expression'
is_vector(object)

## S4 method for signature 'Expression'
is_matrix(object)

## S4 method for signature 'Expression'
nrow(x)

## S4 method for signature 'Expression'
ncol(x)
```

Arguments

`x, object`	An Expression object.

Methods (by generic)

- `value`: The value of the expression.

- `grad`: The (sub/super)-gradient of the expression with respect to each variable.
- `domain`: A list of constraints describing the closure of the region where the expression is finite.
- `name`: The string representation of the expression.
- `curvature`: The curvature of the expression.
- `is_constant`: The expression is constant if it contains no variables or is identically zero.
- `is_affine`: The expression is affine if it is constant or both convex and concave.
- `is_convex`: A logical value indicating whether the expression is convex.
- `is_concave`: A logical value indicating whether the expression is concave.
- `is_dcp`: The expression is DCP if it is convex or concave.
- `is_quadratic`: A logical value indicating whether the expression is quadratic.
- `is_pwl`: A logical value indicating whether the expression is piecewise linear.
- `is_zero`: The expression is zero if it is both positive and negative.
- `is_positive`: A logical value indicating whether the expression is positive.
- `is_negative`: A logical value indicating whether the expression is negative.
- `size`: The `c(row, col)` dimensions of the expression.
- `is_scalar`: The expression is scalar if rows = cols = 1.
- `is_vector`: The expression is a vector if min(rows, cols) = 1.
- `is_matrix`: The expression is a matrix if rows > 1 and cols > 1.
- `nrow`: Number of rows in the expression.
- `ncol`: Number of columns in the expression.

`expression-parts` *Parts of an Expression*

Description

List the variables, parameters, or constants in a canonical expression.

Usage

```
variables(object)

parameters(object)

constants(object)
```

Arguments

`object`	A Canonical expression.

Value

A list of Variable, Parameter, or Constant objects.

Examples

```
m <- 50
n <- 10
beta <- Variable(n)
y <- matrix(rnorm(m), nrow = m)
X <- matrix(rnorm(m*n), nrow = m, ncol = n)
lambda <- Parameter()

expr <- sum_squares(y - X %*% beta) + lambda*p_norm(beta, 1)
variables(expr)
parameters(expr)
constants(expr)
lapply(constants(expr), function(c) { value(c) })
```

---

format_constr *Format Constraints*

Description

Format constraints for the solver.

Usage

```
format_constr(object, eq_constr, leq_constr, dims, solver)
```

Arguments

object	A Constraint object.
eq_constr	A list of the equality constraints in the canonical problem.
leq_constr	A list of the inequality constraints in the canonical problem.
dims	A list with the dimensions of the conic constraints.
solver	A string representing the solver to be called.

Value

A list containing equality constraints, inequality constraints, and dimensions.

format_results *Format Solver Results*

Description

Converts the solver output into standard form.

Usage

```
format_results(solver, results_dict, data, cached_data)
```

Arguments

solver	A Solver object.
results_dict	A list containing the solver output.
data	A list containing information about the problem.
cached_data	A list mapping solver name to cached problem data.

Value

A list containing the solver output in standard form.

GeoMean-class *The GeoMean class.*

Description

This class represents the (weighted) geometric mean of vector x with optional powers given by p.

Usage

```
GeoMean(x, p = NA_real_, max_denom = 1024)

## S4 method for signature 'GeoMean'
validate_args(object)

## S4 method for signature 'GeoMean'
to_numeric(object, values)

## S4 method for signature 'GeoMean'
size_from_args(object)

## S4 method for signature 'GeoMean'
sign_from_args(object)
```

```
## S4 method for signature 'GeoMean'
is_atom_convex(object)

## S4 method for signature 'GeoMean'
is_atom_concave(object)

## S4 method for signature 'GeoMean'
is_incr(object, idx)

## S4 method for signature 'GeoMean'
is_decr(object, idx)

## S4 method for signature 'GeoMean'
get_data(object)

## S4 method for signature 'GeoMean'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric vector.
p	(Optional) A vector of weights for the weighted geometric mean. The default is a vector of ones, giving the **unweighted** geometric mean $x_1^{1/n} \cdots x_n^{1/n}$.
max_denom	(Optional) The maximum denominator to use in approximating p/sum(p) with w. If w is not an exact representation, increasing max_denom may offer a more accurate representation, at the cost of requiring more convex inequalities to represent the geometric mean. Defaults to 1024.
object	A GeoMean object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Details

$$(x_1^{p_1} \cdots x_n^{p_n})^{\frac{1}{\mathbf{1}^T p}}$$

The geometric mean includes an implicit constraint that $x_i \geq 0$ whenever $p_i > 0$. If $p_i = 0, x_i$ will be unconstrained. The only exception to this rule occurs when p has exactly one nonzero element, say p_i, in which case GeoMean(x,p) is equivalent to x_i (without the nonnegativity constraint). A specific case of this is when $x \in \mathbf{R}^1$.

Methods (by generic)

- `validate_args`: Empty function since validation of arguments is done during atom initialization.
- `to_numeric`: The (weighted) geometric mean of the elements of `x`.
- `size_from_args`: The atom is a scalar.
- `sign_from_args`: The atom is non-negative.
- `is_atom_convex`: The atom is not convex.
- `is_atom_concave`: The atom is concave.
- `is_incr`: The atom is weakly increasing in every argument.
- `is_decr`: The atom is not weakly decreasing in any argument.
- `get_data`: Returns `list(w, dyadic completion, tree of dyads)`.
- `graph_implementation`: The graph implementation of the atom.

Slots

`x` An Expression or numeric vector.

`p` (Optional) A vector of weights for the weighted geometric mean. The default is a vector of ones, giving the **unweighted** geometric mean $x_1^{1/n} \cdots x_n^{1/n}$.

`max_denom` (Optional) The maximum denominator to use in approximating `p/sum(p)` with `w`. If `w` is not an exact representation, increasing `max_denom` may offer a more accurate representation, at the cost of requiring more convex inequalities to represent the geometric mean. Defaults to 1024.

`w` (Internal) A list of `bigq` objects that represent a rational approximation of `p/sum(p)`.

`approx_error` (Internal) The error in approximating `p/sum(p)` with `w`, given by $\|p/\mathbf{1}^T p - w\|_\infty$.

`geo_mean` *Geometric Mean*

Description

The (weighted) geometric mean of vector x with optional powers given by p.

Usage

```
geo_mean(x, p = NA_real_, max_denom = 1024)
```

Arguments

`x`	An Expression or vector.
`p`	(Optional) A vector of weights for the weighted geometric mean. Defaults to a vector of ones, giving the **unweighted** geometric mean $x_1^{1/n} \cdots x_n^{1/n}$.
`max_denom`	(Optional) The maximum denominator to use in approximating `p/sum(p)` with `w`. If `w` is not an exact representation, increasing `max_denom` may offer a more accurate representation, at the cost of requiring more convex inequalities to represent the geometric mean. Defaults to 1024.

Details

$$(x_1^{p_1} \cdots x_n^{p_n})^{\frac{1}{1^T p}}$$

The geometric mean includes an implicit constraint that $x_i \geq 0$ whenever $p_i > 0$. If $p_i = 0, x_i$ will be unconstrained. The only exception to this rule occurs when p has exactly one nonzero element, say p_i, in which case `geo_mean(x,p)` is equivalent to x_i (without the nonnegativity constraint). A specific case of this is when $x \in \mathbf{R}^1$.

Value

An Expression representing the geometric mean of the input.

Examples

```
x <- Variable(2)
cost <- geo_mean(x)
prob <- Problem(Maximize(cost), list(sum(x) <= 1))
result <- solve(prob)
result$value
result$getValue(x)

x <- Variable(5)
p <- c(0.07, 0.12, 0.23, 0.19, 0.39)
prob <- Problem(Maximize(geo_mean(x,p)), list(p_norm(x) <= 1))
result <- solve(prob)
result$value
result$getValue(x)
```

---

get_data *Get Expression Data*

Description

Get information needed to reconstruct the expression aside from its arguments.

Usage

```
get_data(object)
```

Arguments

`object`	A Expression object.

Value

A list containing data.

get_gurobiglue *Get our gurobiglue handle*

Description

Get the gurobiglue handle or fail if not available

Usage

```
get_gurobiglue()
```

Value

the gurobiglue handle

Examples

```
## Not run:
   get_gurobiglue

## End(Not run)
```

get_id *Get ID*

Description

Get the next identifier value.

Usage

```
get_id()
```

Value

A new unique integer identifier.

Examples

```
## Not run:
   get_id()

## End(Not run)
```

get_mosekglue *Get our mosekglue handle*

Description

Get the mosekglue handle or fail if not available

Usage

```
get_mosekglue()
```

Value

the mosekglue handle

Examples

```
## Not run:
   get_mosekglue

## End(Not run)
```

get_np *Get numpy handle*

Description

Get the numpy handle or fail if not available

Usage

```
get_np()
```

Value

the numpy handle

Examples

```
## Not run:
   get_np

## End(Not run)
```

get_problem_data *Get Problem Data*

Description

Get the problem data used in the call to the solver.

Usage

```
get_problem_data(object, solver)
```

Arguments

object	A Problem object.
solver	A string indicating the solver that the problem data is for. Call installed_solvers() to see all available.

Value

A list of arguments for the solver.

Examples

```
a <- Variable(name = "a")
data <- get_problem_data(Problem(Maximize(exp(a) + 2)), "SCS")
data[["dims"]]
data[["c"]]
data[["A"]]

x <- Variable(2, name = "x")
data <- get_problem_data(Problem(Minimize(p_norm(x) + 3)), "ECOS")
data[["dims"]]
data[["c"]]
data[["A"]]
data[["G"]]
```

get_sp *Get scipy handle*

Description

Get the scipy handle or fail if not available

Usage

```
get_sp()
```

Value

the scipy handle

Examples

```
## Not run:
   get_sp

## End(Not run)
```

GLPK-class *The GLPK class*

Description

This class is an interface for Gnu Linear Programming Toolkit solver

Usage

```
GLPK()

## S4 method for signature 'GLPK'
lp_capable(solver)

## S4 method for signature 'GLPK'
socp_capable(solver)

## S4 method for signature 'GLPK'
sdp_capable(solver)

## S4 method for signature 'GLPK'
exp_capable(solver)

## S4 method for signature 'GLPK'
mip_capable(solver)

## S4 method for signature 'GLPK'
name(object)

## S4 method for signature 'GLPK'
import_solver(solver)

## S4 method for signature 'GLPK'
Solver.solve(solver, objective, constraints, cached_data,
  warm_start, verbose, ...)

## S4 method for signature 'GLPK'
format_results(solver, results_dict, data, cached_data)
```

Arguments

`object, solver`	A GLPK object.
`objective`	A list representing the canonicalized objective.
`constraints`	A list of canonicalized constraints.
`cached_data`	A list mapping solver name to cached problem data.
`warm_start`	A logical value indicating whether the previous solver result should be used to warm start.
`verbose`	A logical value indicating whether to print solver output.
`...`	Additional arguments to the solver.
`results_dict`	A list containing the solver output.
`data`	A list containing information about the problem.

Methods (by generic)

- `lp_capable`: GLPK can handle linear programs.
- `socp_capable`: GLPK can handle second-order cone programs.
- `sdp_capable`: GLPK can handle semidefinite programs.
- `exp_capable`: GLPK cannot handle exponential cone programs.
- `mip_capable`: GLPK cannot handle mixed-integer programs.
- `name`: The name of the solver.
- `import_solver`: Imports the Rgpkk library.
- `Solver.solve`: Call the solver on the canonicalized problem.
- `format_results`: Convert raw solver output into standard list of results.

See Also

the Gnu GLKP site.

`grad` *Sub/Super-Gradient*

Description

The (sub/super)-gradient of the expression with respect to each variable. Matrix expressions are vectorized, so the gradient is a matrix. `NA` indicates variable values are unknown or outside the domain.

Usage

```
grad(object)
```

Arguments

`object`	An Expression object.

Value

A list mapping each variable to a sparse matrix.

Examples

```
x <- Variable(2, name = "x")
A <- Variable(2, 2, name = "A")

value(x) <- c(-3,4)
expr <- p_norm(x, 2)
grad(expr)

value(A) <- rbind(c(3,-4), c(4,3))
expr <- p_norm(A, 0.5)
grad(expr)

value(A) <- cbind(c(1,2), c(-1,0))
expr <- abs(A)
grad(expr)
```

`graph_implementation` *Graph Implementation*

Description

Reduces the atom to an affine expression and list of constraints.

Usage

```
graph_implementation(object, arg_objs, size, data)
```

Arguments

`object`	An Expression object.
`arg_objs`	A list of linear expressions for each argument.
`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Value

A list of `list(LinOp for objective, list of constraints)`, where LinOp is a list representing the linear operator.

GUROBI-class *The GUROBI class.*

Description

This class is an interface for the commercial GUROBI solver.

Usage

```
GUROBI()

## S4 method for signature 'GUROBI'
lp_capable(solver)

## S4 method for signature 'GUROBI'
socp_capable(solver)

## S4 method for signature 'GUROBI'
sdp_capable(solver)

## S4 method for signature 'GUROBI'
exp_capable(solver)

## S4 method for signature 'GUROBI'
mip_capable(solver)

## S4 method for signature 'GUROBI'
name(object)

## S4 method for signature 'GUROBI'
import_solver(solver)

## S4 method for signature 'GUROBI'
Solver.solve(solver, objective, constraints,
  cached_data, warm_start, verbose, ...)
```

Arguments

`object, solver`	A GUROBI object.
`objective`	A list representing the canonicalized objective.
`constraints`	A list of canonicalized constraints.
`cached_data`	A list mapping solver name to cached problem data.
`warm_start`	A logical value indicating whether the previous solver result should be used to warm start.
`verbose`	A logical value indicating whether to print solver output.
`...`	Additional arguments to the solver.

Methods (by generic)

- `lp_capable`: GUROBI can handle linear programs.
- `socp_capable`: GUROBI can handle second-order cone programs.
- `sdp_capable`: GUROBI cannot handle semidefinite programs.
- `exp_capable`: GUROBI cannot handle exponential cone programs.
- `mip_capable`: GUROBI can handle mixed-integer programs.
- `name`: The name of the solver.
- `import_solver`: Imports the reticulate library to use the python solver.
- `Solver.solve`: Call the solver on the canonicalized problem.

References

Gurobi optimizer reference manual version 5.0, Gurobi Optimization, Inc., Houston, Texas, July 2012.

See Also

the GUROBI Official Site.

`harmonic_mean` *Harmonic Mean*

Description

The harmonic mean, $\left(\frac{1}{n}\sum_{i=1}^{n} x_i^{-1}\right)^{-1}$. For a matrix, the function is applied over all entries.

Usage

```
harmonic_mean(x)
```

Arguments

`x`	An Expression, vector, or matrix.

Value

An Expression representing the harmonic mean of the input.

Examples

```
x <- Variable()
prob <- Problem(Maximize(harmonic_mean(x)), list(x >= 0, x <= 5))
result <- solve(prob)
result$value
result$getValue(x)
```

hstack *Horizontal Concatenation*

Description

The horizontal concatenation of expressions. This is equivalent to cbind when applied to objects with the same number of rows.

Usage

```
hstack(...)
```

Arguments

...	Expression objects, vectors, or matrices. All arguments must have the same number of rows.

Value

An Expression representing the concatenated inputs.

Examples

```
x <- Variable(2)
y <- Variable(3)
c <- matrix(1, nrow = 1, ncol = 5)
prob <- Problem(Minimize(c %*% t(hstack(t(x), t(y)))), list(x == c(1,2), y == c(3,4,5)))
result <- solve(prob)
result$value

c <- matrix(1, nrow = 1, ncol = 4)
prob <- Problem(Minimize(c %*% t(hstack(t(x), t(x)))), list(x == c(1,2)))
result <- solve(prob)
result$value

A <- Variable(2,2)
C <- Variable(3,2)
c <- matrix(1, nrow = 2, ncol = 2)
prob <- Problem(Minimize(sum_entries(hstack(t(A), t(C)))), list(A >= 2*c, C == -2))
result <- solve(prob)
result$value
result$getValue(A)

D <- Variable(3,3)
expr <- hstack(C, D)
obj <- expr[1,2] + sum(hstack(expr, expr))
constr <- list(C >= 0, D >= 0, D[1,1] == 2, C[1,2] == 3)
prob <- Problem(Minimize(obj), constr)
result <- solve(prob)
result$value
```

```
result$getValue(C)
result$getValue(D)
```

`HStack-class` *The HStack class.*

Description

Horizontal concatenation of values.

Usage

```
HStack(...)

## S4 method for signature 'HStack'
validate_args(object)

## S4 method for signature 'HStack'
to_numeric(object, values)

## S4 method for signature 'HStack'
size_from_args(object)

## S4 method for signature 'HStack'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

`...`	Expression objects or matrices. All arguments must have the same number of rows.
`object`	A HStack object.
`values`	A list of arguments to the atom.
`arg_objs`	A list of linear expressions for each argument.
`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Methods (by generic)

- `validate_args`: Check all arguments have the same height.
- `to_numeric`: Horizontally concatenate the values using `cbind`.
- `size_from_args`: The size of the atom.
- `graph_implementation`: The graph implementation of the atom.

Slots

`...` Expression objects or matrices. All arguments must have the same number of rows.

huber *Huber Function*

Description

The elementwise Huber function, $Huber(x, M) =$

- $2M|x| - M^2$for $|x| \geq |M|$
- $|x|^2$for $|x| \leq |M|$.

Usage

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

Arguments

x	An Expression, vector, or matrix.
M	(Optional) A positive scalar value representing the threshold. Defaults to 1.

Value

An Expression representing the Huber function evaluated at the input.

Examples

```
n <- 10
m <- 450
p <- 0.1    # Fraction of responses with sign flipped

# Generate problem data
beta_true <- 5*matrix(stats::rnorm(n), nrow = n)
X <- matrix(stats::rnorm(m*n), nrow = m, ncol = n)
y_true <- X %*% beta_true
eps <- matrix(stats::rnorm(m), nrow = m)

# Randomly flip sign of some responses
factor <- 2*rbinom(m, size = 1, prob = 1-p) - 1
y <- factor * y_true + eps

# Huber regression
beta <- Variable(n)
obj <- sum(huber(y - X %*% beta, 1))
prob <- Problem(Minimize(obj))
result <- solve(prob)
result$getValue(beta)
```

Huber-class *The Huber class.*

Description

This class represents the elementwise Huber function, $Huber(x, M) =$

- $2M|x| - M^2$for $|x| \geq |M|$
- $|x|^2$for $|x| \leq |M|$.

Usage

```
Huber(x, M = 1)

## S4 method for signature 'Huber'
validate_args(object)

## S4 method for signature 'Huber'
to_numeric(object, values)

## S4 method for signature 'Huber'
sign_from_args(object)

## S4 method for signature 'Huber'
is_atom_convex(object)

## S4 method for signature 'Huber'
is_atom_concave(object)

## S4 method for signature 'Huber'
is_incr(object, idx)

## S4 method for signature 'Huber'
is_decr(object, idx)

## S4 method for signature 'Huber'
get_data(object)

## S4 method for signature 'Huber'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression object.
M	A positive scalar value representing the threshold. Defaults to 1.
object	A Huber object.

values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check that M is a non-negative constant.
- to_numeric: The Huber function evaluted elementwise on the input value.
- sign_from_args: The atom is positive.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is not concave.
- is_incr: A logical value indicating whether the atom is weakly increasing.
- is_decr: A logical value indicating whether the atom is weakly decreasing.
- get_data: A list containing the parameter M.
- graph_implementation: The graph implementation of the atom.

Slots

x An Expression or numeric constant.

M A positive scalar value representing the threshold. Defaults to 1.

---

id *Identification Number*

Description

A unique identification number used internally to keep track of variables and constraints. Should not be modified by the user.

Usage

```
id(object)
```

Arguments

object	A Variable or Constraint object.

Value

A non-negative integer identifier.

See Also

get_id setIdCounter

Examples

```
x <- Variable()
constr <- (x >= 5)
id(x)
id(constr)
```

---

import_solver *Import Solver*

Description

Import the R library that interfaces with the specified solver.

Usage

```
import_solver(solver)
```

Arguments

solver	A Solver object.

Examples

```
import_solver(ECOS())
import_solver(SCS())
```

---

installed_solvers *Installed Solvers*

Description

Installed Solvers

Usage

```
installed_solvers()
```

Value

The names of all the installed solvers.

Int-class *The Int class.*

Description

This class represents an integer variable.

Usage

```
Int(rows = 1, cols = 1, name = NA_character_)

## S4 method for signature 'Int'
as.character(x)

## S4 method for signature 'Int'
canonicalize(object)
```

Arguments

rows	The number of rows in the variable.
cols	The number of columns in the variable.
name	(Optional) A character string representing the name of the variable.
x, object	An Int object.

Methods (by generic)

- canonicalize: Enforce that the variable be an integer.

Slots

id (Internal) A unique identification number used internally.

rows The number of rows in the variable.

cols The number of columns in the variable.

name (Optional) A character string representing the name of the variable.

primal_value (Internal) The primal value of the variable stored internally.

Examples

```
x <- Int(3, name = "i") ## 3-int variable
y <- Int(3, 3, name = "j") # Matrix variable
as.character(y)
id(y)
is_positive(x)
is_negative(x)
size(y)
name(y)
```

```
value(y) <- matrix(1:9, nrow = 3)
value(y)
grad(y)
variables(y)
canonicalize(y)
```

IntConstr-class *The IntConstr class.*

Description

This class represents an integer constraint, $X_{ij} \in \mathbf{Z}$ for all i, j.

Usage

```
IntConstr(lin_op)
```

Arguments

lin_op	A list representing the linear operator equal to the .noncvx_var.

Slots

constr_id (Internal) A unique integer identification number used internally.

lin_op A list representing the linear operator equal to the .noncvx_var.

.noncvx_var (Internal) A list representing the variable constrained to be elementwise integer.

---

inv_pos *Reciprocal Function*

Description

The elementwise reciprocal function, $\frac{1}{x}$

Usage

```
inv_pos(x)
```

Arguments

x	An Expression, vector, or matrix.

Value

An Expression representing the reciprocal of the input.

Examples

```
A <- Variable(2,2)
val <- cbind(c(1,2), c(3,4))
prob <- Problem(Minimize(inv_pos(A)[1,2]), list(A == val))
result <- solve(prob)
result$value
```

`is_dcp` *DCP Compliance*

Description

Determine if a problem or expression complies with the disciplined convex programming rules.

Usage

```
is_dcp(object)
```

Arguments

`object`	A Problem or Expression object.

Value

A logical value indicating whether the problem or expression is DCP compliant, i.e. no unknown curvatures.

Examples

```
x <- Variable()
prob <- Problem(Minimize(x^2), list(x >= 5))
is_dcp(prob)
solve(prob)
```

`is_qp` *Is Problem a QP?*

Description

Determine if a problem is a quadratic program.

Usage

```
is_qp(object)
```

Arguments

object	A Problem object.

Value

A logical value indicating whether the problem is a quadratic program.

KLDiv-class *The KLDiv class.*

Description

The elementwise KL-divergence $x \log(x/y) - x + y$.

Usage

```
KLDiv(x, y)

## S4 method for signature 'KLDiv'
to_numeric(object, values)

## S4 method for signature 'KLDiv'
sign_from_args(object)

## S4 method for signature 'KLDiv'
is_atom_convex(object)

## S4 method for signature 'KLDiv'
is_atom_concave(object)

## S4 method for signature 'KLDiv'
is_incr(object, idx)

## S4 method for signature 'KLDiv'
is_decr(object, idx)

## S4 method for signature 'KLDiv'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric constant.
y	An Expression or numeric constant.
object	A KLDiv object.
values	A list of arguments to the atom.

idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The KL-divergence evaluted elementwise on the input value.
- sign_from_args: The atom is positive.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is not concave.
- is_incr: The atom is not monotonic in any argument.
- is_decr: The atom is not monotonic in any argument.
- graph_implementation: The graph implementation of the atom.

Slots

x An Expression or numeric constant.

y An Expression or numeric constant.

---

kl_div *Kullback-Leibler Divergence*

Description

The elementwise Kullback-Leibler divergence, $x \log(x/y) - x + y$.

Usage

```
kl_div(x, y)
```

Arguments

x	An Expression, vector, or matrix.
y	An Expression, vector, or matrix.

Value

An Expression representing the KL-divergence of the input.

Examples

```
n <- 5
alpha <- seq(10, n-1+10)/n
beta <- seq(10, n-1+10)/n
P_tot <- 0.5
W_tot <- 1.0

P <- Variable(n)
W <- Variable(n)
R <- kl_div(alpha*W, alpha*(W + beta*P)) - alpha*beta*P
obj <- sum(R)
constr <- list(P >= 0, W >= 0, sum(P) == P_tot, sum(W) == W_tot)
prob <- Problem(Minimize(obj), constr)
result <- solve(prob)

result$value
result$getValue(P)
result$getValue(W)
```

Kron-class *The Kron class.*

Description

This class represents the kronecker product.

Usage

```
Kron(lh_exp, rh_exp)

## S4 method for signature 'Kron'
validate_args(object)

## S4 method for signature 'Kron'
to_numeric(object, values)

## S4 method for signature 'Kron'
size_from_args(object)

## S4 method for signature 'Kron'
sign_from_args(object)

## S4 method for signature 'Kron'
is_incr(object, idx)

## S4 method for signature 'Kron'
is_decr(object, idx)
```

```
## S4 method for signature 'Kron'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

lh_exp	An Expression or numeric constant representing the left-hand matrix.
rh_exp	An Expression or numeric constant representing the right-hand matrix.
object	A Kron object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check both arguments are vectors and the first is a constant.
- to_numeric: The kronecker product of the two values.
- size_from_args: The size of the atom.
- sign_from_args: The sign of the atom.
- is_incr: Is the left-hand expression positive?
- is_decr: Is the right-hand expression negative?
- graph_implementation: The graph implementation of the atom.

Slots

lh_exp An Expression or numeric constant representing the left-hand matrix.

rh_exp An Expression or numeric constant representing the right-hand matrix.

---

kronecker,Expression,ANY-method

Kronecker Product

Description

The generalized kronecker product of two matrices.

Usage

```
## S4 method for signature 'Expression,ANY'
kronecker(X, Y, FUN = "*",
  make.dimnames = FALSE, ...)

## S4 method for signature 'ANY,Expression'
kronecker(X, Y, FUN = "*",
  make.dimnames = FALSE, ...)

## S4 method for signature 'Expression,ANY'
X %x% Y

## S4 method for signature 'ANY,Expression'
X %x% Y
```

Arguments

X	An Expression or matrix.
Y	An Expression or matrix.
FUN	Hardwired to "*" for the kronecker product.
make.dimnames	(Unimplemented) Dimension names are not supported in Expression objects.
...	(Unimplemented) Optional arguments.

Value

An Expression that represents the kronecker product.

Examples

```
X <- cbind(c(1,2), c(3,4))
Y <- Variable(2,2)
val <- cbind(c(5,6), c(7,8))

obj <- X %x% Y
prob <- Problem(Minimize(kronecker(X,Y)[1,1]), list(Y == val))
result <- solve(prob)
result$value
result$getValue(kronecker(X,Y))
```

LambdaMax-class *The LambdaMax class.*

Description

The maximum eigenvalue of a matrix, $\lambda_{\max}(A)$.

Usage

```
LambdaMax(A)

## S4 method for signature 'LambdaMax'
validate_args(object)

## S4 method for signature 'LambdaMax'
to_numeric(object, values)

## S4 method for signature 'LambdaMax'
size_from_args(object)

## S4 method for signature 'LambdaMax'
sign_from_args(object)

## S4 method for signature 'LambdaMax'
is_atom_convex(object)

## S4 method for signature 'LambdaMax'
is_atom_concave(object)

## S4 method for signature 'LambdaMax'
is_incr(object, idx)

## S4 method for signature 'LambdaMax'
is_decr(object, idx)

## S4 method for signature 'LambdaMax'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

A	An Expression or numeric matrix.
object	A LambdaMax object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check that A is square.
- to_numeric: The largest eigenvalue of A. Requires that A be symmetric.
- size_from_args: The atom is a scalar.

- `sign_from_args`: The sign of the atom is unknown.
- `is_atom_convex`: The atom is convex.
- `is_atom_concave`: The atom is not concave.
- `is_incr`: The atom is not monotonic in any argument.
- `is_decr`: The atom is not monotonic in any argument.
- `graph_implementation`: The graph implementation of the atom.

Slots

`A` An Expression or numeric matrix.

`lambda_max` *Maximum Eigenvalue*

Description

The maximum eigenvalue of a matrix, $\lambda_{\max}(A)$.

Usage

```
lambda_max(A)
```

Arguments

`A`	An Expression or matrix.

Value

An Expression representing the maximum eigenvalue of the input.

Examples

```
A <- Variable(2,2)
prob <- Problem(Minimize(lambda_max(A)), list(A >= 2))
result <- solve(prob)
result$value
result$getValue(A)

obj <- Maximize(A[2,1] - A[1,2])
prob <- Problem(obj, list(lambda_max(A) <= 100, A[1,1] == 2, A[2,2] == 2, A[2,1] == 2))
result <- solve(prob)
result$value
result$getValue(A)
```

lambda_min *Minimum Eigenvalue*

Description

The minimum eigenvalue of a matrix, $\lambda_{\min}(A)$.

Usage

```
lambda_min(A)
```

Arguments

A	An Expression or matrix.

Value

An Expression representing the minimum eigenvalue of the input.

Examples

```
A <- Variable(2,2)
val <- cbind(c(5,7), c(7,-3))
prob <- Problem(Maximize(lambda_min(A)), list(A == val))
result <- solve(prob)
result$value
result$getValue(A)
```

lambda_sum_largest *Sum of Largest Eigenvalues*

Description

The sum of the largest k eigenvalues of a matrix.

Usage

```
lambda_sum_largest(A, k)
```

Arguments

A	An Expression or matrix.
k	The number of eigenvalues to sum over.

Value

An Expression representing the sum of the largest k eigenvalues of the input.

Examples

```
C <- Variable(3,3)
val <- cbind(c(1,2,3), c(2,4,5), c(3,5,6))
prob <- Problem(Minimize(lambda_sum_largest(C,2)), list(C == val))
result <- solve(prob)
result$value
result$getValue(C)
```

---

lambda_sum_smallest *Sum of Smallest Eigenvalues*

Description

The sum of the smallest k eigenvalues of a matrix.

Usage

```
lambda_sum_smallest(A, k)
```

Arguments

A	An Expression or matrix.
k	The number of eigenvalues to sum over.

Value

An Expression representing the sum of the smallest k eigenvalues of the input.

Examples

```
C <- Variable(3,3)
val <- cbind(c(1,2,3), c(2,4,5), c(3,5,6))
prob <- Problem(Maximize(lambda_sum_smallest(C,2)), list(C == val))
result <- solve(prob)
result$value
result$getValue(C)
```

`Leaf-class` *The Leaf class.*

Description

This class represents a leaf node, i.e. a Variable, Constant, or Parameter.

Usage

```
## S4 method for signature 'Leaf'
variables(object)

## S4 method for signature 'Leaf'
parameters(object)

## S4 method for signature 'Leaf'
constants(object)

## S4 method for signature 'Leaf'
is_convex(object)

## S4 method for signature 'Leaf'
is_concave(object)

## S4 method for signature 'Leaf'
is_quadratic(object)

## S4 method for signature 'Leaf'
is_pwl(object)

## S4 method for signature 'Leaf'
domain(object)

## S4 method for signature 'Leaf'
validate_val(object, val)
```

Arguments

`object`	A Leaf object.
`val`	The assigned value.

Methods (by generic)

- `variables`: List of Variable objects in the leaf node.
- `parameters`: List of Parameter objects in the leaf node.
- `constants`: List of Constant objects in the leaf node.

- `is_convex`: A logical value indicating whether the leaf node is convex.
- `is_concave`: A logical value indicating whether the leaf node is concave.
- `is_quadratic`: A logical value indicating whether the leaf node is quadratic.
- `is_pwl`: A logical value indicating whether the leaf node is piecewise linear.
- `domain`: A list of constraints describing the closure of the region where the leaf node is finite. Default is the full domain.
- `validate_val`: Check that `val` satisfies symbolic attributes of leaf.

Slots

`args` A list containing the arguments.

`log,Expression-method` *Logarithms*

Description

The elementwise logarithm. `log` computes the logarithm, by default the natural logarithm, `log10` computes the common (i.e., base 10) logarithm, and `log2` computes the binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`. `log1p` computes elementwise the function $\log(1 + x)$.

Usage

```
## S4 method for signature 'Expression'
log(x, base = exp(1))

## S4 method for signature 'Expression'
log10(x)

## S4 method for signature 'Expression'
log2(x)

## S4 method for signature 'Expression'
log1p(x)
```

Arguments

`x`	An Expression.
`base`	(Optional) A positive number that is the base with respect to which the logarithm is computed. Defaults to e.

Value

An Expression representing the exponentiated input.

Examples

```
# Log in objective
x <- Variable(2)
obj <- Maximize(sum(log(x)))
constr <- list(x <= matrix(c(1, exp(1))))
prob <- Problem(obj, constr)
result <- solve(prob)
result$value
result$getValue(x)

# Log in constraint
obj <- Minimize(sum(x))
constr <- list(log2(x) >= 0, x <= matrix(c(1,1)))
prob <- Problem(obj, constr)
result <- solve(prob)
result$value
result$getValue(x)

# Index into log
obj <- Maximize(log10(x)[2])
constr <- list(x <= matrix(c(1, exp(1))))
prob <- Problem(obj, constr)
result <- solve(prob)
result$value

# Scalar log
obj <- Maximize(log1p(x[2]))
constr <- list(x <= matrix(c(1, exp(1))))
prob <- Problem(obj, constr)
result <- solve(prob)
result$value
```

Log-class *The Log class.*

Description

This class represents the elementwise natural logarithm $\log(x)$.

Usage

```
Log(x)

## S4 method for signature 'Log'
to_numeric(object, values)

## S4 method for signature 'Log'
sign_from_args(object)
```

```
## S4 method for signature 'Log'
is_atom_convex(object)

## S4 method for signature 'Log'
is_atom_concave(object)

## S4 method for signature 'Log'
is_incr(object, idx)

## S4 method for signature 'Log'
is_decr(object, idx)

## S4 method for signature 'Log'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric constant.
object	A Log object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The elementwise natural logarithm of the input value.
- sign_from_args: The sign of the atom is unknown.
- is_atom_convex: The atom is not convex.
- is_atom_concave: The atom is concave.
- is_incr: The atom is weakly increasing.
- is_decr: The atom is not weakly decreasing.
- graph_implementation: The graph implementation of the atom.

Slots

x An Expression or numeric constant.

Log1p-class *The Log1p class.*

Description

This class represents the elementwise operation $\log(1 + x)$.

Usage

```
Log1p(x)

## S4 method for signature 'Log1p'
to_numeric(object, values)

## S4 method for signature 'Log1p'
sign_from_args(object)

## S4 method for signature 'Log1p'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric constant.
object	A Log1p object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The elementwise natural logarithm of one plus the input value.
- sign_from_args: The sign of the atom.
- graph_implementation: The graph implementation of the atom.

Slots

x An Expression or numeric constant.

LogDet-class *The LogDet class.*

Description

The natural logarithm of the determinant of a matrix, $\log\det(A)$.

Usage

```
LogDet(A)

## S4 method for signature 'LogDet'
validate_args(object)

## S4 method for signature 'LogDet'
to_numeric(object, values)

## S4 method for signature 'LogDet'
size_from_args(object)

## S4 method for signature 'LogDet'
sign_from_args(object)

## S4 method for signature 'LogDet'
is_atom_convex(object)

## S4 method for signature 'LogDet'
is_atom_concave(object)

## S4 method for signature 'LogDet'
is_incr(object, idx)

## S4 method for signature 'LogDet'
is_decr(object, idx)

## S4 method for signature 'LogDet'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

A	An Expression or numeric matrix.
object	A LogDet object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.

size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check that A is square.
- to_numeric: The log-determinant of SDP matrix A. This is the sum of logs of the eigenvalues and is equivalent to the nuclear norm of the matrix logarithm of A.
- size_from_args: The atom is a scalar.
- sign_from_args: The atom is non-negative.
- is_atom_convex: The atom is not convex.
- is_atom_concave: The atom is concave.
- is_incr: The atom is not monotonic in any argument.
- is_decr: The atom is not monotonic in any argument.
- graph_implementation: The graph implementation of the atom.

Slots

A An Expression or numeric matrix.

---

logistic *Logistic Function*

Description

The elementwise logistic function, $\log(1+e^x)$. This is a special case of log(sum(exp)) that evaluates to a vector rather than to a scalar, which is useful for logistic regression.

Usage

```
logistic(x)
```

Arguments

x	An Expression, vector, or matrix.

Value

An Expression representing the logistic function evaluated at the input.

Examples

```
n <- 20
m <- 1000
sigma <- 45

beta_true <- stats::rnorm(n)
idxs <- sample(n, size = 0.8*n, replace = FALSE)
beta_true[idxs] <- 0
X <- matrix(stats::rnorm(m*n, 0, 5), nrow = m, ncol = n)
y <- sign(X %*% beta_true + stats::rnorm(m, 0, sigma))

beta <- Variable(n)
X_sign <- apply(X, 2, function(x) { ifelse(y <= 0, -1, 1) * x })
obj <- -sum(logistic(-X[y <= 0,] %*% beta)) - sum(logistic(X[y == 1,] %*% beta))
prob <- Problem(Maximize(obj))
result <- solve(prob)

log_odds <- result$getValue(X %*% beta)
beta_res <- result$getValue(beta)
y_probs <- 1/(1 + exp(-X %*% beta_res))
log(y_probs/(1 - y_probs))
```

Logistic-class *The Logistic class.*

Description

This class represents the elementwise operation $\log(1+e^x)$. This is a special case of log(sum(exp)) that evaluates to a vector rather than to a scalar, which is useful for logistic regression.

Usage

```
Logistic(x)

## S4 method for signature 'Logistic'
to_numeric(object, values)

## S4 method for signature 'Logistic'
sign_from_args(object)

## S4 method for signature 'Logistic'
is_atom_convex(object)

## S4 method for signature 'Logistic'
is_atom_concave(object)

## S4 method for signature 'Logistic'
is_incr(object, idx)
```

```
## S4 method for signature 'Logistic'
is_decr(object, idx)

## S4 method for signature 'Logistic'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric constant.
object	A Logistic object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- `to_numeric`: Evaluates `e^x` elementwise, adds one, and takes the natural logarithm.
- `sign_from_args`: The atom is positive.
- `is_atom_convex`: The atom is convex.
- `is_atom_concave`: The atom is not concave.
- `is_incr`: The atom is weakly increasing.
- `is_decr`: The atom is not weakly decreasing.
- `graph_implementation`: The graph implementation of the atom.

Slots

`x` An Expression or numeric constant.

LogSumExp-class *The LogSumExp class.*

Description

The natural logarithm of the sum of the elementwise exponential, $\log \sum_{i=1}^{n} e^{x_i}$.

Usage

```
LogSumExp(x, axis = NA_real_)

## S4 method for signature 'LogSumExp'
to_numeric(object, values)

## S4 method for signature 'LogSumExp'
sign_from_args(object)

## S4 method for signature 'LogSumExp'
is_atom_convex(object)

## S4 method for signature 'LogSumExp'
is_atom_concave(object)

## S4 method for signature 'LogSumExp'
is_incr(object, idx)

## S4 method for signature 'LogSumExp'
is_decr(object, idx)

## S4 method for signature 'LogSumExp'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression representing a vector or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.
object	A LogSumExp object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: Evaluates e^x elementwise, sums, and takes the natural log.
- sign_from_args: The atom is positive.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is not concave.
- is_incr: The atom is not monotonic in any argument.
- is_decr: The atom is not monotonic in any argument.
- graph_implementation: The graph implementation of the atom.

Slots

`x` An Expression representing a vector or matrix.

`axis` (Optional) The dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `NA`.

`log_det` *Log-Determinant*

Description

The natural logarithm of the determinant of a matrix, $\log\det(A)$.

Usage

```
log_det(A)
```

Arguments

`A`	An Expression or matrix.

Value

An Expression representing the log-determinant of the input.

Examples

```
x <- t(data.frame(c(0.55, 0.25, -0.2, -0.25, -0.0, 0.4),
                  c(0.0, 0.35, 0.2, -0.1, -0.3, -0.2)))
n <- nrow(x)
m <- ncol(x)

A <- Variable(n,n)
b <- Variable(n)
obj <- Maximize(log_det(A))
constr <- lapply(1:m, function(i) { p_norm(A %*% as.matrix(x[,i]) + b) <= 1 })
prob <- Problem(obj, constr)
result <- solve(prob)
result$value
```

log_sum_exp *Log-Sum-Exponential*

Description

The natural logarithm of the sum of the elementwise exponential, $\log \sum_{i=1}^{n} e^{x_i}$.

Usage

```
log_sum_exp(x, axis = NA_real_)
```

Arguments

x	An Expression, vector, or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.

Value

An Expression representing the log-sum-exponential of the input.

Examples

```
A <- Variable(2,2)
val <- cbind(c(5,7), c(0,-3))
prob <- Problem(Minimize(log_sum_exp(A)), list(A == val))
result <- solve(prob)
result$getValue(A)
```

LPSOLVE-class *The LPSOLVE class*

Description

This class is an interface for Gnu Linear Programming Toolkit solver

Usage

```
LPSOLVE()

## S4 method for signature 'LPSOLVE'
lp_capable(solver)

## S4 method for signature 'LPSOLVE'
socp_capable(solver)
```

```
## S4 method for signature 'LPSOLVE'
sdp_capable(solver)

## S4 method for signature 'LPSOLVE'
exp_capable(solver)

## S4 method for signature 'LPSOLVE'
mip_capable(solver)

## S4 method for signature 'LPSOLVE'
name(object)

## S4 method for signature 'LPSOLVE'
import_solver(solver)

## S4 method for signature 'LPSOLVE'
Solver.solve(solver, objective, constraints,
  cached_data, warm_start, verbose, ...)

## S4 method for signature 'LPSOLVE'
format_results(solver, results_dict, data, cached_data)
```

Arguments

`object, solver`	A LPSOLVE object.
`objective`	A list representing the canonicalized objective.
`constraints`	A list of canonicalized constraints.
`cached_data`	A list mapping solver name to cached problem data.
`warm_start`	A logical value indicating whether the previous solver result should be used to warm start.
`verbose`	A logical value indicating whether to print solver output.
`...`	Additional arguments to the solver.
`results_dict`	A list containing the solver output.
`data`	A list containing information about the problem.

Methods (by generic)

- `lp_capable`: LPSOLVE can handle linear programs.
- `socp_capable`: LPSOLVE can handle second-order cone programs.
- `sdp_capable`: LPSOLVE can handle semidefinite programs.
- `exp_capable`: LPSOLVE cannot handle exponential cone programs.
- `mip_capable`: LPSOLVE cannot handle mixed-integer programs.
- `name`: The name of the solver.
- `import_solver`: Imports the Rmosek library.
- `Solver.solve`: Call the solver on the canonicalized problem.
- `format_results`: Convert raw solver output into standard list of results.

See Also

the CRAN lpSolveAPI package.

`MatrixFrac-class` *The MatrixFrac class.*

Description

The matrix fraction function $tr(X^T P^{-1} X)$.

Usage

```
MatrixFrac(X, P)

## S4 method for signature 'MatrixFrac'
validate_args(object)

## S4 method for signature 'MatrixFrac'
to_numeric(object, values)

## S4 method for signature 'MatrixFrac'
size_from_args(object)

## S4 method for signature 'MatrixFrac'
sign_from_args(object)

## S4 method for signature 'MatrixFrac'
is_atom_convex(object)

## S4 method for signature 'MatrixFrac'
is_atom_concave(object)

## S4 method for signature 'MatrixFrac'
is_incr(object, idx)

## S4 method for signature 'MatrixFrac'
is_decr(object, idx)

## S4 method for signature 'MatrixFrac'
is_quadratic(object)

## S4 method for signature 'MatrixFrac'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

X	An Expression or numeric matrix.
P	An Expression or numeric matrix.
object	A MatrixFrac object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check that the dimensions of x and P match.
- to_numeric: The trace of $X^T P^{-1} X$.
- size_from_args: The atom is a scalar.
- sign_from_args: The atom is positive.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is not concave.
- is_incr: The atom is not monotonic in any argument.
- is_decr: The atom is not monotonic in any argument.
- is_quadratic: True if x is affine and P is constant.
- graph_implementation: The graph implementation of the atom.

Slots

X An Expression or numeric matrix.

P An Expression or numeric matrix.

---

matrix_frac *Matrix Fraction*

Description

$tr(X^T P^{-1} X)$.

Usage

```
matrix_frac(X, P)
```

Arguments

X	An Expression or matrix. Must have the same number of rows as P.
P	An Expression or matrix. Must be an invertible square matrix.

Value

An Expression representing the matrix fraction evaluated at the input.

Examples

```
## Not run:
m <- 100
n <- 80
r <- 70

A <- matrix(stats::rnorm(m*n), nrow = m, ncol = n)
b <- matrix(stats::rnorm(m), nrow = m, ncol = 1)
G <- matrix(stats::rnorm(r*n), nrow = r, ncol = n)
h <- matrix(stats::rnorm(r), nrow = r, ncol = 1)

# ||Ax-b||^2 = x^T (A^T A) x - 2(A^T b)^T x + ||b||^2
P <- t(A) %*% A
q <- -2 * t(A) %*% b
r <- t(b) %*% b
Pinv <- base::solve(P)

x <- Variable(n)
obj <- matrix_frac(x, Pinv) + t(q) %*% x + r
constr <- list(G %*% x == h)
prob <- Problem(Minimize(obj), constr)
result <- solve(prob)
result$value

## End(Not run)
```

---

matrix_trace *Matrix Trace*

Description

The sum of the diagonal entries in a matrix.

Usage

```
matrix_trace(expr)
```

Arguments

expr	An Expression or matrix.

Value

An Expression representing the trace of the input.

Examples

```
C <- Variable(3,3)
val <- cbind(3:5, 6:8, 9:11)
prob <- Problem(Maximize(matrix_trace(C)), list(C == val))
result <- solve(prob)
result$value
```

MaxElemwise-class *The MaxElemwise class.*

Description

This class represents the elementwise maximum.

Usage

```
MaxElemwise(arg1, arg2, ...)

## S4 method for signature 'MaxElemwise'
to_numeric(object, values)

## S4 method for signature 'MaxElemwise'
sign_from_args(object)

## S4 method for signature 'MaxElemwise'
is_atom_convex(object)

## S4 method for signature 'MaxElemwise'
is_atom_concave(object)

## S4 method for signature 'MaxElemwise'
is_incr(object, idx)

## S4 method for signature 'MaxElemwise'
is_decr(object, idx)

## S4 method for signature 'MaxElemwise'
is_pwl(object)

## S4 method for signature 'MaxElemwise'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

arg1	The first Expression in the maximum operation.
arg2	The second Expression in the maximum operation.

`...`	Additional Expression objects in the maximum operation.
`object`	A MaxElemwise object.
`values`	A list of arguments to the atom.
`idx`	An index into the atom.
`arg_objs`	A list of linear expressions for each argument.
`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Methods (by generic)

- `to_numeric`: The elementwise maximum.
- `sign_from_args`: The sign of the atom.
- `is_atom_convex`: The atom is convex.
- `is_atom_concave`: The atom is not concave.
- `is_incr`: The atom is weakly increasing.
- `is_decr`: The atom is not weakly decreasing.
- `is_pwl`: Are all the arguments piecewise linear?
- `graph_implementation`: The graph implementation of the atom.

Slots

`arg1` The first Expression in the maximum operation.

`arg2` The second Expression in the maximum operation.

`...` Additional Expression objects in the maximum operation.

`MaxEntries-class` *The MaxEntries class.*

Description

The maximum of an expression.

Usage

```
MaxEntries(x, axis = NA_real_)

## S4 method for signature 'MaxEntries'
to_numeric(object, values)

## S4 method for signature 'MaxEntries'
sign_from_args(object)

## S4 method for signature 'MaxEntries'
```

```
is_atom_convex(object)

## S4 method for signature 'MaxEntries'
is_atom_concave(object)

## S4 method for signature 'MaxEntries'
is_incr(object, idx)

## S4 method for signature 'MaxEntries'
is_decr(object, idx)

## S4 method for signature 'MaxEntries'
is_pwl(object)

## S4 method for signature 'MaxEntries'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression representing a vector or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.
object	A MaxEntries object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The largest entry in x.
- sign_from_args: The sign of the atom.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is not concave.
- is_incr: The atom is weakly increasing in every argument.
- is_decr: The atom is not weakly decreasing in any argument.
- is_pwl: Is x piecewise linear?
- graph_implementation: The graph implementation of the atom.

Slots

x An Expression representing a vector or matrix.

axis (Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.

`Maximize-class` *The Maximize class.*

Description

This class represents an optimization objective for maximization.

Usage

```
Maximize(expr)

## S4 method for signature 'Maximize'
canonicalize(object)

## S4 method for signature 'Maximize'
is_dcp(object)

## S4 method for signature 'Maximize'
is_quadratic(object)
```

Arguments

`expr`	A scalar Expression to maximize.
`object`	A Maximize object.

Methods (by generic)

- `canonicalize`: Negates the target expression's objective.
- `is_dcp`: A logical value indicating whether the objective is concave.
- `is_quadratic`: A logical value indicating whether the objective is quadratic.

Slots

`expr` A scalar Expression to maximize.

Examples

```
x <- Variable(3)
alpha <- c(0.8,1.0,1.2)
obj <- sum(log(alpha + x))
constr <- list(x >= 0, sum(x) == 1)
prob <- Problem(Maximize(obj), constr)
result <- solve(prob)
result$value
result$getValue(x)
```

max_elemwise *Elementwise Maximum*

Description

The elementwise maximum.

Usage

```
max_elemwise(arg1, arg2, ...)
```

Arguments

arg1	An Expression, vector, or matrix.
arg2	An Expression, vector, or matrix.
...	Additional Expression objects, vectors, or matrices.

Value

An Expression representing the elementwise maximum of the inputs.

Examples

```
c <- matrix(c(1,-1))
prob <- Problem(Minimize(max_elemwise(t(c), 2, 2 + t(c))[2]))
result <- solve(prob)
result$value
```

max_entries *Maximum*

Description

The maximum of an expression.

Usage

```
max_entries(x, axis = NA_real_)

## S3 method for class 'Expression'
max(..., na.rm = FALSE)
```

Arguments

x	An Expression, vector, or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.
...	Numeric scalar, vector, matrix, or Expression objects.
na.rm	(Unimplemented) A logical value indicating whether missing values should be removed.

Value

An Expression representing the maximum of the input.

Examples

```
x <- Variable(2)
val <- matrix(c(-5,-10))
prob <- Problem(Minimize(max_entries(x)), list(x == val))
result <- solve(prob)
result$value

A <- Variable(2,2)
val <- rbind(c(-5,2), c(-3,1))
prob <- Problem(Minimize(max_entries(A, axis = 1)[2,1]), list(A == val))
result <- solve(prob)
result$value
```

---

mean.Expression *Arithmetic Mean*

Description

The arithmetic mean of an expression.

Usage

```
## S3 method for class 'Expression'
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	An Expression object.
trim	(Unimplemented) The fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed.
na.rm	(Unimplemented) A logical value indicating whether missing values should be removed.
...	(Unimplemented) Optional arguments.

Value

An Expression representing the mean of the input.

Examples

```
A <- Variable(2,2)
val <- cbind(c(-5,2), c(-3,1))
prob <- Problem(Minimize(mean(A)), list(A == val))
result <- solve(prob)
result$value
```

Minimize-class *The Minimize class.*

Description

This class represents an optimization objective for minimization.

Usage

```
Minimize(expr)

## S4 method for signature 'Minimize'
canonicalize(object)

## S4 method for signature 'Minimize'
variables(object)

## S4 method for signature 'Minimize'
parameters(object)

## S4 method for signature 'Minimize'
constants(object)

## S4 method for signature 'Minimize'
is_dcp(object)

## S4 method for signature 'Minimize'
value(object)
```

Arguments

expr	A scalar Expression to minimize.
object	A Minimize object.

Methods (by generic)

- `canonicalize`: Pass on the target expression's objective and constraints.
- `variables`: List of Variable objects in the objective.
- `parameters`: List of Parameter objects in the objective.
- `constants`: List of Constant objects in the objective.
- `is_dcp`: A logical value indicating whether the objective is convex.
- `value`: The value of the objective expression.

Slots

`expr` A scalar Expression to minimize.

`min_elemwise` *Elementwise Minimum*

Description

The elementwise minimum.

Usage

```
min_elemwise(arg1, arg2, ...)
```

Arguments

`arg1`	An Expression, vector, or matrix.
`arg2`	An Expression, vector, or matrix.
`...`	Additional Expression objects, vectors, or matrices.

Value

An Expression representing the elementwise minimum of the inputs.

Examples

```
a <- cbind(c(-5,2), c(-3,-1))
b <- cbind(c(5,4), c(-1,2))
prob <- Problem(Minimize(min_elemwise(a, 0, b)[1,2]))
result <- solve(prob)
result$value
```

min_entries *Minimum*

Description

The minimum of an expression.

Usage

```
min_entries(x, axis = NA_real_)

## S3 method for class 'Expression'
min(..., na.rm = FALSE)
```

Arguments

x	An Expression, vector, or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.
...	Numeric scalar, vector, matrix, or Expression objects.
na.rm	(Unimplemented) A logical value indicating whether missing values should be removed.

Value

An Expression representing the minimum of the input.

Examples

```
A <- Variable(2,2)
val <- cbind(c(-5,2), c(-3,1))
prob <- Problem(Maximize(min_entries(A)), list(A == val))
result <- solve(prob)
result$value
```

mixed_norm *Mixed Norm*

Description

$l_{p,q}(x) = \left(\sum_{i=1}^{n}(\sum_{j=1}^{m}|x_{i,j}|)^{q/p}\right)^{1/q}$.

Usage

```
mixed_norm(X, p = 2, q = 1)
```

Arguments

X	An Expression, vector, or matrix.
p	The type of inner norm.
q	The type of outer norm.

Value

An Expression representing the $l_{p,q}$ norm of the input.

Examples

```
A <- Variable(2,2)
val <- cbind(c(3,3), c(4,4))
prob <- Problem(Minimize(mixed_norm(A,2,1)), list(A == val))
result <- solve(prob)
result$value
result$getValue(A)

val <- cbind(c(1,4), c(5,6))
prob <- Problem(Minimize(mixed_norm(A,1,Inf)), list(A == val))
result <- solve(prob)
result$value
result$getValue(A)
```

MOSEK-class *The MOSEK class.*

Description

This class is an interface for the commercial MOSEK solver.

Usage

```
MOSEK()

## S4 method for signature 'MOSEK'
lp_capable(solver)

## S4 method for signature 'MOSEK'
socp_capable(solver)

## S4 method for signature 'MOSEK'
sdp_capable(solver)

## S4 method for signature 'MOSEK'
exp_capable(solver)
```

```
## S4 method for signature 'MOSEK'
mip_capable(solver)

## S4 method for signature 'MOSEK'
name(object)

## S4 method for signature 'MOSEK'
import_solver(solver)

## S4 method for signature 'MOSEK'
Solver.solve(solver, objective, constraints, cached_data,
  warm_start, verbose, ...)
```

Arguments

`object, solver`	A MOSEK object.
`objective`	A list representing the canonicalized objective.
`constraints`	A list of canonicalized constraints.
`cached_data`	A list mapping solver name to cached problem data.
`warm_start`	A logical value indicating whether the previous solver result should be used to warm start.
`verbose`	A logical value indicating whether to print solver output.
`...`	Additional arguments to the solver.

Methods (by generic)

- `lp_capable`: MOSEK can handle linear programs.
- `socp_capable`: MOSEK can handle second-order cone programs.
- `sdp_capable`: MOSEK can handle semidefinite programs.
- `exp_capable`: MOSEK cannot handle exponential cone programs.
- `mip_capable`: MOSEK cannot handle mixed-integer programs.
- `name`: The name of the solver.
- `import_solver`: Imports the reticulate library to use the python solver.
- `Solver.solve`: Call the solver on the canonicalized problem.

References

E. Andersen and K. Andersen. "The MOSEK Interior Point Optimizer for Linear Programming: an Implementation of the Homogeneous Algorithm." *High Performance Optimization*, vol. 33, pp. 197-232, 2000.

See Also

the MOSEK Official Site.

MulElemwise-class *The MulElemwise class.*

Description

This class represents the elementwise multiplication of two expressions. The first expression must be constant.

Usage

```
MulElemwise(lh_const, rh_exp)

## S4 method for signature 'MulElemwise'
validate_args(object)

## S4 method for signature 'MulElemwise'
to_numeric(object, values)

## S4 method for signature 'MulElemwise'
size_from_args(object)

## S4 method for signature 'MulElemwise'
sign_from_args(object)

## S4 method for signature 'MulElemwise'
is_incr(object, idx)

## S4 method for signature 'MulElemwise'
is_decr(object, idx)

## S4 method for signature 'MulElemwise'
is_quadratic(object)

## S4 method for signature 'MulElemwise'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

lh_const	A constant Expression or numeric value.
rh_exp	An Expression.
object	A MulElemwise object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.

`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Methods (by generic)

- `validate_args`: Check the first argument is a constant.
- `to_numeric`: Multiply the values elementwise.
- `size_from_args`: The size of the atom.
- `sign_from_args`: The sign of the atom.
- `is_incr`: Is the left-hand constant positive?
- `is_decr`: Is the left-hand constant negative?
- `is_quadratic`: Is the right-hand expression quadratic?
- `graph_implementation`: The graph implementation of the atom.

Slots

`lh_const` A constant Expression or numeric value.

`rh_exp` An Expression.

`name` *Variable, Parameter, or Expression Name*

Description

The string representation of a variable, parameter, or expression.

Usage

```
name(object)
```

Arguments

`object`	A Variable, Parameter, or Expression object.

Value

For Variable or Parameter objects, the value in the name slot. For Expression objects, a string indicating the nested atoms and their respective arguments.

Examples

```
x <- Variable()
y <- Variable(3, name = "yVar")

name(x)
name(y)
```

neg *Elementwise Negative*

Description

The elementwise absolute negative portion of an expression, $-\min(x_i, 0)$. This is equivalent to `-min_elemwise(x,0)`.

Usage

```
neg(x)
```

Arguments

x	An Expression, vector, or matrix.

Value

An Expression representing the negative portion of the input.

Examples

```
x <- Variable(2)
val <- matrix(c(-3,3))
prob <- Problem(Minimize(neg(x)[1]), list(x == val))
result <- solve(prob)
result$value
```

NonlinearConstraint-class *The NonlinearConstraint class.*

Description

This class represents a nonlinear inequality constraint, $f(x) \leq 0$ where f is twice-differentiable.

Usage

```
NonlinearConstraint(f, vars_)

## S4 method for signature 'NonlinearConstraint'
variables(object)
```

Arguments

f	A nonlinear function.
vars_	A list of variables involved in the function.
object	A NonlinearConstraint object.

Methods (by generic)

- variables: The variables involved in the function in order, i.e. f(vars_) = f(vstack(variables)).

Slots

constr_id (Internal) A unique integer identification number used internally.

f A nonlinear function.

vars_ A list of variables involved in the function.

.x_size (Internal) The dimensions of a column vector with number of elements equal to the total elements in all the variables.

NonNegative-class *The NonNegative class.*

Description

This class represents a variable constrained to be non-negative.

Usage

```
NonNegative(rows = 1, cols = 1, name = NA_character_)

## S4 method for signature 'NonNegative'
as.character(x)

## S4 method for signature 'NonNegative'
canonicalize(object)

## S4 method for signature 'NonNegative'
is_positive(object)

## S4 method for signature 'NonNegative'
is_negative(object)
```

Arguments

rows	The number of rows in the variable.
cols	The number of columns in the variable.
name	(Optional) A character string representing the name of the variable.
x, object	A NonNegative object.

Methods (by generic)

- `canonicalize`: Enforce that the variable be non-negative.
- `is_positive`: Always true since the variable is non-negative.
- `is_negative`: Always false since the variable is non-negative.

Slots

`id` (Internal) A unique identification number used internally.

`rows` The number of rows in the variable.

`cols` The number of columns in the variable.

`name` (Optional) A character string representing the name of the variable.

`primal_value` (Internal) The primal value of the variable stored internally.

Examples

```
x <- NonNegative(3, 3)
as.character(x)
canonicalize(x)
is_positive(x)
is_negative(x)
```

`norm,Expression,character-method` *Matrix Norm*

Description

The matrix norm, which can be the 1-norm ("1"), infinity-norm ("I"), Frobenius norm ("F"), maximum modulus of all the entries ("M"), or the spectral norm ("2"), as determined by the value of type.

Usage

```
## S4 method for signature 'Expression,character'
norm(x, type)
```

Arguments

`x`	An Expression.
`type`	A character indicating the type of norm desired. • "O", "o" or "1" specifies the 1-norm (maximum absolute column sum). • "I" or "i" specifies the infinity-norm (maximum absolute row sum). • "F" or "f" specifies the Frobenius norm (Euclidean norm of the vectorized `x`). • "M" or "m" specifies the maximum modulus of all the elements in `x`. • "2" specifies the spectral norm, which is the largest singular value of `x`.

Value

An Expression representing the norm of the input.

See Also

The p_norm function calculates the vector p-norm.

Examples

```
C <- Variable(3,2)
val <- Constant(rbind(c(1,2), c(3,4), c(5,6)))
prob <- Problem(Minimize(norm(C, "F")), list(C == val))
result <- solve(prob, solver = "SCS")
result$value
```

---

norm1 *1-Norm*

Description

$\|x\|_1 = \sum_{i=1}^{n} |x_i|$.

Usage

```
norm1(x, axis = NA_real_)
```

Arguments

x	An Expression, vector, or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.

Value

An Expression representing the 1-norm of the input.

Examples

```
a <- Variable()
prob <- Problem(Minimize(norm1(a)), list(a <= -2))
result <- solve(prob)
result$value
result$getValue(a)

prob <- Problem(Maximize(-norm1(a)), list(a <= -2))
result <- solve(prob)
result$value
result$getValue(a)
```

```
x <- Variable(2)
z <- Variable(2)
prob <- Problem(Minimize(norm1(x - z) + 5), list(x >= c(2,3), z <= c(-1,-4)))
result <- solve(prob)
result$value
result$getValue(x[1] - z[1])
```

`norm2` *Euclidean Norm*

Description

$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$.

Usage

```
norm2(x, axis = NA_real_)
```

Arguments

`x`	An Expression, vector, or matrix.
`axis`	(Optional) The dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `NA`.

Value

An Expression representing the Euclidean norm of the input.

Examples

```
a <- Variable()
prob <- Problem(Minimize(norm2(a)), list(a <= -2))
result <- solve(prob)
result$value
result$getValue(a)

prob <- Problem(Maximize(-norm2(a)), list(a <= -2))
result <- solve(prob)
result$value
result$getValue(a)

x <- Variable(2)
z <- Variable(2)
prob <- Problem(Minimize(norm2(x - z) + 5), list(x >= c(2,3), z <= c(-1,-4)))
result <- solve(prob)
result$value
result$getValue(x)
result$getValue(z)
```

```
prob <- Problem(Minimize(norm2(t(x - z)) + 5), list(x >= c(2,3), z <= c(-1,-4)))
result <- solve(prob)
result$value
result$getValue(x)
result$getValue(z)
```

NormNuc-class *The NormNuc class.*

Description

The nuclear norm, i.e. sum of the singular values of a matrix.

Usage

```
NormNuc(A)

## S4 method for signature 'NormNuc'
to_numeric(object, values)

## S4 method for signature 'NormNuc'
size_from_args(object)

## S4 method for signature 'NormNuc'
sign_from_args(object)

## S4 method for signature 'NormNuc'
is_atom_convex(object)

## S4 method for signature 'NormNuc'
is_atom_concave(object)

## S4 method for signature 'NormNuc'
is_incr(object, idx)

## S4 method for signature 'NormNuc'
is_decr(object, idx)

## S4 method for signature 'NormNuc'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

A	An Expression representing a matrix.
object	A NormNuc object.
values	A list of arguments to the atom.

idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The nuclear norm (i.e., the sum of the singular values) of A.
- size_from_args: The atom is a scalar.
- sign_from_args: The atom is positive.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is not concave.
- is_incr: The atom is not monotonic in any argument.
- is_decr: The atom is not monotonic in any argument.
- graph_implementation: The graph implementation of the atom.

Slots

A An Expression representing a matrix.

---

norm_inf *Infinity-Norm*

Description

$\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$.

Usage

```
norm_inf(x, axis = NA_real_)
```

Arguments

x	An Expression, vector, or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.

Value

An Expression representing the infinity-norm of the input.

Examples

```
a <- Variable()
b <- Variable()
c <- Variable()

prob <- Problem(Minimize(norm_inf(a)), list(a >= 2))
result <- solve(prob)
result$value
result$getValue(a)

prob <- Problem(Minimize(3*norm_inf(a + 2*b) + c), list(a >= 2, b <= -1, c == 3))
result <- solve(prob)
result$value
result$getValue(a + 2*b)
result$getValue(c)

prob <- Problem(Maximize(-norm_inf(a)), list(a <= -2))
result <- solve(prob)
result$value
result$getValue(a)

x <- Variable(2)
z <- Variable(2)
prob <- Problem(Minimize(norm_inf(x - z) + 5), list(x >= c(2,3), z <= c(-1,-4)))
result <- solve(prob)
result$value
result$getValue(x[1] - z[1])
```

---

norm_nuc *Nuclear Norm*

Description

The nuclear norm, i.e. sum of the singular values of a matrix.

Usage

```
norm_nuc(A)
```

Arguments

A	An Expression or matrix.

Value

An Expression representing the nuclear norm of the input.

Examples

```
C <- Variable(3,3)
val <- cbind(3:5, 6:8, 9:11)
prob <- Problem(Minimize(norm_nuc(C)), list(C == val))
result <- solve(prob)
result$value
```

`Objective-arith` *Arithmetic Operations on Objectives*

Description

Add, subtract, multiply, or divide optimization objectives.

Usage

```
## S4 method for signature 'Minimize,missing'
e1 - e2

## S4 method for signature 'Minimize,Minimize'
e1 + e2

## S4 method for signature 'Minimize,Maximize'
e1 + e2

## S4 method for signature 'Minimize,numeric'
e1 + e2

## S4 method for signature 'numeric,Minimize'
e1 + e2

## S4 method for signature 'Minimize,Minimize'
e1 - e2

## S4 method for signature 'Minimize,Maximize'
e1 - e2

## S4 method for signature 'Minimize,numeric'
e1 - e2

## S4 method for signature 'numeric,Minimize'
e1 - e2

## S4 method for signature 'Minimize,numeric'
e1 * e2

## S4 method for signature 'Maximize,numeric'
```

```
e1 * e2

## S4 method for signature 'numeric,Minimize'
e1 * e2

## S4 method for signature 'Minimize,numeric'
e1 / e2

## S4 method for signature 'Maximize,missing'
e1 - e2

## S4 method for signature 'Maximize,Maximize'
e1 + e2

## S4 method for signature 'Maximize,Minimize'
e1 + e2
```

Arguments

e1	The left-hand Minimize, Maximize, or numeric value.
e2	The right-hand Minimize, Maximize, or numeric value.

Value

A Minimize or Maximize object.

Parameter-class *The Parameter class.*

Description

This class represents a parameter, either scalar or a matrix.

Usage

```
Parameter(rows = 1, cols = 1, name = NA_character_, sign = UNKNOWN,
  value = NA_real_)

## S4 method for signature 'Parameter'
as.character(x)

## S4 method for signature 'Parameter'
get_data(object)

## S4 method for signature 'Parameter'
name(object)
```

```
## S4 method for signature 'Parameter'
size(object)

## S4 method for signature 'Parameter'
is_positive(object)

## S4 method for signature 'Parameter'
is_negative(object)

## S4 method for signature 'Parameter'
grad(object)

## S4 method for signature 'Parameter'
parameters(object)

## S4 method for signature 'Parameter'
value(object)

## S4 replacement method for signature 'Parameter'
value(object) <- value

## S4 method for signature 'Parameter'
canonicalize(object)
```

Arguments

rows	The number of rows in the parameter.
cols	The number of columns in the parameter.
name	(Optional) A character string representing the name of the parameter.
sign	(Optional) A character string indicating the sign of the parameter. Must be "ZERO", "POSITIVE", "NEGATIVE", or "UNKNOWN". Defaults to "UNKNOWN".
value	(Optional) A numeric element, vector, matrix, or data.frame. Defaults to NA and may be changed with value<- later.
x, object	A Parameter object.

Methods (by generic)

- get_data: Returns list(rows, cols, name, sign string, value).
- name: The name of the parameter.
- size: The c(rows, cols) dimensions of the parameter.
- is_positive: Is the parameter non-negative?
- is_negative: Is the parameter non-positive?
- grad: An empty list since the gradient of a parameter is zero.
- parameters: Returns itself as a parameter.

- `value`: The value of the parameter.
- `value<-`: Set the value of the parameter.
- `canonicalize`: The canonical form of the parameter.

Slots

`id` (Internal) A unique integer identification number used internally.

`rows` The number of rows in the parameter.

`cols` The number of columns in the parameter.

`name` (Optional) A character string representing the name of the parameter.

`sign_str` A character string indicating the sign of the parameter. Must be "ZERO", "POSITIVE", "NEGATIVE", or "UNKNOWN".

`value` (Optional) A numeric element, vector, matrix, or data.frame. Defaults to `NA` and may be changed with `value<-` later.

Examples

```
x <- Parameter(3, name = "x0", sign="NEGATIVE") ## 3-vec negative
is_positive(x)
is_negative(x)
size(x)
```

`Pnorm-class` *The Pnorm class.*

Description

This class represents the vector p-norm.

Usage

```
Pnorm(x, p = 2, axis = NA_real_, max_denom = 1024)

## S4 method for signature 'Pnorm'
validate_args(object)

## S4 method for signature 'Pnorm'
name(object)

## S4 method for signature 'Pnorm'
to_numeric(object, values)

## S4 method for signature 'Pnorm'
sign_from_args(object)

## S4 method for signature 'Pnorm'
```

```
is_atom_convex(object)

## S4 method for signature 'Pnorm'
is_atom_concave(object)

## S4 method for signature 'Pnorm'
is_incr(object, idx)

## S4 method for signature 'Pnorm'
is_decr(object, idx)

## S4 method for signature 'Pnorm'
is_pwl(object)

## S4 method for signature 'Pnorm'
get_data(object)

## S4 method for signature 'Pnorm'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression representing a vector or matrix.
p	A number greater than or equal to 1, or equal to positive infinity.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.
max_denom	The maximum denominator considered in forming a rational approximation for p.
object	A Pnorm object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Details

If given a matrix variable, Pnorm will treat it as a vector and compute the p-norm of the concatenated columns.

For $p \geq 1$, the p-norm is given by

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

with domain $x \in \mathbf{R}^n$. For $p < 1, p \neq 0$, the p-norm is given by

$$\|x\|_p = \left(\sum_{i=1}^{n} x_i^p\right)^{1/p}$$

with domain $x \in \mathbf{R}_+^n$.

- Note that the "p-norm" is actually a **norm** only when $p \geq 1$ or $p = +\infty$. For these cases, it is convex.
- The expression is undefined when $p = 0$.
- Otherwise, when $p < 1$, the expression is concave, but not a true norm.

Methods (by generic)

- `validate_args`: Check that the arguments are valid.
- `name`: The name and arguments of the atom.
- `to_numeric`: The p-norm of `x`.
- `sign_from_args`: The atom is positive.
- `is_atom_convex`: The atom is convex if $p \geq 1$.
- `is_atom_concave`: The atom is concave if $p < 1$.
- `is_incr`: The atom is weakly increasing if $p < 1$ or $p \geq 1$ and `x` is positive.
- `is_decr`: The atom is weakly decreasing if $p \geq 1$ and `x` is negative.
- `is_pwl`: The atom is piecewise linear only if `x` is piecewise linear, and either $p = 1$ or $p = \infty$.
- `get_data`: Returns `list(p, axis)`.
- `graph_implementation`: The graph implementation of the atom.

Slots

`x` An Expression representing a vector or matrix.

`p` A number greater than or equal to 1, or equal to positive infinity.

`max_denom` The maximum denominator considered in forming a rational approximation for p.

`axis` (Optional) The dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `NA`.

`.approx_error` (Internal) The absolute difference between p and its rational approximation.

pos *Elementwise Positive*

Description

The elementwise positive portion of an expression, $\max(x_i, 0)$. This is equivalent to `max_elemwise(x,0)`.

Usage

```
pos(x)
```

Arguments

x	An Expression, vector, or matrix.

Value

An Expression representing the positive portion of the input.

Examples

```
x <- Variable(2)
val <- matrix(c(-3,2))
prob <- Problem(Minimize(pos(x)[1]), list(x == val))
result <- solve(prob)
result$value
```

Power-class *The Power class.*

Description

This class represents the elementwise power function $f(x) = x^p$. If `expr` is a CVXR expression, then `expr^p` is equivalent to `Power(expr, p)`.

Usage

```
Power(x, p, max_denom = 1024)

## S4 method for signature 'Power'
validate_args(object)

## S4 method for signature 'Power'
get_data(object)

## S4 method for signature 'Power'
```

```
to_numeric(object, values)

## S4 method for signature 'Power'
sign_from_args(object)

## S4 method for signature 'Power'
is_atom_convex(object)

## S4 method for signature 'Power'
is_atom_concave(object)

## S4 method for signature 'Power'
is_constant(object)

## S4 method for signature 'Power'
is_incr(object, idx)

## S4 method for signature 'Power'
is_decr(object, idx)

## S4 method for signature 'Power'
is_quadratic(object)

## S4 method for signature 'Power'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	The Expression to be raised to a power.
p	A numeric value indicating the scalar power.
max_denom	The maximum denominator considered in forming a rational approximation of p.
object	A Power object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Details

#' For $p = 0$, $f(x) = 1$, constant, positive. For $p = 1$, $f(x) = x$, affine, increasing, same sign as x. For $p = 2, 4, 8, ...$, $f(x) = |x|^p$, convex, signed monotonicity, positive. For $p < 0$ and $f(x) =$

- x^p for $x > 0$
- $+\infty x \leq 0$

, this function is convex, decreasing, and positive. For $0 < p < 1$ and $f(x) =$

- x^p for $x \geq 0$
- $-\infty x < 0$

, this function is concave, increasing, and positive. For $p > 1, p \neq 2, 4, 8, \ldots$ and $f(x) =$

- x^p for $x \geq 0$
- $+\infty x < 0$

, this function is convex, increasing, and positive.

Methods (by generic)

- `validate_args`: Verification of arguments happens during initialization.
- `get_data`: A list containing the output of `pow_low, pow_mid`, or `pow_high` depending on the input power.
- `to_numeric`: Throw an error if the power is negative and cannot be handled.
- `sign_from_args`: The sign of the atom.
- `is_atom_convex`: Is $p \leq 0$ or $p \geq 1$?
- `is_atom_concave`: Is $p \geq 0$ or $p \leq 1$?
- `is_constant`: A logical value indicating whether the atom is constant.
- `is_incr`: A logical value indicating whether the atom is weakly increasing.
- `is_decr`: A logical value indicating whether the atom is weakly decreasing.
- `is_quadratic`: A logical value indicating whether the atom is quadratic.
- `graph_implementation`: The graph implementation of the atom.

Slots

`x` The Expression to be raised to a power.

`p` A numeric value indicating the scalar power.

`max_denom` The maximum denominator considered in forming a rational approximation of `p`.

`Problem-arith` *Arithmetic Operations on Problems*

Description

Add, subtract, multiply, or divide DCP optimization problems.

Usage

```
## S4 method for signature 'Problem,missing'
e1 + e2

## S4 method for signature 'Problem,missing'
e1 - e2

## S4 method for signature 'Problem,numeric'
e1 + e2

## S4 method for signature 'numeric,Problem'
e1 + e2

## S4 method for signature 'Problem,Problem'
e1 + e2

## S4 method for signature 'Problem,numeric'
e1 - e2

## S4 method for signature 'numeric,Problem'
e1 - e2

## S4 method for signature 'Problem,Problem'
e1 - e2

## S4 method for signature 'Problem,numeric'
e1 * e2

## S4 method for signature 'numeric,Problem'
e1 * e2

## S4 method for signature 'Problem,numeric'
e1 / e2
```

Arguments

e1	The left-hand Problem object.
e2	The right-hand Problem object.

Value

A Problem object.

Problem-class *The Problem class.*

Description

This class represents a convex optimization problem.

Usage

```
Problem(objective, constraints = list())

## S4 method for signature 'Problem'
objective(object)

## S4 replacement method for signature 'Problem'
objective(object) <- value

## S4 method for signature 'Problem'
constraints(object)

## S4 replacement method for signature 'Problem'
constraints(object) <- value

## S4 method for signature 'Problem'
value(object)

## S4 replacement method for signature 'Problem'
value(object) <- value

## S4 method for signature 'Problem'
is_dcp(object)

## S4 method for signature 'Problem'
is_qp(object)

## S4 method for signature 'Problem'
canonicalize(object)

## S4 method for signature 'Problem'
variables(object)

## S4 method for signature 'Problem'
parameters(object)

## S4 method for signature 'Problem'
constants(object)
```

```
## S4 method for signature 'Problem'
size_metrics(object)

## S4 method for signature 'Problem,character'
get_problem_data(object, solver)

## S4 method for signature 'Problem'
unpack_results(object, solver, results_dict)
```

Arguments

`objective`	A Minimize or Maximize object representing the optimization objective.
`constraints`	(Optional) A list of Constraint objects representing constraints on the optimization variables.
`object`	A Problem object.
`value`	A Minimize or Maximize object (objective), list of Constraint objects (constraints), or numeric scalar (value).
`solver`	A string indicating the solver that the problem data is for. Call `installed_solvers()` to see all available.
`results_dict`	A list containing the solver output.

Methods (by generic)

- `objective`: The objective of the problem.
- `objective<-`: Set the value of the problem objective.
- `constraints`: A list of the constraints of the problem.
- `constraints<-`: Set the value of the problem constraints.
- `value`: The value from the last time the problem was solved.
- `value<-`: Set the value of the optimal objective.
- `is_dcp`: A logical value indicating whether the problem statisfies DCP rules.
- `is_qp`: A logical value indicating whether the problem is a quadratic program.
- `canonicalize`: The graph implementation of the problem.
- `variables`: List of Variable objects in the problem.
- `parameters`: List of Parameter objects in the problem.
- `constants`: List of Constant objects in the problem.
- `size_metrics`: Information about the size of the problem.
- `get_problem_data`: Get the problem data passed to the specified solver.
- `unpack_results`: Parses the output from a solver and updates the problem state, including the status, objective value, and values of the primal and dual variables. Assumes the results are from the given solver.

Slots

`objective` A Minimize or Maximize object representing the optimization objective.

`constraints` (Optional) A list of constraints on the optimization variables.

`value` (Internal) Used internally to hold the value of the optimization objective at the solution.

`status` (Internal) Used internally to hold the status of the problem solution.

`.cached_data` (Internal) Used internally to hold cached matrix data.

`.separable_problems` (Internal) Used internally to hold separable problem data.

`.size_metrics` (Internal) Used internally to hold size metrics.

`.solver_stats` (Internal) Used internally to hold solver statistics.

Examples

```
x <- Variable(2)
p <- Problem(Minimize(p_norm(x, 2)), list(x >= 0))
is_dcp(p)
x <- Variable(2)
A <- matrix(c(1,-1,-1, 1), nrow = 2)
p <- Problem(Minimize(quad_form(x, A)), list(x >= 0))
is_qp(p)
```

---

problem-parts *Parts of a Problem*

Description

Get and set the objective, constraints, or size metrics (get only) of a problem.

Usage

```
objective(object)

objective(object) <- value

constraints(object)

constraints(object) <- value

size_metrics(object)
```

Arguments

`object`	A Problem object.
`value`	The value to assign to the slot.

Value

For getter functions, the requested slot of the object. x <- Variable() prob <- Problem(Minimize(x^2), list(x >= 5)) objective(prob) constraints(prob) size_metrics(prob)

objective(prob) <- Maximize(sqrt(x)) constraints(prob) <- list(x <= 10) objective(prob) constraints(prob)

`psolve` *Solve a DCP Problem*

Description

Solve a DCP compliant optimization problem.

Usage

```
psolve(object, solver, ignore_dcp = FALSE, warm_start = FALSE,
  verbose = FALSE, parallel = FALSE, ...)

## S4 method for signature 'Problem'
psolve(object, solver, ignore_dcp = FALSE,
  warm_start = FALSE, verbose = FALSE, parallel = FALSE, ...)

## S3 method for class 'Problem'
solve(a, b, ...)
```

Arguments

`object, a`	A Problem object.
`solver, b`	(Optional) A string indicating the solver to use. Defaults to "ECOS".
`ignore_dcp`	(Optional) A logical value indicating whether to override the DCP check for a problem.
`warm_start`	(Optional) A logical value indicating whether the previous solver result should be used to warm start.
`verbose`	(Optional) A logical value indicating whether to print additional solver output.
`parallel`	(Optional) A logical value indicating whether to solve in parallel if the problem is separable.
`...`	Additional options that will be passed to the specific solver. In general, these options will override any default settings imposed by CVXR.

Value

A list containing the solution to the problem:

`status` The status of the solution. Can be "optimal", "optimal_inaccurate", "infeasible", "infeasible_inaccurate", "unbounded", "unbounded_inaccurate", or "solver_error".

`value` The optimal value of the objective function.

`solver` The name of the solver.

`solve_time` The time (in seconds) it took for the solver to solve the problem.

`setup_time` The time (in seconds) it took for the solver to set up the problem.

`num_iters` The number of iterations the solver had to go through to find a solution.

`getValue` A function that takes a Variable object and retrieves its primal value.

`getDualValue` A function that takes a Constraint object and retrieves its dual value(s).

Examples

```
a <- Variable(name = "a")
prob <- Problem(Minimize(norm_inf(a)), list(a >= 2))
result <- psolve(prob, solver = "ECOS", verbose = TRUE)
result$status
result$value
result$getValue(a)
result$getDualValue(constraints(prob)[[1]])
```

`p_norm` *P-Norm*

Description

The vector p-norm. If given a matrix variable, `p_norm` will treat it as a vector and compute the p-norm of the concatenated columns.

Usage

```
p_norm(x, p = 2, axis = NA_real_, max_denom = 1024)
```

Arguments

`x`	An Expression, vector, or matrix.
`p`	A number greater than or equal to 1, or equal to positive infinity.
`axis`	(Optional) The dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `NA`.
`max_denom`	The maximum denominator considered in forming a rational approximation for p.

Details

For $p \geq 1$, the p-norm is given by

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

with domain $x \in \mathbf{R}^n$. For $p < 1, p \neq 0$, the p-norm is given by

$$\|x\|_p = \left(\sum_{i=1}^{n} x_i^p \right)^{1/p}$$

with domain $x \in \mathbf{R}_+^n$.

- Note that the "p-norm" is actually a **norm** only when $p \geq 1$ or $p = +\infty$. For these cases, it is convex.
- The expression is undefined when $p = 0$.
- Otherwise, when $p < 1$, the expression is concave, but not a true norm.

Value

An Expression representing the p-norm of the input.

Examples

```
x <- Variable(3)
prob <- Problem(Minimize(p_norm(x,2)))
result <- solve(prob)
result$value
result$getValue(x)

prob <- Problem(Minimize(p_norm(x,Inf)))
result <- solve(prob)
result$value
result$getValue(x)

a <- c(1.0, 2, 3)
prob <- Problem(Minimize(p_norm(x,1.6)), list(t(x) %*% a >= 1))
result <- solve(prob)
result$value
result$getValue(x)

prob <- Problem(Minimize(sum(abs(x - a))), list(p_norm(x,-1) >= 0))
result <- solve(prob)
result$value
result$getValue(x)
```

QuadOverLin-class *The QuadOverLin class.*

Description

This class represents the sum of squared entries in X divided by a scalar y, $\sum_{i,j} X_{i,j}^2 / y$.

Usage

```
QuadOverLin(x, y)

## S4 method for signature 'QuadOverLin'
validate_args(object)

## S4 method for signature 'QuadOverLin'
to_numeric(object, values)

## S4 method for signature 'QuadOverLin'
size_from_args(object)

## S4 method for signature 'QuadOverLin'
sign_from_args(object)

## S4 method for signature 'QuadOverLin'
is_atom_convex(object)

## S4 method for signature 'QuadOverLin'
is_atom_concave(object)

## S4 method for signature 'QuadOverLin'
is_incr(object, idx)

## S4 method for signature 'QuadOverLin'
is_decr(object, idx)

## S4 method for signature 'QuadOverLin'
is_quadratic(object)

## S4 method for signature 'QuadOverLin'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric matrix.
y	A scalar Expression or numeric constant.
object	A QuadOverLin object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- `validate_args`: Check the dimensions of the arguments.
- `to_numeric`: The sum of the entries of `x` squared over `y`.
- `size_from_args`: The atom is a scalar.
- `sign_from_args`: The atom is positive.
- `is_atom_convex`: The atom is convex.
- `is_atom_concave`: The atom is not concave.
- `is_incr`: A logical value indicating whether the atom is weakly increasing.
- `is_decr`: A logical value indicating whether the atom is weakly decreasing.
- `is_quadratic`: True if `x` is affine and `y` is constant.
- `graph_implementation`: The graph implementation of the atom.

Slots

`x` An Expression or numeric matrix.

`y` A scalar Expression or numeric constant.

`quad_form` *Quadratic Form*

Description

The quadratic form, $x^T P x$.

Usage

```
quad_form(x, P)
```

Arguments

`x`	An Expression or vector.
`P`	An Expression or matrix.

Value

An Expression representing the quadratic form evaluated at the input.

Examples

```
x <- Variable(2)
P <- rbind(c(4,0), c(0,9))
prob <- Problem(Minimize(quad_form(x,P)), list(x >= 1))
result <- solve(prob)
result$value
result$getValue(x)

A <- Variable(2,2)
c <- c(1,2)
prob <- Problem(Minimize(quad_form(c,A)), list(A >= 1))
result <- solve(prob)
result$value
result$getValue(A)
```

---

quad_over_lin *Quadratic over Linear*

Description

$\sum_{i,j} X_{i,j}^2/y$.

Usage

```
quad_over_lin(x, y)
```

Arguments

x	An Expression, vector, or matrix.
y	A scalar Expression or numeric constant.

Value

An Expression representing the quadratic over linear function value evaluated at the input.

Examples

```
x <- Variable(3,2)
y <- Variable()
val <- cbind(c(-1,2,-2), c(-1,2,-2))
prob <- Problem(Minimize(quad_over_lin(x,y)), list(x == val, y <= 2))
result <- solve(prob)
result$value
result$getValue(x)
result$getValue(y)
```

Rdict-class *The Rdict class.*

Description

A simple, internal dictionary composed of a list of keys and a list of values. These keys/values can be any type, including nested lists, S4 objects, etc. Incredibly inefficient hack, but necessary for the geometric mean atom, since it requires mixed numeric/gmp objects.

Usage

```
Rdict(keys = list(), values = list())

## S4 method for signature 'Rdict'
x$name

## S4 method for signature 'Rdict'
length(x)

## S4 method for signature 'ANY,Rdict'
is.element(el, set)

## S4 method for signature 'Rdict,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'Rdict,ANY,ANY,ANY'
x[i, j, ...] <- value
```

Arguments

keys	A list of keys.
values	A list of values corresponding to the keys.
x, set	A Rdict object.
name	Either "keys" for a list of keys, "values" for a list of values, or "items" for a list of lists where each nested list is a (key, value) pair.
el	The element to search the dictionary of values for.
i	A key into the dictionary.
j, drop, ...	Unused arguments.
value	The value to assign to key i.

Slots

keys A list of keys.

values A list of values corresponding to the keys.

`Rdictdefault-class` *The Rdictdefault class.*

Description

This is a subclass of Rdict that contains an additional slot for a default function, which assigns a value to an input key. Only partially implemented, but working well enough for the geometric mean. Will be combined with Rdict later.

Usage

```
Rdictdefault(keys = list(), values = list(), default)

## S4 method for signature 'Rdictdefault,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

`keys`	A list of keys.
`values`	A list of values corresponding to the keys.
`default`	A function that takes as input a key and outputs a value to assign to that key.
`x`	A Rdictdefault object.
`i`	A key into the dictionary.
`j, drop, ...`	Unused arguments.

Slots

`keys` A list of keys.

`values` A list of values corresponding to the keys.

`default` A function that takes as input a key and outputs a value to assign to that key.

See Also

Rdict

resetOptions *Reset Options*

Description

Reset the global package variable `.CVXR.options`.

Usage

```
resetOptions()
```

Value

The default value of CVXR package global `.CVXR.options`.

Examples

```
## Not run:
  resetOptions()

## End(Not run)
```

Reshape-class *The Reshape class.*

Description

This class represents the reshaping of an expression. The operator vectorizes the expression, then unvectorizes it into the new shape. Entries are stored in column-major order.

Usage

```
Reshape(expr, rows, cols)

## S4 method for signature 'Reshape'
validate_args(object)

## S4 method for signature 'Reshape'
to_numeric(object, values)

## S4 method for signature 'Reshape'
size_from_args(object)

## S4 method for signature 'Reshape'
get_data(object)
```

```
## S4 method for signature 'Reshape'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

expr	An Expression or numeric matrix.
rows	The new number of rows.
cols	The new number of columns.
object	A Reshape object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check the new shape has the same number of entries as the old.
- to_numeric: Reshape the value into the specified dimensions.
- size_from_args: The c(rows, cols) of the new expression.
- get_data: Returns list(rows, cols).
- graph_implementation: The graph implementation of the atom.

Slots

expr An Expression or numeric matrix.

rows The new number of rows.

cols The new number of columns.

---

reshape_expr *Reshape an Expression*

Description

This function vectorizes an expression, then unvectorizes it into a new shape. Entries are stored in column-major order.

Usage

```
reshape_expr(expr, rows, cols)
```

Arguments

expr	An Expression, vector, or matrix.
rows	The new number of rows.
cols	The new number of columns.

Value

An Expression representing the reshaped input.

Examples

```
x <- Variable(4)
mat <- cbind(c(1,-1), c(2,-2))
vec <- matrix(1:4)
expr <- reshape_expr(x,2,2)
obj <- Minimize(sum(mat %*% expr))
prob <- Problem(obj, list(x == vec))
result <- solve(prob)
result$value

A <- Variable(2,2)
c <- 1:4
expr <- reshape_expr(A,4,1)
obj <- Minimize(t(expr) %*% c)
constraints <- list(A == cbind(c(-1,-2), c(3,4)))
prob <- Problem(obj, constraints)
result <- solve(prob)
result$value
result$getValue(expr)
result$getValue(reshape_expr(expr,2,2))

C <- Variable(3,2)
expr <- reshape_expr(C,2,3)
mat <- rbind(c(1,-1), c(2,-2))
C_mat <- rbind(c(1,4), c(2,5), c(3,6))
obj <- Minimize(sum(mat %*% expr))
prob <- Problem(obj, list(C == C_mat))
result <- solve(prob)
result$value
result$getValue(expr)

a <- Variable()
c <- cbind(c(1,-1), c(2,-2))
expr <- reshape_expr(c * a,1,4)
obj <- Minimize(expr %*% (1:4))
prob <- Problem(obj, list(a == 2))
result <- solve(prob)
result$value
result$getValue(expr)

expr <- reshape_expr(c * a,4,1)
```

```
obj <- Minimize(t(expr) %*% (1:4))
prob <- Problem(obj, list(a == 2))
result <- solve(prob)
result$value
result$getValue(expr)
```

`residual-methods` *Constraint Residual*

Description

The residual expression of a constraint, i.e. the amount by which it is violated, and the value of that violation. For instance, if our constraint is $g(x) \leq 0$, the residual is $max(g(x), 0)$ applied elementwise.

Usage

```
residual(object)

violation(object)
```

Arguments

`object`	A Constraint object.

Value

A Expression representing the residual, or the value of this expression.

`RMulExpression-class` *The RMulExpression class.*

Description

This class represents the matrix product of an expression with a constant on the right.

Usage

```
## S4 method for signature 'RMulExpression'
is_incr(object, idx)

## S4 method for signature 'RMulExpression'
is_decr(object, idx)

## S4 method for signature 'RMulExpression'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

`object`	A RMulExpression object.
`idx`	An index into the atom.
`arg_objs`	A list of linear expressions for each argument.
`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Methods (by generic)

- `is_incr`: Is the right-hand expression positive?
- `is_decr`: Is the right-hand expression negative?
- `graph_implementation`: The graph implementation of the expression.

See Also

MulExpression

`scalene` *Scalene Function*

Description

The elementwise weighted sum of the positive and negative portions of an expression, $\alpha \max(x_i, 0) - \beta \min(x_i, 0)$. This is equivalent to `alpha*pos(x) + beta*neg(x)`.

Usage

```
scalene(x, alpha, beta)
```

Arguments

`x`	An Expression, vector, or matrix.
`alpha`	The weight on the positive portion of `x`.
`beta`	The weight on othe negative portion of `x`.

Value

An Expression representing the scalene function evaluated at the input.

Examples

```
## Not run:
A <- Variable(2,2)
val <- cbind(c(-5,2), c(-3,1))
prob <- Problem(Minimize(scalene(A,2,3)[1,1]), list(A == val))
result <- solve(prob)
result$value
result$getValue(scalene(A, 0.7, 0.3))

## End(Not run)
```

SCS-class *The SCS class.*

Description

This class is an interface for the SCS solver.

Usage

```
SCS()

## S4 method for signature 'SCS'
lp_capable(solver)

## S4 method for signature 'SCS'
socp_capable(solver)

## S4 method for signature 'SCS'
sdp_capable(solver)

## S4 method for signature 'SCS'
exp_capable(solver)

## S4 method for signature 'SCS'
mip_capable(solver)

## S4 method for signature 'SCS'
name(object)

## S4 method for signature 'SCS'
import_solver(solver)

## S4 method for signature 'SCS'
Solver.solve(solver, objective, constraints, cached_data,
  warm_start, verbose, ...)
```

```
## S4 method for signature 'SCS'
format_results(solver, results_dict, data, cached_data)
```

Arguments

`object, solver`	A SCS object.
`objective`	A list representing the canonicalized objective.
`constraints`	A list of canonicalized constraints.
`cached_data`	A list mapping solver name to cached problem data.
`warm_start`	A logical value indicating whether the previous solver result should be used to warm start.
`verbose`	A logical value indicating whether to print solver output.
`...`	Additional arguments to the solver.
`results_dict`	A list containing the solver output.
`data`	A list containing information about the problem.

Methods (by generic)

- `lp_capable`: SCS can handle linear programs.
- `socp_capable`: SCS can handle second-order cone programs.
- `sdp_capable`: SCS can handle semidefinite programs.
- `exp_capable`: SCS can handle exponential cone programs.
- `mip_capable`: SCS cannot handle mixed-integer programs.
- `name`: The name of the solver.
- `import_solver`: Imports the scs library.
- `Solver.solve`: Call the solver on the canonicalized problem.
- `format_results`: Convert raw solver output into standard list of results.

References

B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding." *Journal of Optimization Theory and Applications*, pp. 1-27, 2016. https://doi.org/10.1007/s10957-016-0892-3.

See Also

`scs` and the SCS Github.

Examples

```
scs <- SCS()
lp_capable(scs)
sdp_capable(scs)
socp_capable(scs)
exp_capable(scs)
mip_capable(scs)
```

SDP-class *The SDP class.*

Description

This class represents a semidefinite cone constraint, the set of all symmetric matrices such that the quadratic form $x^T Ax$ is non-negative for all x.

$$\{\text{symmetric } A | x^T Ax \geq 0 \text{ for all } x\}$$

Usage

```
SDP(A, enforce_sym = TRUE, constr_id)

## S4 method for signature 'SDP'
as.character(x)

## S4 method for signature 'SDP'
size(object)

## S4 method for signature 'SDP'
format_constr(object, eq_constr, leq_constr, dims, solver)
```

Arguments

A	The matrix variable constrained to be semidefinite.
enforce_sym	A logical value indicating whether symmetry constraints should be added.
constr_id	(Internal) A unique integer identification number used internally.
x, object	A SDP object.
eq_constr	A list of the equality constraints in the canonical problem.
leq_constr	A list of the inequality constraints in the canonical problem.
dims	A list with the dimensions of the conic constraints.
solver	A string representing the solver to be called.

Methods (by generic)

- size: The dimensions of the semidefinite cone.
- format_constr: Format SDP constraints as inequalities for the solver.

Slots

constr_id (Internal) A unique integer identification number used internally.

A The matrix variable constrained to be semidefinite.

enforce_sym A logical value indicating whether symmetry constraints should be added.

Semidef *Positive Semidefinite Variable*

Description

An expression representing a positive semidefinite matrix.

Usage

```
Semidef(n, name = NA_character_)
```

Arguments

n	The number of rows/columns in the matrix.
name	(Optional) A character string representing the name of the variable.

Value

An Expression representing the positive semidefinite matrix.

Examples

```
x <- Semidef(5) ## 5 by 5 semidefinite matrix expression
```

SemidefUpperTri-class *The SemidefUpperTri class.*

Description

This class represents the upper triangular part of a positive semidefinite variable.

Usage

```
SemidefUpperTri(n, name = NA_character_)

## S4 method for signature 'SemidefUpperTri'
as.character(x)

## S4 method for signature 'SemidefUpperTri'
get_data(object)

## S4 method for signature 'SemidefUpperTri'
canonicalize(object)

## S4 method for signature 'SymmetricUpperTri'
canonicalize(object)
```

Arguments

n	The number of rows/columns in the matrix.
name	(Optional) A character string representing the name of the variable.
x, object	A SemidefUpperTri object.

Methods (by generic)

- get_data: Returns list(n, name).
- canonicalize: Enforce that the variable be positive semidefinite.
- canonicalize: Enforce that the variable be symmetric.

Slots

id (Internal) A unique identification number used internally.

n The number of rows/columns in the matrix.

rows The number of rows in the variable.

cols The number of columns in the variable.

name (Optional) A character string representing the name of the variable.

primal_value (Internal) The primal value of the variable stored internally.

Examples

```
x <- SemidefUpperTri(3)
as.character(x)
get_data(x)
canonicalize(x)
```

---

setIdCounter *Set ID Counter*

Description

Set the CVXR variable/constraint identification number counter.

Usage

```
setIdCounter(value = 0L)
```

Arguments

value	The value to assign as ID.

Value

the changed value of the package global .CVXR.options.

Examples

```
## Not run:
  setIdCounter(value = 0L)

## End(Not run)
```

SigmaMax-class *The SigmaMax class.*

Description

The maximum singular value of a matrix.

Usage

```
SigmaMax(A = A)

## S4 method for signature 'SigmaMax'
to_numeric(object, values)

## S4 method for signature 'SigmaMax'
size_from_args(object)

## S4 method for signature 'SigmaMax'
sign_from_args(object)

## S4 method for signature 'SigmaMax'
is_atom_convex(object)

## S4 method for signature 'SigmaMax'
is_atom_concave(object)

## S4 method for signature 'SigmaMax'
is_incr(object, idx)

## S4 method for signature 'SigmaMax'
is_decr(object, idx)

## S4 method for signature 'SigmaMax'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

A	An Expression or matrix.
object	A SigmaMax object.

values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The largest singular value of A.
- size_from_args: The atom is a scalar.
- sign_from_args: The atom is positive.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is concave.
- is_incr: The atom is not monotonic in any argument.
- is_decr: The atom is not monotonic in any argument.
- graph_implementation: The graph implementation of the atom.

Slots

A An Expression or numeric matrix.

---

sigma_max *Maximum Singular Value*

Description

The maximum singular value of a matrix.

Usage

```
sigma_max(A = A)
```

Arguments

A	An Expression or matrix.

Value

An Expression representing the maximum singular value.

Examples

```
C <- Variable(3,2)
val <- rbind(c(1,2), c(3,4), c(5,6))
obj <- sigma_max(C)
constr <- list(C == val)
prob <- Problem(Minimize(obj), constr)
result <- solve(prob, solver = "SCS")
result$value
result$getValue(C)
```

`sign,Expression-method` *Sign of Expression*

Description

The sign of an expression.

Usage

```
## S4 method for signature 'Expression'
sign(x)
```

Arguments

`x`	An Expression object.

Value

A string indicating the sign of the expression, either "ZERO", "POSITIVE", "NEGATIVE", or "UNKNOWN".

`sign-methods` *Sign Properties*

Description

Determine if an expression is positive, negative, or zero.

Usage

```
is_zero(object)

is_positive(object)

is_negative(object)
```

Arguments

`object`	An Expression object.

Value

A logical value.

Examples

```
pos <- Constant(1)
neg <- Constant(-1)
zero <- Constant(0)
unknown <- Variable()

is_zero(pos)
is_zero(-zero)
is_zero(unknown)
is_zero(pos + neg)

is_positive(pos + zero)
is_positive(pos * neg)
is_positive(pos - neg)
is_positive(unknown)

is_negative(-pos)
is_negative(pos + neg)
is_negative(neg * zero)
is_negative(neg - pos)
```

`sign_from_args` *Atom Sign*

Description

Determine the sign of an atom based on its arguments.

Usage

```
sign_from_args(object)

## S4 method for signature 'Atom'
sign_from_args(object)
```

Arguments

`object`	An Atom object.

Value

A logical vector `c(is positive, is negative)` indicating the sign of the atom.

size *Size of Expression*

Description

The size of an expression.

Usage

```
size(object)

## S4 method for signature 'ListORExpr'
size(object)
```

Arguments

object	An Expression object.

Value

A vector with two elements `c(row, col)` representing the dimensions of the expression.

Examples

```
x <- Variable()
y <- Variable(3)
z <- Variable(3,2)

size(x)
size(y)
size(z)
size(x + y)
size(z - x)
```

size-methods *Size Properties*

Description

Determine if an expression is a scalar, vector, or matrix.

Usage

```
is_scalar(object)

is_vector(object)

is_matrix(object)
```

Arguments

`object`	An Expression object.

Value

A logical value.

Examples

```
x <- Variable()
y <- Variable(3)
z <- Variable(3,2)

is_scalar(x)
is_scalar(y)
is_scalar(x + y)

is_vector(x)
is_vector(y)
is_vector(2*z)

is_matrix(x)
is_matrix(y)
is_matrix(z)
is_matrix(z - x)
```

SizeMetrics-class *The SizeMetrics class.*

Description

This class contains various metrics regarding the problem size.

Usage

```
SizeMetrics(problem)
```

Arguments

`problem`	A Problem object.

Slots

`num_scalar_variables` The number of scalar variables in the problem.

`num_scalar_data` The number of constants used across all matrices and vectors in the problem. Some constants are not apparent when the problem is constructed. For example, the `sum_squares` expression is a wrapper for a `quad_over_lin` expression with a constant `1` in the denominator.

`num_scalar_eq_constr` The number of scalar equality constraints in the problem.

`num_scalar_leq_constr` The number of scalar inequality constraints in the problem.

`max_data_dimension` The longest dimension of any data block constraint or parameter.

`max_big_small_squared` The maximum value of (big)(small)^2 over all data blocks of the problem, where (big) is the larger dimension and (small) is the smaller dimension for each data block.

`size_from_args` *Atom Size*

Description

Determine the size of an atom based on its arguments.

Usage

```
size_from_args(object)

## S4 method for signature 'Atom'
size_from_args(object)
```

Arguments

`object`	A Atom object.

Value

A numeric vector `c(row, col)` indicating the size of the atom.

`SOC-class` *The SOC class.*

Description

This class represents a second-order cone constraint, i.e. $\|x\|_2 \leq t$.

Usage

```
SOC(t, x_elems)

## S4 method for signature 'SOC'
as.character(x)

## S4 method for signature 'SOC'
format_constr(object, eq_constr, leq_constr, dims, solver)

## S4 method for signature 'SOC'
size(object)
```

Arguments

t	The scalar part of the second-order constraint.
x_elems	A list containing the elements of the vector part of the constraint.
x, object	A SOC object.
eq_constr	A list of the equality constraints in the canonical problem.
leq_constr	A list of the inequality constraints in the canonical problem.
dims	A list with the dimensions of the conic constraints.
solver	A string representing the solver to be called.

Methods (by generic)

- format_constr: Format SOC constraints as inequalities for the solver.
- size: The dimensions of the second-order cone.

Slots

constr_id (Internal) A unique integer identification number used internally.

t The scalar part of the second-order constraint.

x_elems A list containing the elements of the vector part of the constraint.

SOCAxis-class *The SOCAxis class.*

Description

This class represents a second-order cone constraint for each row/column. It Assumes t is a vector the same length as X's rows (columns) for axis == 1 (2).

Usage

```
SOCAxis(t, X, axis)

## S4 method for signature 'SOCAxis'
as.character(x)

## S4 method for signature 'SOCAxis'
format_constr(object, eq_constr, leq_constr, dims,
  solver)

## S4 method for signature 'SOCAxis'
num_cones(object)

## S4 method for signature 'SOCAxis'
cone_size(object)

## S4 method for signature 'SOCAxis'
size(object)
```

Arguments

t	The scalar part of the second-order constraint.
X	A matrix whose rows/columns are each a cone.
axis	The dimension across which to take the slice: 1 indicates rows, and 2 indicates columns.
x, object	A SOCAxis object.
eq_constr	A list of the equality constraints in the canonical problem.
leq_constr	A list of the inequality constraints in the canonical problem.
dims	A list with the dimensions of the conic constraints.
solver	A string representing the solver to be called.

Methods (by generic)

- format_constr: Format SOC constraints as inequalities for the solver.
- num_cones: The number of elementwise cones.
- cone_size: The dimensions of a single cone.
- size: The dimensions of the (elementwise) second-order cones.

Slots

constr_id (Internal) A unique integer identification number used internally.

t The scalar part of the second-order constraint.

x_elems A list containing X, a matrix whose rows/columns are each a cone.

axis The dimension across which to take the slice: 1 indicates rows, and 2 indicates columns.

Solver-capable *Solver Capabilities*

Description

Determine if a solver is capable of solving a linear program (LP), second-order cone program (SOCP), semidefinite program (SDP), exponential cone program (EXP), or mixed-integer program (MIP).

Usage

```
lp_capable(solver)

socp_capable(solver)

sdp_capable(solver)

exp_capable(solver)

mip_capable(solver)
```

Arguments

solver	A Solver object.

Value

A logical value.

Examples

```
lp_capable(ECOS())
socp_capable(ECOS())
sdp_capable(ECOS())
exp_capable(ECOS())
mip_capable(ECOS())
```

Solver-class *The Solver class.*

Description

This virtual class represents the generic interface for a solver.

Usage

```
## S4 method for signature 'Solver'
validate_solver(solver, constraints)

## S4 method for signature 'Solver'
nonlin_constr(solver)

## S4 method for signature 'Solver'
Solver.solve(solver, objective, constraints,
  cached_data, warm_start, verbose, ...)

## S4 method for signature 'Solver'
format_results(solver, results_dict, data, cached_data)
```

Arguments

solver	A Solver object.
constraints	A list of canonicalized constraints.
objective	A list representing the canonicalized objective.
cached_data	A list mapping solver name to cached problem data.
warm_start	A logical value indicating whether the previous solver result should be used to warm start.
verbose	A logical value indicating whether to print solver output.

`...`	Additional arguments to the solver.
`results_dict`	A list containing the solver output.
`data`	A list containing information about the problem.

Methods (by generic)

- `validate_solver`: Verify the solver can solve the problem.
- `nonlin_constr`: A logical value indicating whether nonlinear constraints are needed.
- `Solver.solve`: Call the solver on the canonicalized problem.
- `format_results`: Convert raw solver output into standard list of results.

`Solver.choose_solver` *Choose a Solver*

Description

Determines the appropriate solver.

Usage

```
Solver.choose_solver(constraints)
```

Arguments

`constraints`	A list of canonicalized constraints.

Value

A Solver object.

`Solver.solve` *Call to Solver*

Description

Returns the result of the call to the solver.

Usage

```
Solver.solve(solver, objective, constraints, cached_data, warm_start,
  verbose, ...)
```

Arguments

`solver`	A Solver object.
`objective`	A list representing the canonicalized objective.
`constraints`	A list of canonicalized constraints.
`cached_data`	A list mapping solver name to cached problem data.
`warm_start`	A logical value indicating whether the previous solver result should be used to warm start.
`verbose`	A logical value indicating whether to print solver output.
`...`	Additional arguments to the solver.

Value

A list containing the status, optimal value, primal variable, and dual variables for the equality and inequality constraints.

`SolverStats-class` *The SolverStats class.*

Description

This class contains the miscellaneous information that is returned by a solver after solving, but that is not captured directly by the Problem object.

Usage

```
SolverStats(results_dict = list(), solver_name = NA_character_)
```

Arguments

`results_dict`	A list containing the results returned by the solver.
`solver_name`	The name of the solver.

Value

A list containing

`solver_name` The name of the solver.

`solve_time` The time (in seconds) it took for the solver to solve the problem.

`setup_time` The time (in seconds) it took for the solver to set up the problem.

`num_iters` The number of iterations the solver had to go through to find a solution.

Slots

`solver_name` The name of the solver.

`solve_time` The time (in seconds) it took for the solver to solve the problem.

`setup_time` The time (in seconds) it took for the solver to set up the problem.

`num_iters` The number of iterations the solver had to go through to find a solution.

`sqrt,Expression-method` *Square Root*

Description

The elementwise square root.

Usage

```
## S4 method for signature 'Expression'
sqrt(x)
```

Arguments

`x`	An Expression.

Value

An Expression representing the square root of the input. A <- Variable(2,2) val <- cbind(c(2,4), c(16,1)) prob <- Problem(Maximize(sqrt(A)[1,2]), list(A == val)) result <- solve(prob) result$value

---

`Sqrt-class` *The Sqrt class.*

Description

This class represents the elementwise square root $\sqrt{x}$.

Usage

```
Sqrt(x)

## S4 method for signature 'Sqrt'
validate_args(object)

## S4 method for signature 'Sqrt'
to_numeric(object, values)

## S4 method for signature 'Sqrt'
get_data(object)

## S4 method for signature 'Sqrt'
sign_from_args(object)
```

```
## S4 method for signature 'Sqrt'
is_atom_convex(object)

## S4 method for signature 'Sqrt'
is_atom_concave(object)

## S4 method for signature 'Sqrt'
is_incr(object, idx)

## S4 method for signature 'Sqrt'
is_decr(object, idx)

## S4 method for signature 'Sqrt'
is_quadratic(object)

## S4 method for signature 'Sqrt'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression object.
object	A Sqrt object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Verification of arguments happens during initialization.
- to_numeric: The elementwise square root of the input value.
- get_data: A list containing the output of pow_mid.
- sign_from_args: The atom is positive.
- is_atom_convex: The atom is not convex.
- is_atom_concave: The atom is concave.
- is_incr: The atom is weakly increasing.
- is_decr: The atom is not weakly decreasing.
- is_quadratic: Is x constant?
- graph_implementation: The graph implementation of the atom.

Slots

x An Expression object.

square *Square Function*

Description

The elementwise square function. This is equivalent to `power(x,2)`.

Usage

```
square(x)
```

Arguments

x	An Expression, vector, or matrix.

Value

An Expression representing the square of the input.

Examples

```
m <- 30
n <- 20
A <- matrix(stats::rnorm(m*n), nrow = m, ncol = n)
b <- matrix(stats::rnorm(m), nrow = m, ncol = 1)

x <- Variable(n)
obj <- Minimize(sum(square(A %*% x - b)))
constr <- list(0 <= x, x <= 1)
prob <- Problem(obj, constr)
result <- solve(prob)
result$value
result$getValue(x)
```

Square-class *The Square class.*

Description

This class represents the elementwise square x^2.

Usage

```
Square(x)

## S4 method for signature 'Square'
validate_args(object)

## S4 method for signature 'Square'
to_numeric(object, values)

## S4 method for signature 'Square'
get_data(object)

## S4 method for signature 'Square'
sign_from_args(object)

## S4 method for signature 'Square'
is_atom_convex(object)

## S4 method for signature 'Square'
is_atom_concave(object)

## S4 method for signature 'Square'
is_incr(object, idx)

## S4 method for signature 'Square'
is_decr(object, idx)

## S4 method for signature 'Square'
is_quadratic(object)

## S4 method for signature 'Square'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression object.
object	A Square object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Verification of arguments happens during initialization.

- `to_numeric`: The elementwise square of the input value.
- `get_data`: A list containing the output of `pow_high`.
- `sign_from_args`: The atom is positive.
- `is_atom_convex`: The atom is convex.
- `is_atom_concave`: The atom is not concave.
- `is_incr`: A logical value indicating whether the atom is weakly increasing.
- `is_decr`: A logical value indicating whether the atom is weakly decreasing.
- `is_quadratic`: Is `x` affine?
- `graph_implementation`: The graph implementation of the atom.

Slots

`x` An Expression object.

`status_map,ECOS-method`

ECOS Status Map

Description

Map of ECOS status to CVXR status.

Usage

```
## S4 method for signature 'ECOS'
status_map(solver, status)
```

Arguments

`solver`	A ECOS object.
`status`	An exit code returned by ECOS: **ECOS_OPTIMAL (0)** Problem solved to optimality. **ECOS_PINF (1)** Found certificate of primal infeasibility. **ECOS_DINF (2)** Found certificate of dual infeasibility. **ECOS_INACC_OFFSET (10)** Offset exitflag at inaccurate results. **ECOS_MAXIT (-1)** Maximum number of iterations reached. **ECOS_NUMERICS (-2)** Search direction unreliable. **ECOS_OUTCONE (-3)** s or z got outside the cone, numerics? **ECOS_SIGINT (-4)** Solver interrupted by a signal/ctrl-c. **ECOS_FATAL (-7)** Unknown problem in solver.

Value

A string indicating the status, either "optimal", "infeasible", "unbounded", "optimal_inaccurate", "infeasible_inaccurate", "unbounded_inaccurate", or "solver_error".

`status_map,GLPK-method`

GLPK Status Map

Description

Map of GLPK status to CVXR status.

Usage

```
## S4 method for signature 'GLPK'
status_map(solver, status)
```

Arguments

`solver`	A GLPK object.
`status`	An exit code returned by GLPK.

Value

A string indicating the status, either "optimal", "infeasible", "unbounded", "optimal_inaccurate", "infeasible_inaccurate", "unbounded_inaccurate", or "solver_error".

`status_map,GUROBI-method`

GUROBI Status Map

Description

Map of GUROBI status to CVXR status.

Usage

```
## S4 method for signature 'GUROBI'
status_map(solver, status)
```

Arguments

`solver`	A GUROBI object.
`status`	An exit code returned by GUROBI. See the GUROBI documentation for details.

Value

A string indicating the status, either "optimal", "infeasible", "unbounded", "optimal_inaccurate", "infeasible_inaccurate", "unbounded_inaccurate", or "solver_error".

`status_map,LPSOLVE-method`

LPSOLVE Status Map

Description

Map of LPSOLVE status to CVXR status.

Usage

```
## S4 method for signature 'LPSOLVE'
status_map(solver, status)
```

Arguments

`solver`	A LPSOLVE object.
`status`	An exit code returned by LPSOLVE.

Value

A string indicating the status, either "optimal", "infeasible", "unbounded", "optimal_inaccurate", "infeasible_inaccurate", "unbounded_inaccurate", or "solver_error".

`status_map,MOSEK-method`

MOSEK Status Map

Description

Map of MOSEK status to CVXR status.

Usage

```
## S4 method for signature 'MOSEK'
status_map(solver, status)
```

Arguments

`solver`	A MOSEK object.
`status`	An exit code returned by MOSEK. See the MOSEK documentation for details.

Value

A string indicating the status, either "optimal", "infeasible", "unbounded", "optimal_inaccurate", "infeasible_inaccurate", "unbounded_inaccurate", or "solver_error".

status_map,SCS-method *SCS Status Map*

Description

Map of SCS status to CVXR status.

Usage

```
## S4 method for signature 'SCS'
status_map(solver, status)
```

Arguments

solver	A SCS object.
status	An exit code returned by SCS.

Value

A string indicating the status, either "optimal", "infeasible", "unbounded", "optimal_inaccurate", "infeasible_inaccurate", "unbounded_inaccurate", or "solver_error".

SumEntries-class *The SumEntries class.*

Description

This class represents the sum of all entries in a vector or matrix.

Usage

```
SumEntries(expr, axis = NA_real_)

## S4 method for signature 'SumEntries'
to_numeric(object, values)

## S4 method for signature 'SumEntries'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

expr	An Expression representing a vector or matrix.
axis	(Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.
object	A SumEntries object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: Sum the entries along the specified axis.
- graph_implementation: The graph implementation of the atom.

Slots

expr An Expression representing a vector or matrix.

axis (Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and NA indicates rows and columns. The default is NA.

SumLargest-class *The SumLargest class.*

Description

The sum of the largest k values of a matrix.

Usage

```
SumLargest(x, k)

## S4 method for signature 'SumLargest'
validate_args(object)

## S4 method for signature 'SumLargest'
to_numeric(object, values)

## S4 method for signature 'SumLargest'
size_from_args(object)

## S4 method for signature 'SumLargest'
sign_from_args(object)
```

```
## S4 method for signature 'SumLargest'
is_atom_convex(object)

## S4 method for signature 'SumLargest'
is_atom_concave(object)

## S4 method for signature 'SumLargest'
is_incr(object, idx)

## S4 method for signature 'SumLargest'
is_decr(object, idx)

## S4 method for signature 'SumLargest'
get_data(object)

## S4 method for signature 'SumLargest'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x	An Expression or numeric matrix.
k	The number of largest values to sum over.
object	A SumLargest object.
values	A list of arguments to the atom.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check that k is a positive integer.
- to_numeric: The sum of the k largest entries of the vector or matrix.
- size_from_args: The atom is a scalar.
- sign_from_args: The sign of the atom.
- is_atom_convex: The atom is convex.
- is_atom_concave: The atom is not concave.
- is_incr: The atom is weakly increasing in every argument.
- is_decr: The atom is not weakly decreasing in any argument.
- get_data: A list containing k.
- graph_implementation: The graph implementation of the atom.

Slots

`x` An Expression or numeric matrix.

`k` The number of largest values to sum over.

`sum_entries` *Sum of Entries*

Description

The sum of entries in a vector or matrix.

Usage

```
sum_entries(expr, axis = NA_real_)

## S3 method for class 'Expression'
sum(..., na.rm = FALSE)
```

Arguments

`expr`	An Expression, vector, or matrix.
`axis`	(Optional) The dimension across which to apply the function: `1` indicates rows, `2` indicates columns, and `NA` indicates rows and columns. The default is `NA`.
`...`	Numeric scalar, vector, matrix, or Expression objects.
`na.rm`	(Unimplemented) A logical value indicating whether missing values should be removed.

Value

An Expression representing the sum of the entries of the input.

Examples

```
x <- Variable(2)
prob <- Problem(Minimize(sum_entries(x)), list(t(x) >= matrix(c(1,2), nrow = 1, ncol = 2)))
result <- solve(prob)
result$value
result$getValue(x)

C <- Variable(3,2)
prob <- Problem(Maximize(sum_entries(C)), list(C[2:3,] <= 2, C[1,] == 1))
result <- solve(prob)
result$value
result$getValue(C)
```

sum_largest *Sum of Largest Values*

Description

The sum of the largest k values of a vector or matrix.

Usage

```
sum_largest(x, k)
```

Arguments

x	An Expression, vector, or matrix.
k	The number of largest values to sum over.

Value

An Expression representing the sum of the largest k values of the input.

Examples

```
m <- 300
n <- 9
X <- matrix(stats::rnorm(m*n), nrow = m, ncol = n)
X <- cbind(rep(1,m), X)
b <- c(0, 0.8, 0, 1, 0.2, 0, 0.4, 1, 0, 0.7)
y <- X %*% b + stats::rnorm(m)

beta <- Variable(n+1)
obj <- sum_largest((y - X %*% beta)^2, 100)
prob <- Problem(Minimize(obj))
result <- solve(prob)
result$getValue(beta)
```

sum_smallest *Sum of Smallest Values*

Description

The sum of the smallest k values of a vector or matrix.

Usage

```
sum_smallest(x, k)
```

Arguments

x	An Expression, vector, or matrix.
k	The number of smallest values to sum over.

Value

An Expression representing the sum of the smallest k values of the input.

Examples

```
m <- 300
n <- 9
X <- matrix(stats::rnorm(m*n), nrow = m, ncol = n)
X <- cbind(rep(1,m), X)
b <- c(0, 0.8, 0, 1, 0.2, 0, 0.4, 1, 0, 0.7)
factor <- 2*rbinom(m, size = 1, prob = 0.8) - 1
y <- factor * (X %*% b) + stats::rnorm(m)

beta <- Variable(n+1)
obj <- sum_smallest(y - X %*% beta, 200)
prob <- Problem(Maximize(obj), list(0 <= beta, beta <= 1))
result <- solve(prob)
result$getValue(beta)
```

---

sum_squares *Sum of Squares*

Description

The sum of the squared entries in a vector or matrix.

Usage

```
sum_squares(expr)
```

Arguments

expr	An Expression, vector, or matrix.

Value

An Expression representing the sum of squares of the input.

Examples

```
m <- 30
n <- 20
A <- matrix(stats::rnorm(m*n), nrow = m, ncol = n)
b <- matrix(stats::rnorm(m), nrow = m, ncol = 1)

x <- Variable(n)
obj <- Minimize(sum_squares(A %*% x - b))
constr <- list(0 <= x, x <= 1)
prob <- Problem(obj, constr)
result <- solve(prob)

result$value
result$getValue(x)
result$getDualValue(constr[[1]])
```

Symmetric *Symmetric Variable*

Description

An expression representing a symmetric matrix.

Usage

```
Symmetric(n, name = NA_character_)
```

Arguments

n	The number of rows/columns in the matrix.
name	(Optional) A character string representing the name of the variable.

Value

An Expression representing the symmetric matrix.

Examples

```
x <- Symmetric(3, name="s3")
```

SymmetricUpperTri-class *The SymmetricUpperTri class.*

Description

This class represents the upper triangular part of a symmetric variable.

Usage

```
SymmetricUpperTri(n, name = NA_character_)

## S4 method for signature 'SymmetricUpperTri'
as.character(x)

## S4 method for signature 'SymmetricUpperTri'
get_data(object)
```

Arguments

n	The number of rows/columns in the matrix.
name	(Optional) A character string representing the name of the variable.
x, object	A SymmetricUpperTri object.

Methods (by generic)

- get_data: Returns list(n, name).

Slots

id (Internal) A unique identification number used internally.

n The number of rows/columns in the matrix.

rows The number of rows in the variable.

cols The number of columns in the variable.

name (Optional) A character string representing the name of the variable.

primal_value (Internal) The primal value of the variable stored internally.

Examples

```
x <- SymmetricUpperTri(3, name="s3")
name(x)
get_data(x)
```

t.Expression *Matrix Transpose*

Description

The transpose of a matrix.

Usage

```
## S3 method for class 'Expression'
t(x)

## S4 method for signature 'Expression'
t(x)
```

Arguments

x	An Expression representing a matrix.

Value

An Expression representing the transposed matrix.

Examples

```
x <- Variable(3, 4)
t(x)
```

to_numeric *Numeric Value of Atom*

Description

Returns the numeric value of the atom evaluated on the specified arguments.

Usage

```
to_numeric(object, values)
```

Arguments

object	An Atom object.
values	A list of arguments to the atom.

Value

A numeric scalar, vector, or matrix.

`Trace-class` *The Trace class.*

Description

This class represents the sum of the diagonal entries in a matrix.

Usage

```
Trace(expr)

## S4 method for signature 'Trace'
validate_args(object)

## S4 method for signature 'Trace'
to_numeric(object, values)

## S4 method for signature 'Trace'
size_from_args(object)

## S4 method for signature 'Trace'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

`expr`	An Expression representing a matrix.
`object`	A Trace object.
`values`	A list of arguments to the atom.
`arg_objs`	A list of linear expressions for each argument.
`size`	A vector with two elements representing the size of the resulting expression.
`data`	A list of additional data required by the atom.

Methods (by generic)

- `validate_args`: Check the argument is a square matrix.
- `to_numeric`: Sum the diagonal entries.
- `size_from_args`: The atom is a scalar.
- `graph_implementation`: The graph implementation of the atom.

Slots

`expr` An Expression representing a matrix.

Transpose-class *The Transpose class.*

Description

This class represents the matrix transpose.

Usage

```
## S4 method for signature 'Transpose'
to_numeric(object, values)

## S4 method for signature 'Transpose'
size_from_args(object)

## S4 method for signature 'Transpose'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

object	A Transpose object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- to_numeric: The transpose of the given value.
- size_from_args: The size of the atom.
- graph_implementation: The graph implementation of the atom.

tv *Total Variation*

Description

The total variation of a vector, matrix, or list of matrices. Uses L1 norm of discrete gradients for vectors and L2 norm of discrete gradients for matrices.

Usage

```
tv(value, ...)
```

Arguments

value	An Expression, vector, or matrix.
...	(Optional) Expression objects or numeric constants that extend the third dimension of value.

Value

An Expression representing the total variation of the input.

Examples

```
rows <- 10
cols <- 10
Uorig <- matrix(sample(0:255, size = rows * cols, replace = TRUE), nrow = rows, ncol = cols)

# Known is 1 if the pixel is known, 0 if the pixel was corrupted
Known <- matrix(0, nrow = rows, ncol = cols)
for(i in 1:rows) {
   for(j in 1:cols) {
      if(stats::runif(1) > 0.7)
         Known[i,j] <- 1
   }
}
Ucorr <- Known %*% Uorig

# Recover the original image using total variation in-painting
U <- Variable(rows, cols)
obj <- Minimize(tv(U))
constraints <- list(Known * U == Known * Ucorr)
prob <- Problem(obj, constraints)
result <- solve(prob, solver = "SCS")
result$getValue(U)
```

UnaryOperator-class *The UnaryOperator class.*

Description

This base class represents expressions involving unary operators.

Slots

`expr` The Expression that is being operated upon.

`op_name` A `character` string indicating the unary operation.

unpack_results *Parse output from a solver and updates problem state*

Description

Updates problem status, problem value, and primal and dual variable values

Usage

```
unpack_results(object, solver, results_dict)
```

Arguments

object	A Problem object.
solver	A character string specifying the solver such as "ECOS", "SCS" etc.
results_dict	the solver output

Value

A list containing the solution to the problem:

status The status of the solution. Can be "optimal", "optimal_inaccurate", "infeasible", "infeasible_inaccurate", "unbounded", "unbounded_inaccurate", or "solver_error".

value The optimal value of the objective function.

solver The name of the solver.

solve_time The time (in seconds) it took for the solver to solve the problem.

setup_time The time (in seconds) it took for the solver to set up the problem.

num_iters The number of iterations the solver had to go through to find a solution.

getValue A function that takes a Variable object and retrieves its primal value.

getDualValue A function that takes a Constraint object and retrieves its dual value(s).

Examples

```
## Not run:
x <- Variable(2)
obj <- Minimize(x[1] + cvxr_norm(x, 1))
constraints <- list(x >= 2)
prob1 <- Problem(obj, constraints)
# Solve with ECOS.
ecos_data <- get_problem_data(prob1, "ECOS")
# Call ECOS solver interface directly
ecos_output <- ECOSolveR::ECOS_csolve(
                           c = ecos_data[["c"]],
                           G = ecos_data[["G"]],
                           h = ecos_data[["h"]],
                           dims = ecos_data[["dims"]],
```

```
                          A = ecos_data[["A"]],
                          b = ecos_data[["b"]]
                        )
# Unpack raw solver output.
res1 <- unpack_results(prob1, "ECOS", ecos_output)
# Without DCP validation (so be sure of your math), above is equivalent to:
# res1 <- solve(prob1, solver = "ECOS")
X <- Semidef(2)
Fmat <- rbind(c(1,0), c(0,-1))
obj <- Minimize(sum_squares(X - Fmat))
prob2 <- Problem(obj)
scs_data <- get_problem_data(prob2, "SCS")
scs_output <- scs::scs(
                      A = scs_data[['A']],
                      b = scs_data[['b']],
                      obj = scs_data[['c']],
                      cone = scs_data[['dims']]
                    )
res2 <- unpack_results(prob2, "SCS", scs_output)
# Without DCP validation (so be sure of your math), above is equivalent to:
# res2 <- solve(prob2, solver = "SCS")

## End(Not run)
```

`UpperTri-class` *The UpperTri class.*

Description

The vectorized strictly upper triagonal entries of a matrix.

Usage

```
UpperTri(expr)

## S4 method for signature 'UpperTri'
validate_args(object)

## S4 method for signature 'UpperTri'
to_numeric(object, values)

## S4 method for signature 'UpperTri'
size_from_args(object)

## S4 method for signature 'UpperTri'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

expr	An Expression or numeric matrix.
object	An UpperTri object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check the argument is a square matrix.
- to_numeric: Vectorize the upper triagonal entries.
- size_from_args: The size of the atom.
- graph_implementation: The graph implementation of the atom.

Slots

expr An Expression or numeric matrix.

---

upper_tri *Upper Triangle of a Matrix*

Description

The vectorized strictly upper triangular entries of a matrix.

Usage

```
upper_tri(expr)
```

Arguments

expr	An Expression or matrix.

Value

An Expression representing the upper triangle of the input.

Examples

```
C <- Variable(3,3)
val <- cbind(3:5, 6:8, 9:11)
prob <- Problem(Maximize(upper_tri(C)[3,1]), list(C == val))
result <- solve(prob)
result$value
result$getValue(upper_tri(C))
```

validate_args *Validate Arguments*

Description

Validate an atom's arguments, returning an error if any are invalid.

Usage

```
validate_args(object)
```

Arguments

object	An Atom object.

validate_solver *Validate Solver*

Description

Raises an exception if the solver cannot solve the problem.

Usage

```
validate_solver(solver, constraints)
```

Arguments

solver	A Solver object.
constraints	A list of canonicalized constraints

validate_val *Validate Value*

Description

Check that the value satisfies a Leaf's symbolic attributes.

Usage

```
validate_val(object, val)
```

Arguments

`object`	A Leaf object.
`val`	The assigned value.

Value

The value converted to proper matrix type.

`value-methods` *Get or Set Value*

Description

Get or set the value of a variable, parameter, expression, or problem.

Usage

```
value(object)

value(object) <- value
```

Arguments

`object`	A Variable, Parameter, Expression, or Problem object.
`value`	A numeric scalar, vector, or matrix to assign to the object.

Value

The numeric value of the variable, parameter, or expression. If any part of the mathematical object is unknown, return `NA`.

Examples

```
lambda <- Parameter()
value(lambda)

value(lambda) <- 5
value(lambda)
```

Variable-class *The Variable class.*

Description

This class represents an optimization variable.

Usage

```
Variable(rows = 1, cols = 1, name = NA_character_)

## S4 method for signature 'Variable'
as.character(x)

## S4 method for signature 'Variable'
id(object)

## S4 method for signature 'Variable'
is_positive(object)

## S4 method for signature 'Variable'
is_negative(object)

## S4 method for signature 'Variable'
size(object)

## S4 method for signature 'Variable'
get_data(object)

## S4 method for signature 'Variable'
name(object)

## S4 method for signature 'Variable'
value(object)

## S4 replacement method for signature 'Variable'
value(object) <- value

## S4 method for signature 'Variable'
grad(object)

## S4 method for signature 'Variable'
variables(object)

## S4 method for signature 'Variable'
canonicalize(object)
```

Arguments

rows	The number of rows in the variable.
cols	The number of columns in the variable.
name	(Optional) A character string representing the name of the variable.
x, object	A Variable object.
value	The value to assign to the primal variable.

Methods (by generic)

- id: The unique ID of the variable.
- is_positive: A logical value indicating whether the variable is positive.
- is_negative: A logical value indicating whether the variable is negative.
- size: The c(row, col) dimensions of the variable.
- get_data: Returns list(rows, cols, name).
- name: The name of the variable.
- value: The value of the variable.
- value<-: Set the value of the primal variable.
- grad: The sub/super-gradient of the variable represented as a sparse matrix.
- variables: Returns itself as a variable.
- canonicalize: The canonical form of the variable.

Slots

id (Internal) A unique identification number used internally.

rows The number of rows in the variable.

cols The number of columns in the variable.

name (Optional) A character string representing the name of the variable.

primal_value (Internal) The primal value of the variable stored internally.

Examples

```
x <- Variable(3, name = "x0") ## 3-int variable
y <- Variable(3, 3, name = "y0") # Matrix variable
as.character(y)
id(y)
is_positive(x)
is_negative(x)
size(y)
name(y)
value(y) <- matrix(1:9, nrow = 3)
value(y)
grad(y)
variables(y)
canonicalize(y)
```

vec *Vectorization of a Matrix*

Description

Flattens a matrix into a vector in column-major order.

Usage

```
vec(X)
```

Arguments

X	An Expression or matrix.

Value

An Expression representing the vectorized matrix.

Examples

```
A <- Variable(2,2)
c <- 1:4
expr <- vec(A)
obj <- Minimize(t(expr) %*% c)
constraints <- list(A == cbind(c(-1,-2), c(3,4)))
prob <- Problem(obj, constraints)
result <- solve(prob)
result$value
result$getValue(expr)
```

vstack *Vertical Concatenation*

Description

The vertical concatenation of expressions. This is equivalent to `rbind` when applied to objects with the same number of columns.

Usage

```
vstack(...)
```

Arguments

...	Expression objects, vectors, or matrices. All arguments must have the same number of columns.

Value

An Expression representing the concatenated inputs.

Examples

```
x <- Variable(2)
y <- Variable(3)
c <- matrix(1, nrow = 1, ncol = 5)
prob <- Problem(Minimize(c %*% vstack(x, y)), list(x == c(1,2), y == c(3,4,5)))
result <- solve(prob)
result$value

c <- matrix(1, nrow = 1, ncol = 4)
prob <- Problem(Minimize(c %*% vstack(x, x)), list(x == c(1,2)))
result <- solve(prob)
result$value

A <- Variable(2,2)
C <- Variable(3,2)
c <- matrix(1, nrow = 2, ncol = 2)
prob <- Problem(Minimize(sum(vstack(A, C))), list(A >= 2*c, C == -2))
result <- solve(prob)
result$value

B <- Variable(2,2)
c <- matrix(1, nrow = 1, ncol = 2)
prob <- Problem(Minimize(sum(vstack(c %*% A, c %*% B))), list(A >= 2, B == -2))
result <- solve(prob)
result$value
```

VStack-class *The VStack class.*

Description

Vertical concatenation of values.

Usage

```
VStack(...)

## S4 method for signature 'VStack'
validate_args(object)

## S4 method for signature 'VStack'
to_numeric(object, values)

## S4 method for signature 'VStack'
size_from_args(object)
```

```
## S4 method for signature 'VStack'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

...	Expression objects or matrices. All arguments must have the same number of columns.
object	A VStack object.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check all arguments have the same width.
- to_numeric: Vertically concatenate the values using rbind.
- size_from_args: The size of the atom.
- graph_implementation: The graph implementation of the atom.

Slots

... Expression objects or matrices. All arguments must have the same number of columns.

[,Expression,missing,missing,ANY-method

The Index class.

Description

This class represents indexing or slicing into a matrix.

Usage

```
## S4 method for signature 'Expression,missing,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,index,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,missing,index,ANY'
x[i, j, ..., drop = TRUE]
```

```
## S4 method for signature 'Expression,index,index,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,matrix,index,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,index,matrix,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,matrix,matrix,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,matrix,missing,ANY'
x[i, j, ..., drop = TRUE]

Index(expr, key)

## S4 method for signature 'Index'
to_numeric(object, values)

## S4 method for signature 'Index'
size_from_args(object)

## S4 method for signature 'Index'
get_data(object)

## S4 method for signature 'Index'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x, object	An Index object.
i, j	The row and column indices of the slice.
...	(Unimplemented) Optional arguments.
drop	(Unimplemented) A logical value indicating whether the result should be coerced to the lowest possible dimension.
expr	An Expression representing a vector or matrix.
key	A list containing the start index, end index, and step size of the slice.
values	A list of arguments to the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- `to_numeric`: The index/slice into the given value.
- `size_from_args`: The size of the atom.
- `get_data`: A list containing `key`.
- `graph_implementation`: The graph implementation of the atom.

Slots

`expr` An Expression representing a vector or matrix.

`key` A list containing the start index, end index, and step size of the slice.

`%*%,Expression,Expression-method`

The MulExpression class.

Description

This class represents the matrix product of two linear expressions. See MulElemwise for the elementwise product.

Usage

```
## S4 method for signature 'Expression,Expression'
x %*% y

## S4 method for signature 'Expression,ConstVal'
x %*% y

## S4 method for signature 'ConstVal,Expression'
x %*% y

## S4 method for signature 'MulExpression'
validate_args(object)

## S4 method for signature 'MulExpression'
size_from_args(object)

## S4 method for signature 'MulExpression'
is_incr(object, idx)

## S4 method for signature 'MulExpression'
is_decr(object, idx)

## S4 method for signature 'MulExpression'
graph_implementation(object, arg_objs, size,
  data = NA_real_)
```

Arguments

x, y	The Expression objects or numeric constants to multiply.
object	A MulExpression object.
idx	An index into the atom.
arg_objs	A list of linear expressions for each argument.
size	A vector with two elements representing the size of the resulting expression.
data	A list of additional data required by the atom.

Methods (by generic)

- validate_args: Check the dimensions.
- size_from_args: The size of the expression.
- is_incr: Is the left-hand expression positive?
- is_decr: Is the left-hand expression negative?
- graph_implementation: The graph implementation of the expression.

See Also

MulElemwise

%>>% *The PSDConstraint class.*

Description

This class represents the positive semidefinite constraint, $X \succeq Y$, i.e. $z^T(X - Y)z \geq 0$ for all z.

Usage

```
e1 %>>% e2

e1 %<<% e2

## S4 method for signature 'Expression,Expression'
e1 %>>% e2

## S4 method for signature 'Expression,ConstVal'
e1 %>>% e2

## S4 method for signature 'ConstVal,Expression'
e1 %>>% e2

## S4 method for signature 'Expression,Expression'
e1 %<<% e2
```

```
## S4 method for signature 'Expression,ConstVal'
e1 %<<% e2

## S4 method for signature 'ConstVal,Expression'
e1 %<<% e2

PSDConstraint(lh_exp, rh_exp)

## S4 method for signature 'PSDConstraint'
is_dcp(object)

## S4 method for signature 'PSDConstraint'
residual(object)

## S4 method for signature 'PSDConstraint'
canonicalize(object)
```

Arguments

`e1, e2`	The Expression objects or numeric constants to compare.
`lh_exp`	An Expression, numeric element, vector, or matrix representing the left-hand side of the inequality.
`rh_exp`	An Expression, numeric element, vector, or matrix representing the right-hand side of the inequality.
`object`	A PSDConstraint object.

Methods (by generic)

- `is_dcp`: The constraint is DCP if the left-hand and right-hand expressions are affine.
- `residual`: A Expression representing the residual of the constraint.
- `canonicalize`: The graph implementation of the object. Marks the top level constraint as the `dual_holder` so the dual value will be saved to the PSDConstraint.

Slots

`constr_id` (Internal) A unique integer identification number used internally.

`lh_exp` An Expression, numeric element, vector, or matrix representing the left-hand side of the inequality.

`rh_exp` An Expression, numeric element, vector, or matrix representing the right-hand side of the inequality.

`args` (Internal) A list that holds `lh_exp` and `rh_exp` for internal use.

`.expr` (Internal) An Expression representing `lh_exp - rh_exp` for internal use.

`dual_variable` (Internal) A Variable representing the dual variable associated with the constraint.

`^,Expression,numeric-method`
Elementwise Power

Description

Raises each element of the input to the power p. If `expr` is a CVXR expression, then `expr^p` is equivalent to `power(expr,p)`.

Usage

```
## S4 method for signature 'Expression,numeric'
e1 ^ e2

power(x, p, max_denom = 1024)
```

Arguments

`e1`	An Expression object to exponentiate.
`e2`	The power of the exponential. Must be a numeric scalar.
`x`	An Expression, vector, or matrix.
`p`	A scalar value indicating the exponential power.
`max_denom`	The maximum denominator considered in forming a rational approximation of `p`.

Details

For $p = 0$ and $f(x) = 1$, this function is constant and positive. For $p = 1$ and $f(x) = x$, this function is affine, increasing, and the same sign as x. For $p = 2, 4, 8, \ldots$ and $f(x) = |x|^p$, this function is convex, positive, with signed monotonicity. For $p < 0$ and $f(x) =$

- x^p for $x > 0$
- $+\infty x \leq 0$

, this function is convex, decreasing, and positive. For $0 < p < 1$ and $f(x) =$

- x^p for $x \geq 0$
- $-\infty x < 0$

, this function is concave, increasing, and positivea. For $p > 1, p \neq 2, 4, 8, \ldots$ and $f(x) =$

- x^p for $x \geq 0$
- $+\infty x < 0$

, this function is convex, increasing, and positive.

Examples

```
## Not run:
x <- Variable()
prob <- Problem(Minimize(power(x,1.7) + power(x,-2.3) - power(x,0.45)))
result <- solve(prob)
result$value
result$getValue(x)

## End(Not run)
```

Index