Package ‘CVXR’

June 1, 2024

Type Package
Title Disciplined Convex Optimization
Version 1.0-13
VignetteBuilder knitr
BugReports https://github.com/cvxgrp/CVXR/issues

Description An object-oriented modeling language for disciplined convex programming (DCP) as described in Fu, Narasimhan, and Boyd (2020, <doi:10.18637/jss.v094.i14>). 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. Interfaces to solvers on CRAN and elsewhere are provided, both commercial and open source.

Additional_repositories https://bnaras.github.io/drat

Depends R (>= 3.4.0)
Imports methods, R6, Matrix, Rcpp (>= 0.12.12), bit64, gmp, Rmpfr, ECOSolveR (>= 0.5.4), scs (>= 3.0), stats, osqp
Suggests knitr, rmarkdown, testthat, nnls, slam, covr
LinkingTo Rcpp, RcppEigen
License Apache License 2.0 | file LICENSE
LazyData true

Collate 'CVXR-package.R' 'data.R' 'globals.R' 'generics.R'
 'interface.R' 'canonical.R' 'expressions.R' 'constant.R'
 'variable.R' 'lin_ops.R' 'atoms.R' 'affine.R' 'problem.R'
 'constraints.R' 'elementwise.R' 'coeff_extractor.R'
'reductions.R' 'reduction_solvers.R' 'complex2real.R'
'conic_solvers.R' 'eliminate_pwl.R' 'dcp2cone.R' 'dgp2dcp.R'
'qp2quad_form.R' 'qp_solvers.R' 'utilities.R'
'solver_utilities.R' 'transforms.R' 'exports.R' 'rcppUtils.R'

1
R topics documented:

'RcppExports.R' 'CVXcanon-R6.R' 'Deque.R' 'canonInterface.R'

RoxygenNote 7.3.1

Encoding UTF-8

Enhances Rcplex, gurobi, rcbc, cccp, Rmosek, Rglpk

NeedsCompilation yes

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

Maintainer Anqi Fu <anqif@alumni.stanford.edu>

Repository CRAN

Date/Publication 2024-06-01 08:12:52 UTC

R topics documented:

*,Expression,Expression-method  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
+,Expression,missing-method  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
-,Expression,missing-method  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
.build_matrix_0  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
.build_matrix_1  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
.decomp_quad  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
.LinOpVector__new  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
.LinOpVector__push_back  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
.LinOp_at_index  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
.LinOp__args_push_back  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
.LinOp__get_dense_data  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
.LinOp__get_id  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
.LinOp__get_size  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
.LinOp__get_slice  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
.LinOp__get_sparse  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
.LinOp__get_sparse_data  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
.LinOp__get_type  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
.LinOp__new  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
.LinOp__set_dense_data  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
.LinOp__set_size  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
.LinOp__set_slice  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
.LinOp__set_sparse  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
.LinOp__set_sparse_data  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
.LinOp__set_type  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
.LinOp__size_push_back  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
.LinOp__slice_push_back  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
R topics documented:

Complex2Real\_matrix\_frac\_canon ........................................... 62
Complex2Real\_nonpos\_canon ................................................ 62
Complex2Real\_norm\_nuc\_canon ............................................ 63
Complex2Real\_param\_canon ................................................. 63
Complex2Real\_pnorm\_canon ................................................ 64
Complex2Real\_psd\_canon .................................................. 65
Complex2Real\_quad\_canon .................................................. 65
Complex2Real\_quad\_over\_lin\_canon ...................................... 66
Complex2Real\_real\_canon .................................................. 66
Complex2Real\_separable\_canon .......................................... 67
Complex2Real\_soe\_canon .................................................. 68
Complex2Real\_variable\_canon ............................................. 68
Complex2Real\_zero\_canon ................................................ 69
cone-methods ................................................................. 69
ConeDims-class ................................................................. 70
ConeMatrixStuffing-class .................................................. 70
ConicSolver-class ............................................................. 71
ConicSolver\_get\_coeff\_offset ........................................... 72
ConicSolver\_get\_spacing\_matrix ...................................... 72
Conjugate-class ............................................................... 73
Constant-class ............................................................... 74
ConstantSolver-class ........................................................ 76
Constraint-class .............................................................. 78
construct\_intermediate\_chain,Problem,list\_method ..................... 80
construct\_solving\_chain .................................................... 80
constr\_value ................................................................. 81
conv ............................................................................ 81
Conv-class ..................................................................... 82
CPLEX\_CONIC-class .......................................................... 83
CPLEX\_QP-class ............................................................... 85
CumMax-class ................................................................. 87
cummax\_axis ................................................................. 88
CumSum-class ................................................................. 89
cumsum\_axis ................................................................. 90
curvature ....................................................................... 91
curvature\_atom .............................................................. 91
curvature\_comp .............................................................. 93
curvature\_methods .......................................................... 93
CvxAttr2Constr-class .......................................................... 95
CVXOPT-class ................................................................. 95
cvxr\_norm ................................................................. 97
Dcp2Cone-class ............................................................... 98
Dcp2Cone\_entr\_canon .......................................................... 98
Dcp2Cone\_exp\_canon .......................................................... 99
Dcp2Cone\_geo\_mean\_canon ............................................... 99
Dcp2Cone\_huber\_canon ..................................................... 100
Dcp2Cone\_indicator\_canon ............................................... 100
Dcp2Cone\_kl\_div\_canon ................................................... 101
topics documented:

Dcp2Cone.lambda_max_canon .................................................. 101
Dcp2Cone.lambda_sum_largest_canon ...................................... 102
Dcp2Cone.log1p_canon .......................................................... 102
Dcp2Cone.logistic_canon ...................................................... 103
Dcp2Cone.log_canon ............................................................. 103
Dcp2Cone.log_det_canon ....................................................... 104
Dcp2Cone.log_sum_exp_canon ............................................... 104
Dcp2Cone.matrix_frac_canon ................................................. 105
Dcp2Cone.normNuc_canon ..................................................... 105
Dcp2Cone.pnorm_canon ........................................................ 106
Dcp2Cone.power_canon ........................................................ 106
Dcp2Cone.quad_form_canon .................................................. 107
Dcp2Cone.quad_over_lin_canon ............................................. 107
Dcp2Cone.sigma_max_canon ................................................. 108
Dgp2Dcp-class ................................................................. 108
Dgp2Dcp.add_canon ............................................................ 109
Dgp2Dcp.constant_canon ..................................................... 110
Dgp2Dcp.div_canon ............................................................. 110
Dgp2Dcp.exp_canon ............................................................ 111
Dgp2Dcp.eye_minus_inv_canon ............................................. 111
Dgp2Dcp.geo_mean_canon .................................................... 112
Dgp2Dcp.log_canon ............................................................. 112
Dgp2Dcp.mulexpression_canon .............................................. 113
Dgp2Dcp.mul_canon ............................................................ 113
Dgp2Dcp.nonpos_canon ........................................................ 114
Dgp2Dcp.norm1_canon ........................................................ 114
Dgp2Dcp.norm_inf_canon ..................................................... 115
Dgp2Dcp.one_minus_pos_canon ............................................. 115
Dgp2Dcp.parameter_canon ................................................... 116
Dgp2Dcp.pf_eigenvalue_canon .............................................. 116
Dgp2Dcp.pnorm_canon ........................................................ 117
Dgp2Dcp.power_canon ........................................................ 117
Dgp2Dcp.prod_canon ............................................................ 118
Dgp2Dcp.quad_form_canon .................................................. 118
Dgp2Dcp.quad_over_lin_canon ............................................. 119
Dgp2Dcp.sum_canon ............................................................ 119
Dgp2Dcp.trace_canon ........................................................... 120
Dgp2Dcp.zero_constr_canon ............................................... 120
DgpCanonMethods-class ..................................................... 121
Diag .......................................................... 121
diag.Expression-method .................................................... 122
DiagMat-class ............................................................... 122
DiagVec-class ............................................................... 124
Diff ................................................................. 125
diff.Expression-method .................................................... 126
DiffPos ................................................................. 127
dim_from_args ............................................................. 127
domain ............................................................. 128
### R topics documented:

- dspop ...................................................... 129
- dssamp ................................................... 129
- dual_value-methods ................................. 130
- ECOS-class ............................................. 130
- ECOS.dims_to_solver_dict ..................... 131
- ECOS_BB-class ....................................... 132
- Elementwise-class ................................. 133
- EliminatePwl-class ............................... 134
- EliminatePwl.abs_canon ......................... 134
- EliminatePwl.cummax_canon .................... 135
- EliminatePwl.cumsum_canon .................... 135
- EliminatePwl.max_elemwise_canon ............ 136
- EliminatePwl.max_entries_canon ............... 136
- EliminatePwl.min_elemwise_canon ............. 137
- EliminatePwl.min_entries_canon .............. 137
- EliminatePwl.norm1_canon ...................... 138
- EliminatePwl.norm_inf_canon .................. 138
- EliminatePwl.sum_largest_canon .............. 139
- entr ................................................. 139
- Entr-class ............................................. 140
- EvalParams-class ..................................... 141
- exp,Expression-method ............................. 142
- Exp-class .............................................. 142
- ExpCone-class ......................................... 144
- Expression-class ....................................... 145
- expression-parts ..................................... 149
- extract_dual_value ................................. 150
- extract_mip_idx ....................................... 151
- EyeMinusInv-class ...................................... 151
- eye_minus_inv ......................................... 153
- FlipObjective-class .................................... 153
- format_constr ......................................... 154
- GeoMean-class .......................................... 155
- geo_mean ................................................. 157
- get_data ................................................. 158
- get_dual_values ......................................... 159
- get_id ....................................................... 159
- get_np ..................................................... 160
- get_problem_data ....................................... 160
- get_sp ....................................................... 161
- GLPK-class .............................................. 161
- GLPK_MI-class ........................................... 163
- grad ......................................................... 164
- graph_implementation .............................. 165
- group_constraints ...................................... 166
- GUROBI_CONIC-class .................................... 166
- GUROBI_QP-class ........................................ 168
- HarmonicMean ........................................... 169
<table>
<thead>
<tr>
<th>R topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>harmonic_mean</td>
<td>170</td>
</tr>
<tr>
<td>hstack</td>
<td>170</td>
</tr>
<tr>
<td>HStack-class</td>
<td>171</td>
</tr>
<tr>
<td>huber</td>
<td>172</td>
</tr>
<tr>
<td>Huber-class</td>
<td>173</td>
</tr>
<tr>
<td>id</td>
<td>175</td>
</tr>
<tr>
<td>Imag-class</td>
<td>176</td>
</tr>
<tr>
<td>import_solver</td>
<td>177</td>
</tr>
<tr>
<td>installed_solvers</td>
<td>177</td>
</tr>
<tr>
<td>InverseData-class</td>
<td>178</td>
</tr>
<tr>
<td>invert</td>
<td>178</td>
</tr>
<tr>
<td>inv_pos</td>
<td>179</td>
</tr>
<tr>
<td>is_dcp</td>
<td>179</td>
</tr>
<tr>
<td>is_dgp</td>
<td>180</td>
</tr>
<tr>
<td>is_mixed_integer</td>
<td>180</td>
</tr>
<tr>
<td>is_qp</td>
<td>181</td>
</tr>
<tr>
<td>is_stuffed_cone_constraint</td>
<td>181</td>
</tr>
<tr>
<td>is_stuffed_cone_objective</td>
<td>182</td>
</tr>
<tr>
<td>is_stuffed_qp_objective</td>
<td>182</td>
</tr>
<tr>
<td>KLDiv-class</td>
<td>183</td>
</tr>
<tr>
<td>kl_div</td>
<td>184</td>
</tr>
<tr>
<td>Kron-class</td>
<td>185</td>
</tr>
<tr>
<td>kronecker,Expression,ANY-method</td>
<td>186</td>
</tr>
<tr>
<td>LambdaMax-class</td>
<td>187</td>
</tr>
<tr>
<td>LambdaMin</td>
<td>188</td>
</tr>
<tr>
<td>LambdaSumLargest-class</td>
<td>189</td>
</tr>
<tr>
<td>lambda_max</td>
<td>190</td>
</tr>
<tr>
<td>lambda_min</td>
<td>190</td>
</tr>
<tr>
<td>lambda_sum_largest</td>
<td>191</td>
</tr>
<tr>
<td>lambda_sum_smallest</td>
<td>192</td>
</tr>
<tr>
<td>leaf-attr</td>
<td>193</td>
</tr>
<tr>
<td>Leaf-class</td>
<td>193</td>
</tr>
<tr>
<td>linearize</td>
<td>197</td>
</tr>
<tr>
<td>ListORConstr-class</td>
<td>198</td>
</tr>
<tr>
<td>log,Expression-method</td>
<td>198</td>
</tr>
<tr>
<td>Log-class</td>
<td>199</td>
</tr>
<tr>
<td>Log1p-class</td>
<td>201</td>
</tr>
<tr>
<td>LogDet-class</td>
<td>202</td>
</tr>
<tr>
<td>logistic</td>
<td>203</td>
</tr>
<tr>
<td>Logistic-class</td>
<td>204</td>
</tr>
<tr>
<td>LogSumExp-class</td>
<td>205</td>
</tr>
<tr>
<td>log_det</td>
<td>207</td>
</tr>
<tr>
<td>log_log_curvature</td>
<td>208</td>
</tr>
<tr>
<td>log_log_curvature-atom</td>
<td>208</td>
</tr>
<tr>
<td>log_log_curvature-methods</td>
<td>209</td>
</tr>
<tr>
<td>log_sum_exp</td>
<td>209</td>
</tr>
<tr>
<td>MatrixFrac-class</td>
<td>210</td>
</tr>
</tbody>
</table>
topics documented:

MatrixStuffing-class ................................................. 212
matrix_frac ............................................................ 213
matrix_prop-methods .................................................. 214
matrix_trace ........................................................... 214
MaxElemwise-class ..................................................... 215
MaxEntries-class ......................................................... 216
Maximize-class .......................................................... 218
max_elemwise ............................................................ 219
max_entries ............................................................. 220
mean(Expression) ......................................................... 221
MinElemwise-class ...................................................... 222
MinEntries-class ........................................................ 223
Minimize-class ......................................................... 225
min_elemwise ............................................................ 226
min_entries .............................................................. 226
mip_capable .............................................................. 227
MixedNorm ............................................................... 228
mixed_norm .............................................................. 228
MOSEK-class ............................................................. 229
MOSEK.parse_dual_vars ................................................. 231
MOSEK.recover_dual_variables ....................................... 231
multiply ................................................................. 232
Multiply-class .......................................................... 232
name ....................................................................... 234
Neg ....................................................................... 234
gen .................................................................... 235
NonlinearConstraint-class ........................................... 235
NonPosConstraint-class .............................................. 236
Norm ...................................................................... 237
norm,Expression,character-method ................................ 237
norm1 .................................................................... 238
Norm1-class ............................................................. 239
Norm2 ..................................................................... 241
norm2 .................................................................... 242
NormInf-class .......................................................... 243
NormNuc-class .......................................................... 245
norm_inf ............................................................... 246
norm_nuc ............................................................... 247
Objective-arith ........................................................ 248
Objective-class ........................................................ 249
OneMinusPos-class .................................................... 250
one_minus_pos ........................................................ 252
OSQP-class ............................................................. 252
Parameter-class ......................................................... 254
perform ................................................................. 256
PfEigenvalue-class ..................................................... 256
pf_eigenvalue .......................................................... 258
Pnorm-class ............................................................. 259
### R topics documented:

- Pos                                      ........................................ 261
- pos                                      ........................................ 262
- Power-class                              ........................................ 262
- Problem-arith                            ........................................ 265
- Problem-class                            ........................................ 266
- problem-parts                            ........................................ 270
- ProdEntries-class                       ........................................ 270
- prod_entries                             ........................................ 272
- project-methods                          ........................................ 273
- Promote-class                            ........................................ 274
- PSDWrap-class                            ........................................ 275
- psd_coeff_offset                         ........................................ 276
- psolve                                    ........................................ 276
- p_norm                                    ........................................ 278
- Qp2SymbolicQp-class                      ........................................ 280
- QpMatrixStuffing-class                   ........................................ 280
- QpSolver-class                           ........................................ 280
- QuadForm-class                           ........................................ 281
- QuadOverLin-class                        ........................................ 283
- quad_form                                 ........................................ 285
- quad_over_lin                            ........................................ 285
- Rdict-class                              ........................................ 286
- Rdictdefault-class                       ........................................ 287
- Real-class                               ........................................ 288
- reduce                                    ........................................ 289
- Reduction-class                          ........................................ 289
- ReductionSolver-class                    ........................................ 291
- resetOptions                             ........................................ 293
- Reshape-class                            ........................................ 293
- reshape_expr                             ........................................ 294
- residual-methods                         ........................................ 296
- retrieve                                 ........................................ 296
- scaled_lower_tri                         ........................................ 297
- scalene                                  ........................................ 297
- SCS-class                                ........................................ 298
- SCS.dims_to_solver_dict                  ........................................ 300
- SCS.extract_dual_value                   ........................................ 300
- setIdCounter                             ........................................ 301
- SigmaMax-class                           ........................................ 301
- sigma_max                                ........................................ 303
- sign.Expression-method                   ........................................ 303
- sign-methods                             ........................................ 304
- sign_from_args                           ........................................ 305
- size                                      ........................................ 305
- size-methods                             ........................................ 306
- SizeMetrics-class                        ........................................ 307
- SOC-class                                ........................................ 307
- SOCAxis-class                            ........................................ 309
R topics documented:

Solution-class 310
SolverStats-class 311
SolvingChain-class 311
sqrt,Expression-method 313
square,Expression-method 313
SumEntries-class 314
SumLargest-class 315
SumSmallest 317
SumSquares 317
sum_entries 318
sum_largest 319
sum_smallest 320
sum_squares 320
SymbolicQuadForm-class 321
t,Expression 323
TotalVariation 323
to_numeric 324
Trace-class 324
Transpose-class 325
tri_to_full 326
tv 327
UnaryOperator-class 328
unpack_results 328
UpperTri-class 330
upper_tri 331
validate_args 332
validate_val 332
value-methods 333
Variable-class 333
vec 335
vectorized_lower_tri_to_mat 335
vstack 336
VStack-class 337
Wrap-class 338
ZeroConstraint-class 339
[,Expression,index,missing,ANY-method 340
[,Expression,missing,missing,ANY-method 341
%*%,Expression,missing,ANY-method 343
%>>% 345
^,Expression,numeric-method 346

Index 348
### *,Expression,Expression-method

#### Elementwise multiplication operator

**Description**

Elementwise multiplication operator

**Usage**

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

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

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

**Arguments**

- `e1, e2` The `Expression` objects or numeric constants to multiply elementwise.

### +,Expression,missing-method

*The AddExpression class.*

**Description**

This class represents the sum of any number of expressions.

**Usage**

```r
## 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'
```
dim_from_args(object)
```

## S4 method for signature 'AddExpression'
```
name(x)
```

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

## S4 method for signature 'AddExpression'
```
is_atom_log_log_convex(object)
```

## S4 method for signature 'AddExpression'
```
is_atom_log_log_concave(object)
```

## S4 method for signature 'AddExpression'
```
is_symmetric(object)
```

## S4 method for signature 'AddExpression'
```
is_hermitian(object)
```

## S4 method for signature 'AddExpression'
```
copy(object, args = NULL, id_objects = list())
```

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

### Arguments
- **e1, e2** The Expression objects or numeric constants to add.
- **x, object** An AddExpression object.
- **values** A list of arguments to the atom.
- **args** An optional list of arguments to reconstruct the atom. Default is to use current args of the atom.
- **id_objects** Currently unused.
- **arg_objs** A list of linear expressions for each argument.
- **dim** A vector representing the dimensions of the resulting expression.
- **data** A list of additional data required by the atom.

### Methods (by generic)
- **dim_from_args(AddExpression)**: The dimensions of the expression.
- **name(AddExpression)**: The string form of the expression.
- **to_numeric(AddExpression)**: Sum all the values.
- **is_atom_log_log_convex(AddExpression)**: Is the atom log-log convex?
- **is_atom_log_log_concave(AddExpression)**: Is the atom log-log concave?
• is_symmetric(AddExpression): Is the atom symmetric?
• is_hermitian(AddExpression): Is the atom hermitian?
• copy(AddExpression): Returns a shallow copy of the AddExpression atom
• graph_implementation(AddExpression): The graph implementation of the expression.

Slots

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

Description

This class represents the negation of an affine expression.

Usage

```r
# 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'
dim_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'
is_symmetric(object)
```
## S4 method for signature 'NegExpression'
is_hermitian(object)

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

### Arguments

- **e1, e2**
  - The `Expression` objects or numeric constants to subtract.
- **object**
  - A `NegExpression` object.
- **idx**
  - An index into the atom.
- **arg_objs**
  - A list of linear expressions for each argument.
- **dim**
  - A vector representing the dimensions of the resulting expression.
- **data**
  - A list of additional data required by the atom.

### Methods (by generic)

- `dim_from_args(NegExpression)`: The (row, col) dimensions of the expression.
- `sign_from_args(NegExpression)`: The (is positive, is negative) sign of the expression.
- `is_incr(NegExpression)`: The expression is not weakly increasing in any argument.
- `is_decr(NegExpression)`: The expression is weakly decreasing in every argument.
- `is_symmetric(NegExpression)`: Is the expression symmetric?
- `is_hermitian(NegExpression)`: Is the expression Hermitian?
- `graph_implementation(NegExpression)`: 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

---

**decomp_quad**  
*Compute a Matrix Decomposition.*

**Description**

Compute sgn, scale, M such that $P = sgn \ast scale \ast dot(M, t(M))$.

**Usage**

```
decomp_quad(P, cond = NA, rcond = NA)
```

**Arguments**

- **P**  
  A real symmetric positive or negative (semi)definite input matrix
- **cond**  
  Cutoff for small eigenvalues. Singular values smaller than rcond * largest_eigenvalue are considered negligible.
- **rcond**  
  Cutoff for small eigenvalues. Singular values smaller than rcond * largest_eigenvalue are considered negligible.

**Value**

A list consisting of induced matrix 2-norm of P and a rectangular matrix such that $P = scale \ast (dot(M1, t(M1)) - dot(M2, t(M2)))$

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**

```cpp
.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**

```cpp
.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

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

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
### Create a new LinOp object

**Description**
Create a new LinOp object.

**Usage**
```r
.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**
```r
.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**
```r
.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

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

**Usage**
.LINEOp__size_push_back(xp, intVal)

**Arguments**
- **xp**: the LinOp Object XPtr
- **intVal**: the integer value to push back

---

.LinOp__slice_push_back

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

**Usage**
.LINEOp__slice_push_back(xp, intVec)

**Arguments**
- **xp**: the LinOp Object XPtr
- **intVec**: an integer vector to push back
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

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**

```c
.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**

```c
.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

.p_norm    Internal method for calculating the p-norm

Description

Internal method for calculating the p-norm

Usage

.p_norm(x, p)

Arguments

x          A matrix
p          A number greater than or equal to 1, or equal to positive infinity

Value

Returns the specified norm of matrix x
The DivExpression class.

Description

This class represents one expression divided by another expression.

Usage

```r
## 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'
to_numeric(object, values)

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

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

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

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

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

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

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

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

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

## S4 method for signature 'DivExpression'

```r
graph_implementation(object, arg_objs, dim, 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.
- **values** A list of arguments to the atom.
- **idx** An index into the atom.
- **arg_objs** A list of linear expressions for each argument.
- **dim** A vector representing the dimensions of the resulting expression.
- **data** A list of additional data required by the atom.

### Methods (by generic)

- `to_numeric(DivExpression)`: Matrix division by a scalar.
- `is_quadratic(DivExpression)`: Is the left-hand expression quadratic and the right-hand expression constant?
- `is_qpwa(DivExpression)`: Is the expression quadratic of piecewise affine?
- `dim_from_args(DivExpression)`: The (row, col) dimensions of the left-hand expression.
- `is_atom_convex(DivExpression)`: Division is convex (affine) in its arguments only if the denominator is constant.
- `is_atom_concave(DivExpression)`: Division is concave (affine) in its arguments only if the denominator is constant.
- `is_atom_log_log_convex(DivExpression)`: Is the atom log-log convex?
- `is_atom_log_log_concave(DivExpression)`: Is the atom log-log concave?
- `is_incr(DivExpression)`: Is the right-hand expression positive?
- `is_decr(DivExpression)`: Is the right-hand expression negative?
- `graph_implementation(DivExpression)`: The graph implementation of the expression.

---

### Description

The `IneqConstraint` class

*The IneqConstraint class*
Usage

```r
## 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

## S4 method for signature 'IneqConstraint'
name(x)

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

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

## S4 method for signature 'IneqConstraint'
expr(object)
```
Arguments

e1, e2  The Expression objects or numeric constants to compare.
x, object  A IneqConstraint object.

Methods (by generic)

• name(IneqConstraint): The string representation of the constraint.
• dim(IneqConstraint): The dimensions of the constrained expression.
• size(IneqConstraint): The size of the constrained expression.
• expr(IneqConstraint): The expression to constrain.
• is_dcp(IneqConstraint): A non-positive constraint is DCP if its argument is convex.
• is_dgp(IneqConstraint): Is the constraint DGP?
• residual(IneqConstraint): The residual of the constraint.
name(x)

## S4 method for signature 'EqConstraint'

dim(x)

## S4 method for signature 'EqConstraint'

size(object)

## S4 method for signature 'EqConstraint'

expr(object)

## S4 method for signature 'EqConstraint'

is_dcp(object)

## S4 method for signature 'EqConstraint'

is_dgp(object)

## S4 method for signature 'EqConstraint'

residual(object)

Arguments

e1, e2 The Expression objects or numeric constants to compare.
x, object A EqConstraint object.

Methods (by generic)

- name(EqConstraint): The string representation of the constraint.
- dim(EqConstraint): The dimensions of the constrained expression.
- size(EqConstraint): The size of the constrained expression.
- expr(EqConstraint): The expression to constrain.
- is_dcp(EqConstraint): Is the constraint DCP?
- is_dgp(EqConstraint): Is the constraint DGP?
- residual(EqConstraint): The residual of the constraint..

Description

The elementwise absolute value.

Usage

## S4 method for signature 'Expression'

abs(x)
Abs-class

Arguments

\( x \)  
An Expression.

Value

An Expression representing the absolute value of the input.

Examples

\[
A <- \text{Variable}(2,2) 
\]
\[
\text{prob} <- \text{Problem(Minimize(sum(abs(A))), list(A <= -2))} 
\]
\[
\text{result} <- \text{solve(prob)} 
\]
\[
\text{result$getValue} 
\]
\[
\text{result$getValue(A)} 
\]

Abs-class

The Abs class.

Description

This class represents the elementwise absolute value.

Usage

\[
\text{Abs}(x) 
\]

## S4 method for signature 'Abs'
\text{to_numeric}(object, values)

## S4 method for signature 'Abs'
\text{allow_complex}(object)

## S4 method for signature 'Abs'
\text{sign_from_args}(object)

## S4 method for signature 'Abs'
\text{is_atom_convex}(object)

## S4 method for signature 'Abs'
\text{is_atom_concave}(object)

## S4 method for signature 'Abs'
\text{is_incr}(object, idx)

## S4 method for signature 'Abs'
\text{is_decr}(object, idx)

## S4 method for signature 'Abs'
\text{is_pwl}(object)
Arguments

- \( x \) An Expression object.
- object An Abs object.
- values A list of arguments to the atom.
- idx An index into the atom.

Methods (by generic)

- to_numeric(Abs): The elementwise absolute value of the input.
- allow_complex(Abs): Does the atom handle complex numbers?
- sign_from_args(Abs): The atom is positive.
- is_atom_convex(Abs): The atom is convex.
- is_atom_concave(Abs): The atom is not concave.
- is_incr(Abs): A logical value indicating whether the atom is weakly increasing.
- is_decr(Abs): A logical value indicating whether the atom is weakly decreasing.
- is_pwl(Abs): Is \( x \) piecewise linear?

Slots

- \( x \) An Expression object.

---

**accepts**

Description

Determine whether the reduction accepts a problem.

Usage

```r
accepts(object, problem)
```

Arguments

- object A Reduction object.
- problem A Problem to check.

Value

A logical value indicating whether the reduction can be applied.
The AffAtom class.

Description

This virtual class represents an affine atomic expression.

Usage

```r
## S4 method for signature 'AffAtom'
allow_complex(object)

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

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

## S4 method for signature 'AffAtom'
is_complex(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_qpwa(object)

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

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

## S4 method for signature 'AffAtom'
is_nsd(object)
```
## `are_args_affine`

### Description

Are the arguments affine?

### Usage

```r
are_args_affine(constraints)
```

### Arguments

- **constraints**
  - A `Constraint` object.

### Value

All the affine arguments in given constraints.

### Arguments

- **object**
  - An `AffAtom` object.
- **idx**
  - An index into the atom.
- **values**
  - A list of numeric values for the arguments

### Methods (by generic)

- `allow_complex(AffAtom)`: Does the atom handle complex numbers?
- `sign_from_args(AffAtom)`: The sign of the atom.
- `is_imag(AffAtom)`: Is the atom imaginary?
- `is_complex(AffAtom)`: Is the atom complex valued?
- `is_atom_convex(AffAtom)`: The atom is convex.
- `is_atom_concave(AffAtom)`: The atom is concave.
- `is_incr(AffAtom)`: The atom is weakly increasing in every argument.
- `is_decr(AffAtom)`: The atom is not weakly decreasing in any argument.
- `is_quadratic(AffAtom)`: Is every argument quadratic?
- `is_qpwa(AffAtom)`: Is every argument quadratic of piecewise affine?
- `is_pwl(AffAtom)`: Is every argument piecewise linear?
- `is_psd(AffAtom)`: Is the atom a positive semidefinite matrix?
- `is_nsd(AffAtom)`: Is the atom a negative semidefinite matrix?
- `.grad(AffAtom)`: Gives the (sub/super)gradient of the atom w.r.t. each variable
Description

This virtual class represents atomic expressions in CVXR.

Usage

```r
## S4 method for signature 'Atom'
name(x)

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

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

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

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

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

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

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

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

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

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

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

## S4 method for signature 'Atom'
is_log_log_convex(object)
```
## S4 method for signature 'Atom'
is_log_log_concave(object)

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

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

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

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

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

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

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

### Arguments

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

### Methods (by generic)

- **name(Atom)**: Returns the string representation of the function call.
- **validate_args(Atom)**: Raises an error if the arguments are invalid.
- **dim(Atom)**: The (row, col) dimensions of the atom.
- **nrow(Atom)**: The number of rows in the atom.
- **ncol(Atom)**: The number of columns in the atom.
- **allow_complex(Atom)**: Does the atom handle complex numbers?
- **is_nonneg(Atom)**: A logical value indicating whether the atom is nonnegative.
- **is_nonpos(Atom)**: A logical value indicating whether the atom is nonpositive.
- **is_imag(Atom)**: A logical value indicating whether the atom is imaginary.
- **is_complex(Atom)**: A logical value indicating whether the atom is complex valued.
• is_convex(Atom): A logical value indicating whether the atom is convex.
• is_concave(Atom): A logical value indicating whether the atom is concave.
• is_log_log_convex(Atom): A logical value indicating whether the atom is log-log convex.
• is_log_log_concave(Atom): A logical value indicating whether the atom is log-log concave.
• canonicalize(Atom): Represent the atom as an affine objective and conic constraints.
• graph_implementation(Atom): The graph implementation of the atom.
• value_impl(Atom): Returns the value of each of the components in an Atom. Returns an empty matrix if it’s an empty atom
• value(Atom): Returns the value of the atom.
• grad(Atom): The (sub/super)-gradient of the atom with respect to each variable.
• domain(Atom): A list of constraints describing the closure of the region where the expression is finite.
• atoms(Atom): Returns a list of the atom types present amongst this atom’s arguments

---

**AxisAtom-class**  
*The AxisAtom class.*

**Description**

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

**Usage**

```r
## S4 method for signature 'AxisAtom'
dim_from_args(object)

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

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

## S4 method for signature 'AxisAtom'
.axis_grad(object, values)

## S4 method for signature 'AxisAtom'
.column_grad(object, value)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An Atom object.</td>
</tr>
<tr>
<td>values</td>
<td>A list of numeric values for the arguments</td>
</tr>
<tr>
<td>value</td>
<td>A numeric value</td>
</tr>
</tbody>
</table>
Methods (by generic)

- `dim_from_args(AxisAtom)`: The dimensions of the atom determined from its arguments.
- `get_data(AxisAtom)`: A list containing `axis` and `keepdims`.
- `validate_args(AxisAtom)`: Check that the new dimensions have the same number of entries as the old.
- `.axis_grad(AxisAtom)`: Gives the (sub/super)gradient of the atom w.r.t. each variable.
- `.column_grad(AxisAtom)`: Gives the (sub/super)gradient of the atom w.r.t. each column variable.

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.
- `keepdims` (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.

BinaryOperator-class

The `BinaryOperator` class.

Description

This base class represents expressions involving binary operators.

Usage

```r
## S4 method for signature 'BinaryOperator'
name(x)

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

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

## S4 method for signature 'BinaryOperator'
is_imag(object)

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

Arguments

- `x`, `object` A `BinaryOperator` object.
- `values` A list of arguments to the atom.
Methods (by generic)

- `name(BinaryOperator)`: Returns the name of the BinaryOperator object.
- `to_numeric(BinaryOperator)`: Apply the binary operator to the values.
- `sign_from_args(BinaryOperator)`: Default rule for multiplication.
- `is_imag(BinaryOperator)`: Is the expression imaginary?
- `is_complex(BinaryOperator)`: Is the expression complex valued?

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.

### Description

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

### Usage

```r
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

```r
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
```
CallbackParam-class

The CallbackParam class.

Description

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

Usage

CallbackParam(callback, dim = NULL, ...)

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

Arguments

callback A callback function that generates the parameter value.
dim The dimensions of the parameter.
... Additional attribute arguments. See Leaf for details.
object A CallbackParam object.

Slots

callback A callback function that generates the parameter value.
dim The dimensions of the parameter.

Examples

x <- Variable(2)
fun <- function() { value(x) }
y <- CallbackParam(fun, dim(x), nonneg = TRUE)
get_data(y)

Canonical-class

The Canonical class.

Description

This virtual class represents a canonical expression.
## Usage

### S4 method for signature 'Canonical'

- expr(object)
- id(object)
- canonical_form(object)
- variables(object)
- parameters(object)
- constants(object)
- atoms(object)
- get_data(object)

### Arguments

- object: A Canonical object.

### Methods (by generic)

- expr(Canonical): The expression associated with the input.
- id(Canonical): The unique ID of the canonical expression.
- canonical_form(Canonical): The graph implementation of the input.
- variables(Canonical): List of Variable objects in the expression.
- parameters(Canonical): List of Parameter objects in the expression.
- constants(Canonical): List of Constant objects in the expression.
- atoms(Canonical): List of Atom objects in the expression.
- get_data(Canonical): Information needed to reconstruct the expression aside from its arguments.
The Canonicalization class.

Description

This class represents a canonicalization reduction.

Usage

```r
## S4 method for signature 'Canonicalization,Problem'
perform(object, problem)

## S4 method for signature 'Canonicalization,Solution,InverseData'
invert(object, solution, inverse_data)

## S4 method for signature 'Canonicalization'
canonicalize_tree(object, expr)

## S4 method for signature 'Canonicalization'
canonicalize_expr(object, expr, args)
```

Arguments

- **object**: A `Canonicalization` object.
- **problem**: A `Problem` object.
- **solution**: A `Solution` to a problem that generated the inverse data.
- **inverse_data**: An `InverseData` object that contains the data encoding the original problem.
- **expr**: An `Expression` object.
- **args**: List of arguments to canonicalize the expression.

Methods (by generic)

- `perform(object = Canonicalization, problem = Problem)`: Recursively canonicalize the objective and every constraint.
- `invert(object = Canonicalization, solution = Solution, inverse_data = InverseData)`: Performs the reduction on a problem and returns an equivalent problem.
- `canonicalize_tree(Canonicalization)`: Recursively canonicalize an Expression.
- `canonicalize_expr(Canonicalization)`: Canonicalize an expression, w.r.t. canonicalized arguments.
### canonicalize

**Canonicalize**

Computes the graph implementation of a canonical expression.

#### Usage

```r
canonicalize(object)

canonical_form(object)
```

#### Arguments

- `object`: A Canonical object.

#### Value

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

---

### CBC_CONIC-class

**An interface to the CBC solver**

#### Description

An interface to the CBC solver

#### Usage

```r
CBC_CONIC()

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

## S4 method for signature 'CBC_CONIC'
status_map(solver, status)

## S4 method for signature 'CBC_CONIC'
status_map_mip(solver, status)

## S4 method for signature 'CBC_CONIC'
status_map_lp(solver, status)

## S4 method for signature 'CBC_CONIC'
name(x)
```
## S4 method for signature 'CBC_CONIC'
import_solver(solver)

## S4 method for signature 'CBC_CONIC,Problem'
accepts(object, problem)

## S4 method for signature 'CBC_CONIC,Problem'
perform(object, problem)

## S4 method for signature 'CBC_CONIC,list,list'
invert(object, solution, inverse_data)

## S4 method for signature 'CBC_CONIC'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

### Arguments

- **solver, object, x**
  - A **CBC_CONIC** object.
- **status**
  - A status code returned by the solver.
- **problem**
  - A **Problem** object.
- **solution**
  - The raw solution returned by the solver.
- **inverse_data**
  - A list containing data necessary for the inversion.
- **data**
  - Data generated via an apply call.
- **warm_start**
  - A boolean of whether to warm start the solver.
- **verbose**
  - A boolean of whether to enable solver verbosity.
- **feastol**
  - The feasible tolerance.
- **reltol**
  - The relative tolerance.
- **abstol**
  - The absolute tolerance.
- **num_iter**
  - The maximum number of iterations.
- **solver_opts**
  - A list of Solver specific options.
- **solver_cache**
  - Cache for the solver.
**Methods (by generic)**

- `mip_capable(CBC_CONIC)`: Can the solver handle mixed-integer programs?
- `status_map(CBC_CONIC)`: Converts status returned by the CBC solver to its respective CVXPY status.
- `status_map_mip(CBC_CONIC)`: Converts status returned by the CBC solver to its respective CVXPY status for mixed integer problems.
- `status_map_lp(CBC_CONIC)`: Converts status returned by the CBC solver to its respective CVXPY status for linear problems.
- `name(CBC_CONIC)`: Returns the name of the solver
- `import_solver(CBC_CONIC)`: Imports the solver
- `accepts(object = CBC_CONIC, problem = Problem)`: Can CBC_CONIC solve the problem?
- `perform(object = CBC_CONIC, problem = Problem)`: Returns a new problem and data for inverting the new solution.
- `invert(object = CBC_CONIC, solution = list, inverse_data = list)`: Returns the solution to the original problem given the inverse_data.
- `solve_via_data(CBC_CONIC)`: Solve a problem represented by data returned from apply.

---

**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
Chain-class

The Chain class.

Description

This class represents a reduction that replaces symbolic parameters with their constraint values.

Usage

```r
## S4 method for signature 'Chain'
as.character(x)
## S4 method for signature 'Chain,Problem'
accepts(object, problem)
## S4 method for signature 'Chain,Problem'
perform(object, problem)
## S4 method for signature 'Chain,SolutionORList,list'
invert(object, solution, inverse_data)
```

Arguments

- `x`, object: A Chain object.
- `problem`: A Problem object to check.
- `solution`: A Solution or list.
- `inverse_data`: A list that contains the data encoding the original problem.
**Methods (by generic)**

- **accepts(object = Chain, problem = Problem):** A problem is accepted if the sequence of reductions is valid. In particular, the i-th reduction must accept the output of the i-1th reduction, with the first reduction (self.reductions[0]) in the sequence taking as input the supplied problem.

- **perform(object = Chain, problem = Problem):** Applies the chain to a problem and returns an equivalent problem.

- **invert(object = Chain, solution = SolutionORList, inverse_data = list):** Performs the reduction on a problem and returns an equivalent problem.

---

**complex-atoms Complex Numbers**

**Description**

Basic atoms that support complex arithmetic.

**Usage**

```r
## S4 method for signature 'Expression'
Re(z)

## S4 method for signature 'Expression'
Im(z)

## S4 method for signature 'Expression'
Conj(z)
```

**Arguments**

- `z` An Expression object.

**Value**

An Expression object that represents the real, imaginary, or complex conjugate.
complex-methods  Complex Properties

Description
Determine if an expression is real, imaginary, or complex.

Usage
is_real(object)

is_imag(object)

is_complex(object)

Arguments
object  An Expression object.

Value
A logical value.

Complex2Real-class  Lifts complex numbers to a real representation.

Description
This reduction takes in a complex problem and returns an equivalent real problem.

Usage
## S4 method for signature 'Complex2Real,Problem'
accepts(object, problem)

## S4 method for signature 'Complex2Real,Problem'
perform(object, problem)

## S4 method for signature 'Complex2Real,Solution,InverseData'
invert(object, solution, inverse_data)

Arguments
object  A Complex2Real object.

problem  A Problem object.

solution  A Solution object to invert.

inverse_data  A InverseData object containing data necessary for the inversion.
Complex2Real.abs_canon

Methods (by generic)

• accepts(object = Complex2Real, problem = Problem): Checks whether or not the problem involves any complex numbers.

• perform(object = Complex2Real, problem = Problem): Converts a Complex problem into a Real one.

• invert(object = Complex2Real, solution = Solution, inverse_data = InverseData): Returns a solution to the original problem given the inverse data.

Complex2Real.abs_canon

Complex canonicalizer for the absolute value atom

Description

Complex canonicalizer for the absolute value atom

Usage

Complex2Real.abs_canon(expr, real_args, imag_args, real2imag)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An Expression object</td>
</tr>
<tr>
<td>real_args</td>
<td>A list of Constraint objects for the real part of the expression</td>
</tr>
<tr>
<td>imag_args</td>
<td>A list of Constraint objects for the imaginary part of the expression</td>
</tr>
<tr>
<td>real2imag</td>
<td>A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.</td>
</tr>
</tbody>
</table>

Value

A canonicalization of the absolute value atom of a complex expression, where the returned variables are its real and imaginary components parsed out.
Complex2Real.add

Helper function to sum arguments.

Description

Helper function to sum arguments.

Usage

Complex2Real.add(lh_arg, rh_arg, neg = FALSE)

Arguments

lh_arg The arguments for the left-hand side
rh_arg The arguments for the right-hand side
neg Whether to negate the right hand side

Complex2Real.at_least_2D

Upcast 0D and 1D to 2D.

Description

Upcast 0D and 1D to 2D.

Usage

Complex2Real.at_least_2D(expr)

Arguments

expr An Expression object

Value

An expression of dimension at least 2.
Complex2Real.binary_canon

*Complex canonicalizer for the binary atom*

**Description**

Complex canonicalizer for the binary atom

**Usage**

Complex2Real.binary_canon(expr, real_args, imag_args, real2imag)

**Arguments**

- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

**Value**

A canonicalization of a binary atom, where the returned variables are the real component and the imaginary component.

Complex2Real.canonicalize_expr

*Canonicalizes a Complex Expression*

**Description**

Canonicalizes a Complex Expression

**Usage**

Complex2Real.canonicalize_expr(expr, real_args, imag_args, real2imag, leaf_map)

**Arguments**

- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.
- **leaf_map**: A map that consists of a tree representation of the overall expression
**Value**

A list of the parsed out real and imaginary components of the expression at hand.

---

**Complex2Real.canonicalize_tree**

*Recursively Canonicalizes a Complex Expression.*

---

**Description**

Recursively Canonicalizes a Complex Expression.

**Usage**

Complex2Real.canonicalize_tree(expr, real2imag, leaf_map)

**Arguments**

*expr*  
An Expression object.

*real2imag*  
A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

*leaf_map*  
A map that consists of a tree representation of the expression.

**Value**

A list of the parsed out real and imaginary components of the expression that was constructed by performing the canonicalization of each leaf in the tree.

---

**Complex2Realconj_canon**

*Complex canonicalizer for the conjugate atom*

---

**Description**

Complex canonicalizer for the conjugate atom

**Usage**

Complex2Real.conj_canon(expr, real_args, imag_args, real2imag)

**Arguments**

*expr*  
An Expression object

*real_args*  
A list of Constraint objects for the real part of the expression

*imag_args*  
A list of Constraint objects for the imaginary part of the expression

*real2imag*  
A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.
**Complex2Real.constant_canon**

*Description*

Complex canonicalizer for the constant atom

*Usage*

```python
Complex2Real.constant_canon(expr, real_args, imag_args, real2imag)
```

**Arguments**

- `expr`: An Expression object
- `real_args`: A list of Constraint objects for the real part of the expression
- `imag_args`: A list of Constraint objects for the imaginary part of the expression
- `real2imag`: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

**Value**

A canonicalization of a constant atom, where the returned variables are the real component and the imaginary component in the Constant atom.

---

**Complex2Real.hermitian_canon**

*Description*

Complex canonicalizer for the hermitian atom

*Usage*

```python
Complex2Real.hermitian_canon(expr, real_args, imag_args, real2imag)
```

**Arguments**

- `expr`: An Expression object
- `real_args`: A list of Constraint objects for the real part of the expression
- `imag_args`: A list of Constraint objects for the imaginary part of the expression
- `real2imag`: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

**Value**

A canonicalization of a hermitian atom, where the returned variables are the real component and the imaginary component in the Hermitian atom.
Complex2Real.imag_canon

Arguments

- expr: An Expression object
- real_args: A list of Constraint objects for the real part of the expression
- imag_args: A list of Constraint objects for the imaginary part of the expression
- real2imag: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

Value

A canonicalization of a hermitian matrix atom, where the returned variables are the real component and the imaginary component.

Description

Complex canonicalizer for the imaginary atom

Usage

Complex2Real.imag_canon(expr, real_args, imag_args, real2imag)

Arguments

- expr: An Expression object
- real_args: A list of Constraint objects for the real part of the expression
- imag_args: A list of Constraint objects for the imaginary part of the expression
- real2imag: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

Value

A canonicalization of an imaginary atom, where the returned variables are the imaginary component and NULL for the real component.
Complex2Real.join

**Helper function to combine arguments.**

**Description**
Helper function to combine arguments.

**Usage**
Complex2Real.join(expr, lh_arg, rh_arg)

**Arguments**
- **expr**: An Expression object
- **lh_arg**: The arguments for the left-hand side
- **rh_arg**: The arguments for the right-hand side

**Value**
A joined expression of both left and right expressions

Complex2Real.lambda_sum_largest_canon

_Complex canonicalizer for the largest sum atom_

**Description**
Complex canonicalizer for the largest sum atom

**Usage**
Complex2Real.lambda_sum_largest_canon(expr, real_args, imag_args, real2imag)

**Arguments**
- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

**Value**
A canonicalization of the largest sum atom, where the returned variables are the real component and the imaginary component.
Complex2Real.matrix_frac_canon

*Complex canonicalizer for the matrix fraction atom*

**Description**
Complex canonicalizer for the matrix fraction atom

**Usage**
Complex2Real.matrix_frac_canon(expr, real_args, imag_args, real2imag)

**Arguments**
- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

**Value**
A canonicalization of a matrix atom, where the returned variables are converted to real variables.

Complex2Real.nonpos_canon

*Complex canonicalizer for the non-positive atom*

**Description**
Complex canonicalizer for the non-positive atom

**Usage**
Complex2Real.nonpos_canon(expr, real_args, imag_args, real2imag)

**Arguments**
- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.
Value

A canonicalization of a non positive atom, where the returned variables are the real component and the imaginary component.

---

Complex2Real.nuc_canon

*Complex canonicalizer for the nuclear norm atom*

---

Description

Complex canonicalizer for the nuclear norm atom

Usage

Complex2Real.nuc_canon(expr, real_args, imag_args, real2imag)

Arguments

- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

Value

A canonicalization of a nuclear norm matrix atom, where the returned variables are the real component and the imaginary component.

---

Complex2Real.param_canon

*Complex canonicalizer for the parameter matrix atom*

---

Description

Complex canonicalizer for the parameter matrix atom

Usage

Complex2Real.param_canon(expr, real_args, imag_args, real2imag)
Complex canonicalizer for the p norm atom

Arguments

expr An Expression object
real_args A list of Constraint objects for the real part of the expression
imag_args A list of Constraint objects for the imaginary part of the expression
real2imag A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

Value

A canonicalization of a parameter matrix atom, where the returned variables are the real component and the imaginary component.
**Complex2Real.psd_canon**

Complex canonicalizer for the positive semidefinite atom

**Description**

Complex canonicalizer for the positive semidefinite atom

**Usage**

Complex2Real.psd_canon(expr, real_args, imag_args, real2imag)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An Expression object</td>
</tr>
<tr>
<td>real_args</td>
<td>A list of Constraint objects for the real part of the expression</td>
</tr>
<tr>
<td>imag_args</td>
<td>A list of Constraint objects for the imaginary part of the expression</td>
</tr>
<tr>
<td>real2imag</td>
<td>A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.</td>
</tr>
</tbody>
</table>

**Value**

A canonicalization of a positive semidefinite atom, where the returned variables are the real component and the NULL imaginary component.

---

**Complex2Real.quad_canon**

Complex canonicalizer for the quadratic atom

**Description**

Complex canonicalizer for the quadratic atom

**Usage**

Complex2Real.quad_canon(expr, real_args, imag_args, real2imag)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An Expression object</td>
</tr>
<tr>
<td>real_args</td>
<td>A list of Constraint objects for the real part of the expression</td>
</tr>
<tr>
<td>imag_args</td>
<td>A list of Constraint objects for the imaginary part of the expression</td>
</tr>
<tr>
<td>real2imag</td>
<td>A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.</td>
</tr>
</tbody>
</table>
**Complex2Real.real_canon**

Description

A canonicalization of a quadratic atom, where the returned variables are the real component and the imaginary component as NULL.

**Complex2Real.quad_over_lin_canon**

*Complex canonicalizer for the quadratic over linear term atom*

Description

Complex canonicalizer for the quadratic over linear term atom

Usage

Complex2Real.quad_over_lin_canon(expr, real_args, imag_args, real2imag)

Arguments

- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

Value

A canonicalization of a quadratic over a linear term atom, where the returned variables are the real component and the imaginary component.

**Complex2Real.real_canon**

*Complex canonicalizer for the real atom*

Description

Complex canonicalizer for the real atom

Usage

Complex2Real.real_canon(expr, real_args, imag_args, real2imag)
Complex canonicalizer for the separable atom

Description
Complex canonicalizer for the separable atom

Usage
Complex2Real.separable_canon(expr, real_args, imag_args, real2imag)

Arguments
expr
An Expression object
real_args
A list of Constraint objects for the real part of the expression
imag_args
A list of Constraint objects for the imaginary part of the expression
real2imag
A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

Value
A canonicalization of a separable atom, where the returned variables are its real and imaginary components parsed out.
Complex2Real.soc_canon

*Complex canonicalizer for the SOC atom*

**Description**

Complex canonicalizer for the SOC atom

**Usage**

Complex2Real.soc_canon(expr, real_args, imag_args, real2imag)

**Arguments**

- **expr**  
  An Expression object
- **real_args**  
  A list of Constraint objects for the real part of the expression
- **imag_args**  
  A list of Constraint objects for the imaginary part of the expression
- **real2imag**  
  A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

**Value**

A canonicalization of a SOC atom, where the returned variables are the real component and the NULL imaginary component.

Complex2Real.variable_canon

*Complex canonicalizer for the variable atom*

**Description**

Complex canonicalizer for the variable atom

**Usage**

Complex2Real.variable_canon(expr, real_args, imag_args, real2imag)

**Arguments**

- **expr**  
  An Expression object
- **real_args**  
  A list of Constraint objects for the real part of the expression
- **imag_args**  
  A list of Constraint objects for the imaginary part of the expression
- **real2imag**  
  A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.
Complex2Real.zero_canon

Value
A canonicalization of a variable atom, where the returned variables are the real component and the NULL imaginary component.

Description
Complex canonicalizer for the zero atom

Usage
Complex2Real.zero_canon(expr, real_args, imag_args, real2imag)

Arguments
- **expr**: An Expression object
- **real_args**: A list of Constraint objects for the real part of the expression
- **imag_args**: A list of Constraint objects for the imaginary part of the expression
- **real2imag**: A list mapping the ID of the real part of a complex expression to the ID of its imaginary part.

Value
A canonicalization of a zero atom, where the returned variables are the real component and the imaginary component.

cone-methods

Description
The number of elementwise cones or a list of the sizes of the elementwise cones.

Usage
- **num_cones(object)**
- **cone_sizes(object)**

Arguments
- **object**: An SOCAxis object.
Value

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

ConeDims-class

Summary of cone dimensions present in constraints.

Description

Constraints must be formatted as dictionary that maps from constraint type to a list of constraints of that type.

Details

Attributes — zero : int The dimension of the zero cone. nonpos : int The dimension of the non-positive cone. exp : int The dimension of the exponential cone. soc : list of int A list of the second-order cone dimensions. psd : list of int A list of the positive semidefinite cone dimensions, where the dimension of the PSD cone of k by k matrices is k.

ConeMatrixStuffing-class

Construct Matrices for Linear Cone Problems

Description

Linear cone problems are assumed to have a linear objective and cone constraints, which may have zero or more arguments, all of which must be affine.

Usage

## S4 method for signature `ConeMatrixStuffing,Problem`
accepts(object, problem)

## S4 method for signature `ConeMatrixStuffing,Problem,CoeffExtractor`
stuffed_objective(object, problem, extractor)

Arguments

object A ConeMatrixStuffing object.
problem A Problem object.
extractor Used to extract the affine coefficients of the objective.

Details

minimize c^T x subject to cone_constr1(A_1*x + b_1, ...) ... cone_constrK(A_K*x + b_K, ...)
Methods (by generic)

- accepts(object = ConeMatrixStuffing, problem = Problem): Is the solver accepted?
- stuffed_objective( object = ConeMatrixStuffing, problem = Problem, extractor = CoeffExtractor ): Returns a list of the stuffed matrices

ConicSolver-class

The ConicSolver class.

Description

Conic solver class with reduction semantics.

Usage

```r
## S4 method for signature 'ConicSolver,Problem'
accepts(object, problem)

## S4 method for signature 'ConicSolver'
reduction_format_constr(object, problem, constr, exp_cone_order)

## S4 method for signature 'ConicSolver'
group_coeff_offset(object, problem, constraints, exp_cone_order)

## S4 method for signature 'ConicSolver,Solution,InverseData'
invert(object, solution, inverse_data)
```

Arguments

- object: A ConicSolver object.
- problem: A Problem object.
- constr: A Constraint to format.
- exp_cone_order: A list indicating how the exponential cone arguments are ordered.
- constraints: A list of Constraint objects.
- solution: A Solution object to invert.
- inverse_data: A InverseData object containing data necessary for the inversion.

Methods (by generic)

- accepts(object = ConicSolver, problem = Problem): Can the problem be solved with a conic solver?
- reduction_format_constr(ConicSolver): Return a list representing a cone program whose problem data tensors will yield the coefficient "A" and offset "b" for the respective constraints: Linear Equations: \( Ax = b \), Linear inequalities: \( Ax \leq b \), Second order cone: \( Ax \leq_{SOC} b \), Exponential cone: \( Ax \leq_{EXP} b \), Semidefinite cone: \( Ax \leq_{SOP} b \).
ConicSolver.get_spacing_matrix

Returns a sparse matrix that spaces out an expression.

Description

Returns a sparse matrix that spaces out an expression.

Usage

ConicSolver.get_spacing_matrix(dim, spacing, offset)

Arguments

dim A vector outlining the dimensions of the matrix.
spacing An int of the number of rows between the start of each non-zero block.
offset An int of the number of zeros at the beginning of the matrix.

Value

A sparse matrix that spaces out an expression

ConicSolver.get_coeff_offset

Return the coefficient and offset in $Ax + b$.

Description

Return the coefficient and offset in $Ax + b$.

Usage

ConicSolver.get_coeff_offset(expr)

Arguments

expr An Expression object.

Value

The coefficient and offset in $Ax + b$.

ConicSolver.get_spacing_matrix

Returns a sparse matrix that spaces out an expression.

Description

Returns a sparse matrix that spaces out an expression.

Usage

ConicSolver.get_spacing_matrix(dim, spacing, offset)

Arguments

dim A vector outlining the dimensions of the matrix.
spacing An int of the number of rows between the start of each non-zero block.
offset An int of the number of zeros at the beginning of the matrix.

Value

A sparse matrix that spaces out an expression

• group_coeff_offset(ConicSolver): Combine the constraints into a single matrix, offset.
• invert(object = ConicSolver, solution = Solution, inverse_data = InverseData): Returns the solution to the original problem given the inverse_data.
Description

This class represents the complex conjugate of an expression.

Usage

Conjugate(expr)

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

## S4 method for signature 'Conjugate'
dim_from_args(object)

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

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

## S4 method for signature 'Conjugate'
is_symmetric(object)

## S4 method for signature 'Conjugate'
is_hermitian(object)
```

Arguments

- `expr`: An Expression or R numeric data.
- `object`: A Conjugate object.
- `values`: A list of arguments to the atom.
- `idx`: An index into the atom.

Methods (by generic)

- `to_numeric(Conjugate)`: Elementwise complex conjugate of the constant.
- `dim_from_args(Conjugate)`: The (row, col) dimensions of the expression.
- `is_incr(Conjugate)`: Is the composition weakly increasing in argument idx?
- `is_decr(Conjugate)`: Is the composition weakly decreasing in argument idx?
- `is_symmetric(Conjugate)`: Is the expression symmetric?
- `is_hermitian(Conjugate)`: Is the expression hermitian?
**Slots**

- `expr` An Expression or R numeric data.

---

**Constant-class**

*The Constant class.*

---

**Description**

This class represents a constant.

Coerce an R object or expression into the `Constant` class.

**Usage**

```r
Constant(value)
```
is_imag(object)
## S4 method for signature 'Constant'
is_complex(object)
## S4 method for signature 'Constant'
is_symmetric(object)
## S4 method for signature 'Constant'
is_hermitian(object)
## S4 method for signature 'Constant'
is_psd(object)
## S4 method for signature 'Constant'
is_nsd(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)**

- `name(Constant)`: The name of the constant.
- `constants(Constant)`: Returns itself as a constant.
- `value(Constant)`: The value of the constant.
- `is_pos(Constant)`: A logical value indicating whether all elements of the constant are positive.
- `grad(Constant)`: An empty list since the gradient of a constant is zero.
- `dim(Constant)`: The `c(row, col)` dimensions of the constant.
- `canonicalize(Constant)`: The canonical form of the constant.
- `is_nonneg(Constant)`: A logical value indicating whether all elements of the constant are non-negative.
- `is_nonpos(Constant)`: A logical value indicating whether all elements of the constant are non-positive.
- `is_imag(Constant)`: A logical value indicating whether the constant is imaginary.
- `is_complex(Constant)`: A logical value indicating whether the constant is complex-valued.
ConstantSolver-class

- is_symmetric(Constant): A logical value indicating whether the constant is symmetric.
- is_hermitian(Constant): A logical value indicating whether the constant is a Hermitian matrix.
- is_psd(Constant): A logical value indicating whether the constant is a positive semidefinite matrix.
- is_nsd(Constant): A logical value indicating whether the constant is a negative semidefinite matrix.

Slots

value A numeric element, vector, matrix, or data.frame. Vectors are automatically cast into a matrix column.
sparse (Internal) A logical value indicating whether the value is a sparse matrix.
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_nonneg(y)
size(y)
as.Constant(y)

ConstantSolver-class  The ConstantSolver class.

Description

The ConstantSolver class.

Usage

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

## S4 method for signature 'ConstantSolver,Problem'
accepts(object, problem)

## S4 method for signature 'ConstantSolver,Problem'
perform(object, problem)

## S4 method for signature 'ConstantSolver,Solution,list'
invert(object, solution, inverse_data)
## S4 method for signature 'ConstantSolver'
name(x)

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

## S4 method for signature 'ConstantSolver'
is_installed(solver)

## S4 method for signature 'ConstantSolver'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

## S4 method for signature 'ConstantSolver,ANY'
reduction_solve(object, problem, warm_start, verbose, solver_opts)

### Arguments

- `solver`, `object`, `x`  
  A `ConstantSolver` object.
- `problem`  
  A `Problem` object.
- `solution`  
  A `Solution` object to invert.
- `inverse_data`  
  A list containing data necessary for the inversion.
- `data`  
  Data for the solver.
- `warm_start`  
  A boolean of whether to warm start the solver.
- `verbose`  
  A boolean of whether to enable solver verbosity.
- `feastol`  
  The feasible tolerance.
- `reltol`  
  The relative tolerance.
- `abstol`  
  The absolute tolerance.
- `num_iter`  
  The maximum number of iterations.
- `solver_opts`  
  A list of Solver specific options.
- `solver_cache`  
  Cache for the solver.
Methods (by generic)

- `mip_capable(ConstantSolver)`: Can the solver handle mixed-integer programs?
- `accepts(object = ConstantSolver, problem = Problem)`: Is the solver capable of solving the problem?
- `perform(object = ConstantSolver, problem = Problem)`: Returns a list of the ConstantSolver, Problem, and an empty list.
- `invert(object = ConstantSolver, solution = Solution, inverse_data = list)`: Returns the solution.
- `name(ConstantSolver)`: Returns the name of the solver.
- `import_solver(ConstantSolver)`: Imports the solver.
- `is_installed(ConstantSolver)`: Is the solver installed?
- `solve_via_data(ConstantSolver)`: Solve a problem represented by data returned from apply.
- `reduction_solve(object = ConstantSolver, problem = ANY)`: Solve the problem and return a Solution object.

---

**Constraint-class**

### The Constraint class.

#### Description

This virtual class represents a mathematical constraint.

#### Usage

```r
## S4 method for signature 'Constraint'
as.character(x)

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

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

## S4 method for signature 'Constraint'
is_real(object)

## S4 method for signature 'Constraint'
is_imag(object)

## S4 method for signature 'Constraint'
is_complex(object)

## S4 method for signature 'Constraint'
```
is_dcp(object)
## S4 method for signature 'Constraint'
is_dgp(object)
## S4 method for signature 'Constraint'
residual(object)
## S4 method for signature 'Constraint'
vViolation(object)
## S4 method for signature 'Constraint'
constr_value(object, tolerance = 1e-08)
## S4 method for signature 'Constraint'
get_data(object)
## S4 method for signature 'Constraint'
dual_value(object)
## S4 replacement method for signature 'Constraint'
dual_value(object) <- value
## S4 method for signature 'ZeroConstraint'
size(object)

Arguments

x, object A Constraint object.
tolerance The tolerance for checking if the constraint is violated.
value A numeric scalar, vector, or matrix.

Methods (by generic)

- \texttt{dim(Constraint)}: The dimensions of the constrained expression.
- \texttt{size(Constraint)}: The size of the constrained expression.
- \texttt{is\_real(Constraint)}: Is the constraint real?
- \texttt{is\_imag(Constraint)}: Is the constraint imaginary?
- \texttt{is\_complex(Constraint)}: Is the constraint complex?
- \texttt{is\_dcp(Constraint)}: Is the constraint DCP?
- \texttt{is\_dgp(Constraint)}: Is the constraint DGP?
- \texttt{residual(Constraint)}: The residual of a constraint
- \texttt{violation(Constraint)}: The violation of a constraint.
- \texttt{constr\_value(Constraint)}: The value of a constraint.
- \texttt{get\_data(Constraint)}: Information needed to reconstruct the object aside from the args.
• dual_value(Constraint): The dual values of a constraint.
• dual_value(Constraint) <- value: Replaces the dual values of a constraint.
• size(ZeroConstraint): The size of the constrained expression.

---

**construct_intermediate_chain,Problem,list-method**

*Builds a chain that rewrites a problem into an intermediate representation suitable for numeric reductions.*

---

**Description**

Builds a chain that rewrites a problem into an intermediate representation suitable for numeric reductions.

**Usage**

```r
## S4 method for signature 'Problem,list'
construct_intermediate_chain(problem, candidates, gp = FALSE)
```

**Arguments**

- `problem`: The problem for which to build a chain.
- `candidates`: A list of candidate solvers.
- `gp`: A logical value indicating whether the problem is a geometric program.

**Value**

A `Chain` object that can be used to convert the problem to an intermediate form.

---

**construct_solving_chain**

*Build a reduction chain from a problem to an installed solver.*

---

**Description**

Build a reduction chain from a problem to an installed solver.

**Usage**

```r
construct_solving_chain(problem, candidates)
```

**Arguments**

- `problem`: The problem for which to build a chain.
- `candidates`: A list of candidate solvers.
A SolvingChain that can be used to solve the problem.

**Description**
Checks whether the constraint violation is less than a tolerance.

**Usage**
```
constr_value(object, tolerance = 1e-08)
```

**Arguments**
- `object`: A Constraint object.
- `tolerance`: A numeric scalar representing the absolute tolerance to impose on the violation.

**Value**
A logical value indicating whether the violation is less than the `tolerance`. Raises an error if the residual is `NA`.

---

**conv**

**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

```r
set.seed(129)
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)

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

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

## S4 method for signature 'Conv'
dim_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, dim, 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_obj: A list of linear expressions for each argument.
dim: A vector representing the dimensions of the resulting expression.
data: A list of additional data required by the atom.

Methods (by generic)
- to_numeric(Conv): The convolution of the two values.
- validate_args(Conv): Check both arguments are vectors and the first is a constant.
- dim_from_args(Conv): The dimensions of the atom.
- sign_from_args(Conv): The sign of the atom.
- is_incr(Conv): Is the left-hand expression positive?
- is_decr(Conv): Is the left-hand expression negative?
- graph_implementation(Conv): 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.

Description
An interface for the CPLEX solver

Usage
CPLEX_CONIC()

CPLEX_CONIC()

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

## S4 method for signature 'CPLEX_CONIC'
nname(x)

## S4 method for signature 'CPLEX_CONIC'
accepts(object, problem)
## S4 method for signature 'CPLEX_CONIC'
status_map(solver, status)

## S4 method for signature 'CPLEX_CONIC,Problem'
perform(object, problem)

## S4 method for signature 'CPLEX_CONIC,list,list'
invert(object, solution, inverse_data)

## S4 method for signature 'CPLEX_CONIC'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

### Arguments

- **solver, object, x**
  A `CPLEX_CONIC` object.
- **problem**
  A `Problem` object.
- **status**
  A status code returned by the solver.
- **solution**
  The raw solution returned by the solver.
- **inverse_data**
  A list containing data necessary for the inversion.
- **data**
  Data generated via an apply call.
- **warm_start**
  A boolean of whether to warm start the solver.
- **verbose**
  A boolean of whether to enable solver verbosity.
- **feastol**
  The feasible tolerance on the primal and dual residual.
- **reltol**
  The relative tolerance on the duality gap.
- **abstol**
  The absolute tolerance on the duality gap.
- **num_iter**
  The maximum number of iterations.
- **solver_opts**
  A list of Solver specific options
- **solver_cache**
  Cache for the solver.

### Methods (by generic)

- `mip_capable(CPLEX_CONIC)`: Can the solver handle mixed-integer programs?
• `name(CPLEX_CONIC)`: Returns the name of the solver.
• `import_solver(CPLEX_CONIC)`: Imports the solver.
• `accepts(object = CPLEX_CONIC, problem = Problem)`: Can CPLEX solve the problem?
• `status_map(CPLEX_CONIC)`: Converts status returned by the CPLEX solver to its respective CVXPY status.
• `perform(object = CPLEX_CONIC, problem = Problem)`: Returns a new problem and data for inverting the new solution.
• `invert(object = CPLEX_CONIC, solution = list, inverse_data = list)`: Returns the solution to the original problem given the inverse_data.
• `solve_via_data(CPLEX_CONIC)`: Solve a problem represented by data returned from apply.

---

**Description**

An interface for the CPLEX solver.

**Usage**

```r
CPLEX_QP()
```

```r
## S4 method for signature 'CPLEX_QP'
mip_capable(solver)
```

```r
## S4 method for signature 'CPLEX_QP'
status_map(solver, status)
```

```r
## S4 method for signature 'CPLEX_QP'
name(x)
```

```r
## S4 method for signature 'CPLEX_QP'
import_solver(solver)
```

```r
## S4 method for signature 'CPLEX_QP,list,InverseData'
invert(object, solution = list, inverse_data = list)
```

```r
## S4 method for signature 'CPLEX_QP'
solve_via_data(
    object, 
    data, 
    warm_start, 
    verbose, 
    feastol, 
    reltol, 
```
Arguments

status A status code returned by the solver.
x, object, solver A CPLEX_QP object.
solution The raw solution returned by the solver.
inverse_data A InverseData object containing data necessary for the inversion.
data Data generated via an apply call.
warm_start A boolean of whether to warm start the solver.
verbose A boolean of whether to enable solver verbosity.
feastol The feasible tolerance on the primal and dual residual.
reftol The relative tolerance on the duality gap.
abstol The absolute tolerance on the duality gap.
num_iter The maximum number of iterations.
solver_opts A list of Solver specific options
solver_cache Cache for the solver.

Methods (by generic)

- `mip_capable(CPLEX_QP)`: Can the solver handle mixed-integer programs?
- `status_map(CPLEX_QP)`: Converts status returned by the CPLEX solver to its respective CVXPY status.
- `name(CPLEX_QP)`: Returns the name of the solver.
- `import_solver(CPLEX_QP)`: Imports the solver.
- `invert(object = CPLEX_QP, solution = list, inverse_data = InverseData)`: Returns the solution to the original problem given the inverse_data.
- `solve_via_data(CPLEX_QP)`: Solve a problem represented by data returned from apply.
The CumMax class.

Description

This class represents the cumulative maximum of an expression.

Usage

CumMax(expr, axis = 2)

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

## S4 method for signature 'CumMax'
.grad(object, values)

## S4 method for signature 'CumMax'
column_grad(object, value)

## S4 method for signature 'CumMax'
dim_from_args(object)

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

## S4 method for signature 'CumMax'
.get_data(object)

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

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

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

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

Arguments

expr     An Expression.
axis     A numeric vector indicating the axes along which to apply the function. For a
          2D matrix, 1 indicates rows, 2 indicates columns, and c(1,2) indicates rows
          and columns.
cummax_axis

object A CumMax object.
values A list of numeric values for the arguments
value A numeric value.
idx An index into the atom.

Methods (by generic)

- to_numeric(CumMax): The cumulative maximum along the axis.
- .grad(CumMax): Gives the (sub/super)gradient of the atom w.r.t. each variable
- .column_grad(CumMax): Gives the (sub/super)gradient of the atom w.r.t. each column variable
- dim_from_args(CumMax): The dimensions of the atom determined from its arguments.
- sign_from_args(CumMax): The (is positive, is negative) sign of the atom.
- get_data(CumMax): Returns the axis along which the cumulative max is taken.
- is_atom_convex(CumMax): Is the atom convex?
- is_atom_concave(CumMax): Is the atom concave?
- is_incr(CumMax): Is the atom weakly increasing in the index?
- is_decr(CumMax): Is the atom weakly decreasing in the index?

Slots

expr An Expression.
axis A numeric vector indicating the axes along which to apply the function. For a 2D matrix, 1 indicates rows, 2 indicates columns, and c(1,2) indicates rows and columns.

cummax_axis

Cumulative Maximum

Description

The cumulative maximum, \( \max_{i=1,\ldots,k} x_i \) for \( k = 1,\ldots,n \). When calling cummax, matrices are automatically flattened into column-major order before the max is taken.

Usage

cummax_axis(expr, axis = 2)

## S4 method for signature 'Expression'
cummax(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(cummax(Constant(val)))
value(cummax_axis(Constant(val)))

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

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'
dim_from_args(object)

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

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

## S4 method for signature 'CumSum'
graph_implementation(object, arg_objs, dim, 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 numeric values for the arguments
- **arg_objs** A list of linear expressions for each argument.
- **dim** A vector representing the dimensions of the resulting expression.
- **data** A list of additional data required by the atom.
Methods (by generic)
- `to_numeric(CumSum)`: The cumulative sum of the values along the specified axis.
- `dim_from_args(CumSum)`: The dimensions of the atom.
- `get_data(CumSum)`: Returns the axis along which the cumulative sum is taken.
- `.grad(CumSum)`: Gives the (sub/super)gradient of the atom w.r.t. each variable
- `graph_implementation(CumSum)`: 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)`

```r
## 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**

```r
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.

The curvature of an expression.

#### Usage

```r
curvature(object)
```

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

#### Arguments

- `object` An `Expression` object.

#### Value

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

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

#### Examples

```r
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.

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)

## S4 method for signature 'Atom'

is_atom_log_log_convex(object)

## S4 method for signature 'Atom'

is_atom_log_log_concave(object)

## S4 method for signature 'Atom'

is_atom_log_log_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)
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)
Usage

is_constant(object)

is_affine(object)

is_convex(object)

is_concave(object)

is_quadratic(object)

is_pwl(object)

is_qpwa(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)
CvxAttr2Constr-class

Description

This class represents a reduction that expands convex variable attributes into constraints.

Usage

```r
## S4 method for signature 'CvxAttr2Constr,Problem'
perform(object, problem)
## S4 method for signature 'CvxAttr2Constr,Solution,list'
invert(object, solution, inverse_data)
```

Arguments

- object: A CvxAttr2Constr object.
- problem: A Problem object.
- solution: A Solution to a problem that generated the inverse data.
- inverse_data: The inverse data returned by an invocation to apply.

Methods (by generic)

- `perform(object = CvxAttr2Constr, problem = Problem)`: Expand convex variable attributes to constraints.
- `invert(object = CvxAttr2Constr, solution = Solution, inverse_data = list)`: Performs the reduction on a problem and returns an equivalent problem.

CVXOPT-class

An interface for the CVXOPT solver.

Description

An interface for the CVXOPT solver.

Usage

```r
## S4 method for signature 'CVXOPT'
mip_capable(solver)
## S4 method for signature 'CVXOPT'
status_map(solver, status)
```
## S4 method for signature 'CVXOPT'
name(x)

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

## S4 method for signature 'CVXOPT,Problem'
accepts(object, problem)

## S4 method for signature 'CVXOPT,Problem'
perform(object, problem)

## S4 method for signature 'CVXOPT,list,list'
invert(object, solution, inverse_data)

## S4 method for signature 'CVXOPT'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

### Arguments

- **solver, object, x**
  - A CVXOPT object.
- **status**
  - A status code returned by the solver.
- **problem**
  - A Problem object.
- **solution**
  - The raw solution returned by the solver.
- **inverse_data**
  - A list containing data necessary for the inversion.
- **data**
  - Data generated via an apply call.
- **warm_start**
  - A boolean of whether to warm start the solver.
- **verbose**
  - A boolean of whether to enable solver verbosity.
- **feastol**
  - The feasible tolerance on the primal and dual residual.
- **realtol**
  - The relative tolerance on the duality gap.
- **abstol**
  - The absolute tolerance on the duality gap.
- **num_iter**
  - The maximum number of iterations.
- **solver_opts**
  - A list of Solver specific options
- **solver_cache**
  - Cache for the solver.
cvxr_norm

Methods (by generic)

- mip_capable(CVXOPT): Can the solver handle mixed-integer programs?
- status_map(CVXOPT): Converts status returned by the CVXOPT solver to its respective CVXPY status.
- name(CVXOPT): Returns the name of the solver.
- import_solver(CVXOPT): Imports the solver.
- accepts(object = CVXOPT, problem = Problem): Can CVXOPT solve the problem?
- perform(object = CVXOPT, problem = Problem): Returns a new problem and data for inverting the new solution.
- invert(object = CVXOPT, solution = list, inverse_data = list): Returns the solution to the original problem given the inverse_data.
- solve_via_data(CVXOPT): Solve a problem represented by data returned from apply.

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 \( p = 2 \), \( \text{axis} = \text{NA} \), and \( x \) is a matrix, this returns the maximum singular value.

Usage

cvxr_norm(x, p = 2, axis = NA_real_, keepdims = FALSE)

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 \( p = 2 \).
- \( \text{axis} \): (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.
- \( \text{keepdims} \): (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an \( n \times 1 \) column vector. The default is FALSE.

Value

An Expression representing the norm.

See Also

- norm
Dcp2Cone-class  
Reduce DCP Problem to Conic Form

Description

This reduction takes as input (minimization) DCP problems and converts them into problems with affine objectives and conic constraints whose arguments are affine.

Usage

```r
## S4 method for signature 'Dcp2Cone,Problem'
accepts(object, problem)
## S4 method for signature 'Dcp2Cone,Problem'
perform(object, problem)
```

Arguments

- object  
  A Dcp2Cone object.
- problem  
  A Problem object.

Methods (by generic)

- accepts(object = Dcp2Cone, problem = Problem): A problem is accepted if it is a minimization and is DCP.
- perform(object = Dcp2Cone, problem = Problem): Converts a DCP problem to a conic form.

Dcp2Cone.entr_canon  
Dcp2Cone canonicalizer for the entropy atom

Description

Dcp2Cone canonicalizer for the entropy atom

Usage

```r
Dcp2Cone.entr_canon(expr, args)
```

Arguments

- expr  
  An Expression object
- args  
  A list of Constraint objects
Value

A cone program constructed from an entropy atom where the objective function is just the variable t with an ExpCone constraint.

Description

Dcp2Cone canonicalizer for the exponential atom

Usage

Dcp2Cone.exp_canon(expr, args)

Arguments

expr An Expression object
args A list of Constraint objects

Value

A cone program constructed from an exponential atom where the objective function is the variable t with an ExpCone constraint.

Description

Dcp2Cone canonicalizer for the geometric mean atom

Usage

Dcp2Cone.geo_mean_canon(expr, args)

Arguments

expr An Expression object
args A list of Constraint objects

Value

A cone program constructed from a geometric mean atom where the objective function is the variable t with geometric mean constraints
**Dcp2Cone.huber_canon**  
*Dcp2Cone canonicalizer for the huber atom*

**Description**

Dcp2Cone canonicalizer for the huber atom

**Usage**

Dcp2Cone.huber_canon(expr, args)

**Arguments**

- expr: An Expression object
- args: A list of Constraint objects

**Value**

A cone program constructed from a huber atom where the objective function is the variable t with square and absolute constraints

---

**Dcp2Cone.indicator_canon**  
*Dcp2Cone canonicalizer for the indicator atom*

**Description**

Dcp2Cone canonicalizer for the indicator atom

**Usage**

Dcp2Cone.indicator_canon(expr, args)

**Arguments**

- expr: An Expression object
- args: A list of Constraint objects

**Value**

A cone program constructed from an indicator atom and where 0 is the objective function with the given constraints in the function.
**Dcp2Cone.kl_div_canon**  
*Dcp2Cone canonicalizer for the KL Divergence atom*

**Description**
Dcp2Cone canonicalizer for the KL Divergence atom

**Usage**
Dcp2Cone.kl_div_canon(expr, args)

**Arguments**
- **expr**  
  An Expression object
- **args**  
  A list of Constraint objects

**Value**
A cone program constructed from a KL divergence atom where t is the objective function with the ExpCone constraints.

---

**Dcp2Cone.lambda_max_canon**  
*Dcp2Cone canonicalizer for the lambda maximization atom*

**Description**
Dcp2Cone canonicalizer for the lambda maximization atom

**Usage**
Dcp2Cone.lambda_max_canon(expr, args)

**Arguments**
- **expr**  
  An Expression object
- **args**  
  A list of Constraint objects

**Value**
A cone program constructed from a lambda maximization atom where t is the objective function and a PSD constraint and a constraint requiring I*t to be symmetric.
Dcp2Cone.lambda_sum_largest_canon

*Description*

Dcp2Cone canonicalizer for the largest lambda sum atom

*Usage*

Dcp2Cone.lambda_sum_largest_canon(expr, args)

*Arguments*

<table>
<thead>
<tr>
<th>expr</th>
<th>An Expression object</th>
</tr>
</thead>
<tbody>
<tr>
<td>args</td>
<td>A list of Constraint objects</td>
</tr>
</tbody>
</table>

*Value*

A cone program constructed from a lambda sum of the k largest elements atom where k*t + trace(Z) is the objective function. t denotes the variable subject to constraints and Z is a PSD matrix variable whose dimensions consist of the length of the vector at hand. The constraints require the the diagonal matrix of the vector to be symmetric and PSD.

Dcp2Cone.log1p_canon

*Description*

Dcp2Cone canonicalizer for the log(1p) atom

*Usage*

Dcp2Cone.log1p_canon(expr, args)

*Arguments*

<table>
<thead>
<tr>
<th>expr</th>
<th>An Expression object</th>
</tr>
</thead>
<tbody>
<tr>
<td>args</td>
<td>A list of Constraint objects</td>
</tr>
</tbody>
</table>

*Value*

A cone program constructed from a log 1p atom where t is the objective function and the constraints consist of ExpCone constraints + 1.
Dcp2Cone.logistic_canon

Description
Dcp2Cone canonicalizer for the logistic function atom

Usage
Dcp2Cone.logistic_canon(expr, args)

Arguments
expr     An Expression object
args     A list of Constraint objects

Value
A cone program constructed from the logistic atom where the objective function is given by t0 and the constraints consist of the ExpCone constraints.

Dcp2Cone.log_canon

Description
Dcp2Cone canonicalizer for the log atom

Usage
Dcp2Cone.log_canon(expr, args)

Arguments
expr     An Expression object
args     A list of Constraint objects

Value
A cone program constructed from a log atom where t is the objective function and the constraints consist of ExpCone constraints.
Dcp2Cone.log_sum_exp_canon

Dcp2Cone canonicalizer for the log sum of the exp atom

Description
Dcp2Cone canonicalizer for the log sum of the exp atom

Usage
Dcp2Cone.log_sum_exp_canon(expr, args)

Arguments
expr An Expression object
args A list of Constraint objects

Value
A cone program constructed from the log sum of the exp atom where the objective is the t variable and the constraints consist of the ExpCone constraints and requiring t to be less than a matrix of ones of the same size.
**Description**

Dcp2Cone canonicalizer for the matrix fraction atom

**Usage**

Dcp2Cone.matrix_frac_canon(expr, args)

**Arguments**

- **expr** An Expression object
- **args** A list of Constraint objects

**Value**

A cone program constructed from the matrix fraction atom, where the objective function is the trace of Tvar, a m by m matrix where the constraints consist of the matrix of the Schur complement of Tvar to consist of P, an n by n, given matrix, X, an n by m given matrix, and Tvar.

---

**Description**

Dcp2Cone canonicalizer for the nuclear norm atom

**Usage**

Dcp2Cone.normNuc_canon(expr, args)

**Arguments**

- **expr** An Expression object
- **args** A list of Constraint objects

**Value**

A cone program constructed from a nuclear norm atom, where the objective function consists of .5 times the trace of a matrix X of size m+n by m+n where the constraint consist of the top right corner of the matrix being the original matrix.
Dcp2Cone.power_canon

Dcp2Cone canonicalizer for the power atom

Description
Dcp2Cone canonicalizer for the power atom

Usage
Dcp2Cone.power_canon(expr, args)

Arguments

- **expr**: An Expression object
- **args**: A list of Constraint objects

Value
A cone program constructed from a power atom, where the objective function consists of the variable t which is of the dimension of the original vector from the power atom and the constraints consist of geometric mean constraints.

---

Dcp2Cone.pnorm_canon

Dcp2Cone canonicalizer for the p norm atom

Description
Dcp2Cone canonicalizer for the p norm atom

Usage
Dcp2Cone.pnorm_canon(expr, args)

Arguments

- **expr**: An Expression object
- **args**: A list of Constraint objects

Value
A cone program constructed from a pnorm atom, where the objective is a variable t of dimension of the original vector in the problem and the constraints consist of geometric mean constraints.
**Dcp2Cone.quad_form_canon**

*Dcp2Cone canonicalizer for the quadratic form atom*

**Description**

Dcp2Cone canonicalizer for the quadratic form atom

**Usage**

Dcp2Cone.quad_form_canon(expr, args)

**Arguments**

- **expr**: An Expression object
- **args**: A list of Constraint objects

**Value**

A cone program constructed from a quadratic form atom, where the objective function consists of the scaled objective function from the quadratic over linear canonicalization and same with the constraints.

---

**Dcp2Cone.quad_over_lin_canon**

*Dcp2Cone canonicalizer for the quadratic over linear term atom*

**Description**

Dcp2Cone canonicalizer for the quadratic over linear term atom

**Usage**

Dcp2Cone.quad_over_lin_canon(expr, args)

**Arguments**

- **expr**: An Expression object
- **args**: A list of Constraint objects

**Value**

A cone program constructed from a quadratic over linear term atom where the objective function consists of a one dimensional variable t with SOC constraints.
Dcp2Cone.sigma_max_canon

*Dcp2Cone canonicalizer for the sigma max atom*

**Description**
Dcp2Cone canonicalizer for the sigma max atom

**Usage**
Dcp2Cone.sigma_max_canon(expr, args)

**Arguments**
- *expr*: An Expression object
- *args*: A list of Constraint objects

**Value**
A cone program constructed from a sigma max atom where the objective function consists of the variable \( t \) that is of the same dimension as the original expression with specified constraints in the function.

---

Dgp2Dcp-class

*Reduce DGP problems to DCP problems.*

**Description**
This reduction takes as input a DGP problem and returns an equivalent DCP problem. Because every (generalized) geometric program is a DGP problem, this reduction can be used to convert geometric programs into convex form.

**Usage**

```r
## S4 method for signature 'Dgp2Dcp,Problem'
accepts(object, problem)

## S4 method for signature 'Dgp2Dcp,Problem'
perform(object, problem)

## S4 method for signature 'Dgp2Dcp'
canonicalize_expr(object, expr, args)

## S4 method for signature 'Dgp2Dcp,Solution,InverseData'
invert(object, solution, inverse_data)
```
**Arguments**

- **object**: A Dgp2Dcp object.
- **problem**: A Problem object.
- **expr**: An Expression object corresponding to the DGP problem.
- **args**: A list of values corresponding to the DGP expression.
- **solution**: A Solution object to invert.
- **inverse_data**: An InverseData object containing data necessary for the inversion.

**Methods (by generic)**

- **accepts(object = Dgp2Dcp, problem = Problem)**: Is the problem DGP?
- **perform(object = Dgp2Dcp, problem = Problem)**: Converts the DGP problem to a DCP problem.
- **canonicalize_expr(Dgp2Dcp)**: Canonicalizes each atom within an Dgp2Dcp expression.
- **invert(object = Dgp2Dcp, solution = Solution, inverse_data = InverseData)**: Returns the solution to the original problem given the inverse_data.

---

**Dgp2Dcp.add_canon**  
*Dgp2Dcp canonicalizer for the addition atom*

**Description**

Dgp2Dcp canonicalizer for the addition atom

**Usage**

Dgp2Dcp.add_canon(expr, args)

**Arguments**

- **expr**: An Expression object
- **args**: A list of values for the expr variable

**Value**

A canonicalization of the addition atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
**Dgp2Dcp.constant_canon**

_Dgp2Dcp canonicalizer for the constant atom_

**Description**

_Dgp2Dcp canonicalizer for the constant atom_

**Usage**

_Dgp2Dcp.constant_canon(expr, args)_

**Arguments**

- **expr**  
  An Expression object
- **args**  
  A list of values for the expr variable

**Value**

A canonicalization of the constant atom of a DGP expression, where the returned expression is the DCP equivalent resulting from the log of the expression.

---

**Dgp2Dcp.div_canon**

_Dgp2Dcp canonicalizer for the division atom_

**Description**

_Dgp2Dcp canonicalizer for the division atom_

**Usage**

_Dgp2Dcp.div_canon(expr, args)_

**Arguments**

- **expr**  
  An Expression object
- **args**  
  A list of values for the expr variable

**Value**

A canonicalization of the division atom of a DGP expression, where the returned expression is the log transformed DCP equivalent.
Dgp2Dcp.canonizer for the `exp` atom

**Description**

Dgp2Dcp canonicalizer for the `exp` atom

**Usage**

Dgp2Dcp.exp_canon(expr, args)

**Arguments**

- **expr**: An `Expression` object
- **args**: A list of values for the `expr` variable

**Value**

A canonicalization of the `exp` atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

---

Dgp2Dcp.canonizer for the `(I − X)^{-1}` atom

**Description**

Dgp2Dcp canonicalizer for the `(I − X)^{-1}` atom

**Usage**

Dgp2Dcp.eye_minus_inv_canon(expr, args)

**Arguments**

- **expr**: An `Expression` object
- **args**: A list of values for the `expr` variable

**Value**

A canonicalization of the `(I − X)^{-1}` atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
**Dgp2Dcp.geo_mean_canon**

*Dgp2Dcp canonicalizer for the geometric mean atom*

**Description**

Dgp2Dcp canonicalizer for the geometric mean atom

**Usage**

Dgp2Dcp.geo_mean_canon(expr, args)

**Arguments**

- **expr**: An Expression object
- **args**: A list of values for the expr variable

**Value**

A canonicalization of the geometric mean atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

---

**Dgp2Dcp.log_canon**

*Dgp2Dcp canonicalizer for the log atom*

**Description**

Dgp2Dcp canonicalizer for the log atom

**Usage**

Dgp2Dcp.log_canon(expr, args)

**Arguments**

- **expr**: An Expression object
- **args**: A list of values for the expr variable

**Value**

A canonicalization of the log atom of a DGP expression, where the returned expression is the log of the original expression.
**Dgp2Dcp.mulexpression_canon**

_Dgp2Dcp canonicalizer for the multiplication expression atom_

**Description**

Dgp2Dcp canonicalizer for the multiplication expression atom

**Usage**

Dgp2Dcp.mulexpression_canon(expr, args)

**Arguments**

- **expr**
  - An Expression object
- **args**
  - A list of values for the expr variable

**Value**

A canonicalization of the multiplication expression atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

---

**Dgp2Dcp.mul_canon**

_Dgp2Dcp canonicalizer for the multiplication atom_

**Description**

Dgp2Dcp canonicalizer for the multiplication atom

**Usage**

Dgp2Dcp.mul_canon(expr, args)

**Arguments**

- **expr**
  - An Expression object
- **args**
  - A list of values for the expr variable

**Value**

A canonicalization of the multiplication atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
**Dgp2Dcp.nonpos_constr_canon**

*Dgp2Dcp canonicalizer for the non-positive constraint atom*

**Description**

Dgp2Dcp canonicalizer for the non-positive constraint atom

**Usage**

Dgp2Dcp.nonpos_constr_canon(expr, args)

**Arguments**

- **expr**
  - An *Expression* object
- **args**
  - A list of values for the expr variable

**Value**

A canonicalization of the non-positive contraint atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

---

**Dgp2Dcp.norm1_canon**

*Dgp2Dcp canonicalizer for the 1 norm atom*

**Description**

Dgp2Dcp canonicalizer for the 1 norm atom

**Usage**

Dgp2Dcp.norm1_canon(expr, args)

**Arguments**

- **expr**
  - An *Expression* object
- **args**
  - A list of values for the expr variable

**Value**

A canonicalization of the norm1 atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
Dgp2Dcp canonicalizer for the infinite norm atom

Description

Dgp2Dcp canonicalizer for the infinite norm atom

Usage

Dgp2Dcp.norm_inf_canon(expr, args)

Arguments

expr An Expression object
args A list of values for the expr variable

Value

A canonicalization of the infinity norm atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

Dgp2Dcp canonicalizer for the 1-x atom

Description

Dgp2Dcp canonicalizer for the 1-x atom

Usage

Dgp2Dcp.one_minus_pos_canon(expr, args)

Arguments

expr An Expression object
args A list of values for the expr variable

Value

A canonicalization of the 1-x with 0 < x < 1 atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
Dgp2Dcp.parameter_canon

Dgp2Dcp canonicalizer for the parameter atom

Description
Dgp2Dcp canonicalizer for the parameter atom

Usage
Dgp2Dcp.parameter_canon(expr, args)

Arguments
expr  An Expression object
args  A list of values for the expr variable

Value
A canonicalization of the parameter atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

Dgp2Dcp.pf_eigenvalue_canon

Dgp2Dcp canonicalizer for the spectral radius atom

Description
Dgp2Dcp canonicalizer for the spectral radius atom

Usage
Dgp2Dcp.pf_eigenvalue_canon(expr, args)

Arguments
expr  An Expression object
args  A list of values for the expr variable

Value
A canonicalization of the spectral radius atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
Dgp2Dcp.pnorm_canon

Dgp2Dcp canonicalizer for the p norm atom

Description
Dgp2Dcp canonicalizer for the p norm atom

Usage
Dgp2Dcp.pnorm_canon(expr, args)

Arguments
- expr: An Expression object
- args: A list of values for the expr variable

Value
A canonicalization of the pnorm atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

Dgp2Dcp.power_canon

Dgp2Dcp canonicalizer for the power atom

Description
Dgp2Dcp canonicalizer for the power atom

Usage
Dgp2Dcp.power_canon(expr, args)

Arguments
- expr: An Expression object
- args: A list of values for the expr variable

Value
A canonicalization of the power atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
**Dgp2Dcp.prod_canon**  
*Dgp2Dcp canonicalizer for the product atom*

**Description**

Dgp2Dcp canonicalizer for the product atom

**Usage**

Dgp2Dcp.prod_canon(expr, args)

**Arguments**

- **expr**: An *Expression* object
- **args**: A list of values for the expr variable

**Value**

A canonicalization of the product atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

---

**Dgp2Dcp.quad_form_canon**  
*Dgp2Dcp canonicalizer for the quadratic form atom*

**Description**

Dgp2Dcp canonicalizer for the quadratic form atom

**Usage**

Dgp2Dcp.quad_form_canon(expr, args)

**Arguments**

- **expr**: An *Expression* object
- **args**: A list of values for the expr variable

**Value**

A canonicalization of the quadratic form atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
**Dgp2Dcp.quad_over_lin_canon**

*Dgp2Dcp canonicalizer for the quadratic over linear term atom*

**Description**

Dgp2Dcp canonicalizer for the quadratic over linear term atom

**Usage**

```
Dgp2Dcp.quad_over_lin_canon(expr, args)
```

**Arguments**

- `expr` An Expression object
- `args` A list of values for the expr variable

**Value**

A canonicalization of the quadratic over linear atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

---

**Dgp2Dcp.sum_canon**

*Dgp2Dcp canonicalizer for the sum atom*

**Description**

Dgp2Dcp canonicalizer for the sum atom

**Usage**

```
Dgp2Dcp.sum_canon(expr, args)
```

**Arguments**

- `expr` An Expression object
- `args` A list of values for the expr variable

**Value**

A canonicalization of the sum atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
**Dgp2Dcp.trace_canon**

**Description**

Dgp2Dcp canonicalizer for the trace atom

**Usage**

```python
Dgp2Dcp.trace_canon(expr, args)
```

**Arguments**

- `expr`: An Expression object
- `args`: A list of values for the `expr` variable

**Value**

A canonicalization of the trace atom of a DGP expression, where the returned expression is the transformed DCP equivalent.

---

**Dgp2Dcp.zero_constr_canon**

**Description**

Dgp2Dcp canonicalizer for the zero constraint atom

**Usage**

```python
Dgp2Dcp.zero_constr_canon(expr, args)
```

**Arguments**

- `expr`: An Expression object
- `args`: A list of values for the `expr` variable

**Value**

A canonicalization of the zero constraint atom of a DGP expression, where the returned expression is the transformed DCP equivalent.
DgpCanonMethods-class  

**DGP canonical methods class.**

**Description**

Canonicalization of DGPs is a stateful procedure, hence the need for a class.

**Usage**

```r
## S4 method for signature 'DgpCanonMethods'
names(x)

## S4 method for signature 'DgpCanonMethods'
x$name
```

**Arguments**

- `x` A `DgpCanonMethods` object.
- `name` The name of the atom or expression to canonicalize.

**Methods (by generic)**

- `names(DgpCanonMethods)`: Returns the name of all the canonicalization methods
- `$(DgpCanonMethods)`: Returns either a canonicalized variable or a corresponding Dgp2Dcp canonicalization method

---

**Diag**  

**Description**

Turns an expression into a DiagVec object

**Usage**

```r
Diag(expr)
```

**Arguments**

- `expr` An `Expression` that represents a vector or square matrix.

**Value**

An `Expression` representing the diagonal vector/matrix.
Description

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

Usage

```r
## 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

```r
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 == Variable(3,3, PSD = TRUE))
prob <- Problem(obj, constraints)
result <- solve(prob)
result$value
result$get_value(C)
```

DiagMat-class

The `DiagMat` class.

Description

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

Usage

DiagMat(expr)

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

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

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

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

## S4 method for signature 'DiagMat'
graph_implementation(object, arg_objs, dim, 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.
dim A vector representing the dimensions of the resulting expression.
data A list of additional data required by the atom.

Methods (by generic)

- to_numeric(DiagMat): Extract the diagonal from a square matrix constant.
- dim_from_args(DiagMat): The size of the atom.
- is_atom_log_log_convex(DiagMat): Is the atom log-log convex?
- is_atom_log_log_concave(DiagMat): Is the atom log-log concave?
- graph_implementation(DiagMat): 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'
dim_from_args(object)
```

```
## S4 method for signature 'DiagVec'
is_atom_log_log_convex(object)
```

```
## S4 method for signature 'DiagVec'
is_atom_log_log_concave(object)
```

```
## S4 method for signature 'DiagVec'
is_symmetric(object)
```

```
## S4 method for signature 'DiagVec'
is_hermitian(object)
```

```
## S4 method for signature 'DiagVec'
graph_implementation(object, arg_objs, dim, 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.

- **dim**  
  A vector representing the dimensions of the resulting expression.

- **data**  
  A list of additional data required by the atom.

**Methods (by generic)**

- **to_numeric(DiagVec)**: Convert the vector constant into a diagonal matrix.

- **dim_from_args(DiagVec)**: The dimensions of the atom.
- `is_atom_log_log_convex(DiagVec)`: Is the atom log-log convex?
- `is_atom_log_log_concave(DiagVec)`: Is the atom log-log concave?
- `is_symmetric(DiagVec)`: Is the expression symmetric?
- `is_hermitian(DiagVec)`: Is the expression hermitian?
- `graph_implementation(DiagVec)`: The graph implementation of the atom.

**Slots**

`expr` An *Expression* representing the vector to convert.

---

**Diff**

*Takes the k-th order differences*

### Description

Takes the k-th order differences

### Usage

```
Diff(x, lag = 1, k = 1, axis = 2)
```

### Arguments

- `x` An *Expression* that represents a vector
- `lag` The degree of lag between differences
- `k` The integer value of the order of differences
- `axis` The axis along which to apply the function. For a 2D matrix, 1 indicates rows and 2 indicates columns.

### Value

Takes in a vector of length n and returns a vector of length n-k of the kth order differences
### 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 \( k \)th order difference between the lagged terms. \( \text{diff}(x) \) returns the vector of differences between adjacent elements in the vector, i.e. \([x[2] - x[1], x[3] - x[2], ...]\). \( \text{diff}(x, 1, 2) \) is the second-order differences vector, equivalently \( \text{diff}(\text{diff}(x)) \). \( \text{diff}(x, 1, 0) \) returns the vector x unchanged. \( \text{diff}(x, 2) \) returns the vector of differences \([x[3] - x[1], x[4] - x[2], ...]\), equivalent to \( x[(1+\text{lag}):n] - x[1:(n-\text{lag})] \).

**Usage**

```r
## 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 \( k \)th order difference.

**Examples**

```r
## 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

```r
xs <- result$getValue(x)
y <- result$getValue(y)
plot(c(0, 1), c(0, 1), type = "n", xlab = "x", ylab = "y")
lines(xs, ys, col = "blue", lwd = 2)
grid()
```

---

### DiffPos

**The DiffPos atom.**

#### Description

The difference between expressions, \( x - y \), where \( x > y > 0 \).

#### Usage

```r
DiffPos(x, y)
```

#### Arguments

- **x** An Expression
- **y** An Expression

#### Value

The difference \( x - y \) with domain \( x, y : x > y > 0 \).

---

### dim_from_args

**Atom Dimensions**

#### Description

Determine the dimensions of an atom based on its arguments.

#### Usage

```r
dim_from_args(object)
```

```r
## S4 method for signature 'Atom'
dim_from_args(object)
```

#### Arguments

- **object** A Atom object.

#### Value

A numeric vector \( c(row, col) \) indicating the dimensions of the atom.
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

```r
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
### dual_value-methods  
*Get and Set Dual Value*

**Description**

Get and set the value of the dual variable in a constraint.

**Usage**

```r
dual_value(object)
dual_value(object) <- value
```

**Arguments**

- `object`  
  A `Constraint` object.
- `value`  
  A numeric scalar, vector, or matrix to assign to the object.

### ECOS-class  
*An interface for the ECOS solver*

**Description**

An interface for the ECOS solver

**Usage**

```r
ECOS()
```
Arguments

solver, object, x
A ECOS object.

status
A status code returned by the solver.

problem
A Problem object.

solution
The raw solution returned by the solver.

inverse_data
A list containing data necessary for the inversion.

Methods (by generic)

- `mip_capable(ECOS)`: Can the solver handle mixed-integer programs?
- `status_map(ECOS)`: Converts status returned by the ECOS solver to its respective CVXPY status.
- `import_solver(ECOS)`: Imports the solver
- `name(ECOS)`: Returns the name of the solver
- `perform(object = ECOS, problem = Problem)`: Returns a new problem and data for inverting the new solution.
- `invert(object = ECOS, solution = list, inverse_data = list)`: Returns the solution to the original problem given the inverse_data.

---

**ECOS.dims_to_solver_dict**

Utility method for formatting a ConeDims instance into a dictionary that can be supplied to ECOS.

---

**Description**

Utility method for formatting a ConeDims instance into a dictionary that can be supplied to ECOS.

**Usage**

`ECOS.dims_to_solver_dict(cone_dims)`

**Arguments**

cone_dims A ConeDims instance.

**Value**

A dictionary of cone dimensions
ECOS_BB-class

An interface for the ECOS BB solver.

Description

An interface for the ECOS BB solver.

Usage

ECOS_BB()

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

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

## S4 method for signature 'ECOS_BB,Problem'
perform(object, problem)

## S4 method for signature 'ECOS_BB'
solve_via_data(
  object,
data,
warm_start,
verbose,
feastol,
reitol,
abstol,
num_iter,
solver_opts,
solver_cache
)

Arguments

solver, object, x
  A ECOS_BB object.
problem
  A Problem object.
data
  Data generated via an apply call.
warm_start
  A boolean of whether to warm start the solver.
verbose
  A boolean of whether to enable solver verbosity.
feastol
  The feasible tolerance.
reitol
  The relative tolerance.
abstol
  The absolute tolerance.
Elementwise-class

num_iter        The maximum number of iterations.
solver_opts    A list of Solver specific options
solver_cache   Cache for the solver.

Methods (by generic)

- mip_capable(ECOS_BB): Can the solver handle mixed-integer programs?
- name(ECOS_BB): Returns the name of the solver.
- perform(object = ECOS_BB, problem = Problem): Returns a new problem and data for inverting the new solution.
- solve_via_data(ECOS_BB): Solve a problem represented by data returned from apply.

## S4 method for signature 'Elementwise'
dim_from_args(object)

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

## S4 method for signature 'Elementwise'
is_symmetric(object)

Arguments

object        An Elementwise object.

Methods (by generic)

- dim_from_args(Elementwise): Dimensions is the same as the sum of the arguments’ dimensions.
- validate_args(Elementwise): Verify that all the dimensions are the same or can be promoted.
- is_symmetric(Elementwise): Is the expression symmetric?
EliminatePwl-class

The EliminatePwl class.

Description

This class eliminates piecewise linear atoms.

Usage

```r
## S4 method for signature 'EliminatePwl,Problem'
accepts(object, problem)
```

Arguments

- `object` An `EliminatePwl` object.
- `problem` A `Problem` object.

Methods (by generic)

- `accepts(object = EliminatePwl, problem = Problem)`: Does this problem contain piecewise linear atoms?

EliminatePwl.abs_canon

EliminatePwl canonicalizer for the absolute atom

Description

EliminatePwl canonicalizer for the absolute atom

Usage

```r
EliminatePwl.abs_canon(expr, args)
```

Arguments

- `expr` An `Expression` object
- `args` A list of `Constraint` objects

Value

A canonicalization of the piecewise-linear atom constructed from an absolute atom where the objective function consists of the variable that is of the same dimension as the original expression and the constraints consist of splitting the absolute value into two inequalities.
EliminatePwl.cummax_canon

EliminatePwl canonicalizer for the cumulative max atom

Description

EliminatePwl canonicalizer for the cumulative max atom

Usage

EliminatePwl.cummax_canon(expr, args)

Arguments

expr An Expression object
args A list of Constraint objects

Value

A canonicalization of the piecewise-linear atom constructed from a cumulative max atom where the objective function consists of the variable Y which is of the same dimension as the original expression and the constraints consist of row/column constraints depending on the axis.

EliminatePwl.cumsum_canon

EliminatePwl canonicalizer for the cumulative sum atom

Description

EliminatePwl canonicalizer for the cumulative sum atom

Usage

EliminatePwl.cumsum_canon(expr, args)

Arguments

expr An Expression object
args A list of Constraint objects

Value

A canonicalization of the piecewise-linear atom constructed from a cumulative sum atom where the objective is Y that is of the same dimension as the matrix of the expression and the constraints consist of various row constraints.
EliminatePwl.max_entries_canon

EliminatePwl canonicalizer for the max entries atom

Description
EliminatePwl canonicalizer for the max entries atom

Usage
EliminatePwl.max_entries_canon(expr, args)

Arguments
expr An Expression object
args A list of Constraint objects

Value
A canonicalization of the piecewise-linear atom constructed from the max entries atom where the objective function consists of the variable $t$ of the same size as the original expression and the constraints consist of a vector multiplied by a vector of $1$'s.
**EliminatePwl.min_elemwise_canon**

*EliminatePwl canonicalizer for the elementwise minimum atom*

**Description**

EliminatePwl canonicalizer for the elementwise minimum atom

**Usage**

```python
EliminatePwl.min_elemwise_canon(expr, args)
```

**Arguments**

- `expr`: An `Expression` object
- `args`: A list of `Constraint` objects

**Value**

A canonicalization of the piecewise-linear atom constructed by a minimum elementwise atom where the objective function is the negative of variable $t$ produced by `max_elemwise_canon` of the same dimension as the expression and the constraints consist of a simple inequality.

---

**EliminatePwl.min_entries_canon**

*EliminatePwl canonicalizer for the minimum entries atom*

**Description**

EliminatePwl canonicalizer for the minimum entries atom

**Usage**

```python
EliminatePwl.min_entries_canon(expr, args)
```

**Arguments**

- `expr`: An `Expression` object
- `args`: A list of `Constraint` objects

**Value**

A canonicalization of the piecewise-linear atom constructed by a minimum entries atom where the objective function is the negative of variable $t$ produced by `max_elemwise_canon` of the same dimension as the expression and the constraints consist of a simple inequality.
EliminatePwl.norm1_canon

*EliminatePwl canonicalizer for the 1 norm atom*

**Description**

EliminatePwl canonicalizer for the 1 norm atom

**Usage**

EliminatePwl.norm1_canon(expr, args)

**Arguments**

- **expr**: An Expression object
- **args**: A list of Constraint objects

**Value**

A canonicalization of the piecewise-linear atom constructed by the norm1 atom where the objective function consists of the sum of the variables created by the abs_canon function and the constraints consist of constraints generated by abs_canon.

----------

EliminatePwl.norm_inf_canon

*EliminatePwl canonicalizer for the infinite norm atom*

**Description**

EliminatePwl canonicalizer for the infinite norm atom

**Usage**

EliminatePwl.norm_inf_canon(expr, args)

**Arguments**

- **expr**: An Expression object
- **args**: A list of Constraint objects

**Value**

A canonicalization of the piecewise-linear atom constructed by the infinite norm atom where the objective function consists variable t of the same dimension as the expression and the constraints consist of a vector constructed by multiplying t to a vector of 1’s
EliminatePwl.sum_largest_canon

Description
EliminatePwl canonicalizer for the largest sum atom

Usage
EliminatePwl.sum_largest_canon(expr, args)

Arguments
expr : An Expression object
args : A list of Constraint objects

Value
A canonicalization of the piecewise-linear atom constructed by the k largest sums atom where the objective function consists of the sum of variables t that is of the same dimension as the expression plus k

entr

Description
The elementwise entropy function, \(-x log(x)\).

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 \(-x \log(x)\).

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'
.grad(object, values)
## S4 method for signature 'Entr'
.domain(object)
```

Arguments

- **x**: An Expression or numeric constant.
- **object**: An Entr object.
- **values**: A list of numeric values for the arguments
- **idx**: An index into the atom.
Methods (by generic)

- `to_numeric(Entr)`: The elementwise entropy function evaluated at the value.
- `sign_from_args(Entr)`: The sign of the atom is unknown.
- `is_atom_convex(Entr)`: The atom is not convex.
- `is_atom_concave(Entr)`: The atom is concave.
- `is_incr(Entr)`: The atom is weakly increasing.
- `is_decr(Entr)`: The atom is weakly decreasing.
- `.grad(Entr)`: Gives the (sub/super)gradient of the atom w.r.t. each variable
- `.domain(Entr)`: Returns constraints describing the domain of the node

Slots

- x: An Expression or numeric constant.

---

EvalParams-class

The EvalParams class.

Description

This class represents a reduction that replaces symbolic parameters with their constant values.

Usage

```r
## S4 method for signature 'EvalParams,Problem'
perform(object, problem)
```

```r
## S4 method for signature 'EvalParams,Solution,list'
invert(object, solution, inverse_data)
```

Arguments

- **object**: A EvalParams object.
- **problem**: A Problem object.
- **solution**: A Solution to a problem that generated the inverse data.
- **inverse_data**: The inverse data returned by an invocation to apply.

Methods (by generic)

- `perform(object = EvalParams, problem = Problem)`: Replace parameters with constant values.
- `invert(object = EvalParams, solution = Solution, inverse_data = list)`: Returns a solution to the original problem given the inverse_data.
### Description

The elementwise natural exponential.

### Usage

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

### Arguments

- `x`: An `Expression`.

### Value

An `Expression` representing the natural exponential of the input.

### Examples

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

---

### Exp-class

#### Description

This class represents the elementwise natural exponential $e^x$.

#### Usage

```r
Exp(x)
```

```r
to_numeric(object, values)
```

```r
sign_from_args(object)
```

```r
```

---
is_atom_convex(object)

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

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

## S4 method for signature 'Exp'
is_atom_log_log_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'
.grad(object, values)

Arguments

x An Expression object.
object An Exp object.
values A list of numeric values for the arguments
idx An index into the atom.

Methods (by generic)

- `to_numeric(Exp)`: The matrix with each element exponentiated.
- `sign_from_args(Exp)`: The atom is positive.
- `is_atom_convex(Exp)`: The atom is convex.
- `is_atom_concave(Exp)`: The atom is not concave.
- `is_atom_log_log_convex(Exp)`: Is the atom log-log convex?
- `is_atom_log_log_concave(Exp)`: Is the atom log-log concave?
- `is_incr(Exp)`: The atom is weakly increasing.
- `is_decr(Exp)`: The atom is not weakly decreasing.
- `.grad(Exp)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

Slots

x An Expression object.
The ExpCone class.

Description

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

Usage

\[
\text{ExpCone}(x, y, z, \text{id = NA\_integer\_})
\]

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

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

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

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

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

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

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

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

Arguments

- **x**
  - The variable \(x\) in the exponential cone.

- **y**
  - The variable \(y\) in the exponential cone.

- **z**
  - The variable \(z\) in the exponential cone.

- **id**
  - (Optional) A numeric value representing the constraint ID.

- **object**
  - A \texttt{ExpCone} object.
Details

Original cone:

\[ K = \{(x, y, z) | y > 0, ye^{-y} \leq z\} \cup \{(x, y, z) | x \leq 0, y = 0, z \geq 0\} \]

Reformulated cone:

\[ K = \{(x, y, z) | y > 0, y \log(y) + x \leq y \log(z)\} \cup \{(x, y, z) | x \leq 0, y = 0, z \geq 0\} \]

Methods (by generic)

- `residual(ExpCone)`: The size of the x argument.
- `size(ExpCone)`: The number of entries in the combined cones.
- `num_cones(ExpCone)`: The number of elementwise cones.
- `cone_sizes(ExpCone)`: The dimensions of the exponential cones.
- `is_dcp(ExpCone)`: An exponential constraint is DCP if each argument is affine.
- `is_dgp(ExpCone)`: Is the constraint DGP?
- `canonicalize(ExpCone)`: Canonicalizes by converting expressions to LinOps.

Slots

- `x`: The variable x in the exponential cone.
- `y`: The variable y in the exponential cone.
- `z`: The variable z in the exponential cone.

---

Description

This class represents a mathematical expression.

Usage

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

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

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

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

## S4 method for signature 'Expression'
expr(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_log_log_constant(object)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## S4 method for signature 'Expression'
flatten(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(Expression)`: The value of the expression.
- `grad(Expression)`: The (sub/super)-gradient of the expression with respect to each variable.
- `domain(Expression)`: A list of constraints describing the closure of the region where the expression is finite.
- `as.character(Expression)`: The string representation of the expression.
- `name(Expression)`: The name of the expression.
- `expr(Expression)`: The expression itself.
- `is_constant(Expression)`: The expression is constant if it contains no variables or is identically zero.
- `is_affine(Expression)`: The expression is affine if it is constant or both convex and concave.
- `is_convex(Expression)`: A logical value indicating whether the expression is convex.
- `is_concave(Expression)`: A logical value indicating whether the expression is concave.
- `is_dcp(Expression)`: The expression is DCP if it is convex or concave.
- `is_log_log_constant(Expression)`: Is the expression log-log constant, i.e., elementwise positive?
- `is_log_log_affine(Expression)`: Is the expression log-log affine?
- `is_log_log_convex(Expression)`: Is the expression log-log convex?
- `is_log_log_concave(Expression)`: Is the expression log-log concave?
- `is_dgp(Expression)`: The expression is DGP if it is log-log DCP.
- `is_hermitian(Expression)`: A logical value indicating whether the expression is a Hermitian matrix.
- `is_psd(Expression)`: A logical value indicating whether the expression is a positive semidefinite matrix.
- `is_nsd(Expression)`: A logical value indicating whether the expression is a negative semidefinite matrix.
- `is_quadratic(Expression)`: A logical value indicating whether the expression is quadratic.
- `is_symmetric(Expression)`: A logical value indicating whether the expression is symmetric.
expression-parts

- **is_pwl(Expression)**: A logical value indicating whether the expression is piecewise linear.
- **is_qpwa(Expression)**: A logical value indicating whether the expression is quadratic of piecewise affine.
- **is_zero(Expression)**: The expression is zero if it is both nonnegative and nonpositive.
- **is_nonneg(Expression)**: A logical value indicating whether the expression is nonnegative.
- **is_nonpos(Expression)**: A logical value indicating whether the expression is nonpositive.
- **dim(Expression)**: The (row, col) dimensions of the expression.
- **is_real(Expression)**: A logical value indicating whether the expression is real.
- **is_imag(Expression)**: A logical value indicating whether the expression is imaginary.
- **is_complex(Expression)**: A logical value indicating whether the expression is complex.
- **size(Expression)**: The number of entries in the expression.
- **ndim(Expression)**: The number of dimensions of the expression.
- **flatten(Expression)**: Vectorizes the expression.
- **is_scalar(Expression)**: A logical value indicating whether the expression is a scalar.
- **is_vector(Expression)**: A logical value indicating whether the expression is a row or column vector.
- **is_matrix(Expression)**: A logical value indicating whether the expression is a matrix.
- **nrow(Expression)**: Number of rows in the expression.
- **ncol(Expression)**: Number of columns in the expression.

---

**expression-parts**  
*Parts of an Expression Leaf*

**Description**

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

**Usage**

```r
variables(object)
parameters(object)
constants(object)
atoms(object)
```

**Arguments**

- `object`  
  A *Leaf* object.
Value

A list of Variable, Parameter, Constant, or Atom objects.

Examples

```r
set.seed(67)
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) })
```

extract_dual_value  

Gets a specified value of a dual variable.

Description

Gets a specified value of a dual variable.

Usage

```r
extract_dual_value(result_vec, offset, constraint)
```

Arguments

- `result_vec`  
  A vector containing the dual variable values.
- `offset`  
  An offset to get correct index of dual values.
- `constraint`  
  A list of the constraints in the problem.

Value

A list of a dual variable value and its offset.
extract_mip_idx  

Coalesces bool, int indices for variables.

**Description**
Coalesces bool, int indices for variables.

**Usage**
```
extract_mip_idx(variables)
```

**Arguments**
- **variables** A list of Variable objects.

**Value**
Coalesces bool, int indices for variables. The indexing scheme assumes that the variables will be coalesced into a single one-dimensional variable, with each variable being reshaped in Fortran order.

**EyeMinusInv-class**
The EyeMinusInv class.

This class represents the unity resolvent of an elementwise positive matrix $X$, i.e., $(I - X)^{-1}$, and it enforces the constraint that the spectral radius of $X$ is at most 1. This atom is log-log convex.

**Usage**
```
EyeMinusInv(X)
```

```r
## S4 method for signature 'EyeMinusInv'
to_numeric(object, values)
## S4 method for signature 'EyeMinusInv'
name(x)
## S4 method for signature 'EyeMinusInv'
dim_from_args(object)
## S4 method for signature 'EyeMinusInv'
sign_from_args(object)
## S4 method for signature 'EyeMinusInv'
```
is_atom_convex(object)

## S4 method for signature 'EyeMinusInv'

is_atom_concave(object)

## S4 method for signature 'EyeMinusInv'

is_atom_log_log_convex(object)

## S4 method for signature 'EyeMinusInv'

is_atom_log_log_concave(object)

## S4 method for signature 'EyeMinusInv'

is_incr(object, idx)

## S4 method for signature 'EyeMinusInv'

is_decr(object, idx)

## S4 method for signature 'EyeMinusInv'

.grad(object, values)

Arguments

X An Expression or numeric matrix.

object, x An EyeMinusInv object.

dvalues A list of numeric values for the arguments

idz An index into the atom.

Methods (by generic)

- to_numeric(EyeMinusInv): The unity resolvent of the matrix.
- name(EyeMinusInv): The name and arguments of the atom.
- dim_from_args(EyeMinusInv): The dimensions of the atom determined from its arguments.
- sign_from_args(EyeMinusInv): The (is positive, is negative) sign of the atom.
- is_atom_convex(EyeMinusInv): Is the atom convex?
- is_atom_concave(EyeMinusInv): Is the atom concave?
- is_atom_log_log_convex(EyeMinusInv): Is the atom log-log convex?
- is_atom_log_log_concave(EyeMinusInv): Is the atom log-log concave?
- is_incr(EyeMinusInv): Is the atom weakly increasing in the index?
- is_decr(EyeMinusInv): Is the atom weakly decreasing in the index?
- .grad(EyeMinusInv): Gives EyeMinusInv the (sub/super)gradient of the atom w.r.t. each variable

Slots

X An Expression or numeric matrix.
**eye_minus_inv**

**Unity Resolvent**

**Description**

The unity resolvent of a positive matrix. For an elementwise positive matrix $X$, this atom represents $(I - X)^{-1}$, and it enforces the constraint that the spectral radius of $X$ is at most 1.

**Usage**

`eye_minus_inv(X)`

**Arguments**

$X$  
An Expression or positive square matrix.

**Details**

This atom is log-log convex.

**Value**

An Expression representing the unity resolvent of the input.

**Examples**

```r
A <- Variable(2,2, pos = TRUE)
prob <- Problem(Minimize(matrix_trace(A)), list(eye_minus_inv(A) <=1))
result <- solve(prob, gp = TRUE)
result$value
result$getValue(A)
```

---

**FlipObjective-class**  
The FlipObjective class.

**Description**

This class represents a reduction that flips a minimization objective to a maximization and vice versa.

**Usage**

```r
## S4 method for signature 'FlipObjective,Problem'
perform(object, problem)

## S4 method for signature 'FlipObjective,Solution,list'
invert(object, solution, inverse_data)
```
Arguments

- **object**: A `FlipObjective` object.
- **problem**: A `Problem` object.
- **solution**: A `Solution` to a problem that generated the inverse data.
- **inverse_data**: The inverse data returned by an invocation to apply.

Methods (by generic)

- **perform** (`object = FlipObjective, problem = Problem`): Flip a minimization objective to a maximization and vice versa.
- **invert** (`object = FlipObjective, solution = Solution, inverse_data = list`): Map the solution of the flipped problem to that of the original.

---

**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.
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'
to_numeric(object, values)

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

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

## S4 method for signature 'GeoMean'
name(x)

## S4 method for signature 'GeoMean'
dim_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_atom_log_log_convex(object)

## S4 method for signature 'GeoMean'
is_atom_log_log_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'
copy(object, args = NULL, id_objects = list())

### 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 $p/\text{sum}(p)$ 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 numeric values for the arguments
- **idx**: An index into the atom.
- **args**: An optional list that contains the arguments to reconstruct the atom. Default is to use current arguments of the atom.
- **id_objects**: Currently unused.

### Details

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

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 $x \in \mathbb{R}^1$.

### Methods (by generic)

- `to_numeric(GeoMean)`: The (weighted) geometric mean of the elements of $x$.
- `.domain(GeoMean)`: Returns constraints describing the domain of the node
- `.grad(GeoMean)`: Gives the (sub/super)gradient of the atom w.r.t. each variable
- `name(GeoMean)`: The name and arguments of the atom.
- `dim_from_args(GeoMean)`: The atom is a scalar.
- `sign_from_args(GeoMean)`: The atom is non-negative.
- `is_atom_convex(GeoMean)`: The atom is not convex.
- `is_atom_concave(GeoMean)`: The atom is concave.
- `is_atom_log_log_convex(GeoMean)`: Is the atom log-log convex?
- `is_atom_log_log_concave(GeoMean)`: Is the atom log-log concave?
geo_mean

- `is_incr(GeoMean)`: The atom is weakly increasing in every argument.
- `is_decr(GeoMean)`: The atom is not weakly decreasing in any argument.
- `get_data(GeoMean)`: Returns \( \text{list(w, dyadic completion, tree of dyads)} \).
- `copy(GeoMean)`: Returns a shallow copy of the GeoMean 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 \( \| p/1^T p - w \|_\infty \).

---

### Description

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

### Usage

```r
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

\[
(x_1^{p_1} \cdots x_n^{p_n})^{1/p}
\]

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 \( x \in \mathbb{R}^1 \).
get_data

Value

An Expression representing the geometric mean of the input.

Examples

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

## Not run:
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)

## End(Not run)
```

---

get_data  

Get Expression Data

Description

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

Usage

```r
get_data(object)
```

Arguments

```r
object  
A Expression object.
```

Value

A list containing data.
get_dual_values

Description

Gets the values of the dual variables.

Usage

get_dual_values(result_vec, parse_func, constraints)

Arguments

- result_vec: A vector containing the dual variable values.
- parse_func: Function handle for the parser.
- constraints: A list of the constraints in the problem.

Value

A map of constraint ID to dual variable value.

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_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, gp)
```

**Arguments**
- **object**  
  A Problem object.
- **solver**  
  A string indicating the solver that the problem data is for. Call `installed_solvers()` to see all available.
- **gp**  
  (Optional) A logical value indicating whether the problem is a geometric program.

**Value**
A list containing the data for the solver, the solving chain for the problem, and the inverse data needed to invert the solution.
Examples

```r
a <- Variable(name = "a")
data <- get_problem_data(Problem(Minimize(exp(a) + 2)), "SCS")[[1]]
data["dims"]
data["c"]
data["A"]

x <- Variable(2, name = "x")
data <- get_problem_data(Problem(Minimize(p_norm(x) + 3)), "ECOS")[[1]]
data["dims"]
data["c"]
data["A"]
data["G"]
```

---

**get_sp**

*Get scipy handle*

---

**Description**

Get the scipy handle or fail if not available

**Usage**

```r
get_sp()
```

**Value**

the scipy handle

**Examples**

```r
## Not run:
get_sp

## End(Not run)
```

---

**GLPK-class**

*An interface for the GLPK solver.*

---

**Description**

An interface for the GLPK solver.
Usage

GLPK()

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

## S4 method for signature 'GLPK'
status_map(solver, status)

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

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

## S4 method for signature 'GLPK,list,list'
invert(object, solution, inverse_data)

## S4 method for signature 'GLPK'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

Arguments

solver, object, x
  A GLPK object.

status
  A status code returned by the solver.

solution
  The raw solution returned by the solver.

inverse_data
  A list containing data necessary for the inversion.

data
  Data generated via an apply call.

warm_start
  A boolean of whether to warm start the solver.

verbose
  A boolean of whether to enable solver verbosity.

feastol
  The feasible tolerance.

reltol
  The relative tolerance.

abstol
  The absolute tolerance.
GLPK_MI-class

An interface for the GLPK MI solver.

Description

An interface for the GLPK MI solver.

Usage

GLPK_MI()

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

## S4 method for signature 'GLPK_MI'
status_map(solver, status)

## S4 method for signature 'GLPK_MI'
name(x)

## S4 method for signature 'GLPK_MI'
solve_via_data(
    object, data, warm_start, verbose, feastol, reltol, abstol, num_iter, solver_opts, solver_cache, num_iter
    The maximum number of iterations.
    solver_opts A list of Solver specific options
    solver_cache Cache for the solver.

Methods (by generic)

• mip_capable(GLPK): Can the solver handle mixed-integer programs?
• status_map(GLPK): Converts status returned by the GLPK solver to its respective CVXPY status.
• name(GLPK): Returns the name of the solver.
• import_solver(GLPK): Imports the solver.
• invert(object = GLPK, solution = list, inverse_data = list): Returns the solution to the original problem given the inverse_data.
• solve_via_data(GLPK): Solve a problem represented by data returned from apply.
Arguments

solver, object, x

A GLPK_MI object.

status

A status code returned by the solver.

data

Data generated via an apply call.

warm_start

A boolean of whether to warm start the solver.

verbose

A boolean of whether to enable solver verbosity.

feastol

The feasible tolerance.

reftol

The relative tolerance.

abstol

The absolute tolerance.

num_iter

The maximum number of iterations.

solver_opts

A list of Solver specific options

solver_cache

Cache for the solver.

Methods (by generic)

• mip_capable(GLPK_MI)

  Can the solver handle mixed-integer programs?

• status_map(GLPK_MI)

  Converts status returned by the GLPK_MI solver to its respective CVXPY status.

• name(GLPK_MI)

  Returns the name of the solver.

• solve_via_data(GLPK_MI)

  Solve a problem represented by data returned from apply.

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
```r
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
```r
graph_implementation(object, arg_objs, dim, data)
```

Arguments
- **object** An Expression object.
- **arg_objs** A list of linear expressions for each argument.
- **dim** A vector representing the dimensions 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.
**group_constraints**  
*Organize the constraints into a dictionary keyed by constraint names.*

**Description**  
Organize the constraints into a dictionary keyed by constraint names.

**Usage**  
group_constraints(constraints)

**Arguments**  
constraints  
a list of constraints.

**Value**  
A list of constraint types where constr_map[[cone_type]] maps to a list.

---

**GUROBI_CONIC-class**  
*An interface for the GUROBI conic solver.*

**Description**  
An interface for the GUROBI conic solver.

**Usage**  
GUROBI_CONIC()

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

## S4 method for signature 'GUROBI_CONIC'
name(x)

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

## S4 method for signature 'GUROBI_CONIC'
status_map(solver, status)

## S4 method for signature 'GUROBI_CONIC,Problem'
accepts(object, problem)

## S4 method for signature 'GUROBI_CONIC,Problem'
perform(object, problem)

## S4 method for signature 'GUROBI_CONIC,list,list'
invert(object, solution, inverse_data)

## S4 method for signature 'GUROBI_CONIC'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

Arguments

- solver, object, x
  A GUROBI_CONIC object.
- status
  A status code returned by the solver.
- problem
  A Problem object.
- solution
  The raw solution returned by the solver.
- inverse_data
  A list containing data necessary for the inversion.
- data
  Data generated via an apply call.
- warm_start
  A boolean of whether to warm start the solver.
- verbose
  A boolean of whether to enable solver verbosity.
- feastol
  The feasible tolerance.
- reltol
  The relative tolerance.
- abstol
  The absolute tolerance.
- num_iter
  The maximum number of iterations.
- solver_opts
  A list of Solver specific options
- solver_cache
  Cache for the solver.

Methods (by generic)

- `mip_capable(GUROBI_CONIC)`: Can the solver handle mixed-integer programs?
- `name(GUROBI_CONIC)`: Returns the name of the solver.
- `import_solver(GUROBI_CONIC)`: Imports the solver.
- `status_map(GUROBI_CONIC)`: Converts status returned by the GUROBI solver to its respective CVXPY status.
• accepts(object = GUROBI_CONIC, problem = Problem): Can GUROBI_CONIC solve the problem?
• perform(object = GUROBI_CONIC, problem = Problem): Returns a new problem and data for inverting the new solution.
• invert(object = GUROBI_CONIC, solution = list, inverse_data = list): Returns the solution to the original problem given the inverse_data.
• solve_via_data(GUROBI_CONIC): Solve a problem represented by data returned from apply.

GUROBI_QP-class

An interface for the GUROBI_QP solver.

Description

An interface for the GUROBI_QP solver.

Usage

GUROBI_QP()

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

## S4 method for signature 'GUROBI_QP'
status_map(solver, status)

## S4 method for signature 'GUROBI_QP'
name(x)

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

## S4 method for signature 'GUROBI_QP'
solve_via_data(
    object,
data,
warm_start,
verbose,
feastol,
reltol,
abstol,
um_iter,
solver_opts,
solver_cache
)

## S4 method for signature 'GUROBI_QP,list,InverseData'
invert(object, solution, inverse_data)
HarmonicMean

Arguments

- solver, object, x
  - A GUROBI_QP object.
- status
  - A status code returned by the solver.
- data
  - Data generated via an apply call.
- warm_start
  - A boolean of whether to warm start the solver.
- verbose
  - A boolean of whether to enable solver verbosity.
- feastol
  - The feasible tolerance.
- reltol
  - The relative tolerance.
- abstol
  - The absolute tolerance.
- num_iter
  - The maximum number of iterations.
- solver_opts
  - A list of Solver specific options
- solver_cache
  - Cache for the solver.
- solution
  - The raw solution returned by the solver.
- inverse_data
  - A InverseData object containing data necessary for the inversion.

Methods (by generic)

- mip_capable(GUROBI_QP): Can the solver handle mixed-integer programs?
- status_map(GUROBI_QP): Converts status returned by the GUROBI solver to its respective CVXPY status.
- name(GUROBI_QP): Returns the name of the solver.
- import_solver(GUROBI_QP): Imports the solver.
- solve_via_data(GUROBI_QP): Solve a problem represented by data returned from apply.
- invert(object = GUROBI_QP, solution = list, inverse_data = InverseData): Returns the solution to the original problem given the inverse_data.

HarmonicMean

The HarmonicMean atom.

Description

The harmonic mean of x, \( \frac{1}{n} \sum_{i=1}^{n} x_i^{-1} \), where n is the length of x.

Usage

HarmonicMean(x)

Arguments

- x
  - An expression or number whose harmonic mean is to be computed. Must have positive entries.

Value

The harmonic mean of x.
**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**

```r
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

```r
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

```r
HStack(...)

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

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

## S4 method for signature 'HStack'
```
is_atom_log_log_convex(object)

## S4 method for signature 'HStack'

is_atom_log_log_concave(object)

## S4 method for signature 'HStack'

validate_args(object)

## S4 method for signature 'HStack'

graph_implementation(object, arg_objs, dim, data = NA_real_)

Arguments

... Expression objects or matrices. All arguments must have the same dimensions except for axis 2 (columns).

object A HStack object.

values A list of arguments to the atom.

arg_objs A list of linear expressions for each argument.

dim A vector representing the dimensions of the resulting expression.

data A list of additional data required by the atom.

Methods (by generic)

• to_numeric(HStack): Horizontally concatenate the values using cbind.

• dim_from_args(HStack): The dimensions of the atom.

• is_atom_log_log_convex(HStack): Is the atom log-log convex?

• is_atom_log_log_concave(HStack): Is the atom log-log concave?

• validate_args(HStack): Check all arguments have the same height.

• graph_implementation(HStack): The graph implementation of the atom.

Slots

... Expression objects or matrices. All arguments must have the same dimensions except for axis 2 (columns).

---

huber

Huber Function

Description

The elementwise Huber function, \( Huber(x, M) = 1 

2M|x| - M^2 \text{ for } |x| \geq |M|

|x|^2 \text{ 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

set.seed(11)
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, \( Huber(x, M = 1) \)

\[
2M|x| - M^2 \quad \text{for} \quad |x| \geq |M| \\
|x|^2 \quad \text{for} \quad |x| \leq |M|.
\]
Usage

Huber(x, M = 1)

Arguments

x    An Expression object.
M    A positive scalar value representing the threshold. Defaults to 1.
object    A Huber object.
values    A list of numeric values for the arguments
idx    An index into the atom.

Methods (by generic)

- `to_numeric(Huber)`: The Huber function evaluated elementwise on the input value.
- `sign_from_args(Huber)`: The atom is positive.
- `is_atom_convex(Huber)`: The atom is convex.
• `is_atom_concave(Huber)`: The atom is not concave.
• `is_incr(Huber)`: A logical value indicating whether the atom is weakly increasing.
• `is_decr(Huber)`: A logical value indicating whether the atom is weakly decreasing.
• `is_quadratic(Huber)`: The atom is quadratic if \( x \) is affine.
• `get_data(Huber)`: A list containing the parameter \( M \).
• `validate_args(Huber)`: Check that \( M \) is a non-negative constant.
• `.grad(Huber)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

Slots

- \( x \) An Expression or numeric constant.
- \( M \) A positive scalar value representing the threshold. Defaults to 1.

---

**Description**

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

**Usage**

```r
id(object)
```

**Arguments**

- `object` A Variable or Constraint object.

**Value**

A non-negative integer identifier.

**See Also**

- `get_id`
- `setIdCounter`

**Examples**

```r
x <- Variable()
constr <- (x >= 5)
id(x)
id(constr)
```
**Imag-class**

The Imag class.

**Description**

This class represents the imaginary part of an expression.

**Usage**

Imag(expr)

```r
## S4 method for signature 'Imag'
to_numeric(object, values)
## S4 method for signature 'Imag'
dim_from_args(object)
## S4 method for signature 'Imag'
is_imag(object)
## S4 method for signature 'Imag'
is_complex(object)
## S4 method for signature 'Imag'
is_symmetric(object)
```

**Arguments**

- **expr** An `Expression` representing a vector or matrix.
- **object** An `Imag` object.
- **values** A list of arguments to the atom.

**Methods (by generic)**

- `to_numeric(Imag)`: The imaginary part of the given value.
- `dim_from_args(Imag)`: The dimensions of the atom.
- `is_imag(Imag)`: Is the atom imaginary?
- `is_complex(Imag)`: Is the atom complex valued?
- `is_symmetric(Imag)`: Is the atom symmetric?

**Slots**

- **expr** An `Expression` representing a vector or matrix.
**import_solver**

**Import Solver**

**Description**
Import the R library that interfaces with the specified solver.

**Usage**
import_solver(solver)

**Arguments**
solver A ReductionSolver object.

**Examples**
import_solver(ECOS())
import_solver(SCS())

**installed_solvers**

**List installed solvers**

**Description**
List available solvers, taking currently blacklisted solvers into account.

**Usage**
installed_solvers()
add_to_solver_blacklist(solvers)
remove_from_solver_blacklist(solvers)
set_solver_blacklist(solvers)

**Arguments**
solvers a character vector of solver names, default character(0)

**Value**
The names of all the installed solvers as a character vector.
The current blacklist (character vector), invisibly.
Functions

- add_to_solver_blacklist(): Add to solver blacklist, useful for temporarily disabling a solver
- remove_from_solver_blacklist(): Remove solvers from blacklist
- set_solver_blacklist(): Set solver blacklist to a value

InverseData-class  

The InverseData class.

Description

This class represents the data encoding an optimization problem.

invert  

Return Original Solution

Description

Returns a solution to the original problem given the inverse data.

Usage

invert(object, solution, inverse_data)

Arguments

- object  A Reduction object.
- solution  A Solution to a problem that generated inverse_data.
- inverse_data  A InverseData object encoding the original problem.

Value

A Solution to the original problem.
**inv_pos**  

*Reciprocal Function*

**Description**

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

**Usage**

```r
inv_pos(x)
```

**Arguments**

- `x`  
  An *Expression*, vector, or matrix.

**Value**

An *Expression* representing the reciprocal of the input.

**Examples**

```r
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**

```r
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.
is_mixed_integer

Examples

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

is_dgp

**DGP Compliance**

**Description**

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

**Usage**

```r
is_dgp(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**

```r
x <- Variable(pos = TRUE)
y <- Variable(pos = TRUE)
prob <- Problem(Minimize(x*y), list(x >= 5, y >= 5))
is_dgp(prob)
solve(prob, gp = TRUE)
```

is_mixed_integer

**Is Problem Mixed Integer?**

**Description**

Determine if a problem is a mixed-integer program.

**Usage**

```r
is_mixed_integer(object)
```
is_qp

Arguments
object A Problem object.

Value
A logical value indicating whether the problem is a mixed-integer program

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.

is_stuffed_cone_constraint

Is the constraint a stuffed cone constraint?

Description
Is the constraint a stuffed cone constraint?

Usage
is_stuffed_cone_constraint(constraint)

Arguments
constraint A Constraint object.

Value
Is the constraint a stuffed-cone constraint?
is_stuffed_cone_objective

Is the objective a stuffed cone objective?

Description
Is the objective a stuffed cone objective?

Usage
is_stuffed_cone_objective(objective)

Arguments
objective An Objective object.

Value
Is the objective a stuffed-cone objective?

is_stuffed_qp_objective

Is the QP objective stuffed?

Description
Is the QP objective stuffed?

Usage
is_stuffed_qp_objective(objective)

Arguments
objective A Minimize or Maximize object representing the optimization objective.

Value
Is the objective a stuffed QP?
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'
.grad(object, values)

## S4 method for signature 'KLDiv'
.domain(object)

Arguments

x  An Expression or numeric constant.
y  An Expression or numeric constant.
object  A KLDiv object.
values  A list of numeric values for the arguments
idx  An index into the atom.
Methods (by generic)

- to_numeric(KLDiv): The KL-divergence evaluated elementwise on the input value.
- sign_from_args(KLDiv): The atom is positive.
- is_atom_convex(KLDiv): The atom is convex.
- is_atom_concave(KLDiv): The atom is not concave.
- is_incr(KLDiv): The atom is not monotonic in any argument.
- is_decr(KLDiv): The atom is not monotonic in any argument.
- .grad(KLDiv): Gives the (sub/super)gradient of the atom w.r.t. each variable
- .domain(KLDiv): Returns constraints describing the domain of the node

Slots

- \( x \) An Expression or numeric constant.
- \( y \) An Expression or numeric constant.

kl_div  

\textit{Kullback-Leibler Divergence}

Description

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

Usage

\[
\text{kl_div}(x, y)
\]

Arguments

- \( x \) An Expression, vector, or matrix.
- \( y \) An Expression, vector, or matrix.

Value

An Expression representing the KL-divergence of the input.

Examples

```r
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'
to_numeric(object, values)

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

## S4 method for signature 'Kron'
dim_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, dim, 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.
dim 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(Kron): The kronecker product of the two values.
- validate_args(Kron): Check both arguments are vectors and the first is a constant.
- dim_from_args(Kron): The dimensions of the atom.
- sign_from_args(Kron): The sign of the atom.
- is_incr(Kron): Is the left-hand expression positive?
- is_decr(Kron): Is the right-hand expression negative?
- graph_implementation(Kron): The graph implementation of the atom.

Slots

1h_exp An Expression or numeric constant representing the left-hand matrix.
rh_exp An Expression or numeric constant representing the right-hand matrix.

Description

The generalized kronecker product of two matrices.

Usage

```r
# 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, ...)
```

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.
LambdaMax-class

Value

An Expression that represents the kronecker product.

Examples

```r
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, \( \lambda_{\text{max}}(A) \).

Usage

LambdaMax(A)

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

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

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

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

## S4 method for signature 'LambdaMax'
dim_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)

### 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.

### Methods (by generic)

- **to_numeric(LambdaMax)**: The largest eigenvalue of $A$. Requires that $A$ be symmetric.
- **domain(LambdaMax)**: Returns the constraints describing the domain of the atom.
- **grad(LambdaMax)**: Gives the (sub/super)gradient of the atom with respect to each argument. Matrix expressions are vectorized, so the gradient is a matrix.
- **validate_args(LambdaMax)**: Check that $A$ is square.
- **dim_from_args(LambdaMax)**: The atom is a scalar.
- **sign_from_args(LambdaMax)**: The sign of the atom is unknown.
- **is_atom_convex(LambdaMax)**: The atom is convex.
- **is_atom_concave(LambdaMax)**: The atom is not concave.
- **is_incr(LambdaMax)**: The atom is not monotonic in any argument.
- **is_decr(LambdaMax)**: The atom is not monotonic in any argument.

### Slots

- **A**: An Expression or numeric matrix.

---

**LambdaMin**  
*The LambdaMin atom.*

### Description

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

### Usage

LambdaMin(A)
Arguments

A  An Expression or numeric matrix.

Value

Returns the minimum eigenvalue of a matrix.

Description

This class represents the sum of the $k$ largest eigenvalues of a matrix.

Usage

LambdaSumLargest(A, k)

# S4 method for signature 'LambdaSumLargest'
allow_complex(object)

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

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

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

# S4 method for signature 'LambdaSumLargest'
.grad(object, values)

Arguments

A  An Expression or numeric matrix.

k  A positive integer.

object  A LambdaSumLargest object.

values  A list of numeric values for the arguments
**Methods (by generic)**

- `allow_complex(LambdaSumLargest)`: Does the atom handle complex numbers?
- `to_numeric(LambdaSumLargest)`: Returns the largest eigenvalue of $A$, which must be symmetric.
- `validate_args(LambdaSumLargest)`: Verify that the argument $A$ is square.
- `get_data(LambdaSumLargest)`: Returns the parameter $k$.
- `.grad(LambdaSumLargest)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

**Slots**

- $k$ A positive integer.

---

**LambdaSumSmallest**

*The LambdaSumSmallest atom.*

**Description**

This class represents the sum of the $k$ smallest eigenvalues of a matrix.

**Usage**

```
LambdaSumSmallest(A, k)
```

**Arguments**

- $A$ An `Expression` or numeric matrix.
- $k$ A positive integer.

**Value**

Returns the sum of the $k$ smallest eigenvalues of a matrix.

---

**lambda_max**

*Maximum Eigenvalue*

**Description**

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

**Usage**

```
lambda_max(A)
```
**lambda_min**

**Arguments**

\( A \)  
An Expression or matrix.

**Value**

An Expression representing the maximum eigenvalue of the input.

**Examples**

```r
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])
result <- solve(prob)
result$value
result$getValue(A)
```

---

**lambda_min**  
**Minimum Eigenvalue**

**Description**

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

**Usage**

\( \text{lambda_min}(A) \)

**Arguments**

\( A \)  
An Expression or matrix.

**Value**

An Expression representing the minimum eigenvalue of the input.

**Examples**

```r
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**

\[ \text{lambda_sum_largest}(A, k) \]

**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**

```r
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**

\[ \text{lambda_sum_smallest}(A, k) \]

**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

```r
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)
```

---

### Attributes of an Expression Leaf

**Description**

Determine if an expression is positive or negative.

**Usage**

```r
is_pos(object)
```

```r
is_neg(object)
```

**Arguments**

- `object` A Leaf object.

**Value**

A logical value.

---

### The Leaf class

**Description**

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

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

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

## 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'
atoms(object)

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

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

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

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

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

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

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

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

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

## S4 method for signature 'Leaf'
is_symmetric(object)
## S4 method for signature 'Leaf' is_imag(object)
## S4 method for signature 'Leaf' is_complex(object)
## S4 method for signature 'Leaf' domain(object)
## S4 method for signature 'Leaf' project(object, value)
## S4 method for signature 'Leaf' project_and_assign(object, value)
## S4 method for signature 'Leaf' value(object)
## S4 replacement method for signature 'Leaf' value(object) <- value
## S4 method for signature 'Leaf' validate_val(object, val)
## S4 method for signature 'Leaf' is_psd(object)
## S4 method for signature 'Leaf' is_nsd(object)
## S4 method for signature 'Leaf' is_quadratic(object)
## S4 method for signature 'Leaf' is_pwl(object)

### Arguments

- **object**, **x**  
  A Leaf object.
- **value**  
  A numeric scalar, vector, or matrix.
- **val**  
  The assigned value.

### Methods (by generic)

- `get_data(Leaf)`: Leaves are not copied.
- `dim(Leaf)`: The dimensions of the leaf node.
- `variables(Leaf)`: List of Variable objects in the leaf node.
• parameters(Leaf): List of Parameter objects in the leaf node.
• constants(Leaf): List of Constant objects in the leaf node.
• atoms(Leaf): List of Atom objects in the leaf node.
• is_convex(Leaf): A logical value indicating whether the leaf node is convex.
• is_concave(Leaf): A logical value indicating whether the leaf node is concave.
• is_log_log_convex(Leaf): Is the expression log-log convex?
• is_log_log_concave(Leaf): Is the expression log-log concave?
• is_nonneg(Leaf): A logical value indicating whether the leaf node is nonnegative.
• is_nonpos(Leaf): A logical value indicating whether the leaf node is nonpositive.
• is_pos(Leaf): Is the expression positive?
• is_neg(Leaf): Is the expression negative?
• is_hermitian(Leaf): A logical value indicating whether the leaf node is hermitian.
• is_symmetric(Leaf): A logical value indicating whether the leaf node is symmetric.
• is_imag(Leaf): A logical value indicating whether the leaf node is imaginary.
• is_complex(Leaf): A logical value indicating whether the leaf node is complex.
• domain(Leaf): A list of constraints describing the closure of the region where the leaf node is finite. Default is the full domain.
• project(Leaf): Project value onto the attribute set of the leaf.
• project_and_assign(Leaf): Project and assign a value to the leaf.
• value(Leaf): Get the value of the leaf.
• value(Leaf) <- value: Set the value of the leaf.
• validate_val(Leaf): Check that val satisfies symbolic attributes of leaf.
• is_psd(Leaf): A logical value indicating whether the leaf node is a positive semidefinite matrix.
• is_nsd(Leaf): A logical value indicating whether the leaf node is a negative semidefinite matrix.
• is_quadratic(Leaf): Leaf nodes are always quadratic.
• is_pwl(Leaf): Leaf nodes are always piecewise linear.

Slots

id (Internal) A unique integer identification number used internally.
dim The dimensions of the leaf.
value The numeric value of the leaf.
nonneg Is the leaf nonnegative?
nonpos Is the leaf nonpositive?
complex Is the leaf a complex number?
imag Is the leaf imaginary?
symmetric Is the leaf a symmetric matrix?
diag  Is the leaf a diagonal matrix?
PSD  Is the leaf positive semidefinite?
NSD  Is the leaf negative semidefinite?
hermitian  Is the leaf hermitian?
boolean  Is the leaf boolean? Is the variable boolean? May be TRUE = entire leaf is boolean, FALSE = entire leaf is not boolean, or a vector of indices which should be constrained as boolean, where each index is a vector of length exactly equal to the length of dim.
integer  Is the leaf integer? The semantics are the same as the boolean argument.
sparsity  A matrix representing the fixed sparsity pattern of the leaf.
pos  Is the leaf strictly positive?
neg  Is the leaf strictly negative?

linearize  Affine Approximation to an Expression

Description
Gives an elementwise lower (upper) bound for convex (concave) expressions that is tight at the current variable/parameter values. No guarantees for non-DCP expressions.

Usage
linearize(expr)

Arguments
expr  An Expression to linearize.

Details
If f and g are convex, the objective f-g can be (heuristically) minimized using the implementation below of the convex-concave method:
for(iters in 1:N) solve(Problem(Minimize(f - linearize(g))))

Value
An affine expression or NA if cannot be linearized.
ListORConstr-class  A Class Union of List and Constraint

Description

A Class Union of List and Constraint

Usage

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

Arguments

object A list or Constraint object.

Methods (by generic)

• id(ListORConstr): Returns the ID associated with the list or constraint.

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 = 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**

```r
# 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_atom_log_log_convex(object)

## S4 method for signature 'Log'
is_atom_log_log_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'
.grad(object, values)

## S4 method for signature 'Log'
.domain(object)

Arguments

x An Expression or numeric constant.
object A Log object.
values A list of numeric values for the arguments
idx An index into the atom.

Methods (by generic)

- to_numeric(Log): The elementwise natural logarithm of the input value.
- sign_from_args(Log): The sign of the atom is unknown.
- is_atom_convex(Log): The atom is not convex.
- is_atom_concave(Log): The atom is concave.
Log1p-class

- is_atom_log_log_convex(Log): Is the atom log-log convex?
- is_atom_log_log_concave(Log): Is the atom log-log concave?
- is_incr(Log): The atom is weakly increasing.
- is_decr(Log): The atom is not weakly decreasing.
- .grad(Log): Gives the (sub/super)gradient of the atom w.r.t. each variable
- .domain(Log): Returns constraints describing the domain of the node

Slots

- x An Expression or numeric constant.

Description

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

Usage

Log1p(x)

```r
## S4 method for signature 'Log1p'
to_numeric(object, values)
## S4 method for signature 'Log1p'
sign_from_args(object)
## S4 method for signature 'Log1p'
.grad(object, values)
## S4 method for signature 'Log1p'
.domain(object)
```

Arguments

- x An Expression or numeric constant.
- object A Log1p object.
- values A list of numeric values for the arguments

Methods (by generic)

- to_numeric(Log1p): The elementwise natural logarithm of one plus the input value.
- sign_from_args(Log1p): The sign of the atom.
- .grad(Log1p): Gives the (sub/super)gradient of the atom w.r.t. each variable
- .domain(Log1p): Returns constraints describing the domain of the node
Slots

- An Expression or numeric constant.

Description

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

Usage

LogDet(A)

to_numeric(object, values)

validate_args(object)

dim_from_args(object)

sign_from_args(object)

is_atom_convex(object)

is_atom_concave(object)

is_incr(object, idx)

is_decr(object, idx)

.grad(object, values)

.domain(object)
Arguments

- **A**: An Expression or numeric matrix.
- **object**: A LogDet object.
- **values**: A list of numeric values for the arguments
- **idx**: An index into the atom.

Methods (by generic)

- **to_numeric(LogDet)**: 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 $A$.
- **validate_args(LogDet)**: Check that $A$ is square.
- **dim_from_args(LogDet)**: The atom is a scalar.
- **sign_from_args(LogDet)**: The atom is non-negative.
- **is_atom_convex(LogDet)**: The atom is not convex.
- **is_atom_concave(LogDet)**: The atom is concave.
- **is_incr(LogDet)**: The atom is not monotonic in any argument.
- **is_decr(LogDet)**: The atom is not monotonic in any argument.
- **.grad(LogDet)**: Gives the (sub/super)gradient of the atom w.r.t. each variable
- **.domain(LogDet)**: Returns constraints describing the domain of the node

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 $\log(\text{sum}(\exp))$ that evaluates to a vector rather than to a scalar, which is useful for logistic regression.

Usage

```r
logistic(x)
```

Arguments

- **x**: An Expression, vector, or matrix.

Value

An Expression representing the logistic function evaluated at the input.
Examples

```r
set.seed(92)
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 \( \log(\text{sum}(\exp)) \) that evaluates to a vector rather than to a scalar, which is useful for logistic regression.

Usage

Logistic(x)

```r
## 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'

.grad(object, values)

Arguments

x An Expression or numeric constant.

object A Logistic object.

values A list of numeric values for the arguments

idx An index into the atom.

Methods (by generic)

- to_numeric(Logistic): Evaluates \( e^x \) elementwise, adds one, and takes the natural logarithm.
- sign_from_args(Logistic): The atom is positive.
- is_atom_convex(Logistic): The atom is convex.
- is_atom_concave(Logistic): The atom is not concave.
- is_incr(Logistic): The atom is weakly increasing.
- is_decr(Logistic): The atom is not weakly decreasing.
- .grad(Logistic): Gives the (sub/super)gradient of the atom w.r.t. each variable

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_, keepdims = FALSE)

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

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

## S4 method for signature 'LogSumExp'
column_grad(object, value)

## 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)

### 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.
- **keepdims**: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.
- **object**: A `LogSumExp` object.
- **values**: A list of numeric values.
- **value**: A numeric value.
- **idx**: An index into the atom.

### Methods (by generic)

- **to_numeric(LogSumExp)**: Evaluates \( e^x \) elementwise, sums, and takes the natural log.
- **.grad(LogSumExp)**: Gives the (sub/super)gradient of the atom w.r.t. each variable
- **.column_grad(LogSumExp)**: Gives the (sub/super)gradient of the atom w.r.t. each column variable.
- **sign_from_args(LogSumExp)**: Returns sign (is positive, is negative) of the atom.
- **is_atom_convex(LogSumExp)**: The atom is convex.
- **is_atom_concave(LogSumExp)**: The atom is not concave.
- **is_incr(LogSumExp)**: The atom is weakly increasing in the index.
- **is_decr(LogSumExp)**: The atom is not weakly decreasing in the index.
\textit{log\_det}

\textbf{Slots}

- \texttt{x} An \texttt{Expression} representing a vector or matrix.
- \texttt{axis} (Optional) The dimension across which to apply the function: 1 indicates rows, 2 indicates columns, and \texttt{NA} indicates rows and columns. The default is \texttt{NA}.
- \texttt{keepdims} (Optional) Should dimensions be maintained when applying the atom along an axis? If \texttt{FALSE}, result will be collapsed into an \(n \times 1\) column vector. The default is \texttt{FALSE}.

\begin{center}
\begin{tabular}{ll}
\textbf{log\_det} & \textit{Log-Determinant} \\
\end{tabular}
\end{center}

\textbf{Description}

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

\textbf{Usage}

\texttt{log\_det(A)}

\textbf{Arguments}

- \texttt{A} An \texttt{Expression} or matrix.

\textbf{Value}

An \texttt{Expression} representing the log-determinant of the input.

\textbf{Examples}

\begin{verbatim}
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
\end{verbatim}
### log_log_curvature

**Log-Log Curvature of Expression**

**Description**

The log-log curvature of an expression.

**Usage**

```r
log_log_curvature(object)
```

#### Arguments

- `object`: An `Expression` object.

#### Value

A string indicating the log-log curvature of the expression, either "LOG_LOG_CONSTANT", "LOG_LOG_AFFINE", "LOG_LOG_CONVEX", "LOG_LOG_CONCAVE", or "UNKNOWN".

### log_log_curvature-atom

**Log-Log Curvature of an Atom**

**Description**

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

**Usage**

```r
is_atom_log_log_convex(object)

is_atom_log_log_concave(object)

is_atom_log_log_affine(object)
```

#### Arguments

- `object`: A `Atom` object.
Value

A logical value.

Log-Log Curvature Properties

Description

Determine if an expression is log-log constant, log-log affine, log-log convex, or log-log concave.

Usage

is_log_log_constant(object)

is_log_log_affine(object)

is_log_log_convex(object)

is_log_log_concave(object)

Arguments

object An Expression object.

Value

A logical value.

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_, keepdims = FALSE)
MatrixFrac-class

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.
- **keepdims**: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an \(nx1\) column vector. The default is FALSE.

Value

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

Examples

```r
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)
```

MatrixFrac-class

The MatrixFrac class.

Description

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

Usage

```r
MatrixFrac(X, P)
```

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

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

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

# S4 method for signature 'MatrixFrac'
dim_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'
is_qpwa(object)

## S4 method for signature 'MatrixFrac'
.domain(object)

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>An Expression or numeric matrix.</td>
</tr>
<tr>
<td>P</td>
<td>An Expression or numeric matrix.</td>
</tr>
<tr>
<td>object</td>
<td>A MatrixFrac object.</td>
</tr>
<tr>
<td>values</td>
<td>A list of numeric values for the arguments</td>
</tr>
<tr>
<td>idx</td>
<td>An index into the atom.</td>
</tr>
</tbody>
</table>

**Methods (by generic)**

- allow_complex(MatrixFrac): Does the atom handle complex numbers?
- to_numeric(MatrixFrac): The trace of $X^T P^{-1} X$.
- validate_args(MatrixFrac): Check that the dimensions of x and P match.
- dim_from_args(MatrixFrac): The atom is a scalar.
- sign_from_args(MatrixFrac): The atom is positive.
- is_atom_convex(MatrixFrac): The atom is convex.
- is_atom_concave(MatrixFrac): The atom is not concave.
- is_incr (MatrixFrac): The atom is not monotonic in any argument.
- is_decr (MatrixFrac): The atom is not monotonic in any argument.
- is_quadratic(MatrixFrac): True if x is affine and P is constant.
- is_qpwa (MatrixFrac): True if x is piecewise linear and P is constant.
- .domain (MatrixFrac): Returns constraints describing the domain of the node
- .grad (MatrixFrac): Gives the (sub/super)gradient of the atom w.r.t. each variable
MatrixStuffing-class

Description

The MatrixStuffing class.

Usage

## S4 method for signature 'MatrixStuffing,Problem'
perform(object, problem)

## S4 method for signature 'MatrixStuffing,Solution,InverseData'
invert(object, solution, inverse_data)

Arguments

- **object**: A MatrixStuffing object.
- **problem**: A Problem object to stuff; the arguments of every constraint must be affine.
- **solution**: A Solution to a problem that generated the inverse data.
- **inverse_data**: The data encoding the original problem.

Methods (by generic)

- **perform(object = MatrixStuffing, problem = Problem)**: Returns a stuffed problem. The returned problem is a minimization problem in which every constraint in the problem has affine arguments that are expressed in the form $A$

- **invert(object = MatrixStuffing, solution = Solution, inverse_data = InverseData)**: Returns the solution to the original problem given the inverse_data.

Slots

- X: An Expression or numeric matrix.
- P: An Expression or numeric matrix.
Description

\[ tr(X^TP^{-1}X). \]

Usage

\[ \text{matrix\_frac}(X, P) \]

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

```r
## Not run:
set.seed(192)
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 Properties

Description
Determine if an expression is positive semidefinite, negative semidefinite, hermitian, and/or symmetric.

Usage
- is_psd(object)
- is_nsd(object)
- is_hermitian(object)
- is_symmetric(object)

Arguments
- object: An Expression object.

Value
A logical value.

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.
MaxElemwise-class

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_atom_log_log_convex(object)

# S4 method for signature 'MaxElemwise'
is_atom_log_log_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'
.grad(object, values)
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 numeric values for the arguments
idx   An index into the atom.

Methods (by generic)

• to_numeric(MaxElemwise): The elementwise maximum.
• sign_from_args(MaxElemwise): The sign of the atom.
• is_atom_convex(MaxElemwise): The atom is convex.
• is_atom_concave(MaxElemwise): The atom is not concave.
• is_atom_log_log_convex(MaxElemwise): Is the atom log-log convex?
• is_atom_log_log_concave(MaxElemwise): Is the atom log-log concave?
• is_incr(MaxElemwise): The atom is weakly increasing.
• is_decr(MaxElemwise): The atom is not weakly decreasing.
• is_pwl(MaxElemwise): Are all the arguments piecewise linear?
• .grad(MaxElemwise): Gives the (sub/super)gradient of the atom w.r.t. each variable

Slots

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

Description

The maximum of an expression.
Usage

MaxEntries(x, axis = NA_real_, keepdims = FALSE)

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.
- **keepdims**: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.
- **object**: A MaxEntries object.
- **values**: A list of numeric values for the arguments
- **idx**: An index into the atom.
value A numeric value

Methods (by generic)

- `to_numeric(MaxEntries)`: The largest entry in x.
- `sign_from_args(MaxEntries)`: The sign of the atom.
- `is_atom_convex(MaxEntries)`: The atom is convex.
- `is_atom_concave(MaxEntries)`: The atom is not concave.
- `is_atom_log_log_convex(MaxEntries)`: Is the atom log-log convex.
- `is_atom_log_log_concave(MaxEntries)`: Is the atom log-log concave.
- `is_incr(MaxEntries)`: The atom is weakly increasing in every argument.
- `is_decr(MaxEntries)`: The atom is not weakly decreasing in any argument.
- `is_pwl(MaxEntries)`: Is \( x \) piecewise linear?
- `.grad(MaxEntries)`: Gives the (sub/super)gradient of the atom w.r.t. each variable
- `.column_grad(MaxEntries)`: Gives the (sub/super)gradient of the atom w.r.t. each column variable

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.
- `keepdims` (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an \( n \times 1 \) column vector. The default is FALSE.

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_dgp(object)
Arguments

expr A scalar Expression to maximize.
object A Maximize object.

Methods (by generic)

- canonicalize(Maximize): Negates the target expression's objective.
- is_dcp(Maximize): A logical value indicating whether the objective is concave.
- is_dgp(Maximize): A logical value indicating whether the objective is log-log concave.

Slots

expr A scalar Expression to maximize.

Examples

```r
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

```r
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**

```r
max_entries(x, axis = NA_real_, keepdims = FALSE)
```

```r
## 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.
- `keepdims`: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an nx1 column vector. The default is FALSE.
- `...`: 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**

```r
x <- Variable(2)
val <- matrix(c(-5,-10))
prob <- Problem(Minimize(max_entries(x)), list(x == val))
result <- solve(prob)
result$value
```

```r
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$value
x <- Variable(2)
val <- matrix(c(-5,-10))
prob <- Problem(Minimize(max_entries(x)), list(x == val))
result <- solve(prob)
result$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**

```r
## 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**

```r
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
```
MinElemwise-class

The MinElemwise class.

**Description**

This class represents the elementwise minimum.

**Usage**

MinElemwise(arg1, arg2, ...)

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

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

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

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

## S4 method for signature 'MinElemwise' is_atom_log_log_convex(object)

## S4 method for signature 'MinElemwise' is_atom_log_log_concave(object)

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

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

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

## S4 method for signature 'MinElemwise' .grad(object, values)

**Arguments**

- **arg1** The first Expression in the minimum operation.
- **arg2** The second Expression in the minimum operation.
- **...** Additional Expression objects in the minimum operation.
MinEntries-class

- **object**: A MinElemwise object.
- **values**: A list of numeric values for the arguments
- **idx**: An index into the atom.

### Methods (by generic)
- `to_numeric(MinElemwise)`: The elementwise minimum.
- `sign_from_args(MinElemwise)`: The sign of the atom.
- `is_atom_convex(MinElemwise)`: The atom is not convex.
- `is_atom_concave(MinElemwise)`: The atom is not concave.
- `is_atom_log_log_convex(MinElemwise)`: Is the atom log-log convex?
- `is_atom_log_log_concave(MinElemwise)`: Is the atom log-log concave?
- `is_incr(MinElemwise)`: The atom is weakly increasing.
- `is_decr(MinElemwise)`: The atom is not weakly decreasing.
- `is_pwl(MinElemwise)`: Are all the arguments piecewise linear?
- `.grad(MinElemwise)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

### Slots
- **arg1**: The first Expression in the minimum operation.
- **arg2**: The second Expression in the minimum operation.
- **...**: Additional Expression objects in the minimum operation.

---

**MinEntries-class**

*The MinEntries class.*

### Description
The minimum of an expression.

### Usage
```
MinEntries(x, axis = NA_real_, keepdims = FALSE)
```

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

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

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

```
## S4 method for signature 'MinEntries'
is_atom_concave(object)
```

```
## S4 method for signature 'MinEntries'
.is_pwl(MinElemwise)
```

```
## S4 method for signature 'MinEntries'
.grad(MinElemwise)
```
is_atom_concave(object)

## S4 method for signature 'MinEntries'
is_atom_log_log_convex(object)

## S4 method for signature 'MinEntries'
is_atom_log_log_concave(object)

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

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

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

## S4 method for signature 'MinEntries'
.grad(object, values)

## S4 method for signature 'MinEntries'
column_grad(object, value)

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.

keepdims
(Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.

object
A MinEntries object.

values
A list of numeric values for the arguments

idx
An index into the atom.

value
A numeric value

Methods (by generic)

• to_numeric(MinEntries): The largest entry in x.
• sign_from_args(MinEntries): The sign of the atom.
• is_atom_convex(MinEntries): The atom is not convex.
• is_atom_concave(MinEntries): The atom is concave.
• is_atom_log_log_convex(MinEntries): Is the atom log-log convex?
• is_atom_log_log_concave(MinEntries): Is the atom log-log concave?
• is_incr(MinEntries): The atom is weakly increasing in every argument.
Minimize-class

- `is_decr(MinEntries)`: The atom is not weakly decreasing in any argument.
- `is_pwl(MinEntries)`: Is x piecewise linear?
- `.grad(MinEntries)`: Gives the (sub/super)gradient of the atom w.r.t. each variable
- `.column_grad(MinEntries)`: Gives the (sub/super)gradient of the atom w.r.t. each column variable

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.
keepdims (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.

---

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'
is_dcp(object)

## S4 method for signature 'Minimize'
is_dgp(object)

Arguments

expr A scalar Expression to minimize.
object A Minimize object.

Methods (by generic)

- canonicalize(Minimize): Pass on the target expression’s objective and constraints.
- is_dcp(Minimize): A logical value indicating whether the objective is convex.
- is_dgp(Minimize): A logical value indicating whether the objective is log-log convex.

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**

```r
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_, keepdims = FALSE)`

```r
## 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.
- **keepdims** (Optional): Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n.x.1 column vector. The default is FALSE.
- **...**: 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

```r
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$value
A <- Variable(2,2)
val <- cbind(c(-5,2), c(-3,1))
prob <- Problem(Maximize(min_entries(A)), list(A == val))
result <- solve(prob)
result$value
```

---

### mip_capable

**Solver Capabilities**

**Description**

Determine if a solver is capable of solving a mixed-integer program (MIP).

**Usage**

`mip_capable(solver)`

**Arguments**

- **solver**: A ReductionSolver object.

**Value**

A logical value.
Examples
mip_capable(ECOS())

MixedNorm

Description
The \( l_{p,q} \) norm of \( X \), \( (\sum_k (\sum_l |X_{k,l}|^p)^{q/p})^{1/q} \).

Usage
MixedNorm(X, p = 2, q = 1)

Arguments
- \( X \) : The matrix to take the \( l_{p,q} \) norm of.
- \( p \) : The type of inner norm.
- \( q \) : The type of outer norm.

Value
Returns the mixed norm of \( X \) with specified parameters \( p \) and \( q \).

mixed_norm

Description
\[ l_{p,q}(x) = \left( \sum_{i=1}^{n} \left( \sum_{j=1}^{m} |x_{i,j}|^{q/p} \right)^{p/q} \right)^{1/q}. \]

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

```r
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

An interface for the MOSEK solver.

Description

An interface for the MOSEK solver.

Usage

```r
MOSEK()

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

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

## S4 method for signature 'MOSEK'
name(x)

## S4 method for signature 'MOSEK,Problem'
accepts(object, problem)

## S4 method for signature 'MOSEK'
block_format(object, problem, constraints, exp_cone_order = NA)

## S4 method for signature 'MOSEK,Problem'
perform(object, problem)

## S4 method for signature 'MOSEK'
solve_via_data(
    object, data,
)```
warm_start,
verbose,
feastol,
reltol,
abstol,
num_iter,
solver_opts,
solver_cache
)

## S4 method for signature 'MOSEK,ANY,ANY'
invert(object, solution, inverse_data)

Arguments

solver, object, x
A MOSEK object.

problem A Problem object.

constraints A list of Constraint objects for which coefficient andd offset data ("G", "h" re-
spectively) is needed

exp_cone_order A parameter that is only used when a Constraint object describes membership
in the exponential cone.

data Data generated via an apply call.

warm_start A boolean of whether to warm start the solver.

verbose A boolean of whether to enable solver verbosity.

feastol The feasible tolerance.

reltol The relative tolerance.

abstol The absolute tolerance.

num_iter The maximum number of iterations.

solver_opts A list of Solver specific options

solver_cache Cache for the solver.

solution The raw solution returned by the solver.

inverse_data A list containing data necessary for the inversion.

Methods (by generic)

- `mip_capable(MOSEK)`: Can the solver handle mixed-integer programs?
- `import_solver(MOSEK)`: Imports the solver.
- `name(MOSEK)`: Returns the name of the solver.
- `accepts(object = MOSEK, problem = Problem)`: Can MOSEK solve the problem?
- `block_format(MOSEK)`: Returns a large matrix "coeff" and a vector of constants "offset" such
  that every Constraint in "constraints" holds at z in R^n iff "coeff" * z <=_K offset", where
  K is a product of cones supported by MOSEK and CVXR (zero cone, nonnegative orthant,
  second order cone, exponential cone). The nature of K is inferred later by accessing the data
  in "lengths" and "ids".
**MOSEK.parse_dual_vars**

Parses MOSEK dual variables into corresponding CVXR constraints and dual values

**Description**

Parses MOSEK dual variables into corresponding CVXR constraints and dual values

**Usage**

```r
MOSEK.parse_dual_vars(dual_var, constr_id_to_constr_dim)
```

**Arguments**

- `dual_var`: List of the dual variables returned by the MOSEK solution.
- `constr_id_to_constr_dim`: A list that contains the mapping of entry "id" that is the index of the CVXR `Constraint` object to which the next "dim" entries of the dual variable belong.

**Value**

A list with the mapping of the CVXR `Constraint` object indices with the corresponding dual values.

**MOSEK.recover_dual_variables**

Recovering MOSEK solutions dual variables

**Description**

Recovering MOSEK solutions dual variables

**Usage**

```r
MOSEK.recover_dual_variables(sol, inverse_data)
```

**Arguments**

- `sol`: List of the solutions returned by the MOSEK solver.
- `inverse_data`: A list of the data returned by the perform function.
Value

A list containing the mapping of CVXR’s Constraint object’s id to its corresponding dual variables in the current solution.

multiply

Elementwise Multiplication

Description

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

Usage

multiply(lh_exp, rh_exp)

Arguments

lh_exp  An 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 <- multiply(c, A)
obj <- Minimize(norm_inf(expr))
prob <- Problem(obj, list(A == 5))
result <- solve(prob)
result$value
result$getValue(expr)

Multiply-class

The Multiply class.

Description

This class represents the elementwise product of two expressions.
Multiply-class

Usage

Multiply(lh_exp, rh_exp)

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

## S4 method for signature 'Multiply'
dim_from_args(object)

## S4 method for signature 'Multiply'
is_atom_log_log_convex(object)

## S4 method for signature 'Multiply'
is_atom_log_log_concave(object)

## S4 method for signature 'Multiply'
is_psd(object)

## S4 method for signature 'Multiply'
is_nsd(object)

## S4 method for signature 'Multiply'
graph_implementation(object, arg_objs, dim, data = NA_real_)

Arguments

- **lh_exp**: An `Expression` or R numeric data.
- **rh_exp**: An `Expression` or R numeric data.
- **object**: A `Multiply` object.
- **values**: A list of arguments to the atom.
- **arg_objs**: A list of linear expressions for each argument.
- **dim**: A vector representing the dimensions of the resulting expression.
- **data**: A list of additional data required by the atom.

Methods (by generic)

- `to_numeric(Multiply)`: Multiplies the values elementwise.
- `dim_from_args(Multiply)`: The sum of the argument dimensions - 1.
- `is_atom_log_log_convex(Multiply)`: Is the atom log-log convex?
- `is_atom_log_log_concave(Multiply)`: Is the atom log-log concave?
- `is_psd(Multiply)`: Is the expression a positive semidefinite matrix?
- `is_nsd(Multiply)`: Is the expression a negative semidefinite matrix?
- `graph_implementation(Multiply)`: The graph implementation of the expression.
**name**  
*Variable, Parameter, or Expression Name*

**Description**

The string representation of a variable, parameter, or expression.

**Usage**

```r
name(x)
```

**Arguments**

- `x`  
  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**

```r
x <- Variable()
y <- Variable(3, name = "yVar")

name(x)
nome(y)
```

---

**Neg**  
*An alias for -MinElemwise(x, 0)*

**Description**

An alias for -MinElemwise(x, 0)

**Usage**

```r
Neg(x)
```

**Arguments**

- `x`  
  An R numeric value or Expression.

**Value**

An alias for -MinElemwise(x, 0)
Elementwise Negative

Description

The elementwise absolute negative portion of an expression, $-\min(x, 0)$. This is equivalent to $-\text{min\_elemwise}(x, 0)$.

Usage

neg(x)

Arguments

x An Expression, vector, or matrix.

Value

An Expression representing the negative portion of the input.

Examples

```r
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_, id = NA_integer_)

Arguments

f A nonlinear function.
vars_ A list of variables involved in the function.
id (Optional) An integer representing the unique ID of the constraint.
NonPosConstraint-class

Slots

  f  A nonlinear function.
  vars_  A list of variables involved in the function.
  .x_dim (Internal) The dimensions of a column vector with number of elements equal to the total
       elements in all the variables.

NonPosConstraint-class

The NonPosConstraint class

Description

The NonPosConstraint class

Usage

## S4 method for signature 'NonPosConstraint'
name(x)

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

## S4 method for signature 'NonPosConstraint'
is_dgp(object)

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

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

Arguments

  x, object  A NonPosConstraint object.

Methods (by generic)

- name(NonPosConstraint): The string representation of the constraint.
- is_dcp(NonPosConstraint): Is the constraint DCP?
- is_dgp(NonPosConstraint): Is the constraint DGP?
- canonicalize(NonPosConstraint): The graph implementation of the object.
- residual(NonPosConstraint): The residual of the constraint.
**Norm**

The Norm atom.

### Description

Wrapper around the different norm atoms.

### Usage

```r
Norm(x, p = 2, axis = NA_real_, keepdims = FALSE)
```

### Arguments

- **x**: The matrix to take the norm of
- **p**: The type of norm. Valid options include any positive integer, 'fro' (for Frobenius), 'nuc' (sum of singular values), np.inf or 'inf' (infinity norm).
- **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.
- **keepdims**: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an nx1 column vector. The default is FALSE.

### Value

Returns the specified norm of 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

```r
## 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

```r
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
```

<table>
<thead>
<tr>
<th><strong>norm1</strong></th>
<th>1-Norm</th>
</tr>
</thead>
</table>

Description

\[ \|x\|_1 = \sum_{i=1}^n |x_i|. \]

Usage

`norm1(x, axis = NA_real_, keepdims = 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.
- **keepdims**  
  (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.
Value

An Expression representing the 1-norm of the input.

Examples

```r
a <- Variable()
prob <- Problem(Minimize(norm1(a)), list(a <= -2))
result <- solve(prob)
result$value
result$getValue(a)
```

```r
prob <- Problem(Maximize(-norm1(a)), list(a <= -2))
result <- solve(prob)
result$value
result$getValue(a)
```

```r
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])
```

---

**Norm1-class**

*The Norm1 class.*

Description

This class represents the 1-norm of an expression.

Usage

```r
Norm1(x, axis = NA_real_, keepdims = FALSE)
```

## S4 method for signature 'Norm1'
name(x)

## S4 method for signature 'Norm1'
to_numeric(object, values)

## S4 method for signature 'Norm1'
allow_complex(object)

## S4 method for signature 'Norm1'
sign_from_args(object)

## S4 method for signature 'Norm1'
is_atom_convex(object)
## S4 method for signature 'Norm1'
is_atom_concave(object)

## S4 method for signature 'Norm1'
is_incr(object, idx)

## S4 method for signature 'Norm1'
is_decr(object, idx)

## S4 method for signature 'Norm1'
is_pwl(object)

## S4 method for signature 'Norm1'
get_data(object)

## S4 method for signature 'Norm1'
domain(object)

## S4 method for signature 'Norm1'
.grad(object, values)

## S4 method for signature 'Norm1'
column_grad(object, value)

### Arguments

- **x**: An `Expression` object.
- **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.
- **keepdims**: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.
- **object**: A `Norm1` object.
- **values**: A list of numeric values for the arguments
- **idx**: An index into the atom.
- **value**: A numeric value

### Methods (by generic)

- `name(Norm1)`: The name and arguments of the atom.
- `to_numeric(Norm1)`: Returns the 1-norm of x along the given axis.
- `allow_complex(Norm1)`: Does the atom handle complex numbers?
- `sign_from_args(Norm1)`: The atom is always positive.
- `is_atom_convex(Norm1)`: The atom is convex.
- `is_atom_concave(Norm1)`: The atom is not concave.
- **isincr(Norm1)**: Is the composition weakly increasing in argument idx?
- **isdecr(Norm1)**: Is the composition weakly decreasing in argument idx?
- **ispwl(Norm1)**: Is the atom piecewise linear?
- **get_data(Norm1)**: Returns the axis.
- **.domain(Norm1)**: Returns constraints describing the domain of the node
- **.grad(Norm1)**: Gives the (sub/super)gradient of the atom w.r.t. each variable
- **.column_grad(Norm1)**: Gives the (sub/super)gradient of the atom w.r.t. each column variable

**Slots**

- **x** An **Expression** object.

---

**Norm2**

*The Norm2 atom.*

**Description**

The 2-norm of an expression.

**Usage**

Norm2(x, axis = NA_real_, keepdims = FALSE)

**Arguments**

- **x** An **Expression** object.
- **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.
- **keepdims** (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.

**Value**

Returns the 2-norm of x.
**Description**

\[ \|x\|_2 = \left( \sum_{i=1}^{n} x_i^2 \right)^{1/2}. \]

**Usage**

\[
\text{norm2}(x, \text{axis} = \text{NA}_\text{real}_-, \text{keepdims} = \text{FALSE})
\]

**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.
- **keepdims**
  - (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an \(nx1\) column vector. The default is FALSE.

**Value**

An **Expression** representing the Euclidean norm of the input.

**Examples**

```r
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)
```
The NormInf class.

Description

This class represents the infinity-norm.

Usage

```r
## S4 method for signature 'NormInf'
name(x)

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

## S4 method for signature 'NormInf'
allow_complex(object)

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

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

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

## S4 method for signature 'NormInf'
is_atom_log_log_convex(object)

## S4 method for signature 'NormInf'
is_atom_log_log_concave(object)

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

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

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

## S4 method for signature 'NormInf'
get_data(object)
```
## S4 method for signature 'NormInf'
.domain(object)

## S4 method for signature 'NormInf'
.grad(object, values)

## S4 method for signature 'NormInf'
.column_grad(object, value)

### Arguments

- **x**, **object**
  A `NormInf` object.

- **values**
  A list of numeric values for the arguments

- **idx**
  An index into the atom.

- **value**
  A numeric value

### Methods (by generic)

- **name(NormInf)**: The name and arguments of the atom.
- **to_numeric(NormInf)**: Returns the infinity norm of `x`.
- **allow_complex(NormInf)**: Does the atom handle complex numbers?
- **sign_from_args(NormInf)**: The atom is always positive.
- **is_atom_convex(NormInf)**: The atom is convex.
- **is_atom_concave(NormInf)**: The atom is not concave.
- **is_atom_log_log_convex(NormInf)**: Is the atom log-log convex?
- **is_atom_log_log_concave(NormInf)**: Is the atom log-log concave?
- **is_incr(NormInf)**: Is the composition weakly increasing in argument `idx`?
- **is_decr(NormInf)**: Is the composition weakly decreasing in argument `idx`?
- **is_pwl(NormInf)**: Is the atom piecewise linear?
- **get_data(NormInf)**: Returns the axis.
- **.domain(NormInf)**: Returns constraints describing the domain of the node
- **.grad(NormInf)**: Gives the (sub/super)gradient of the atom w.r.t. each variable
- **.column_grad(NormInf)**: Gives the (sub/super)gradient of the atom w.r.t. each column variable
**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)`
- `allow_complex(object)`
- `dim_from_args(object)`
- `sign_from_args(object)`
- `is_atom_convex(object)`
- `is_atom_concave(object)`
- `is_incr(object, idx)`
- `is_decr(object, idx)`
- `.grad(object, values)`

**Arguments**

- **A**
  - An Expression or numeric matrix.
- **object**
  - A NormNuc object.
- **values**
  - A list of numeric values for the arguments
- **idx**
  - An index into the atom.
Methods (by generic)

- `to_numeric(NormNuc)`: The nuclear norm (i.e., the sum of the singular values) of $A$.
- `allow_complex(NormNuc)`: Does the atom handle complex numbers?
- `dim_from_args(NormNuc)`: The atom is a scalar.
- `sign_from_args(NormNuc)`: The atom is positive.
- `is_atom_convex(NormNuc)`: The atom is convex.
- `is_atom_concave(NormNuc)`: The atom is not concave.
- `is_incr(NormNuc)`: The atom is not monotonic in any argument.
- `is_decr(NormNuc)`: The atom is not monotonic in any argument.
- `.grad(NormNuc)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

Slots

- `A` An Expression or numeric matrix.

---

<table>
<thead>
<tr>
<th>norm_inf</th>
<th>Infinity-Norm</th>
</tr>
</thead>
</table>

Description

$$\|x\|_{\infty} = \max_{i=1,...,n} |x_i|.$$  

Usage

`norm_inf(x, axis = NA_real_, keepdims = 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.
- `keepdims`: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an $n \times 1$ column vector. The default is FALSE.

Value

An Expression representing the infinity-norm of the input.
Examples

```r
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])
```

---

**Description**

The nuclear norm, i.e. sum of the singular values of a matrix.

**Usage**

```r
norm_nuc(A)
```

**Arguments**

- `A` An Expression or matrix.

**Value**

An Expression representing the nuclear norm of the input.
Example

```r
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

```r
## S4 method for signature 'Objective,numeric'
e1 + e2

## S4 method for signature 'numeric,Objective'
e1 + e2

## 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 'Objective,Minimize'
e1 - e2

## S4 method for signature 'Objective,Maximize'
e1 - e2

## S4 method for signature 'Minimize,Objective'
e1 - e2

## S4 method for signature 'Maximize,Objective'
e1 - e2

## S4 method for signature 'Objective,numeric'
e1 - e2

## S4 method for signature 'numeric,Objective'
e1 - e2
```
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 'numeric,Maximize'
e1 * e2

## S4 method for signature 'Objective,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.

Description

This class represents an optimization objective.

Usage

Objective(expr)

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

## S4 method for signature 'Objective'

is_quadratic(object)

## S4 method for signature 'Objective'

is_qpwa(object)

### Arguments

- **expr**  
  A scalar `Expression` to optimize.

- **object**  
  An `Objective` object.

### Methods (by generic)

- `value(Objective)`: The value of the objective expression.
- `is_quadratic(Objective)`: Is the objective a quadratic function?
- `is qpwa(Objective)`: Is the objective a quadratic of piecewise affine function?

### Slots

- **expr**  
  A scalar `Expression` to optimize.

---

**OneMinusPos-class**  
*The OneMinusPos class.*

### Description

This class represents the difference $1 - x$ with domain $\{x : 0 < x < 1\}$

### Usage

```r
OneMinusPos(x)
```

```r
## S4 method for signature 'OneMinusPos'
name(x)
```

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

```r
## S4 method for signature 'OneMinusPos'
dim_from_args(object)
```

```r
## S4 method for signature 'OneMinusPos'
sign_from_args(object)
```
OneMinusPos-class

is_atom_convex(object)
## S4 method for signature 'OneMinusPos'
is_atom_concave(object)
## S4 method for signature 'OneMinusPos'
is_atom_log_log_convex(object)
## S4 method for signature 'OneMinusPos'
is_atom_log_log_concave(object)
## S4 method for signature 'OneMinusPos'
is_incr(object, idx)
## S4 method for signature 'OneMinusPos'
is_decr(object, idx)
## S4 method for signature 'OneMinusPos'
.grad(object, values)

Methods (by generic)

• name(OneMinusPos): The name and arguments of the atom.
• to_numeric(OneMinusPos): Returns one minus the value.
• dim_from_args(OneMinusPos): The dimensions of the atom.
• sign_from_args(OneMinusPos): Returns the sign (is positive, is negative) of the atom.
• is_atom_convex(OneMinusPos): Is the atom convex?
• is_atom_concave(OneMinusPos): Is the atom concave?
• is_atom_log_log_convex(OneMinusPos): Is the atom log-log convex?
• is_atom_log_log_concave(OneMinusPos): Is the atom log-log concave?
• is_incr(OneMinusPos): Is the atom weakly increasing in the argument idx?
• is_decr(OneMinusPos): Is the atom weakly decreasing in the argument idx?
• .grad(OneMinusPos): Gives the (sub/super)gradient of the atom w.r.t. each variable

Arguments

x An Expression or numeric matrix.
object A OneMinusPos object.
values A list of numeric values for the arguments
idx An index into the atom.

Slots

x An Expression or numeric matrix.
**one_minus_pos**  
*Difference on Restricted Domain*

### Description

The difference $1 - x$ with domain $\{x : 0 < x < 1\}$.

### Usage

`one_minus_pos(x)`

### Arguments

- **x**  
  An *Expression*, vector, or matrix.

### Details

This atom is log-log concave.

### Value

An *Expression* representing one minus the input restricted to $(0, 1)$.

### Examples

```r
x <- Variable(pos = TRUE)  
y <- Variable(pos = TRUE)  
prob <- Problem(Maximize(one_minus_pos(x*y)), list(x <= 2 * y^2, y >= .2))  
result <- solve(prob, gp = TRUE)  
result$value  
result$getValue(x)  
result$getValue(y)
```

---

**OSQP-class**  
*An interface for the OSQP solver.*

### Description

An interface for the OSQP solver.
Usage

OSQP()

## S4 method for signature 'OSQP'
status_map(solver, status)

## S4 method for signature 'OSQP'
name(x)

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

## S4 method for signature 'OSQP,list,InverseData'
invert(object, solution, inverse_data)

## S4 method for signature 'OSQP'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  retol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

Arguments

solver, object, x
  A OSQP object.
status
  A status code returned by the solver.
solution
  The raw solution returned by the solver.
inverse_data
  A InverseData object containing data necessary for the inversion.
data
  Data generated via an apply call.
warm_start
  A boolean of whether to warm start the solver.
verbose
  A boolean of whether to enable solver verbosity.
feastol
  The feasible tolerance.
retol
  The relative tolerance.
abstol
  The absolute tolerance.
num_iter
  The maximum number of iterations.
solver_opts
  A list of Solver specific options
solver_cache
  Cache for the solver.
Methods (by generic)

- status_map(OSQP): Converts status returned by the OSQP solver to its respective CVXPY status.
- name(OSQP): Returns the name of the solver.
- import_solver(OSQP): Imports the solver.
- invert(object = OSQP, solution = list, inverse_data = InverseData): Returns the solution to the original problem given the inverse_data.
- solve_via_data(OSQP): Solve a problem represented by data returned from apply.

Parameter-class

The Parameter class.

Description

This class represents a parameter, either scalar or a matrix.

Usage

Parameter(
  rows = NULL,
  cols = NULL,
  name = NA_character_,
  value = NA_real_,
  ...
)

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

## S4 method for signature 'Parameter'
name(x)

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

## S4 replacement method for signature 'Parameter'
value(object) <- value

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

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

## S4 method for signature 'Parameter'
canonicalize(object)
Parameter-class

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.
value (Optional) A numeric element, vector, matrix, or data.frame. Defaults to NA and may be changed with value<- later.
... Additional attribute arguments. See Leaf for details.
object, x A Parameter object.

Methods (by generic)

• get_data(Parameter): Returns list(dim, name, value, attributes).
• name(Parameter): The name of the parameter.
• value(Parameter): The value of the parameter.
• value(Parameter) <- value: Set the value of the parameter.
• grad(Parameter): An empty list since the gradient of a parameter is zero.
• parameters(Parameter): Returns itself as a parameter.
• canonicalize(Parameter): The canonical form of the parameter.

Slots

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.
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", nonpos = TRUE) ## 3-vec negative
is_nonneg(x)
is_nonpos(x)
size(x)
perform  

Perform Reduction

Description

Performs the reduction on a problem and returns an equivalent problem.

Usage

perform(object, problem)

Arguments

object  
A Reduction object.

problem  
A Problem on which the reduction will be performed.

Value

A list containing

"problem"  
A Problem or list representing the equivalent problem.

"inverse_data"  
A InverseData or list containing the data needed to invert this particular reduction.

PfEigenvalue-class

The PfEigenvalue class.

Description

This class represents the Perron-Frobenius eigenvalue of a positive matrix.

Usage

PfEigenvalue(X)

## S4 method for signature 'PfEigenvalue'
name(x)

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

## S4 method for signature 'PfEigenvalue'
dim_from_args(object)

## S4 method for signature 'PfEigenvalue'
sign_from_args(object)
## S4 method for signature 'PfEigenvalue'
is_atom_convex(object)

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

## S4 method for signature 'PfEigenvalue'
is_atom_log_log_convex(object)

## S4 method for signature 'PfEigenvalue'
is_atom_log_log_concave(object)

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

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

## S4 method for signature 'PfEigenvalue'
.grad(object, values)

### Arguments

- **X**: An Expression or numeric matrix.
- **x, object**: A PfEigenvalue object.
- **values**: A list of numeric values for the arguments
- **idx**: An index into the atom.

### Methods (by generic)

- `name(PfEigenvalue)`: The name and arguments of the atom.
- `to_numeric(PfEigenvalue)`: Returns the Perron-Frobenius eigenvalue of $X$. 
- `dim_from_args(PfEigenvalue)`: The dimensions of the atom.
- `sign_from_args(PfEigenvalue)`: Returns the sign (is positive, is negative) of the atom.
- `is_atom_convex(PfEigenvalue)`: Is the atom convex?
- `is_atom_concave(PfEigenvalue)`: Is the atom concave?
- `is_atom_log_log_convex(PfEigenvalue)`: Is the atom log-log convex?
- `is_atom_log_log_concave(PfEigenvalue)`: Is the atom log-log concave?
- `is_incr(PfEigenvalue)`: Is the atom weakly increasing in the argument idx?
- `is_decr(PfEigenvalue)`: Is the atom weakly decreasing in the argument idx?
- `.grad(PfEigenvalue)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

### Slots

- **X**: An Expression or numeric matrix.
pf_eigenvalue  

Perron-Frobenius Eigenvalue

Description

The Perron-Frobenius eigenvalue of a positive matrix.

Usage

pf_eigenvalue(X)

Arguments

X  An Expression or positive square matrix.

Details

For an elementwise positive matrix X, this atom represents its spectral radius, i.e., the magnitude of its largest eigenvalue. Because X is positive, the spectral radius equals its largest eigenvalue, which is guaranteed to be positive.

This atom is log-log convex.

Value

An Expression representing the largest eigenvalue of the input.

Examples

n <- 3
X <- Variable(n, n, pos=TRUE)
objective_fn <- pf_eigenvalue(X)
constraints <- list( X[1,1] == 1.0,
                     X[1,3] == 1.9,
                     X[2,2] == .8,
                     X[3,1] == 3.2,
                     X[3,2] == 5.9,
problem <- Problem(Minimize(objective_fn), constraints)
result <- solve(problem, gp=TRUE)
result$value
result$getValue(X)
The Pnorm class.

Description

This class represents the vector p-norm.

Usage

Pnorm(x, p = 2, axis = NA_real_, keepdims = FALSE, max_denom = 1024)

## S4 method for signature 'Pnorm'
allow_complex(object)

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

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

## 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_atom_log_log_convex(object)

## S4 method for signature 'Pnorm'
is_atom_log_log_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)
## 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.
- **keepdims**: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an nx1 column vector. The default is FALSE.
- **max_denom**: (Optional) The maximum denominator considered in forming a rational approximation for \( p \). The default is 1024.
- **object**: A `Pnorm` object.
- **values**: A list of numeric values for the arguments
- **idx**: An index into the atom.
- **value**: A numeric value

## Details

If given a matrix variable, \( \text{Pnorm} \) 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 \( x \in \mathbb{R}^n \). 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 \( x \in \mathbb{R}_{++}^n \).

- 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)

• allow_complex(Pnorm): Does the atom handle complex numbers?
• to_numeric(Pnorm): The p-norm of \( x \).
• validate_args(Pnorm): Check that the arguments are valid.
• sign_from_args(Pnorm): The atom is positive.
• is_atom_convex(Pnorm): The atom is convex if \( p \geq 1 \).
• is_atom_concave(Pnorm): The atom is concave if \( p < 1 \).
• is_atom_log_log_convex(Pnorm): Is the atom log-log convex?
• is_atom_log_log_concave(Pnorm): Is the atom log-log concave?
• is_incr(Pnorm): The atom is weakly increasing if \( p < 1 \) or \( p > 1 \) and \( x \) is positive.
• is_decr(Pnorm): The atom is weakly decreasing if \( p > 1 \) and \( x \) is negative.
• is_pwl(Pnorm): The atom is not piecewise linear unless \( p = 1 \) or \( p = \infty \).
• get_data(Pnorm): Returns list\((p, \text{axis})\).
• name(Pnorm): The name and arguments of the atom.
• .domain(Pnorm): Returns constraints describing the domain of the node
• .grad(Pnorm): Gives the (sub/super)gradient of the atom w.r.t. each variable
• .column_grad(Pnorm): Gives the (sub/super)gradient of the atom w.r.t. each column variable

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.
keepdims (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an \( n \times 1 \) column vector. The default is FALSE.
.approx_error (Internal) The absolute difference between \( p \) and its rational approximation.
.original_p (Internal) The original input \( p \).

---

Pos  An alias for MaxElemwise\((x, 0)\)

Description
An alias for MaxElemwise\((x, 0)\)

Usage
Pos\((x)\)
Arguments

\( x \)  
An R numeric value or Expression.

Value

An alias for MaxElemwise\( (x, 0) \)

---

\( pos \)  
Elementwise Positive

---

Description

The elementwise positive portion of an expression, \( \max(x_i, 0) \). This is equivalent to \( \text{max}_\text{elemwise}(x, 0) \).

Usage

\( \text{pos}(x) \)

Arguments

\( x \)  
An Expression, vector, or matrix.

Value

An Expression representing the positive portion of the input.

Examples

\[
\begin{align*}
x & \leftarrow \text{Variable}(2) \\
\text{val} & \leftarrow \text{matrix}(c(-3,2)) \\
\text{prob} & \leftarrow \text{Problem} (\text{Minimize} (\text{pos}(x)[1]), \text{list}(x == \text{val})) \\
\text{result} & \leftarrow \text{solve} (\text{prob}) \\
\text{result}$\text{value} & \\
\end{align*}
\]

---

Power-class  
The Power class.

---

Description

This class represents the elementwise power function \( f(x) = x^p \). If \( \text{expr} \) is a CVXR expression, then \( \text{expr}^p \) is equivalent to \( \text{Power} (\text{expr}, p) \).
**Usage**

```r
Power(x, p, max_denom = 1024)
```

## 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_atom_log_log_convex(object)

## S4 method for signature 'Power'
is_atom_log_log_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'
is_qpwa(object)

## S4 method for signature 'Power'
.grad(object, values)

## S4 method for signature 'Power'
.domain(object)

## S4 method for signature 'Power'
.get_data(object)

## S4 method for signature 'Power'
copy(object, args = NULL, id_objects = list())
## S4 method for signature 'Power'

name(x)

### 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 numeric values for the arguments
- **idx**: An index into the atom.
- **args**: A list of arguments to reconstruct the atom. If args=NULL, use the current args of the atom
- **id_objects**: Currently unused.

### Details

For \( p = 0 \), \( f(x) = 1 \), constant, positive.

For \( p = 1 \), \( f(x) = x \), affine, increasing, same sign as \( x \).

For \( p = 2, 4, 8, \ldots \), \( f(x) = |x|^p \), convex, signed monotonicity, positive.

For \( p < 0 \) and \( f(x) =

\[ x^p \text{ for } x > 0 \]
\[ +\infty \text{ for } x \leq 0 \]

, this function is convex, decreasing, and positive.

For \( 0 < p < 1 \) and \( f(x) =

\[ x^p \text{ for } x \geq 0 \]
\[ -\infty \text{ for } x < 0 \]

, this function is concave, increasing, and positive.

For \( p > 1, p \neq 2, 4, 8, \ldots \) and \( f(x) =

\[ x^p \text{ for } x \geq 0 \]
\[ +\infty \text{ for } x < 0 \]

, this function is convex, increasing, and positive.
Methods (by generic)

- to_numeric(Power): Throw an error if the power is negative and cannot be handled.
- sign_from_args(Power): The sign of the atom.
- is_atom_convex(Power): Is \( p \leq 0 \) or \( p \geq 1 \)?
- is_atom_concave(Power): Is \( p \geq 0 \) or \( p \leq 1 \)?
- is_atom_log_log_convex(Power): Is the atom log-log convex?
- is_atom_log_log_concave(Power): Is the atom log-log concave?
- is_constant(Power): A logical value indicating whether the atom is constant.
- is_incr(Power): A logical value indicating whether the atom is weakly increasing.
- is_decr(Power): A logical value indicating whether the atom is weakly decreasing.
- is_quadratic(Power): A logical value indicating whether the atom is quadratic.
- is_qpwa(Power): A logical value indicating whether the atom is quadratic of piecewise affine.
- .grad(Power): Gives the (sub/super)gradient of the atom w.r.t. each variable
- .domain(Power): Returns constraints describing the domain of the node
- get_data(Power): A list containing the output of pow_low, pow_mid, or pow_high depending on the input power.
- copy(Power): Returns a shallow copy of the power atom
- name(Power): Returns the expression in string form.

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**

```r
# 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'
status(object)

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

## S4 method for signature 'Problem'
is_dgp(object)

## S4 method for signature 'Problem'
is_qp(object)

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

## S4 method for signature 'Problem'
is_mixed_integer(object)

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

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

## S4 method for signature 'Problem'
references(object)

## S4 method for signature 'Problem'
atoms(object)

## S4 method for signature 'Problem'
Problem-class

size_metrics(object)
## S4 method for signature 'Problem'
solver_stats(object)
## S4 replacement method for signature 'Problem'
solver_stats(object) <- value
## S4 method for signature 'Problem,character,logical'
get_problem_data(object, solver, gp)
## S4 method for signature 'Problem,character,missing'
get_problem_data(object, solver, gp)
## S4 method for signature 'Problem'
unpack_results(object, solution, chain, inverse_data)

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 class.
- 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.
- gp: Is the problem a geometric problem?
- solution: A Solution object.
- chain: The corresponding solving Chain.
- inverse_data: A InverseData object or list containing data necessary for the inversion.

Methods (by generic)

- objective(Problem): The objective of the problem.
- objective(Problem) <- value: Set the value of the problem objective.
- constraints(Problem): A list of the constraints of the problem.
- constraints(Problem) <- value: Set the value of the problem constraints.
- value(Problem): The value from the last time the problem was solved (or NA if not solved).
- value(Problem) <- value: Set the value of the optimal objective.
- status(Problem): The status from the last time the problem was solved.
- is_dcp(Problem): A logical value indicating whether the problem satisfies DCP rules.
- is_dgp(Problem): A logical value indicating whether the problem satisfies DGP rules.
- is_qp(Problem): A logical value indicating whether the problem is a quadratic program.
- **canonicalize(Problem)**: The graph implementation of the problem.
- **is_mixed_integer(Problem)**: Logical value indicating whether the problem is a mixed integer program.
- **variables(Problem)**: List of Variable objects in the problem.
- **parameters(Problem)**: List of Parameter objects in the problem.
- **constants(Problem)**: List of Constant objects in the problem.
- **atoms(Problem)**: List of Atom objects in the problem.
- **size_metrics(Problem)**: Information about the size of the problem.
- **solver_stats(Problem)**: Additional information returned by the solver.
- **solver_stats(Problem) <- value**: Set the additional information returned by the solver in the problem.
- **get_problem_data(object = Problem, solver = character, gp = logical)**: Get the problem data passed to the specified solver.
- **get_problem_data(object = Problem, solver = character, gp = missing)**: Get the problem data passed to the specified solver.
- **unpack_results(Problem)**: 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**

```r
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)
```
description
Get and set the objective, constraints, or size metrics (get only) of a problem.

Usage

function(object)

function(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)

ProdEntries-class

The ProdEntries class.

Description

The product of the entries in an expression.

Usage

ProdEntries(..., axis = NA_real_, keepdims = FALSE)

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

## S4 method for signature 'ProdEntries'
sign_from_args(object)
## S4 method for signature 'ProdEntries'
is_atom_convex(object)
## S4 method for signature 'ProdEntries'
is_atom_concave(object)
## S4 method for signature 'ProdEntries'
is_atom_log_log_convex(object)
## S4 method for signature 'ProdEntries'
is_atom_log_log_concave(object)
## S4 method for signature 'ProdEntries'
is_incr(object, idx)
## S4 method for signature 'ProdEntries'
is_decr(object, idx)
## S4 method for signature 'ProdEntries'
.column_grad(object, value)
## S4 method for signature 'ProdEntries'
.grad(object, values)

Arguments

...  Expression objects, vectors, or matrices.
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.
keepdims  (Optional) Should dimensions be maintained when applying the atom along an
axis? If FALSE, result will be collapsed into an n x 1 column vector. The default
is FALSE.
object  A ProdEntries object.
values  A list of numeric values for the arguments
idx  An index into the atom.
value  A numeric value.

Methods (by generic)

- to_numeric(ProdEntries): The product of all the entries.
- sign_from_args(ProdEntries): Returns the sign (is positive, is negative) of the atom.
- is_atom_convex(ProdEntries): Is the atom convex?
- is_atom_concave(ProdEntries): Is the atom concave?
- is_atom_log_log_convex(ProdEntries): Is the atom log-log convex?
• is_atom_log_log_concave(ProdEntries): is the atom log-log concave?
• is_incr(ProdEntries): Is the atom weakly increasing in the argument idx?
• is_decr(ProdEntries): Is the atom weakly decreasing in the argument idx?
• .column_grad(ProdEntries): Gives the (sub/super)gradient of the atom w.r.t. each column variable
• .grad(ProdEntries): Gives the (sub/super)gradient of the atom w.r.t. each variable

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.

prod_entries

Description

The product of entries in a vector or matrix.

Usage

prod_entries(..., axis = NA_real_, keepdims = FALSE)

## S3 method for class 'Expression'
prod(..., na.rm = FALSE)

Arguments

... Numeric scalar, vector, matrix, or Expression objects.
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.
keepdims (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.
na.rm (Unimplemented) A logical value indicating whether missing values should be removed.

Details

This atom is log-log affine, but it is neither convex nor concave.

Value

An Expression representing the product of the entries of the input.
Examples

```r
n <- 2
X <- Variable(n, n, pos=TRUE)
obj <- sum(X)
constraints <- list(prod_entries(X) == 4)
prob <- Problem(Minimize(obj), constraints)
result <- solve(prob, gp=TRUE)
result$value
result$getValue(X)
```

```r
n <- 2
X <- Variable(n, n, pos=TRUE)
obj <- sum(X)
constraints <- list(prod(X) == 4)
prob <- Problem(Minimize(obj), constraints)
result <- solve(prob, gp=TRUE)
result$value
```

---

**project-methods**

<table>
<thead>
<tr>
<th><strong>Project Value</strong></th>
</tr>
</thead>
</table>

**Description**

Project a value onto the attribute set of a Leaf. A sensible idiom is `value(leaf) = project(leaf, val)`.

**Usage**

```r
project(object, value)
```

```r
project_and_assign(object, value)
```

**Arguments**

- **object** A Leaf object.
- **value** The assigned value.

**Value**

The value rounded to the attribute type.
The Promote class.

Description

This class represents the promotion of a scalar expression into a vector/matrix.

Usage

Promote(expr, promoted_dim)

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

## S4 method for signature 'Promote'
is_symmetric(object)

## S4 method for signature 'Promote'
dim_from_args(object)

## S4 method for signature 'Promote'
is_atom_log_log_convex(object)

## S4 method for signature 'Promote'
is_atom_log_log_concave(object)

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

## S4 method for signature 'Promote'
graph_implementation(object, arg_objs, dim, data = NA_real_)

Arguments

- **expr**: An Expression or numeric constant.
- **promoted_dim**: The desired dimensions.
- **object**: A Promote object.
- **values**: A list containing the value to promote.
- **arg_objs**: A list of linear expressions for each argument.
- **dim**: A vector representing the dimensions of the resulting expression.
- **data**: A list of additional data required by the atom.
Methods (by generic)

- `to_numeric(Promote)`: Promotes the value to the new dimensions.
- `is_symmetric(Promote)`: Is the expression symmetric?
- `dim_from_args(Promote)`: Returns the (row, col) dimensions of the expression.
- `is_atom_log_log_convex(Promote)`: Is the atom log-log convex?
- `is_atom_log_log_concave(Promote)`: Is the atom log-log concave?
- `get_data(Promote)`: Returns information needed to reconstruct the expression besides the args.
- `graph_implementation(Promote)`: The graph implementation of the atom.

Slots

expr  An `Expression` or numeric constant.

`promoted_dim`  The desired dimensions.

Description

A no-op wrapper to assert the input argument is positive semidefinite.

Usage

```r
PSDWrap(arg)
```

```r
## S4 method for signature 'PSDWrap'
is_psd(object)
```

Arguments

- `arg`  A `Expression` object or matrix.
- `object`  A `PSDWrap` object.

Methods (by generic)

- `is_psd(PSDWrap)`: Is the atom positive semidefinite?
psd_coeff_offset

Given a problem returns a PSD constraint

Description

Given a problem returns a PSD constraint

Usage

psd_coeff_offset(problem, c)

Arguments

problem A Problem object.
c A vector of coefficients.

Value

Returns an array G and vector h such that the given constraint is equivalent to $G \ast z \leq_{PSD} h$.

psolve

Solve a DCP Problem

Description

Solve a DCP compliant optimization problem.

Usage

psolve(
  object, 
solver = NA,
ignore_dcp = FALSE,
warm_start = FALSE,
verbose = FALSE,
parallel = FALSE,
gp = FALSE,
feastol = NULL,
reltol = NULL,
abstol = NULL,
num_iter = NULL,
...
)

## S4 method for signature 'Problem'
```r
psolve(
  object,
  solver = NA,
  ignore_dcp = FALSE,
  warm_start = FALSE,
  verbose = FALSE,
  parallel = FALSE,
  gp = FALSE,
  feastol = NULL,
  reltol = NULL,
  abstol = NULL,
  num_iter = NULL,
  ...
)
```

## S4 method for signature 'Problem,ANY'
solve(a, b = NA, ...)  

### 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.

- **gp**
  (Optional) A logical value indicating whether the problem is a geometric program. Defaults to FALSE.

- **feastol**
  The feasible tolerance on the primal and dual residual.

- **realtol**
  The relative tolerance on the duality gap.

- **abstol**
  The absolute tolerance on the duality gap.

- **num_iter**
  The maximum number of iterations.

- **...**
  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.
p_norm

$solver$ 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

```r
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

<table>
<thead>
<tr>
<th>$P-Norm$</th>
</tr>
</thead>
</table>

**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_, keepdims = FALSE, 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.
- **keepdims**: (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an $n \times 1$ column vector. The default is FALSE.
- **max_denom**: (Optional) The maximum denominator considered in forming a rational approximation for $p$. The default is 1024.
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 $x \in \mathbb{R}^n$. 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 $x \in \mathbb{R}^n_+$.  

- 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

```r
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)

## Not run:
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)

## End(Not run)
```
Qp2SymbolicQp-class

The Qp2SymbolicQp class.

Description

This class reduces a quadratic problem to a problem that consists of affine expressions and symbolic quadratic forms.

QpMatrixStuffing-class

The QpMatrixStuffing class.

Description

This class fills in numeric values for the problem instance and outputs a DCP-compliant minimization problem with an objective of the form

Details

QuadForm(x, p) + t(q)

and Zero/NonPos constraints, both of which exclusively carry affine arguments

QpSolver-class

A QP solver interface.

Description

A QP solver interface.

Usage

## S4 method for signature 'QpSolver,Problem'
accepts(object, problem)

## S4 method for signature 'QpSolver,Problem'
perform(object, problem)

Arguments

object A QpSolver object.

problem A Problem object.
QuadForm-class

Methods (by generic)

- accepts(object = QpSolver, problem = Problem): Is this a QP problem?
- perform(object = QpSolver, problem = Problem): Constructs a QP problem data stored in a list

---

QuadForm-class

The QuadForm class.

---

Description

This class represents the quadratic form $x^T P x$

Usage

QuadForm(x, P)

```r
## S4 method for signature 'QuadForm'
name(x)
```

```r
## S4 method for signature 'QuadForm'
allow_complex(object)
```

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

```r
## S4 method for signature 'QuadForm'
validate_args(object)
```

```r
## S4 method for signature 'QuadForm'
sign_from_args(object)
```

```r
## S4 method for signature 'QuadForm'
dim_from_args(object)
```

```r
## S4 method for signature 'QuadForm'

is_atom_convex(object)
```

```r
## S4 method for signature 'QuadForm'

is_atom_concave(object)
```

```r
## S4 method for signature 'QuadForm'

is_atom_log_log_convex(object)
```

```r
## S4 method for signature 'QuadForm'

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

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

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

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

## S4 method for signature 'QuadForm'
.grad(object, values)

### Arguments

- `x` An Expression or numeric vector.
- `P` An Expression, numeric matrix, or vector.
- `object` A QuadForm object.
- `values` A list of numeric values for the arguments
- `idx` An index into the atom.

### Methods (by generic)

- `name(QuadForm)`: The name and arguments of the atom.
- `allow_complex(QuadForm)`: Does the atom handle complex numbers?
- `to_numeric(QuadForm)`: Returns the quadratic form.
- `validate_args(QuadForm)`: Checks the dimensions of the arguments.
- `sign_from_args(QuadForm)`: Returns the sign (is positive, is negative) of the atom.
- `dim_from_args(QuadForm)`: The dimensions of the atom.
- `is_atom_convex(QuadForm)`: Is the atom convex?
- `is_atom_concave(QuadForm)`: Is the atom concave?
- `is_atom_log_log_convex(QuadForm)`: Is the atom log-log convex?
- `is_atom_log_log_concave(QuadForm)`: Is the atom log-log concave?
- `is_incr(QuadForm)`: Is the atom weakly increasing in the argument idx?
- `is_decr(QuadForm)`: Is the atom weakly decreasing in the argument idx?
- `is_quadratic(QuadForm)`: Is the atom quadratic?
- `is_pwl(QuadForm)`: Is the atom piecewise linear?
- `.grad(QuadForm)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

### Slots

- `x` An Expression or numeric vector.
- `P` An Expression, numeric matrix, or vector.
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'
allow_complex(object)

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

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

## S4 method for signature 'QuadOverLin'
dim_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_atom_log_log_convex(object)

## S4 method for signature 'QuadOverLin'
is_atom_log_log_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'
is_qpwa(object)

## S4 method for signature 'QuadOverLin'.domain(object)

## S4 method for signature 'QuadOverLin'.grad(object, values)

### Arguments

- **x** An Expression or numeric matrix.
- **y** A scalar Expression or numeric constant.
- **object** A QuadOverLin object.
- **values** A list of numeric values for the arguments
- **idx** An index into the atom.

### Methods (by generic)

- `allow_complex(QuadOverLin)`: Does the atom handle complex numbers?
- `to_numeric(QuadOverLin)`: The sum of the entries of \( x \) squared over \( y \).
- `validate_args(QuadOverLin)`: Check the dimensions of the arguments.
- `dim_from_args(QuadOverLin)`: The atom is a scalar.
- `sign_from_args(QuadOverLin)`: The atom is positive.
- `is_atom_convex(QuadOverLin)`: The atom is convex.
- `is_atom_concave(QuadOverLin)`: The atom is not concave.
- `is_atom_log_log_convex(QuadOverLin)`: Is the atom log-log convex?
- `is_atom_log_log_concave(QuadOverLin)`: Is the atom log-log concave?
- `is_incr(QuadOverLin)`: A logical value indicating whether the atom is weakly increasing in argument \( \text{idx} \).
- `is_decr(QuadOverLin)`: A logical value indicating whether the atom is weakly decreasing in argument \( \text{idx} \).
- `is_quadratic(QuadOverLin)`: Quadratic if \( x \) is affine and \( y \) is constant.
- `is_qpwa(QuadOverLin)`: Quadratic of piecewise affine if \( x \) is piecewise linear and \( y \) is constant.
- `.domain(QuadOverLin)`: Returns constraints describing the domain of the node
- `.grad(QuadOverLin)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

### 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**

```r
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

```r
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

```r
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
```
**Rdictdefault-class**

**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.
Real-class

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

Real-class

The Real class.

Description

This class represents the real part of an expression.

Usage

Real(expr)

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

## S4 method for signature 'Real'
dim_from_args(object)

## S4 method for signature 'Real'
is_imag(object)

## S4 method for signature 'Real'
is_complex(object)

## S4 method for signature 'Real'
is_symmetric(object)

Arguments

expr     An Expression representing a vector or matrix.
object   An Real object.
values   A list of arguments to the atom.
Methods (by generic)

- `to_numeric(Real)`: The imaginary part of the given value.
- `dim_from_args(Real)`: The dimensions of the atom.
- `is_imag(Real)`: Is the atom imaginary?
- `is_complex(Real)`: Is the atom complex valued?
- `is_symmetric(Real)`: Is the atom symmetric?

Slots

- `expr` An `Expression` representing a vector or matrix.

---

`reduce` `Reduce a Problem`

**Description**

Reduces the owned problem to an equivalent problem.

**Usage**

`reduce(object)`

**Arguments**

- `object` A `Reduction` object.

**Value**

An equivalent problem, encoded either as a `Problem` object or a list.

---

**Reduction-class** `The Reduction class.`

**Description**

This virtual class represents a reduction, an actor that transforms a problem into an equivalent problem. By equivalent, we mean that there exists a mapping between solutions of either problem: if we reduce a problem $A$ to another problem $B$ and then proceed to find a solution to $B$, we can convert it to a solution of $A$ with at most a moderate amount of effort.
Usage

```r
## S4 method for signature 'Reduction,Problem'
accepts(object, problem)
## S4 method for signature 'Reduction'
reduce(object)
## S4 method for signature 'Reduction,Solution'
retrieve(object, solution)
## S4 method for signature 'Reduction,Problem'
perform(object, problem)
## S4 method for signature 'Reduction,Solution,list'
invert(object, solution, inverse_data)
```

Arguments

- `object`: A `Reduction` object.
- `problem`: A `Problem` object.
- `solution`: A `Solution` to a problem that generated the inverse data.
- `inverse_data`: The data encoding the original problem.

Details

Every reduction supports three methods: accepts, perform, and invert. The accepts method of a particular reduction codifies the types of problems that it is applicable to, the perform method takes a problem and reduces it to a (new) equivalent form, and the invert method maps solutions from reduced-to problems to their problems of provenance.

Methods (by generic)

- `accepts(object = Reduction, problem = Problem)`: States whether the reduction accepts a problem.
- `reduce(Reduction)`: Reduces the owned problem to an equivalent problem.
- `retrieve(object = Reduction, solution = Solution)`: Retrieves a solution to the owned problem.
- `perform(object = Reduction, problem = Problem)`: Performs the reduction on a problem and returns an equivalent problem.
- `invert(object = Reduction, solution = Solution, inverse_data = list)`: Returns a solution to the original problem given the inverse data.
ReductionSolver-class

The ReductionSolver class.

Description
The ReductionSolver class.

Usage
## S4 method for signature 'ReductionSolver'
mip_capable(solver)

## S4 method for signature 'ReductionSolver'
name(x)

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

## S4 method for signature 'ReductionSolver'
is_installed(solver)

## S4 method for signature 'ReductionSolver'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

## S4 method for signature 'ReductionSolver,ANY'
reduction_solve(
  object,
  problem,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts
)
## S4 method for signature 'ECOS'
solve_via_data(
  object,
  data,
  warm_start,
  verbose,
  feastol,
  reltol,
  abstol,
  num_iter,
  solver_opts,
  solver_cache
)

### Arguments

- **solver, object, x**
  - A `ReductionSolver` object.
- **data**
  - Data generated via an apply call.
- **warm_start**
  - A boolean of whether to warm start the solver.
- **verbose**
  - An integer number indicating level of solver verbosity.
- **feastol**
  - The feasible tolerance on the primal and dual residual.
- **reltol**
  - The relative tolerance on the duality gap.
- **abstol**
  - The absolute tolerance on the duality gap.
- **num_iter**
  - The maximum number of iterations.
- **solver_opts**
  - A list of Solver specific options
- **solver_cache**
  - Cache for the solver.
- **problem**
  - A `Problem` object.

### Methods (by generic)

- **mip_capable(ReductionSolver)**: Can the solver handle mixed-integer programs?
- **name(ReductionSolver)**: Returns the name of the solver
- **import_solver(ReductionSolver)**: Imports the solver
- **is_installed(ReductionSolver)**: Is the solver installed?
- **solve_via_data(ReductionSolver)**: Solve a problem represented by data returned from apply.
- **reduction_solve(object = ReductionSolver, problem = ANY)**: Solve a problem represented by data returned from apply.
- **solve_via_data(ECOS)**: Solve a problem represented by data returned from apply.
resetOptions

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 dimensions. Entries are stored in column-major order.

Usage

Reshape(expr, new_dim)

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

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

## S4 method for signature 'Reshape'
dim_from_args(object)

## S4 method for signature 'Reshape'
is_atom_log_log_convex(object)
## S4 method for signature 'Reshape'

`is_atom_log_log_concave(object)`

## S4 method for signature 'Reshape'

`get_data(object)`

## S4 method for signature 'Reshape'

`graph_implementation(object, arg_objs, dim, data = NA_real_)`

### Arguments

- **expr**: An `Expression` or numeric matrix.
- **new_dim**: The new dimensions.
- **object**: A `Reshape` object.
- **values**: A list of arguments to the atom.
- **arg_objs**: A list of linear expressions for each argument.
- **dim**: A vector representing the dimensions of the resulting expression.
- **data**: A list of additional data required by the atom.

### Methods (by generic)

- `to_numeric(Reshape)`: Reshape the value into the specified dimensions.
- `validate_args(Reshape)`: Check the new shape has the same number of entries as the old.
- `dim_from_args(Reshape)`: The c(rows, cols) dimensions of the new expression.
- `is_atom_log_log_convex(Reshape)`: Is the atom log-log convex?
- `is_atom_log_log_concave(Reshape)`: Is the atom log-log concave?
- `get_data(Reshape)`: Returns a list containing the new shape.
- `graph_implementation(Reshape)`: The graph implementation of the atom.

### Slots

- **expr**: An `Expression` or numeric matrix.
- **new_dim**: The new dimensions.

---

**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.
reshape_expr

Usage

reshape_expr(expr, new_dim)

Arguments

expr An Expression, vector, or matrix.
new_dim The new dimensions.

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,c(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,c(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,c(2,2)))

C <- Variable(3,2)
expr <- reshape_expr(C,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,c(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, c(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.

---

**retrieve** | **Retrieve Solution**

**Description**

Retrieves a solution to the owned problem.

**Usage**

retrieve(object, solution)

**Arguments**

object | A Reduction object.
solution | A Solution object.
Value

A Solution to the problem emitted by reduce.

---

### scaled_lower_tri

**Utility methods for special handling of semidefinite constraints.**

**Description**

Utility methods for special handling of semidefinite constraints.

**Usage**

```python
scaled_lower_tri(matrix)
```

**Arguments**

- `matrix`  
  The matrix to get the lower triangular matrix for

**Value**

The lower triangular part of the matrix, stacked in column-major order

---

### 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 $\alpha \times \text{pos}(x) + \beta \times \text{neg}(x)$.

**Usage**

```python
scalene(x, alpha, beta)
```

**Arguments**

- `x`  
  An Expression, vector, or matrix.
- `alpha`  
  The weight on the positive portion of $x$.
- `beta`  
  The weight on the negative portion of $x$.

**Value**

An Expression representing the scalene function evaluated at the input.
Examples

```r
## 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

An interface for the SCS solver

Description

An interface for the SCS solver

Usage

```r
SCS()
```

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

## S4 method for signature 'SCS'
status_map(solver, status)

## S4 method for signature 'SCS'
name(x)

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

## S4 method for signature 'SCS'
reduction_format_constr(object, problem, constr, exp_cone_order)

## S4 method for signature 'SCS,Problem'
perform(object, problem)

## S4 method for signature 'SCS,list,list'
invert(object, solution, inverse_data)

## S4 method for signature 'SCS'
solve_via_data(
object,
data,
```
Arguments

solver, object, x

A SCS object.

status

A status code returned by the solver.

problem

A Problem object.

constr

A Constraint to format.

exp_cone_order

A list indicating how the exponential cone arguments are ordered.

solution

The raw solution returned by the solver.

inverse_data

A list containing data necessary for the inversion.

data

Data generated via an apply call.

warm_start

A boolean of whether to warm start the solver.

verbose

A boolean of whether to enable solver verbosity.

feastol

The feasible tolerance on the primal and dual residual.

reltol

The relative tolerance on the duality gap.

abstol

The absolute tolerance on the duality gap.

num_iter

The maximum number of iterations.

solver_opts

A list of Solver specific options

solver_cache

Cache for the solver.

Methods (by generic)

• mip_capable(SCS): Can the solver handle mixed-integer programs?
• status_map(SCS): Converts status returned by SCS solver to its respective CVXPY status.
• name(SCS): Returns the name of the solver
• import_solver(SCS): Imports the solver
• reduction_format_constr(SCS): Return a linear operator to multiply by PSD constraint coefficients.
• perform(object = SCS, problem = Problem): Returns a new problem and data for inverting the new solution
• invert(object = SCS, solution = list, inverse_data = list): Returns the solution to the original problem given the inverse_data.
• solve_via_data(SCS): Solve a problem represented by data returned from apply.
SCS.dims_to_solver_dict

Utility method for formatting a ConeDims instance into a dictionary that can be supplied to SCS.

Description
Utility method for formatting a ConeDims instance into a dictionary that can be supplied to SCS.

Usage
SCS.dims_to_solver_dict(cone_dims)

Arguments
cone_dims A ConeDims instance.

Value
The dimensions of the cones.

SCS.extract_dual_value

Extracts the dual value for constraint starting at offset.

Description
Special cases PSD constraints, as per the SCS specification.

Usage
SCS.extract_dual_value(result_vec, offset, constraint)

Arguments
result_vec The vector to extract dual values from.
offset The starting point of the vector to extract from.
constraint A Constraint object.

Value
The dual values for the corresponding PSD constraints
**setIdCounter**  

*Set ID Counter*

**Description**

Set the CVXR variable/constraint identification number counter.

**Usage**

```r
setIdCounter(value = 0L)
```

**Arguments**

- `value` The value to assign as ID.

**Value**

the changed value of the package global .CVXR.options.

**Examples**

```r
## Not run:
setIdCounter(value = 0L)
## End(Not run)
```

---

**SigmaMax-class**  

*The SigmaMax class.*

**Description**

The maximum singular value of a matrix.

**Usage**

```r
SigmaMax(A = A)
```

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

```r
## S4 method for signature 'SigmaMax'
allow_complex(object)
```

```r
## S4 method for signature 'SigmaMax'
dim_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'
.grad(object, values)

**Arguments**

- **A**
  - An Expression or matrix.
- **object**
  - A SigmaMax object.
- **values**
  - A list of numeric values for the arguments
- **idx**
  - An index into the atom.

**Methods (by generic)**

- `to_numeric(SigmaMax)`: The largest singular value of A.
- `allow_complex(SigmaMax)`: Does the atom handle complex numbers?
- `dim_from_args(SigmaMax)`: The atom is a scalar.
- `sign_from_args(SigmaMax)`: The atom is positive.
- `is_atom_convex(SigmaMax)`: The atom is convex.
- `is_atom_concave(SigmaMax)`: The atom is concave.
- `is_incr(SigmaMax)`: The atom is not monotonic in any argument.
- `is_decr(SigmaMax)`: The atom is not monotonic in any argument.
- `.grad(SigmaMax)`: Gives the (sub/super)gradient of the atom w.r.t. each variable

**Slots**

- **A**
  - An Expression or numeric matrix.
Maximum Singular Value

Description

The maximum singular value of a matrix.

Usage

\[ \text{sigma_max}(A = A) \]

Arguments

- \( A \)  
  An Expression or matrix.

Value

An Expression representing the maximum singular value.

Examples

```r
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 of Expression

Description

The sign of an expression.

Usage

```r
## S4 method for signature 'Expression'
sign(x)
```

Arguments

- \( x \)  
  An Expression object.
Value
A string indicating the sign of the expression, either "ZERO", "NONNEGATIVE", "NONPOSITIVE", or "UNKNOWN".

sign-methods

<table>
<thead>
<tr>
<th>Sign Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Determine if an expression is positive, negative, or zero.</td>
</tr>
<tr>
<td>Usage</td>
</tr>
<tr>
<td>is_zero(object)</td>
</tr>
<tr>
<td>is_nonneg(object)</td>
</tr>
<tr>
<td>is_nonpos(object)</td>
</tr>
<tr>
<td>Arguments</td>
</tr>
<tr>
<td>object</td>
</tr>
<tr>
<td>An Expression object.</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>A logical value.</td>
</tr>
<tr>
<td>Examples</td>
</tr>
<tr>
<td>pos &lt;- Constant(1)</td>
</tr>
<tr>
<td>neg &lt;- Constant(-1)</td>
</tr>
<tr>
<td>zero &lt;- Constant(0)</td>
</tr>
<tr>
<td>unknown &lt;- Variable()</td>
</tr>
<tr>
<td>is_zero(pos)</td>
</tr>
<tr>
<td>is_zero(-zero)</td>
</tr>
<tr>
<td>is_zero(unknown)</td>
</tr>
<tr>
<td>is_zero(pos + neg)</td>
</tr>
<tr>
<td>is_nonneg(pos + zero)</td>
</tr>
<tr>
<td>is_nonneg(pos * neg)</td>
</tr>
<tr>
<td>is_nonneg(pos - neg)</td>
</tr>
<tr>
<td>is_nonneg(unknown)</td>
</tr>
<tr>
<td>is_nonpos(-pos)</td>
</tr>
<tr>
<td>is_nonpos(pos + neg)</td>
</tr>
<tr>
<td>is_nonpos(neg * zero)</td>
</tr>
<tr>
<td>is_nonpos(neg - pos)</td>
</tr>
</tbody>
</table>
sign_from_args

<table>
<thead>
<tr>
<th>Description</th>
<th>Atom Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine the sign of an atom based on its arguments.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

```r
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

<table>
<thead>
<tr>
<th>Size of Expression</th>
</tr>
</thead>
</table>

Description

The size of an expression.

Usage

```r
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.
size-methods  

Examples

\begin{verbatim}
x <- Variable()
y <- Variable(3)
z <- Variable(3,2)

size(x)
size(y)
size(z)
size(x + y)
size(z - x)
\end{verbatim}

---

Description

Determine if an expression is a scalar, vector, or matrix.

Usage

\begin{verbatim}
is_scalar(object)
is_vector(object)
is_matrix(object)
\end{verbatim}

Arguments

\begin{verbatim}
object  
\end{verbatim}

An Expression object.

Value

A logical value.

Examples

\begin{verbatim}
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)
\end{verbatim}
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.

SOC-class

The SOC class.

Description

This class represents a second-order cone constraint, i.e. \( \|x\|_2 \leq t \).
Usage

SOC(t, X, axis = 2, id = NA_integer_)

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

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

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

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

## S4 method for signature 'SOC'
um_cones(object)

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

## S4 method for signature 'SOC'
cone_sizes(object)

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

## S4 method for signature 'SOC'
is_dgp(object)

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

Arguments

t       The scalar part of the second-order constraint.
X       A matrix whose rows/columns are each a cone.
axis    The dimension along which to slice: 1 indicates rows, and 2 indicates columns.
The default is 2.
id      (Optional) A numeric value representing the constraint ID.
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)

- residual(SOC): The residual of the second-order constraint.
- get_data(SOC): Information needed to reconstruct the object aside from the args.
- format_constr(SOC): Format SOC constraints as inequalities for the solver.
- num_cones(SOC): The number of elementwise cones.
- size(SOC): The number of entries in the combined cones.
- cone_sizes(SOC): The dimensions of the second-order cones.
- is_dcp(SOC): An SOC constraint is DCP if each of its arguments is affine.
- is_dgp(SOC): Is the constraint DGP?
- canonicalize(SOC): The canonicalization of the constraint.

Slots

t  The scalar part of the second-order constraint.
X  A matrix whose rows/columns are each a cone.
axis The dimension along which to slice: 1 indicates rows, and 2 indicates columns. The default is 2.

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, id = NA_integer_)

## 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_sizes(object)

## S4 method for signature 'SOCAxis'
size(object)
Solution-class

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>The scalar part of the second-order constraint.</td>
</tr>
<tr>
<td>x</td>
<td>A matrix whose rows/columns are each a cone.</td>
</tr>
<tr>
<td>axis</td>
<td>The dimension across which to take the slice: 1 indicates rows, and 2 indicates columns.</td>
</tr>
<tr>
<td>id</td>
<td>(Optional) A numeric value representing the constraint ID.</td>
</tr>
<tr>
<td>x, object</td>
<td>A SOCAxis object.</td>
</tr>
<tr>
<td>eq_constr</td>
<td>A list of the equality constraints in the canonical problem.</td>
</tr>
<tr>
<td>leq_constr</td>
<td>A list of the inequality constraints in the canonical problem.</td>
</tr>
<tr>
<td>dims</td>
<td>A list with the dimensions of the conic constraints.</td>
</tr>
<tr>
<td>solver</td>
<td>A string representing the solver to be called.</td>
</tr>
</tbody>
</table>

Methods (by generic)

- `format_constr(SOCAxis)`: Format SOC constraints as inequalities for the solver.
- `num_cones(SOCAxis)`: The number of elementwise cones.
- `cone_sizes(SOCAxis)`: The dimensions of a single cone.
- `size(SOCAxis)`: The dimensions of the (elementwise) second-order cones.

Slots

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>The scalar part of the second-order constraint.</td>
</tr>
<tr>
<td>x_elems</td>
<td>A list containing X, a matrix whose rows/columns are each a cone.</td>
</tr>
<tr>
<td>axis</td>
<td>The dimension across which to take the slice: 1 indicates rows, and 2 indicates columns.</td>
</tr>
</tbody>
</table>

Description

This class represents a solution to an optimization problem.

Usage

```r
## S4 method for signature 'Solution'
as.character(x)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A Solution object.</td>
</tr>
</tbody>
</table>
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.
ssetup_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.

Description
This class represents a reduction chain that ends with a solver.
Usage

## S4 method for signature 'SolvingChain,Chain'
prepend(object, chain)

## S4 method for signature 'SolvingChain,Problem'
reduction_solve(
  object, 
  problem, 
  warm_start, 
  verbose, 
  feastol, 
  reltol, 
  abstol, 
  num_iter, 
  solver_opts
)

## S4 method for signature 'SolvingChain'
reduction_solve_via_data(
  object, 
  problem, 
  data, 
  warm_start, 
  verbose, 
  feastol, 
  reltol, 
  abstol, 
  num_iter, 
  solver_opts
)

Arguments

object: A SolvingChain object.
chain: A Chain to prepend.
problem: The problem to solve.
warm_start: A boolean of whether to warm start the solver.
verbose: A boolean of whether to enable solver verbosity.
feastol: The feasible tolerance.
reltol: The relative tolerance.
abstol: The absolute tolerance.
um_iter: The maximum number of iterations.
solver_opts: A list of Solver specific options
data: Data for the solver.
Methods (by generic)

- prepend(object = SolvingChain, chain = Chain): Create and return a new SolvingChain by concatenating chain with this instance.
- reduction_solve(object = SolvingChain, problem = Problem): Applies each reduction in the chain to the problem, solves it, and then inverts the chain to return a solution of the supplied problem.
- reduction_solve_via_data(SolvingChain): Solves the problem using the data output by the an apply invocation.

--

sqrt,Expression-method

Square Root

Description

The elementwise square root.

Usage

```r
## 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

--

square,Expression-method

Square

Description

The elementwise square.

Usage

```r
## S4 method for signature 'Expression'
square(x)
```
SumEntries-class

Arguments

x An Expression.

Value

An Expression representing the square of the input. A <- Variable(2, 2) val <- cbind(c(2, 4), c(16, 1)) prob <- Problem(Minimize(square(A)[1, 2]), list(A == val)) result <- solve(prob) result$value

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, keepdims = FALSE)

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

## S4 method for signature 'SumEntries'
is_atom_log_log_convex(object)

## S4 method for signature 'SumEntries'
is_atom_log_log_concave(object)

## S4 method for signature 'SumEntries'
graph_implementation(object, arg_objs, dim, 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.
keepdims (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.
object A SumEntries object.
values A list of arguments to the atom.
arg_objs A list of linear expressions for each argument.
dim A vector representing the dimensions of the resulting expression.
data A list of additional data required by the atom.
Methods (by generic)

- `to_numeric(SumEntries)`: Sum the entries along the specified axis.
- `is_atom_log_log_convex(SumEntries)`: Is the atom log-log convex?
- `is_atom_log_log_concave(SumEntries)`: Is the atom log-log concave?
- `graph_implementation(SumEntries)`: 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.
- `keepdims` (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an \( n \times 1 \) column vector. The default is FALSE.

Description

The sum of the largest k values of a matrix.

Usage

SumLargest(x, k)

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

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

## S4 method for signature 'SumLargest
 dim_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'
.grad(object, values)

### Arguments

- **x**
  - An Expression or numeric matrix.

- **k**
  - The number of largest values to sum over.

- **object**
  - A SumLargest object.

- **values**
  - A list of numeric values for the arguments

- **idx**
  - An index into the atom.

### Methods (by generic)

- **to_numeric(SumLargest)**: The sum of the $k$ largest entries of the vector or matrix.
- **validate_args(SumLargest)**: Check that $k$ is a positive integer.
- **dim_from_args(SumLargest)**: The atom is a scalar.
- **sign_from_args(SumLargest)**: The sign of the atom.
- **is_atom_convex(SumLargest)**: The atom is convex.
- **is_atom_concave(SumLargest)**: The atom is not concave.
- **is_incr(SumLargest)**: The atom is weakly increasing in every argument.
- **is_decr(SumLargest)**: The atom is not weakly decreasing in any argument.
- **get_data(SumLargest)**: A list containing $k$.
- **.grad(SumLargest)**: Gives the (sub/super)gradient of the atom w.r.t. each variable

### Slots

- **x**
  - An Expression or numeric matrix.

- **k**
  - The number of largest values to sum over.
**SumSmallest**

*The SumSmallest atom.*

**Description**

The sum of the smallest $k$ values of a matrix.

**Usage**

`SumSmallest(x, k)`

**Arguments**

- `x` An Expression or numeric matrix.
- `k` The number of smallest values to sum over.

**Value**

Sum of the smallest $k$ values

---

**SumSquares**

*The SumSquares atom.*

**Description**

The sum of the squares of the entries.

**Usage**

`SumSquares(expr)`

**Arguments**

- `expr` An Expression or numeric matrix.

**Value**

Sum of the squares of the entries in the expression.
sum_entries

**Sum of Entries**

**Description**

The sum of entries in a vector or matrix.

**Usage**

```r
sum_entries(expr, axis = NA_real_, keepdims = FALSE)
```

```r
## 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.
- `keepdims` (Optional) Should dimensions be maintained when applying the atom along an axis? If FALSE, result will be collapsed into an n x 1 column vector. The default is FALSE.
- `...` 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**

```r
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)
	x <- Variable(2)
prob <- Problem(Minimize(sum_entries(x)), list(t(x) >= matrix(c(1,2), nrow = 1, ncol = 2)))
result <- solve(prob)
result$value
```
\begin{verbatim}
result$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)

\end{verbatim}

---

### sum_largest

**Sum of Largest Values**

**Description**

The sum of the largest $k$ values of a vector or matrix.

**Usage**

\[
\text{sum\_largest}(x, k)
\]

**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**

```
set.seed(122)
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**

```r
set.seed(1323)
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)`
SymbolicQuadForm-class

Arguments

expr       An Expression, vector, or matrix.

Value

An Expression representing the sum of squares of the input.

Examples

set.seed(212)
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]])

SymbolicQuadForm-class

The SymbolicQuadForm class.

Description

The SymbolicQuadForm class.

Usage

SymbolicQuadForm(x, P, expr)

## S4 method for signature 'SymbolicQuadForm'
dim_from_args(object)

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

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

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

is_atom_concave(object)

## S4 method for signature 'SymbolicQuadForm'

is_incr(object, idx)

## S4 method for signature 'SymbolicQuadForm'

is_decr(object, idx)

## S4 method for signature 'SymbolicQuadForm'

is_quadratic(object)

## S4 method for signature 'SymbolicQuadForm'

.grad(object, values)

### Arguments

- **x**: An Expression or numeric vector.
- **P**: An Expression, numeric matrix, or vector.
- **expr**: The original Expression.
- **object**: A SymbolicQuadForm object.
- **idx**: An index into the atom.
- **values**: A list of numeric values for the arguments.

### Methods (by generic)

- `dim_from_args(SymbolicQuadForm)`: The dimensions of the atom.
- `sign_from_args(SymbolicQuadForm)`: The sign (is positive, is negative) of the atom.
- `get_data(SymbolicQuadForm)`: The original expression.
- `is_atom_convex(SymbolicQuadForm)`: Is the original expression convex?
- `is_atom_concave(SymbolicQuadForm)`: Is the original expression concave?
- `is_incr(SymbolicQuadForm)`: Is the original expression weakly increasing in argument `idx`?
- `is_decr(SymbolicQuadForm)`: Is the original expression weakly decreasing in argument `idx`?
- `is_quadratic(SymbolicQuadForm)`: The atom is quadratic.
- `.grad(SymbolicQuadForm)`: Gives the (sub/super)gradient of the atom w.r.t. each variable.

### Slots

- **x**: An Expression or numeric vector.
- **P**: An Expression, numeric matrix, or vector.
- **original_expression**: The original Expression.
### t.Expression

**Description**

The transpose of a matrix.

**Usage**

```r
## 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**

```r
x <- Variable(3, 4)
t(x)
```

### TotalVariation

**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**

```r
TotalVariation(value, ...)
```

**Arguments**

- `value`: An `Expression` representing the value to take the total variation of.
- `...`: Additional matrices extending the third dimension of `value`.

**Value**

An expression representing the total variation.
to_numeric

Numeric Value of Atom

Description
Returns the numeric value of the atom evaluated on the specified arguments.

Usage
\[
to\_numeric(\text{object}, \text{values})
\]

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
\[
\text{Trace(}\text{expr})
\]

## S4 method for signature 'Trace'
\[
to\_numeric(\text{object}, \text{values})
\]

## S4 method for signature 'Trace'
\[
\text{validate\_args(}\text{object})
\]

## S4 method for signature 'Trace'
\[
\text{dim\_from\_args(}\text{object})
\]

## S4 method for signature 'Trace'
\[
\text{is\_atom\_log\_log\_convex(}\text{object})
\]

## S4 method for signature 'Trace'
\[
\text{is\_atom\_log\_log\_concave(}\text{object})
\]

## S4 method for signature 'Trace'
\[
\text{graph\_implementation(}\text{object, arg\_objs, dim, data = NA\_real_})
\]
**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.
- **dim**: A vector representing the dimensions of the resulting expression.
- **data**: A vector representing additional data required by the atom.

**Methods (by generic)**

- `to_numeric(Trace)`: Sum the diagonal entries.
- `validate_args(Trace)`: Check the argument is a square matrix.
- `dim_from_args(Trace)`: The atom is a scalar.
- `is_atom_log_log_convex(Trace)`: Is the atom log-log convex?
- `is_atom_log_log_concave(Trace)`: Is the atom log-log concave?
- `graph_implementation(Trace)`: The graph implementation of the atom.

**Slots**

- **expr**: An `Expression` representing a matrix.

---

**Description**

This class represents the matrix transpose.

**Usage**

```r
## S4 method for signature 'Transpose'
transp <- function(expr) {
  expr
transp <- function(object, values) {
  object
}
```

```r
## S4 method for signature 'Transpose'
is_symmetric(object)

## S4 method for signature 'Transpose'
is_hermitian(object)

## S4 method for signature 'Transpose'
dim_from_args(object)

## S4 method for signature 'Transpose'
is_atom_log_log_convex(object)
```
## S4 method for signature 'Transpose'
is_atom_log_log_concave(object)

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

## S4 method for signature 'Transpose'
graph_implementation(object, arg_objs, dim, 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.
- **dim**: A vector representing the dimensions of the resulting expression.
- **data**: A list of additional data required by the atom.

Methods (by generic)

- `to_numeric(Transpose)`: The transpose of the given value.
- `is_symmetric(Transpose)`: Is the expression symmetric?
- `is_hermitian(Transpose)`: Is the expression hermitian?
- `dim_from_args(Transpose)`: The dimensions of the atom.
- `is_atom_log_log_convex(Transpose)`: Is the atom log-log convex?
- `is_atom_log_log_concave(Transpose)`: Is the atom log-log concave?
- `get_data(Transpose)`: Returns the axes for transposition.
- `graph_implementation(Transpose)`: The graph implementation of the atom.

---

tri_to_full

Expands lower triangular to full matrix.

Description

Expands lower triangular to full matrix.

Usage

```r
tri_to_full(lower_tri, n)
```

Arguments

- **lower_tri**: A matrix representing the lower triangular part of the matrix, stacked in column-major order.
- **n**: The number of rows (columns) in the full square matrix.
**Value**

A matrix that is the scaled expansion of the lower triangular matrix.

---

**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**

r
```r
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.

Usage

## S4 method for signature 'UnaryOperator'
name(x)

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

Arguments

x, object  AUnaryOperator object.
values  A list of arguments to the atom.

Methods (by generic)

• name(UnaryOperator): Returns the expression in string form.
• to_numeric(UnaryOperator): Applies the unary operator to the value.

Slots

eexpr  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, solution, chain, inverse_data)
unpack_results

Arguments

object A Problem object.
solution A Solution object.
chain The corresponding solving Chain.
inverse_data A InverseData object or list containing data necessary for the inversion.

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

```r
## 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 <- Variable(2,2, PSD = TRUE)
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(
```

```r
```
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 triangular entries of a matrix.

Usage

UpperTri(expr)

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

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

## S4 method for signature 'UpperTri'
dim_from_args(object)

## S4 method for signature 'UpperTri'
is_atom_log_log_convex(object)

## S4 method for signature 'UpperTri'
is_atom_log_log_concave(object)

## S4 method for signature 'UpperTri'
graph_implementation(object, arg_objs, dim, 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.
- **dim**: A vector representing the dimensions of the resulting expression.
- **data**: A list of additional data required by the atom.
Methods (by generic)

- `to_numeric(UpperTri)`: Vectorize the upper triagonal entries.
- `validate_args(UpperTri)`: Check the argument is a square matrix.
- `dim_from_args(UpperTri)`: The dimensions of the atom.
- `is_atom_log_log_convex(UpperTri)`: Is the atom log-log convex?
- `is_atom_log_log_concave(UpperTri)`: Is the atom log-log concave?
- `graph_implementation(UpperTri)`: 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

```r
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_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 = NULL, cols = NULL, name = NA_character_, ...)

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

## S4 method for signature 'Variable'
name(x)

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

## 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.
...
(Optional) Additional attribute arguments. See Leaf for details.
x, object
A Variable object.

Methods (by generic)

• name(Variable): The name of the variable.
• value(Variable): Get the value of the variable.
• grad(Variable): The sub/super-gradient of the variable represented as a sparse matrix.
• variables(Variable): Returns itself as a variable.
• canonicalize(Variable): The canonical form of the variable.

Slots

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

Examples

x <- Variable(3, name = "x0")  ## 3-int variable
y <- Variable(3, 3, name = "y0")  ## Matrix variable
as.character(y)
id(y)
is_nonneg(x)
is_nonpos(x)
size(y)
name(y)
value(y) <- matrix(1:9, nrow = 3)
value(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**

```r
A <- Variable(2,2)
c <- 1:4
eexpr <- vec(A)
obj <- Minimize(t(eexpr) %*% c)
constraints <- list(A == cbind(c(-1,-2), c(3,4)))
prob <- Problem(obj, constraints)
result <- solve(prob)
result$value
result$getValue(expr)
```

---

**vectorized_lower_tri_to_mat**

*Turns symmetric 2D array into a lower triangular matrix*

**Description**

Turns symmetric 2D array into a lower triangular matrix

**Usage**

```r
vectorized_lower_tri_to_mat(v, dim)
```
Arguments

v  A list of length \((\text{dim} \times (\text{dim} + 1) / 2)\).

dim  The number of rows (equivalently, columns) in the output array.

Value

Return the symmetric 2D array defined by taking "v" to specify its lower triangular matrix.

Description

The vertical concatenation of expressions. This is equivalent to \(\text{rbind}\) 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

```r
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'
to_numeric(object, values)

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

## S4 method for signature 'VStack'
dim_from_args(object)

## S4 method for signature 'VStack'
is_atom_log_log_convex(object)

## S4 method for signature 'VStack'
is_atom_log_log_concave(object)

## S4 method for signature 'VStack'
graph_implementation(object, arg_objs, dim, 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.
dim A vector representing the dimensions of the resulting expression.
data A list of additional data required by the atom.
Methods (by generic)

- to_numeric(VStack): Vertically concatenate the values using rbind.
- validate_args(VStack): Check all arguments have the same width.
- dim_from_args(VStack): The dimensions of the atom.
- is_atom_log_log_convex(VStack): Is the atom log-log convex?
- is_atom_log_log_concave(VStack): Is the atom log-log concave?
- graph_implementation(VStack): The graph implementation of the atom.

Slots

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

---

Wrap-class

The Wrap class.

Description

This virtual class represents a no-op wrapper to assert properties.

Usage

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

# S4 method for signature 'Wrap'
dim_from_args(object)

# S4 method for signature 'Wrap'
is_atom_log_log_convex(object)

# S4 method for signature 'Wrap'
is_atom_log_log_concave(object)

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

Arguments

- **object** A Wrap object.
- **values** A list of arguments to the atom.
- **arg_objs** A list of linear expressions for each argument.
- **dim** A vector representing the dimensions of the resulting expression.
- **data** A list of additional data required by the atom.
Methods (by generic)

- `to_numeric(Wrap)`: Returns the input value.
- `dim_from_args(Wrap)`: The dimensions of the atom.
- `is_atom_log_log_convex(Wrap)`: Is the atom log-log convex?
- `is_atom_log_log_concave(Wrap)`: Is the atom log-log concave?
- `graph_implementation(Wrap)`: The graph implementation of the atom.

---

**ZeroConstraint-class**  
The ZeroConstraint class

**Description**

The ZeroConstraint class

**Usage**

```r
## S4 method for signature 'ZeroConstraint'
name(x)

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

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

## S4 method for signature 'ZeroConstraint'
is_dgp(object)

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

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

**Arguments**

- `x`, `object`  
  A ZeroConstraint object.

**Methods (by generic)**

- `name(ZeroConstraint)`: The string representation of the constraint.
- `dim(ZeroConstraint)`: The dimensions of the constrained expression.
- `is_dcp(ZeroConstraint)`: Is the constraint DCP?
- `is_dgp(ZeroConstraint)`: Is the constraint DGP?
- `residual(ZeroConstraint)`: The residual of a constraint
- `canonicalize(ZeroConstraint)`: The graph implementation of the object.
The SpecialIndex class.

Description

This class represents indexing using logical indexing or a list of indices into a matrix.

Usage

```
## 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]
```

`SpecialIndex(expr, key)`

```
## S4 method for signature 'SpecialIndex'
name(x)
```

```
## S4 method for signature 'SpecialIndex'
is_atom_log_log_convex(object)
```

```
## S4 method for signature 'SpecialIndex'
is_atom_log_log_concave(object)
```

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

```
## S4 method for signature 'SpecialIndex'
.grad(object)
```
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.

Methods (by generic)

- **name**(SpecialIndex): Returns the index in string form.
- **is_atom_log_log_convex**(SpecialIndex): Is the atom log-log convex?
- **is_atom_log_log_concave**(SpecialIndex): Is the atom log-log concave?
- **get_data**(SpecialIndex): A list containing key.
- **.grad**(SpecialIndex): Gives the (sub/super)gradient of the atom w.r.t. each variable

Slots

- **expr**
  - An Expression representing a vector or matrix.
- **key**
  - A list containing the start index, end index, and step size of the slice.

Description

This class represents indexing or slicing into a matrix.

Usage

```r
## S4 method for signature 'Expression,missing,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,numeric,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,missing,numeric,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'Expression,numeric,numeric,ANY'
x[i, j, ..., drop = TRUE]
```
Index(expr, key)

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

## S4 method for signature 'Index'
dim_from_args(object)

## S4 method for signature 'Index'
is_atom_log_log_convex(object)

## S4 method for signature 'Index'
is_atom_log_log_concave(object)

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

## S4 method for signature 'Index'
graph_implementation(object, arg_objs, dim, data = NA_real_)

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

## S4 method for signature 'SpecialIndex'
dim_from_args(object)

**Arguments**

- **x**: A *Expression* 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.
- **object**: An *Index* object.
- **values**: A list of arguments to the atom.
- **arg_objs**: A list of linear expressions for each argument.
- **dim**: A vector representing the dimensions of the resulting expression.
- **data**: A list of additional data required by the atom.

**Methods (by generic)**

- `to_numeric(Index)`: The index/slice into the given value.
- `dim_from_args(Index)`: The dimensions of the atom.
• **is_atom_log_log_convex**(*Index*): Is the atom log-log convex?
• **is_atom_log_log_concave**(*Index*): Is the atom log-log concave?
• **get_data**(*Index*): A list containing `key`.
• **graph_implementation**(*Index*): The graph implementation of the atom.
• **to_numeric**(*SpecialIndex*): The index/slice into the given value.
• **dim_from_args**(*SpecialIndex*): The dimensions 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.

---

**Description**

This class represents the matrix product of two linear expressions. See *Multiply* for the elementwise product.

**Usage**

```r
## 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'
to_numeric(object, values)

## S4 method for signature 'MulExpression'
dim_from_args(object)

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

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

## S4 method for signature 'MulExpression'
```
is_atom_log_log_convex(object)

## S4 method for signature 'MulExpression'

is_atom_log_log_concave(object)

## S4 method for signature 'MulExpression'

is_incr(object, idx)

## S4 method for signature 'MulExpression'

is_decr(object, idx)

## S4 method for signature 'MulExpression'

.grad(object, values)

## S4 method for signature 'MulExpression'

graph_implementation(object, arg_objs, dim, data = NA_real_)

Arguments

x, y
The Expression objects or numeric constants to multiply.

object
A MulExpression object.

values
A list of numeric values for the arguments

idx
An index into the atom.

arg_objs
A list of linear expressions for each argument.

dim
A vector representing the dimensions of the resulting expression.

data
A list of additional data required by the atom.

Methods (by generic)

• to_numeric(MulExpression): Matrix multiplication.

• dim_from_args(MulExpression): The (row, col) dimensions of the expression.

• is_atom_convex(MulExpression): Multiplication is convex (affine) in its arguments only if one of the arguments is constant.

• is_atom_concave(MulExpression): If the multiplication atom is convex, then it is affine.

• is_atom_log_log_convex(MulExpression): Is the atom log-log convex?

• is_atom_log_log_concave(MulExpression): Is the atom log-log concave?

• is_incr(MulExpression): Is the left-hand expression positive?

• is_decr(MulExpression): Is the left-hand expression negative?

• .grad(MulExpression): Gives the (sub/super)gradient of the atom w.r.t. each variable

• graph_implementation(MulExpression): The graph implementation of the expression.

See Also

Multiply
The PSDConstraint class.

Description

This class represents the positive semidefinite constraint, \( \frac{1}{2} (X + X^T) \succeq 0 \), i.e. \( z^T (X + X^T) z \geq 0 \) for all \( z \).

Usage

\begin{verbatim}
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(expr, id = NA_integer_)

## S4 method for signature 'PSDConstraint'
name(x)

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

## S4 method for signature 'PSDConstraint'
is_dgp(object)

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

## S4 method for signature 'PSDConstraint'
\end{verbatim}
canonicalize(object)

**Arguments**

e1, e2  The Expression objects or numeric constants to compare.
expr  An Expression, numeric element, vector, or matrix representing $X$.
id  (Optional) A numeric value representing the constraint ID.
x, object  A PSDConstraint object.

**Methods (by generic)**

- `name(PSDConstraint)`: The string representation of the constraint.
- `is_dcp(PSDConstraint)`: The constraint is DCP if the left-hand and right-hand expressions are affine.
- `is_dgp(PSDConstraint)`: Is the constraint DGP?
- `residual(PSDConstraint)`: A Expression representing the residual of the constraint.
- `canonicalize(PSDConstraint)`: 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**

- `expr`  An Expression, numeric element, vector, or matrix representing $X$.

---

**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**

```r
## 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) =
\begin{align*}
x^p & \quad \text{for } x > 0 \\
+\infty & \quad x \leq 0
\end{align*}
, this function is convex, decreasing, and positive. For $0 < p < 1$ and $f(x) =
\begin{align*}
x^p & \quad \text{for } x \geq 0 \\
-\infty & \quad x < 0
\end{align*}
, this function is concave, increasing, and positive. For $p > 1, p \neq 2, 4, 8, \ldots$ and $f(x) =
\begin{align*}
x^p & \quad \text{for } x \geq 0 \\
+\infty & \quad x < 0
\end{align*}
, this function is convex, increasing, and positive.

Examples
```r
## 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

* data
  cdic, 51
dspop, 129
dssamp, 129
*(multiply), 232
*,ConstVal,Expression-method
  (*,Expression,Expression-method), 11
*,Expression,ConstVal-method
  (*,Expression,Expression-method), 11
*,Maximize,Expression-method, 11
*,Minimize,numeric-method
  (Objective-arith), 248
*,Minimize,numeric-method
  (Objective-arith), 248
*,Problem,numeric-method
  (Problem-arith), 265
*,numeric,Maximize-method
  (Objective-arith), 248
*,numeric,Minimize-method
  (Objective-arith), 248
*,numeric,Problem-method
  (Problem-arith), 265
+,ConstVal,Expression-method
  (+,Expression,missing-method), 11
+,Expression,ConstVal-method
  (+,Expression,missing-method), 11
+,Expression,Expression-method
  (+,Expression,missing-method), 11
+,Expression,missing-method, 11
+,Maximize,Maximize-method
  (Objective-arith), 248
+,Maximize,Minimize-method
  (Objective-arith), 248
+,Minimize,Minimize-method
  (Objective-arith), 248
+,numeric,numeric-method
  (Objective-arith), 248
+,numeric,Problem-method
  (Problem-arith), 265
-,ConstVal,Expression-method
  (-,Expression,missing-method), 13
-,Expression,ConstVal-method
  (-,Expression,missing-method), 13
-,Expression,Expression-method
  (-,Expression,missing-method), 13
-,Expression,missing-method, 13
-,Maximize,Objective-method
  (Objective-arith), 248
-,Maximize,missing-method
  (Objective-arith), 248
-,Minimize,Objective-method
  (Objective-arith), 248
-,Minimize,missing-method
  (Objective-arith), 248
-,numeric,numeric-method
  (Objective-arith), 248
-,Objective,Maximize-method
  (Objective-arith), 248
-,Objective,Minimize-method
  (Objective-arith), 248
-,Objective,numeric-method
  (Objective-arith), 248
<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>Abs-class</td>
<td>37</td>
</tr>
<tr>
<td>AddExpression</td>
<td>(+,Expression,missing-method), 11</td>
<td></td>
</tr>
<tr>
<td>CallbackParam</td>
<td>CallbackParam-class</td>
<td>46</td>
</tr>
<tr>
<td>Canonicalization</td>
<td>Canonicalization-class</td>
<td>48</td>
</tr>
<tr>
<td>Chain</td>
<td>Chain-class</td>
<td>52</td>
</tr>
<tr>
<td>ConeDims</td>
<td>ConeDims-class</td>
<td>70</td>
</tr>
<tr>
<td>Conjugate</td>
<td>Conjugate-class</td>
<td>73</td>
</tr>
<tr>
<td>Constant</td>
<td>Constant-class</td>
<td>74</td>
</tr>
<tr>
<td>Conv</td>
<td>Conv-class</td>
<td>82</td>
</tr>
<tr>
<td>CumMax</td>
<td>CumMax-class</td>
<td>87</td>
</tr>
<tr>
<td>CumSum</td>
<td>CumSum-class</td>
<td>89</td>
</tr>
<tr>
<td>Dcp2Cone</td>
<td>Dcp2Cone-class</td>
<td>98</td>
</tr>
<tr>
<td>DgpCanonMethods</td>
<td>DgpCanonMethods-class</td>
<td>121</td>
</tr>
<tr>
<td>DiagMat</td>
<td>DiagMat-class</td>
<td>122</td>
</tr>
<tr>
<td>DiagVec</td>
<td>DiagVec-class</td>
<td>124</td>
</tr>
<tr>
<td>DivExpression</td>
<td>(/,Expression,Expression-method), 32</td>
<td></td>
</tr>
<tr>
<td>EliminatePwl</td>
<td>EliminatePwl-class</td>
<td>134</td>
</tr>
<tr>
<td>Entr</td>
<td>Entr-class</td>
<td>140</td>
</tr>
<tr>
<td>EqConstraint</td>
<td>(==,Expression,Expression-method), 35</td>
<td></td>
</tr>
<tr>
<td>Exp</td>
<td>Exp-class</td>
<td>142</td>
</tr>
<tr>
<td>ExpCone</td>
<td>ExpCone-class</td>
<td>144</td>
</tr>
<tr>
<td>EyeMinusInv</td>
<td>EyeMinusInv-class</td>
<td>151</td>
</tr>
<tr>
<td>GeoMean</td>
<td>GeoMean-class</td>
<td>155</td>
</tr>
<tr>
<td>HStack</td>
<td>HStack-class</td>
<td>171</td>
</tr>
<tr>
<td>Huber</td>
<td>Huber-class</td>
<td>173</td>
</tr>
<tr>
<td>Imag</td>
<td>Imag-class</td>
<td>176</td>
</tr>
<tr>
<td>Index</td>
<td>([,Expression,missing,missing,ANY-method), 341</td>
<td></td>
</tr>
<tr>
<td>IneqConstraint</td>
<td>(-,Expression,missing-method), 13</td>
<td></td>
</tr>
<tr>
<td>InverseData</td>
<td>InverseData-class</td>
<td>178</td>
</tr>
<tr>
<td>KLDiv</td>
<td>KLDiv-class</td>
<td>183</td>
</tr>
<tr>
<td>Kron</td>
<td>Kron-class</td>
<td>185</td>
</tr>
<tr>
<td>LambdaMax</td>
<td>LambdaMax-class</td>
<td>187</td>
</tr>
<tr>
<td>LambdaSumLargest</td>
<td>(LambdaSumLargest-class), 189</td>
<td></td>
</tr>
<tr>
<td>LinOpVector__new</td>
<td>LinOpVector__new</td>
<td>16</td>
</tr>
<tr>
<td>LinOpVector__push_back</td>
<td>LinOpVector__push_back</td>
<td>16</td>
</tr>
<tr>
<td>LinOp__args_push_back</td>
<td>LinOp__args_push_back</td>
<td>17</td>
</tr>
<tr>
<td>LinOp__get_dense_data</td>
<td>LinOp__get_dense_data</td>
<td>17</td>
</tr>
<tr>
<td>LinOp__get_id</td>
<td>LinOp__get_id</td>
<td>18</td>
</tr>
<tr>
<td>LinOp__get_size</td>
<td>LinOp__get_size</td>
<td>18</td>
</tr>
<tr>
<td>LinOp__get_slice</td>
<td>LinOp__get_slice</td>
<td>19</td>
</tr>
<tr>
<td>LinOp__get_sparse</td>
<td>LinOp__get_sparse</td>
<td>19</td>
</tr>
<tr>
<td>LinOp__get_sparse_data</td>
<td>LinOp__get_sparse_data</td>
<td>20</td>
</tr>
<tr>
<td>LinOp__get_set_type</td>
<td>LinOp__get_set_type</td>
<td>20</td>
</tr>
<tr>
<td>LinOp__new</td>
<td>LinOp__new</td>
<td>21</td>
</tr>
<tr>
<td>LinOp__set_dense_data</td>
<td>LinOp__set_dense_data</td>
<td>21</td>
</tr>
<tr>
<td>LinOp__set_size</td>
<td>LinOp__set_size</td>
<td>21</td>
</tr>
<tr>
<td>LinOp__set_slice</td>
<td>LinOp__set_slice</td>
<td>22</td>
</tr>
<tr>
<td>LinOp__set_sparse</td>
<td>LinOp__set_sparse</td>
<td>22</td>
</tr>
<tr>
<td>LinOp__set_sparse_data</td>
<td>LinOp__set_sparse_data</td>
<td>23</td>
</tr>
<tr>
<td>LinOp__set_type</td>
<td>LinOp__set_type</td>
<td>23</td>
</tr>
<tr>
<td>LinOp__size_push_back</td>
<td>LinOp__size_push_back</td>
<td>24</td>
</tr>
<tr>
<td>LinOp__slice_push_back</td>
<td>LinOp__slice_push_back</td>
<td>24</td>
</tr>
<tr>
<td>LinOp_at_index</td>
<td>LinOp_at_index</td>
<td>16</td>
</tr>
<tr>
<td>Log</td>
<td>Log-class</td>
<td>199</td>
</tr>
<tr>
<td>Log1p</td>
<td>Log1p-class</td>
<td>201</td>
</tr>
<tr>
<td>LogDet</td>
<td>LogDet-class</td>
<td>202</td>
</tr>
<tr>
<td>LogSumExp</td>
<td>LogSumExp-class</td>
<td>205</td>
</tr>
<tr>
<td>Logistic</td>
<td>Logistic-class</td>
<td>204</td>
</tr>
<tr>
<td>MatrixFrac</td>
<td>MatrixFrac-class</td>
<td>210</td>
</tr>
<tr>
<td>MaxElemwise</td>
<td>MaxElemwise-class</td>
<td>215</td>
</tr>
<tr>
<td>MaxEntries</td>
<td>MaxEntries-class</td>
<td>216</td>
</tr>
<tr>
<td>Maximize</td>
<td>Maximize-class</td>
<td>218</td>
</tr>
<tr>
<td>MinElemwise</td>
<td>MinElemwise-class</td>
<td>222</td>
</tr>
<tr>
<td>MinEntries</td>
<td>MinEntries-class</td>
<td>223</td>
</tr>
<tr>
<td>Minimize</td>
<td>Minimize-class</td>
<td>225</td>
</tr>
<tr>
<td>MulExpression</td>
<td>(%*,Expression,Expression-method), 343</td>
<td></td>
</tr>
<tr>
<td>Multiply</td>
<td>Multiply-class</td>
<td>232</td>
</tr>
<tr>
<td>NegExpression</td>
<td>NegExpression</td>
<td></td>
</tr>
<tr>
<td>NonPosConstraint</td>
<td>NonPosConstraint</td>
<td></td>
</tr>
</tbody>
</table>
.NonPosConstraint (NonPosConstraint-class), 236
  .NonlinearConstraint (NonlinearConstraint-class), 235
  .Norm1 (Norm1-class), 239
  .NormInf (NormInf-class), 243
  .NormNuc (NormNuc-class), 245
  .Objective (Objective-class), 249
  .OneMinusPos (OneMinusPos-class), 250
  .PSDConstraint (%>>%), 345
  .PSDWrap (PSDWrap-class), 275
  .Parameter (Parameter-class), 254
  .PFEigenvalue (PFEigenvalue-class), 256
  .Pnorm (Pnorm-class), 259
  .Power (Power-class), 262
  .Problem (Problem-class), 266
  .ProblemData__get_I, 26
  .ProblemData__get_J, 27
  .ProblemData__get_V, 27
  .ProblemData__get_const_to_row, 25
  .ProblemData__get_const_vec, 25
  .ProblemData__get_id_to_col, 26
  .ProblemData__new, 28
  .ProblemData__set_I, 29
  .ProblemData__set_J, 30
  .ProblemData__set_V, 31
  .ProblemData__set_const_to_row, 28
  .ProblemData__set_const_vec, 29
  .ProblemData__set_id_to_col, 30
  .ProdEntries (ProdEntries-class), 270
  .Promote (Promote-class), 274
  .Qp2SymbolicQp (Qp2SymbolicQp-class), 280
  .QuadForm (QuadForm-class), 281
  .QuadOverLin (QuadOverLin-class), 283
  .Real (Real-class), 288
  .Reshape (Reshape-class), 293
  .SOC (SOC-class), 307
  .SOCAxis (SOCAxis-class), 309
  .SigmaMax (SigmaMax-class), 301
  .SizeMetrics (SizeMetrics-class), 307
  .Solution (Solution-class), 310
  .SolverStats (SolverStats-class), 311
  .SolvingChain (SolvingChain-class), 311
  .SpecialIndex (SpecialIndex-class), 314
  .SumEntries (SumEntries-class), 314
  .SumLargest (SumLargest-class), 315
  .SymbolicQuadForm (SymbolicQuadForm-class), 321
  .Trace (Trace-class), 324
  .Transpose (Transpose-class), 325
  .UpperTri (UpperTri-class), 330
  .VStack (VStack-class), 337
  .Variable (Variable-class), 333
  .ZeroConstraint (ZeroConstraint-class), 339
  .axis_grad, AxisAtom-method (AxisAtom-class), 43
  .build_matrix_0, 14
  .build_matrix_1, 15
  .column_grad, AxisAtom-method (AxisAtom-class), 43
  .column_grad, CumMax-method (CumMax-class), 87
  .column_grad, LogSumExp-method (LogSumExp-class), 205
  .column_grad, MaxEntries-method (MaxEntries-class), 216
  .column_grad, MinEntries-method (MinEntries-class), 223
  .column_grad, Norm1-method (Norm1-class), 239
  .column_grad, NormInf-method (NormInf-class), 243
  .column_grad, Pnorm-method (Pnorm-class), 259
  .column_grad, ProdEntries-method (ProdEntries-class), 270
  .decomp_quad, 15
  .domain,Entr-method (Entr-class), 140
  .domain,GeoMean-method (GeoMean-class), 155
  .domain,KLDiv-method (KLDiv-class), 183
  .domain,LambdaMax-method (LambdaMax-class), 187
  .domain,Log-method (Log-class), 199
  .domain,Log1p-method (Log1p-class), 201
  .domain,LogDet-method (LogDet-class), 202
  .domain,MatrixFrac-method (MatrixFrac-class), 210
  (Expression, index, missing, ANY-method)
  .domain, Norm1-method (Norm1-class), 239
  .domain, NormInf-method (NormInf-class), 243
INDEX

351

.domain, Pnorm-method (Pnorm-class), 259
.domain, Power-method (Power-class), 262
.domain, QuadOverLin-method
   (QuadOverLin-class), 283
.grad, AffAtom-method (AffAtom-class), 39
.grad, CumMax-method (CumMax-class), 87
.grad, CumSum-method (CumSum-class), 89
.grad, Entr-method (Entr-class), 140
.grad, Exp-method (Exp-class), 142
.grad, EyeMinusInv-method
   (EyeMinusInv-class), 151
.grad, GeoMean-method (GeoMean-class), 155
.grad, Huber-method (Huber-class), 173
.grad, KLDiv-method (KLDiv-class), 183
.grad, LambdaMax-method
   (LambdaMax-class), 187
.grad, LambdaSumLargest-method
   (LambdaSumLargest-class), 189
.grad, Log-method (Log-class), 199
.grad, Log1p-method (Log1p-class), 201
.grad, LogDet-method (LogDet-class), 202
.grad, LogSumExp-method
   (LogSumExp-class), 205
.grad, Logistic-method (Logistic-class), 204
.grad, MatrixFrac-method
   (MatrixFrac-class), 210
.grad, MaxElemwise-method
   (MaxElemwise-class), 215
.grad, MaxEntries-method
   (MaxEntries-class), 216
.grad, MinElewise-method
   (MinElewise-class), 222
.grad, MinEntries-method
   (MinEntries-class), 223
.grad, MulExpression-method
   (%*%, Expression, Expression-method), 343
.grad, Norm1-method (Norm1-class), 239
.grad, NormInf-method (NormInf-class), 243
.grad, NormNuc-method (NormNuc-class), 245
.grad, OneMinusPos-method
   (OneMinusPos-class), 250
.grad, PfEigenvalue-method
   (PfEigenvalue-class), 256
.grad, Pnorm-method (Pnorm-class), 259
.grad, Power-method (Power-class), 262
.grad, ProdEntries-method
   (ProdEntries-class), 270
.grad, QuadForm-method (QuadForm-class), 281
.grad, QuadOverLin-method
   (QuadOverLin-class), 283
.grad, SigmaMax-method (SigmaMax-class), 301
.grad, SpecialIndex-method
   ([, Expression, index, missing, ANY-method), 340
.grad, SumLargest-method
   (SumLargest-class), 315
.grad, SymbolicQuadForm-method
   (SymbolicQuadForm-class), 321
.p_norm, 31
/., ConstVal, Expression-method
   (/, Expression, Expression-method), 32
/., Expression, ConstVal-method
   (/, Expression, Expression-method), 32
/., Expression, Expression-method, 32
/., Objective, numeric-method
   (Objective-arith), 248
/., Problem, numeric-method
   (Problem-arith), 265
<., ConstVal, Expression-method
   (<, Expression, Expression-method), 33
<., Expression, ConstVal-method
   (<, Expression, Expression-method), 33
<., Expression, Expression-method
   (<, Expression, Expression-method), 33
<=, ConstVal, Expression-method
   (<=, Expression, Expression-method), 33
<=, Expression, ConstVal-method
   (<=, Expression, Expression-method), 33
<=, Expression, Expression-method
   (<=, Expression, Expression-method), 33
==, ConstVal, Expression-method
   (==, Expression, Expression-method), 35
<=, Expression, ConstVal-method
   (<=, Expression, Expression-method), 33
<=, Expression, Expression-method
   (<=, Expression, Expression-method), 33
==, ConstVal, Expression-method
   (==, Expression, Expression-method), 35
<=, Expression, Expression-method
   (<=, Expression, Expression-method), 33
<=, Expression, Expression-method
   (<=, Expression, Expression-method), 33
accepts, Chain, Problem-method  
(Chain-class), 52
accepts, Complex2Real, Problem-method  
(Complex2Real-class), 54
accepts, ConeMatrixStuffing, Problem-method  
(ConeMatrixStuffing-class), 70
accepts, ConicSolver, Problem-method  
(ConicSolver-class), 71
accepts, ConstantSolver, Problem-method  
(ConstantSolver-class), 76
accepts, CPLEX_CONIC, Problem-method  
(CPLEX_CONIC-class), 83
accepts, CVXOPT, Problem-method  
(CVXOPT-class), 95
accepts, Dcp2Cone, Problem-method  
(Dcp2Cone-class), 98
accepts, Dgp2Dcp, Problem-method  
(Dgp2Dcp-class), 108
accepts, EliminatePwl, Problem-method  
(EliminatePwl-class), 134
accepts, GUROBI_CONIC, Problem-method  
(GUROBI_CONIC-class), 166
accepts, MOSEK, Problem-method  
(MOSEK-class), 229
accepts, QpSolver, Problem-method  
(QpSolver-class), 280
accepts, Reduction, Problem-method  
(Reduction-class), 289
add_to_solver_blacklist  
(installed_solvers), 177
AddExpression, 12
AddExpression  
(+, Expression, missing-method), 11
AddExpression-class  
(+, Expression, missing-method), 11
AffAtom, 40
AffAtom (AffAtom-class), 39
AffAtom-class, 39
allow_complex, Abs-method (Abs-class), 37
allow_complex, AffAtom-method  
(AffAtom-class), 39
allow_complex, Atom-method (Atom-class), 41
allow_complex, LambdaSumLargest-method  
(LambdaSumLargest-class), 189
allow_complex, MatrixFrac-method
allow_complex, Norm1-method  
(Norm1-class), 239
allow_complex, NormInf-method  
(NormInf-class), 243
allow_complex, NormNuc-method  
(NormNuc-class), 245
allow_complex, Pnorm-method  
(Pnorm-class), 259
allow_complex, QuadForm-method  
(QuadForm-class), 281
allow_complex, QuadOverLin-method  
(QuadOverLin-class), 283
allow_complex, SigmaMax-method  
(SigmaMax-class), 301
are_args_affine, 40
as.character, Chain-method  
(Chain-class), 52
as.character, Constraint-method  
(Constraint-class), 78
as.character, ExpCone-method  
(ExpCone-class), 144
as.character, Expression-method  
(Expression-class), 145
as.character, SOC-method (SOC-class), 307
as.character, SOCAxis-method  
(SOCAxis-class), 309
as.character, Solution-method  
(Solution-class), 310
as.character, Variable-method  
(Variable-class), 333
as.Constant (Constant-class), 74
Atom, 42, 43, 47, 92, 93, 127, 150, 196, 208, 269, 305, 324, 332
Atom (Atom-class), 41
Atom-class, 41
atoms (expression-parts), 149
atoms, Atom-method (Atom-class), 41
atoms, Canonical-method  
(Canonical-class), 46
atoms, Leaf-method (Leaf-class), 193
atoms, Problem-method (Problem-class), 266
AxisAtom (AxisAtom-class), 43
AxisAtom-class, 43
BinaryOperator, 44
BinaryOperator (BinaryOperator-class), 44
BinaryOperator-class, 44
block_format, MOSEK-method
  (MOSEK-class), 229
bmat, 45
CallbackParam, 46
CallbackParam (CallbackParam-class), 46
CallbackParam-class, 46
Canonical, 47, 49
Canonical-class, 46
canonical_form (canonicalize), 49
canonical_form, Canonical-method
  (Canonical-class), 46
Canonicalization, 48
Canonicalization-class, 48
canonicalize, 49
canonicalize, Atom-method (Atom-class), 41
canonicalize, Constant-method
  (Constant-class), 74
canonicalize, ExpCone-method
  (ExpCone-class), 144
canonicalize, Maximize-method
  (Maximize-class), 218
canonicalize, Minimize-method
  (Minimize-class), 225
canonicalize, NonPosConstraint-method
  (NonPosConstraint-class), 236
canonicalize, Parameter-method
  (Parameter-class), 254
canonicalize, Problem-method
  (Problem-class), 266
canonicalize, PSDConstraint-method
  (%>>%), 345
canonicalize, SOC-method (SOC-class), 307
canonicalize, Variable-method
  (Variable-class), 333
canonicalize, ZeroConstraint-method
  (ZeroConstraint-class), 339
canonicalize_expr, Canonicalization-method
  (Canonicalization-class), 48
canonicalize_expr, Dgp2Dcp-method
  (Dgp2Dcp-class), 108
canonicalize_tree, Canonicalization-method
  (Canonicalization-class), 48
CBC_CONIC, 50
CBC_CONIC (CBC_CONIC-class), 49
CBC_CONIC-class, 49
cdiac, 51
Chain, 52, 80, 268, 312, 329
Chain-class, 52
complex-atoms, 53
complex-methods, 54
Complex2Real, 54
Complex2Real (Complex2Real-class), 54
Complex2Real-class, 54
Complex2Real.abs_canon, 55
Complex2Real.add, 56
Complex2Real.at_least_2D, 56
Complex2Real.binary_canon, 57
Complex2Real.canonize_expr, 57
Complex2Real.canonicalize_tree, 58
Complex2Real.conj_canon, 58
Complex2Real.constant_canon, 59
Complex2Real.hermitian_canon, 59
Complex2Real.imag_canon, 60
Complex2Real.join, 61
Complex2Real.lambda_sum_largest_canon, 61
Complex2Real.matrix_frac_canon, 62
Complex2Real.nonpos_canon, 62
Complex2Real.norm_nuc_canon, 63
Complex2Real.param_canon, 63
Complex2Real.pnorm_canon, 64
Complex2Real.psd_canon, 65
Complex2Real.quad_canon, 65
Complex2Real.quad_over_lin_canon, 66
Complex2Real.real_canon, 66
Complex2Real.separable_canon, 67
Complex2Real.soc_canon, 68
Complex2Real.variable_canon, 68
Complex2Real.zero_canon, 69
cone-methods, 69
cone_sizes (cone-methods), 69
cone_sizes, ExpCone-method
  (ExpCone-class), 144
cone_sizes, SOC-method (SOC-class), 307
cone_sizes, SOCAxis-method
  (SOCAxis-class), 309
ConeDims, 131, 300
ConeDims-class, 70
ConeMatrixStuffing, 70
ConeMatrixStuffing (ConeMatrixStuffing-class), 70
ConeMatrixStuffing-class, 70
ConicSolver, 71
ConicSolver (ConicSolver-class), 71
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConicSolver-class</td>
<td>71</td>
</tr>
<tr>
<td>ConicSolver.get_coeff_offset</td>
<td>72</td>
</tr>
<tr>
<td>ConicSolver.get_spacing_matrix</td>
<td>72</td>
</tr>
<tr>
<td>Conj,Expression-method (complex-atoms)</td>
<td>53</td>
</tr>
<tr>
<td>Conjugate</td>
<td>73</td>
</tr>
<tr>
<td>Conjugate (Conjugate-class)</td>
<td>73</td>
</tr>
<tr>
<td>Conjugate-class</td>
<td>73</td>
</tr>
<tr>
<td>Constant</td>
<td>47, 59, 74, 75, 150, 196, 269</td>
</tr>
<tr>
<td>Constant (Constant-class)</td>
<td>74</td>
</tr>
<tr>
<td>Constant-class</td>
<td>74</td>
</tr>
<tr>
<td>constants (expression-parts)</td>
<td>149</td>
</tr>
<tr>
<td>constants,Canonical-method (Canonical-class)</td>
<td>46</td>
</tr>
<tr>
<td>constants,Constant-method (Constant-class)</td>
<td>74</td>
</tr>
<tr>
<td>constants,Leaf-method (Leaf-class)</td>
<td>193</td>
</tr>
<tr>
<td>constants,Problem-method (Problem-class)</td>
<td>266</td>
</tr>
<tr>
<td>ConstantSolver</td>
<td>76</td>
</tr>
<tr>
<td>ConstantSolver (ConstantSolver-class)</td>
<td>76</td>
</tr>
<tr>
<td>ConstantSolver-class</td>
<td>76</td>
</tr>
<tr>
<td>constr_value</td>
<td>81</td>
</tr>
<tr>
<td>constr_value,Constraint-method (Constraint-class)</td>
<td>78</td>
</tr>
<tr>
<td>Constraint (Constraint-class)</td>
<td>78</td>
</tr>
<tr>
<td>Constraint-class</td>
<td>78</td>
</tr>
<tr>
<td>constraints (problem-parts)</td>
<td>270</td>
</tr>
<tr>
<td>constraints,Problem-method (Problem-class)</td>
<td>266</td>
</tr>
<tr>
<td>constraints&lt;-(problem-parts)</td>
<td>270</td>
</tr>
<tr>
<td>constraints&lt;-,Problem-method (Problem-class)</td>
<td>266</td>
</tr>
<tr>
<td>construct_intermediate_chain,Problem,list-method</td>
<td>80</td>
</tr>
<tr>
<td>construct_solving_chain</td>
<td>80</td>
</tr>
<tr>
<td>Conv</td>
<td>82</td>
</tr>
<tr>
<td>Conv (Conv-class)</td>
<td>82</td>
</tr>
<tr>
<td>conv</td>
<td>81</td>
</tr>
<tr>
<td>Conv-class</td>
<td>82</td>
</tr>
<tr>
<td>copy,AddExpression-method (+,Expression,missing-method)</td>
<td>11</td>
</tr>
<tr>
<td>copy,GeoMean-method (GeoMean-class)</td>
<td>155</td>
</tr>
<tr>
<td>copy,Power-method (Power-class)</td>
<td>262</td>
</tr>
<tr>
<td>CPLEX_CONIC</td>
<td>84</td>
</tr>
<tr>
<td>CPLEX_CONIC (PLEX_CONIC-class)</td>
<td>83</td>
</tr>
<tr>
<td>CPLEX_CONIC-class</td>
<td>83</td>
</tr>
<tr>
<td>CPLEX_QP</td>
<td>86</td>
</tr>
<tr>
<td>CPLEX_QP (PLEX_QP-class)</td>
<td>85</td>
</tr>
<tr>
<td>CPLEX_QP-class</td>
<td>85</td>
</tr>
<tr>
<td>CumMax</td>
<td>88</td>
</tr>
<tr>
<td>CumMax (CumMax-class)</td>
<td>87</td>
</tr>
<tr>
<td>cummax,Expression-method (cummax_axis)</td>
<td>88</td>
</tr>
<tr>
<td>cummax_axis</td>
<td>88</td>
</tr>
<tr>
<td>CumSum</td>
<td>89</td>
</tr>
<tr>
<td>CumSum (CumSum-class)</td>
<td>89</td>
</tr>
<tr>
<td>cumsum (cumsum_axis)</td>
<td>90</td>
</tr>
<tr>
<td>cumsum,Expression-method (cumsum_axis)</td>
<td>90</td>
</tr>
<tr>
<td>CumSum-class</td>
<td>89</td>
</tr>
<tr>
<td>cumsum_axis</td>
<td>90</td>
</tr>
<tr>
<td>curvature</td>
<td>91</td>
</tr>
<tr>
<td>curvature,Expression-method (curvature)</td>
<td>91</td>
</tr>
<tr>
<td>curvature-atom</td>
<td>91</td>
</tr>
<tr>
<td>curvature-comp</td>
<td>93</td>
</tr>
<tr>
<td>curvature-methods</td>
<td>93</td>
</tr>
<tr>
<td>CvxAttr2Constr</td>
<td>95</td>
</tr>
<tr>
<td>CvxAttr2Constr (CvxAttr2Constr-class)</td>
<td>95</td>
</tr>
<tr>
<td>CvxAttr2Constr-class</td>
<td>95</td>
</tr>
<tr>
<td>CVXOPT</td>
<td>96</td>
</tr>
<tr>
<td>CVXOPT-class</td>
<td>95</td>
</tr>
<tr>
<td>cvxr_norm</td>
<td>97</td>
</tr>
<tr>
<td>Dcp2Cone</td>
<td>98</td>
</tr>
<tr>
<td>Dcp2Cone-class</td>
<td>98</td>
</tr>
<tr>
<td>Dcp2Cone.entr_canon</td>
<td>98</td>
</tr>
<tr>
<td>Dcp2Cone.exp_canon</td>
<td>99</td>
</tr>
<tr>
<td>Dcp2Cone.geo_mean_canon</td>
<td>99</td>
</tr>
<tr>
<td>Dcp2Cone.huber_canon</td>
<td>100</td>
</tr>
<tr>
<td>Dcp2Cone.indicator_canon</td>
<td>100</td>
</tr>
<tr>
<td>Dcp2Cone.kl_div_canon</td>
<td>101</td>
</tr>
<tr>
<td>Dcp2Cone.lambda_max_canon</td>
<td>101</td>
</tr>
<tr>
<td>Dcp2Cone.lambda_sum_largest_canon</td>
<td>102</td>
</tr>
<tr>
<td>Dcp2Cone.log1p_canon</td>
<td>102</td>
</tr>
<tr>
<td>Dcp2Cone.log_canon</td>
<td>103</td>
</tr>
<tr>
<td>Dcp2Cone.log_det_canon</td>
<td>104</td>
</tr>
</tbody>
</table>
Dcp2Cone.log_sum_exp_canon, 104  
Dcp2Cone.logistic_canon, 103  
Dcp2Cone.matrix_frac_canon, 105  
Dcp2Cone.normNuc_canon, 105  
Dcp2Cone.pnorm_canon, 106  
Dcp2Cone.power_canon, 106  
Dcp2Cone.quad_form_canon, 107  
Dcp2Cone.quad_over_lin_canon, 107  
Dcp2Cone.sigma_max_canon, 108  
dgCMatrix-class, 20, 23  
Dgp2Dcp, 109  
Dgp2Dcp(Dgp2Dcp-class), 108  
Dgp2Dcp-class, 108  
Dgp2Dcp.add_canon, 109  
Dgp2Dcp.constant_canon, 110  
Dgp2Dcp.div_canon, 110  
Dgp2Dcp.exp_canon, 111  
Dgp2Dcp.eye_minus_inv_canon, 111  
Dgp2Dcp.geo_mean_canon, 112  
Dgp2Dcp.log_canon, 112  
Dgp2Dcp.mul_canon, 113  
Dgp2Dcp.mul_expression_canon, 113  
Dgp2Dcp.nonpos_constr_canon, 114  
Dgp2Dcp.norm1_canon, 114  
Dgp2Dcp.norm_inf_canon, 115  
Dgp2Dcp.one_minus_pos_canon, 115  
Dgp2Dcp.parameter_canon, 116  
Dgp2Dcp.pf_eigenvalue_canon, 116  
Dgp2Dcp.pnorm_canon, 117  
Dgp2Dcp.power_canon, 117  
Dgp2Dcp.prod_canon, 118  
Dgp2Dcp.quad_form_canon, 118  
Dgp2Dcp.quad_over_lin_canon, 119  
Dgp2Dcp.sum_canon, 119  
Dgp2Dcp.trace_canon, 120  
Dgp2Dcp.zero_constr_canon, 120  
DgpCanonMethods, 121  
DgpCanonMethods-class, 121  
Diag, 121  
diag(diag,Expression-method), 122  
diag,Expression-method, 122  
DiagMat, 123  
DiagMat(DiagMat-class), 123  
DiagMat-class, 122  
DiagVec, 124  
DiagVec(DiagVec-class), 124  
DiagVec-class, 124  
Diff, 125  
diff(diff,Expression-method), 126  
diff,Expression-method, 126  
DiffPos, 127  
dim,Atom-method (Atom-class), 41  
dim,Constant-method (Constant-class), 74  
dim,Constraint-method (Constraint-class), 78  
dim,EqConstraint-method (==,Expression,Expression-method), 35  
dim,Expression-method (Expression-class), 145  
dim,IneqConstraint-method (<=,Expression,Expression-method), 33  
dim,Leaf-method (Leaf-class), 193  
dim,ZeroConstraint-method (ZeroConstraint-class), 339  
dim_from_args, 127  
dim_from_args,AddExpression-method (+,Expression,missing-method), 11  
dim_from_args,Atom-method (dim_from_args), 127  
dim_from_args,AxisAtom-method (AxisAtom-class), 43  
dim_from_args,Conjugate-method (Conjugate-class), 73  
dim_from_args,Conv-method (Conv-class), 82  
dim_from_args,CumMax-method (CumMax-class), 87  
dim_from_args,CumSum-method (CumSum-class), 89  
dim_from_args,DiagMat-method (DiagMat-class), 122  
dim_from_args,DiagVec-method (DiagVec-class), 124  
dim_from_args,DivExpression-method (/,Expression,Expression-method), 32  
dim_from_args,Elementwise-method (Elementwise-class), 133  
dim_from_args,EyeMinusInv-method (EyeMinusInv-class), 151  
dim_from_args,GeoMean-method (GeoMean-class), 155  
dim_from_args,HStack-method
Dim_from_args, HStack-method (HStack-class), 171
Dim_from_args, Imag-method (Imag-class), 176
Dim_from_args, Index-method ([,Expression,missing,missing,ANY-method), 341
Dim_from_args, Kron-method (Kron-class), 185
Dim_from_args, LambdaMax-method (LambdaMax-class), 187
Dim_from_args, LogDet-method (LogDet-class), 202
Dim_from_args, MatrixFrac-method (MatrixFrac-class), 210
Dim_from_args, MulExpression-method (%*%,Expression,Expression-method), 343
Dim_from_args, Multiply-method (Multiply-class), 232
Dim_from_args, NegExpression-method (-,Expression,missing-method), 13
Dim_from_args, NormNuc-method (NormNuc-class), 245
Dim_from_args, OneMinusPos-method (OneMinusPos-class), 250
Dim_from_args, PfEigenvalue-method (PfEigenvalue-class), 256
Dim_from_args, Promote-method (Promote-class), 274
Dim_from_args, QuadForm-method (QuadForm-class), 281
Dim_from_args, QuadOverLin-method (QuadOverLin-class), 283
Dim_from_args, Real-method (Real-class), 288
Dim_from_args, Reshape-method (Reshape-class), 293
Dim_from_args, SigmaMax-method (SigmaMax-class), 301
Dim_from_args, SpecialIndex-method ([,Expression,missing,missing,ANY-method), 341
Dim_from_args, SumLargest-method (SumLargest-class), 315
Dim_from_args, SymbolicQuadForm-method (SymbolicQuadForm-class), 321
Dim_from_args, Trace-method (Trace-class), 324
Dim_from_args, Transpose-method (Transpose-class), 325
Dim_from_args, UpperTri-method (UpperTri-class), 330
Dim_from_args, VStack-method (VStack-class), 337
Dim_from_args, Wrap-method (Wrap-class), 338
DivExpression, 33
DivExpression (/,Expression,Expression-method), 32
DivExpression (/,Expression,Expression-method), 32
domain, 128
domain, Atom-method (Atom-class), 41
domain, Expression-method (Expression-class), 145
domain, Leaf-method (Leaf-class), 193
dspop, 129, 129
dssamp, 129, 129
dual_value (dual_value-methods), 130
dual_value, Constraint-method (Constraint-class), 78
dual_value-methods, 130
dual_value<-(dual_value-methods), 130
dual_value<-,Constraint-method (Constraint-class), 78
ECOS, 131
ECOS(ECOS-class), 130
ECOS-class, 130
ECOS.dims_to_solver_dict, 131
ECOS_BB, 132
ECOS_BB (ECOS_BB-class), 132
ECOS_BB-class, 132
Elementwise, 133
Elementwise (Elementwise-class), 133
Elementwise-class, 133
EliminatePwl, 134
EliminatePwl-class, 134
EliminatePwl.abs_canon, 134
EliminatePwl.cummax_canon, 135
EliminatePwl.cumsum_canon, 135
EliminatePwl.max_elemwise_canon, 136
EliminatePwl.max_entries_canon, 136
EliminatePwl.min_elemwise_canon, 137
EliminatePwl\_min\_entries\_canon, 137
EliminatePwl\_norm1\_canon, 138
EliminatePwl\_norm\_inf\_canon, 138
EliminatePwl\_sum\_largest\_canon, 139
Entr, 140
Entr\ (Entr\-class), 140
entr, 139
Entr\-class, 140
entropy\ (entr), 139
EqConstraint, 36
EqConstraint\-class
\(==,\)Expression,Expression\-method, 35
EvalParams, 141
EvalParams\ (EvalParams\-class), 141
EvalParams\-class, 141
Exp, 143
Exp\ (Exp\-class), 142
exp\,exp,Expression\-method, 142
exp,Expression\-method, 142
Exp\-class, 142
ExpCone, 144
ExpCone\ (ExpCone\-class), 144
ExpCone\-class, 144
expr,Canonical\-method
\(\)Canonical\-class, 46
expr,EqConstraint\-method
\(==,\)Expression,Expression\-method, 35
expr,Expression\-method
\(\)Expression\-class, 145
expr,IneqConstraint\-method
\(<=,\)Expression,Expression\-method, 33
Expression\ (Expression\-class), 145
Expression\-class, 145
expression\-parts, 149
extract\_dual\_value, 150
extract\_mip\_idx, 151
eye\_minus\_inv, 153
EyeMinusInv, 152
EyeMinusInv\ (EyeMinusInv\-class), 151
EyeMinusInv\-class, 151
flatten\,Expression\-method
\(\)Expression\-class, 145
FlipObjective, 154
FlipObjective\ (FlipObjective\-class), 153
FlipObjective\-class, 153
format\_constr, 154
format\_constr\,SOC\-method\ (SOC\-class), 307
format\_constr\,SOC\_Axis\-method
\(\)SOC\_Axis\-class, 309
geo\_mean, 157
GeoMean, 156
GeoMean\ (GeoMean\-class), 155
GeoMean\-class, 155
get\_data, 158
get\_data\,AxisAtom\-method
\(\)AxisAtom\-class, 43
get\_data\,Canonical\-method
\(\)Canonical\-class, 46
get\_data\,Constraint\-method
\(\)Constraint\-class, 78
get\_data\,CumMax\-method\ (CumMax\-class), 87
get\_data\,CumSum\-method\ (CumSum\-class), 89
get\_data\,GeoMean\-method
\(\)GeoMean\-class, 155
get\_data\,Huber\-method\ (Huber\-class), 173
get\_data\,Index\-method
\(\[,\)Expression,missing,missing,ANY\-method), 341
get\_data\,LambdaSumLargest\-method
\(\)LambdaSumLargest\-class, 189
get\_data\,Leaf\-method\ (Leaf\-class), 193
get\_data\,Norm1\-method\ (Norm1\-class), 239
get\_data\,NormInf\-method
\(\)NormInf\-class, 243
get\_data\,Parameter\-method
\(\)Parameter\-class, 254
get\_data\,Pnorm\-method\ (Pnorm\-class), 259
INDEX

359

get_data, Power-method (Power-class), 262
get_data, Promote-method (Promote-class), 274
get_data, Reshape-method (Reshape-class), 293
get_data, SOC-method (SOC-class), 307
get_data, SpecialIndex-method (\[, Expression, index, missing, ANY-method), 340
get_data, SumLargest-method (SumLargest-class), 315
get_data, SymbolicQuadForm-method (SymbolicQuadForm-class), 321
get_data, Transpose-method (Transpose-class), 325
get_dual_values, 159
get_id, 159, 175
get_np, 160
get_problem_data, 160
get_problem_data, Problem, character, logical-method (Problem-class), 266
get_problem_data, Problem, character, missing-method (Problem-class), 266
get_sp, 161
GLPK, 162
GLPK (GLPK-class), 161
GLPK-class, 161
GLPK_MI (GLPK_MI-class), 163
GLPK_MI-class, 163
grad, 164
grad, Atom-method (Atom-class), 41
grad, Constant-method (Constant-class), 74
grad, Expression-method (Expression-class), 145
grad, Parameter-method (Parameter-class), 254
grad, Variable-method (Variable-class), 333
graph_implementation, 165
graph_implementation, AddExpression-method (+, Expression, missing-method), 11
graph_implementation, Atom-method (Atom-class), 41
graph_implementation, Conv-method (Conv-class), 82
GUROBI_QP-class, 168
harmonic_mean, 170
HarmonicMean, 169
HStack, 172
HStack (HStack-class), 171
hstack, 170
HStack-class, 171
Huber, 174
Huber (Huber-class), 173
huber, 172
Huber-class, 173
id, 175
id, Canonical-method (Canonical-class), 46
id, ListORConstr-method (ListORConstr-class), 198
Im, Expression-method (complex-atoms), 53
Imag, 176
Imag (Imag-class), 176
Imag-class, 176
import_solver, 177
import_solver, CBC_CONIC-method (CBC_CONIC-class), 49
import_solver, ConstantSolver-method (ConstantSolver-class), 76
import_solver, CPLEX_CONIC-method (CPLEX_CONIC-class), 83
import_solver, CPLEX_QP-method (CPLEX_QP-class), 85
import_solver, CVXOPT-method (CVXOPT-class), 95
import_solver, ECOS-method (ECOS-class), 130
import_solver, GLPK-method (GLPK-class), 161
import_solver, GUROBI_CONIC-method (GUROBI_CONIC-class), 166
import_solver, GUROBI_QP-method (GUROBI_QP-class), 168
import_solver, MOSEK-method (MOSEK-class), 229
import_solver, OSQP-method (OSQP-class), 252
import_solver, ReductionSolver-method (ReductionSolver-class), 291
import_solver, SCS-method (SCS-class), 298
Index, 341, 342
Index
([,Expression,missing,missing,ANY-method), 341
Index-class
([,Expression,missing,missing,ANY-method), 341
IneqConstraint, 35
IneqConstraint-class
(<,Expression,Expression-method), 33
installed_solvers, 177
inv_pos, 179
InverseData, 48, 54, 71, 86, 109, 169, 178, 253, 256, 268, 329
InverseData-class, 178
invert, 178
invert, Canonicalization, Solution, InverseData-method (Canonicalization-class), 48
invert, CBC_CONIC, list, list-method (CBC_CONIC-class), 49
invert, Chain, SolutionORList, list-method (Chain-class), 52
invert, Complex2Real, Solution, InverseData-method (Complex2Real-class), 54
invert, ConicSolver, Solution, InverseData-method (ConicSolver-class), 71
invert, ConstantSolver, Solution, list-method (ConstantSolver-class), 76
invert, CPLEX_CONIC, list, list-method (CPLEX_CONIC-class), 83
invert, CPLEX_QP, list, InverseData-method (CPLEX_QP-class), 85
invert, CvxAttr2Constr, Solution, list-method (CvxAttr2Constr-class), 95
invert, CVXOPT, list, list-method (CVXOPT-class), 95
invert, Dgp2Dcp, Solution, InverseData-method (Dgp2Dcp-class), 108
invert, ECOS, list, list-method (ECOS-class), 130
invert, EvalParams, Solution, list-method (EvalParams-class), 141
invert, FlipObjective, Solution, list-method (FlipObjective-class), 153
invert, GLPK, list, list-method (GLPK-class), 161
invert, GUROBI_CONIC, list, list-method
invert, GUROBI_QP-class, 168
invert, MatrixStuffing-class, 212
invert, MOSEK-class, 229
invert, OSQP-class, 252
invert, Reduction-class, 289
invert, SCS-class, 298
is.element, ANY-class, 286
is_affine, Expression-class, 145
is_atom_affine, Atom-class, 91
is_atom_concave, Atom-class, 91
is_atom_concave, Abs-class, 37
is_atom_concave, AffAtom-class, 39
is_atom_concave, Atom-class, 91
is_atom_concave, CumMax-class, 87
is_atom_concave, DivExpression-class, 32
is_atom_concave, Entr-class, 140
is_atom_concave, Exp-class, 142
is_atom_concave, EyeMinusInv-class, 151
is_atom_concave, GeoMean-class, 155
is_atom_concave, Huber-class, 173
is_atom_concave, KLDiv-class, 183
is_atom_concave, LambdaMax-class, 187
is_atom_concave, Log-class, 199
is_atom_concave, LogDet-class, 202
is_atom_concave, Logistic-class, 204
is_atom_concave, LogSumExp-class, 205
is_atom_concave, MatrixExp-class, 210
is_atom_concave, MaxElemwise-class, 215
is_atom_concave, MaxEntries-class, 216
is_atom_concave, MinElemwise-class, 222
is_atom_concave, MinEntries-class, 223
is_atom_concave, MulExpression-class, 343
is_atom_concave, Norm1-class, 239
is_atom_concave, NormInf-class, 243
is_atom_concave, NormNuc-class, 245
is_atom_concave, OneMinusPos-class, 250
is_atom_concave, PFEigenvvalue-class, 256
is_atom_concave, Pnorm-class, 259
is_atom_concave, Power-class, 262
is_atom_concave, ProdEntries-class, 270
is_atom_concave, QuadForm-class, 281
is_atom_concave, QuadOverLin-class, 283
is_atom_concave, SigmaMax-class, 301
is_atom_concave, SumLargest-class, 315
is_atom_concave, SymbolicQuadForm-class, 321
is_atom_convex, curvature-atom, 91
is_atom_convex, Abs-class, 91
<table>
<thead>
<tr>
<th>is_atom_log_log_concave</th>
<th>(HStack-class), 171</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index-method</td>
<td>(UpperTri-class), 330</td>
</tr>
<tr>
<td>Log-method</td>
<td>(UpperTri-class), 338</td>
</tr>
<tr>
<td>MaxElemwise-method</td>
<td>(Wrap-class), 337</td>
</tr>
<tr>
<td>MaxEntries-method</td>
<td>(VStack-class), 338</td>
</tr>
<tr>
<td>MinElemwise-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>MinEntries-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>MulExpression-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>Multiply-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>NormInf-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>OneMinusPos-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>PfEigenvalue-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>DiagMat-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>DiagVec-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>DivExpression-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>Exp-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>EyeMinusInv-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>GeoMean-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>HStack-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>HStack-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>HStack-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>Index-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>Log-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>MaxElemwise-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>MaxEntries-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>MinElemwise-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>MinEntries-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>MulExpression-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>Multiply-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>NormInf-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>OneMinusPos-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>PfEigenvalue-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>DiagMat-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>DiagVec-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>DivExpression-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>Exp-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>EyeMinusInv-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>GeoMean-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>HStack-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>HStack-method</td>
<td>(Wrap-class), 338</td>
</tr>
<tr>
<td>HStack-method</td>
<td>(Wrap-class), 338</td>
</tr>
</tbody>
</table>
is_atom_log_log_convex, OneMinusPos-method (OneMinusPos-class), 250
is_atom_log_log_convex, PfEigenvalue-method (PfEigenvalue-class), 256
is_atom_log_log_convex, Pnorm-method (Pnorm-class), 259
is_atom_log_log_convex, Power-method (Power-class), 262
is_atom_log_log_convex, ProdEntries-method (ProdEntries-class), 270
is_atom_log_log_convex, Promote-method (Promote-class), 274
is_atom_log_log_convex, QuadForm-method (QuadForm-class), 281
is_atom_log_log_convex, QuadOverLin-method (QuadOverLin-class), 283
is_atom_log_log_convex, Reshape-method (Reshape-class), 293
is_atom_log_log_convex, SpecialIndex-method ([,Expression,index,missing,ANY-method), 340
is_atom_log_log_convex, SumEntries-method (SumEntries-class), 314
is_atom_log_log_convex, Trace-method (Trace-class), 324
is_atom_log_log_convex, Transpose-method (Transpose-class), 325
is_atom_log_log_convex, UpperTri-method (UpperTri-class), 330
is_atom_log_log_convex, VStack-method (VStack-class), 337
is_atom_log_log_convex, Wrap-method (Wrap-class), 338
is_complex (complex-methods), 54
is_complex, AffAtom-method (AffAtom-class), 39
is_complex, Atom-method (Atom-class), 41
is_complex, BinaryOperator-method (BinaryOperator-class), 44
is_complex, Constant-method (Constant-class), 74
is_complex, Constraint-method (Constraint-class), 78
is_complex, Expression-method (Expression-class), 145
is_complex, Imag-method (Imag-class), 176
is_complex, Leaf-method (Leaf-class), 193
is_complex, Real-method (Real-class), 288
is_concave (curvature-methods), 93
is_concave, Atom-method (Atom-class), 41
is_concave, Expression-method (Expression-class), 145
is_concave, Leaf-method (Leaf-class), 193
is_constant (curvature-methods), 93
is_constant, Expression-method (Expression-class), 145
is_constant, Power-method (Power-class), 262
is_complex, Expression-method (Expression-class), 145
is_complex, Leaf-method (Leaf-class), 193
is_dcp, 179
is_dcp, Constraint-method (Constraint-class), 78
is_dcp, EqConstraint-method (==,Expression,Expression-method), 35
is_dcp, ExpCone-method (ExpCone-class), 144
is_dcp, Expression-method (Expression-class), 145
is_dcp, IneqConstraint-method (<=,Expression,Expression-method), 33
is_dcp, Maximize-method (Maximize-class), 218
is_dcp, Minimize-method (Minimize-class), 225
is_dcp, NonPosConstraint-method (NonPosConstraint-class), 236
is_dcp, Problem-method (Problem-class), 266
is_dcp, PSDConstraint-method (>>,%), 345
is_dcp, SOC-method (SOC-class), 307
is_dcp, ZeroConstraint-method (ZeroConstraint-class), 339
is_decr (curvature-comp), 93
is_decr, Abs-method (Abs-class), 37
is_decr, AffAtom-method (AffAtom-class), 39
is_decr, Atom-method (curvature-comp), 93
is_decr, Conjugate-method (Conjugate-class), 73
is_decr, Conv-method (Conv-class), 82
<table>
<thead>
<tr>
<th>is_decr, CumMax-method (CumMax-class), 87</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_decr, DivExpression-method ((/), Expression, Expression-method), 32</td>
</tr>
<tr>
<td>is_decr, Entr-method (Entr-class), 140</td>
</tr>
<tr>
<td>is_decr, Exp-method (Exp-class), 142</td>
</tr>
<tr>
<td>is_decr, EyeMinusInv-method (EyeMinusInv-class), 151</td>
</tr>
<tr>
<td>is_decr, GeoMean-method (GeoMean-class), 155</td>
</tr>
<tr>
<td>is_decr, Huber-method (Huber-class), 173</td>
</tr>
<tr>
<td>is_decr, Kron-method (Kron-class), 185</td>
</tr>
<tr>
<td>is_decr, LambdaMax-method (LambdaMax-class), 187</td>
</tr>
<tr>
<td>is_decr, Log-method (Log-class), 199</td>
</tr>
<tr>
<td>is_decr, LogDet-method (LogDet-class), 202</td>
</tr>
<tr>
<td>is_decr, Logistic-method (Logistic-class), 204</td>
</tr>
<tr>
<td>is_decr, LogSumExp-method (LogSumExp-class), 205</td>
</tr>
<tr>
<td>is_decr, MatrixFrac-method (MatrixFrac-class), 210</td>
</tr>
<tr>
<td>is_decr, MaxElemwise-method (MaxElemwise-class), 215</td>
</tr>
<tr>
<td>is_decr, MaxEntries-method (MaxEntries-class), 216</td>
</tr>
<tr>
<td>is_decr, MinElemwise-method (MinElemwise-class), 222</td>
</tr>
<tr>
<td>is_decr, MinEntries-method (MinEntries-class), 223</td>
</tr>
<tr>
<td>is_decr, MulExpression-method ((%*%), Expression, Expression-method), 343</td>
</tr>
<tr>
<td>is_decr, NegExpression-method ((-), Expression, missing-method), 13</td>
</tr>
<tr>
<td>is_decr, Norm1-method (Norm1-class), 239</td>
</tr>
<tr>
<td>is_decr, NormInf-method (NormInf-class), 243</td>
</tr>
<tr>
<td>is_decr, NormNuc-method (NormNuc-class), 245</td>
</tr>
<tr>
<td>is_decr, OneMinusPos-method (OneMinusPos-class), 250</td>
</tr>
<tr>
<td>is_decr, PFEigenvalue-method (PFEigenvalue-class), 256</td>
</tr>
<tr>
<td>is_decr, Pnorm-method (Pnorm-class), 259</td>
</tr>
<tr>
<td>is_decr, Power-method (Power-class), 262</td>
</tr>
<tr>
<td>is_decr, ProdEntries-method (ProdEntries-class), 270</td>
</tr>
<tr>
<td>is_decr, QuadForm-method (QuadForm-class), 281</td>
</tr>
<tr>
<td>is_decr, QuadOverLin-method (QuadOverLin-class), 283</td>
</tr>
<tr>
<td>is_decr, SigmaMax-method (SigmaMax-class), 301</td>
</tr>
<tr>
<td>is_decr, SumLargest-method (SumLargest-class), 315</td>
</tr>
<tr>
<td>is_decr, SymbolicQuadForm-method (SymbolicQuadForm-class), 321</td>
</tr>
<tr>
<td>is_dgp, 180</td>
</tr>
<tr>
<td>is_dgp, Constraint-method (Constraint-class), 78</td>
</tr>
<tr>
<td>is_dgp, EqConstraint-method (==, Expression, Expression-method), 35</td>
</tr>
<tr>
<td>is_dgp, ExpCone-method (ExpCone-class), 144</td>
</tr>
<tr>
<td>is_dgp, Expression-method (Expression-class), 145</td>
</tr>
<tr>
<td>is_dgp, IneqConstraint-method (&lt;=, Expression, Expression-method), 33</td>
</tr>
<tr>
<td>is_dgp, Maximize-method (Maximize-class), 218</td>
</tr>
<tr>
<td>is_dgp, Minimize-method (Minimize-class), 225</td>
</tr>
<tr>
<td>is_dgp, NonPosConstraint-method (NonPosConstraint-class), 236</td>
</tr>
<tr>
<td>is_dgp, Problem-method (Problem-class), 266</td>
</tr>
<tr>
<td>is_dgp, PSDConstraint-method ((%*%)), 345</td>
</tr>
<tr>
<td>is_dgp, SOC-method (SOC-class), 307</td>
</tr>
<tr>
<td>is_dgp, ZeroConstraint-method (ZeroConstraint-class), 339</td>
</tr>
<tr>
<td>is_hermitian (matrix_prop-methods), 214</td>
</tr>
<tr>
<td>is_hermitian, AddExpression-method (+, Expression, missing-method), 11</td>
</tr>
<tr>
<td>is_hermitian, Conjugate-method (Conjugate-class), 73</td>
</tr>
<tr>
<td>is_hermitian, Constant-method (Constant-class), 74</td>
</tr>
<tr>
<td>is_hermitian, DiagVec-method (DiagVec-class), 124</td>
</tr>
</tbody>
</table>
is_log_log_affine (log_log_curvature-methods), 209
is_log_log_affine,Expression-method (Expression-class), 145
is_log_log_concave (log_log_curvature-methods), 209
is_log_log_concave,Atom-method (Atom-class), 41
is_log_log_concave,Expression-method (Expression-class), 145
is_log_log_concave,Leaf-method (Leaf-class), 193
is_log_log_constant (log_log_curvature-methods), 209
is_log_log_constant,Expression-method (Expression-class), 145
is_log_log_convex (log_log_curvature-methods), 209
is_log_log_convex,Atom-method (Atom-class), 41
is_log_log_convex,Expression-method (Expression-class), 145
is_log_log_convex,Leaf-method (Leaf-class), 193
is_matrix (size-methods), 306
is_matrix,Expression-method (Expression-class), 145
is_mixed_integer, 180
is_mixed_integer,Problem-method (Problem-class), 266
is_neg (leaf-attr), 193
is_neg,Leaf-method (Leaf-class), 193
is_nonneg (sign-methods), 304
is_nonneg,Atom-method (Atom-class), 41
is_nonneg,Constant-method (Constant-class), 74
is_nonneg,Expression-method (Expression-class), 145
is_nonneg,Leaf-method (Leaf-class), 193
is_nonneg,MaxElemwise-method (MaxElemwise-class), 215
is_pwl,MaxEntries-method (MaxEntries-class), 216
is_pwl,MinElemwise-method (MinElemwise-class), 222
is_pwl,MinEntries-method (MinEntries-class), 223
is_pwl,Norm1-method (Norm1-class), 239
is_pwl,NormInf-method (NormInf-class), 243
is_pwl,Pnorm-method (Pnorm-class), 259
is_pwl, QuadForm-method (QuadForm-class), 281
is_qp, 181
is_qp, Problem-method (Problem-class), 281
is_qpwa (curvature-methods), 93
is_qpwa, AffAtom-method (AffAtom-class), 39
is_qpwa, DivExpression-method (/, Expression, Expression-method), 32
is_qpwa, Expression-method (Expression-class), 145
is_qpwa, MatrixFrac-method (MatrixFrac-class), 210
is_qpwa, Objective-method (Objective-class), 249
is_qpwa, Power-method (Power-class), 262
is_qpwa, QuadOverLin-method (QuadOverLin-class), 283
is_quadratic (curvature-methods), 93
is_quadratic, AffAtom-method (AffAtom-class), 39
is_quadratic, DivExpression-method (/, Expression, Expression-method), 32
is_quadratic, Expression-method (Expression-class), 145
is_quadratic, Huber-method (Huber-class), 173
is_quadratic, Leaf-method (Leaf-class), 193
is_quadratic, MatrixFrac-method (MatrixFrac-class), 210
is_quadratic, Objective-method (Objective-class), 249
is_quadratic, Power-method (Power-class), 262
is_quadratic, QuadForm-method (QuadForm-class), 281
is_quadratic, QuadOverLin-method (QuadOverLin-class), 283
is_quadratic, SymbolicQuadForm-method (SymbolicQuadForm-class), 321
is_real (complex-methods), 54
is_real, Constraint-method (Constraint-class), 78
is_real, Expression-method (Expression-class), 145

kl_div, 184
KLDiv, 183
KLDiv (KLDiv-class), 183
KLDiv-class, 183
Kron, 185
Kron (Kron-class), 185
Kron-class, 185
INDEX 369

kronecker
  (kronecker, Expression, ANY-method), 186
kronecker, ANY, Expression-method
  (kronecker, Expression, ANY-method), 186
kronecker, Expression, ANY-method, 186

lambda_max, 190
lambda_min, 191
lambda_sum_largest, 192
LambdaMax, 188
LambdaMax (LambdaMax-class), 187
LambdaMax-class, 187
LambdaMin, 188
LambdaSumLargest, 189
LambdaSumLargest
  (LambdaSumLargest-class), 189
LambdaSumLargest-class, 189
LambdaSumSmallest, 190
Leaf, 46, 149, 193, 195, 255, 273, 332, 334
Leaf (Leaf-class), 193
leaf-atr, 193
Leaf-class, 193
length, Rdict-method (Rdict-class), 286
linearize, 197

ListORConstr-class, 198

Log, 200
Log (Log-class), 199
log (log, Expression-method), 198
log, Expression-method, 198
Log-class, 199
log10 (log, Expression-method), 198
log10, Expression-method
  (log, Expression-method), 198
Log1p, 201
Log1p (Log1p-class), 201
log1p (log, Expression-method), 198
log1p, Expression-method
  (log, Expression-method), 198
Log1p-class, 201
log2 (log, Expression-method), 198
log2, Expression-method
  (log, Expression-method), 198
log_det, 207
log_log_curvature, 208
log_log_curvature, Expression-method
  (log_log_curvature), 208
log_log_curvature-atom, 208
log_log_curvature-methods, 209
log_sum_exp, 209
LogDet, 203
LogDet (LogDet-class), 202
LogDet-class, 202
Logistic, 205
Logistic (Logistic-class), 204
logistic, 203
Logistic-class, 204
LogSumExp, 206
LogSumExp (LogSumExp-class), 205
LogSumExp-class, 205

matrix_frac, 213
matrix_prop-methods, 214
matrix_trace, 214
MatrixFrac, 211
MatrixFrac (MatrixFrac-class), 210
MatrixFrac-class, 210
MatrixStuffing, 212
MatrixStuffing (MatrixStuffing-class), 212
MatrixStuffing-class, 212
max (max_entries), 220
max_elemwise, 219
max_entries, 220
MaxElemwise, 216
MaxElemwise (MaxElemwise-class), 215
MaxElemwise-class, 215
MaxEntries, 217
MaxEntries (MaxEntries-class), 216
MaxEntries-class, 216
Maximize, 182, 219, 249, 268, 269
Maximize (Maximize-class), 218
Maximize-class, 218
mean (mean, Expression), 221
mean, Expression, 221
min (min_entries), 226
min_elemwise, 226
min_entries, 226
MinElemwise, 223
MinElemwise (MinElemwise-class), 222
MinElemwise-class, 222
MinEntries, 224
MinEntries (MinEntries-class), 223
MinEntries-class, 223
Minimize, 182, 225, 249, 268, 269
Minimize (Minimize-class), 225
Minimize-class, 225
mip_capable, 227
mip_capable,CBC_CONIC-method
(CBC_CONIC-class), 49
mip_capable,ConstantSolver-method
(ConstantSolver-class), 76
mip_capable,CPLEX_CONIC-method
(CPLEX_CONIC-class), 83
mip_capable,CPLEX_QP-method
(CPLEX_QP-class), 85
mip_capable,CVXOPT-method
(CVXOPT-class), 95
mip_capable,ECOS-method
(ECOS-class), 130
mip_capable,ECOS_BB-method
(ECOS_BB-class), 132
mip_capable,GLPK-method
(GLPK-class), 161
mip_capable,GLPK_MI-method
(GLPK_MI-class), 163
mip_capable,GUROBI_CONIC-method
(GUROBI_CONIC-class), 166
mip_capable,GUROBI_QP-method
(GUROBI_QP-class), 168
mip_capable,MOSEK-method
(MOSEK-class), 229
mip_capable,ReductionSolver-method
(ReductionSolver-class), 291
mip_capable,SCS-method
(SCS-class), 298
mixed_norm, 228
MixedNorm, 228
MOSEK, 230
MOSEK (MOSEK-class), 229
MOSEK-class, 229
MOSEK.parse_dual_vars, 231
MOSEK.recover_dual_vars, 231
MulExpression, 344
MulExpression
(%,%,Expression,Expression-method), 343
MulExpression-class
(%,%,Expression,Expression-method), 343
Multiply, 232, 343, 344
Multiply (Multiply-class), 232
multiply, 232
Multiply-class, 232
name, 234
name,AddExpression-method
(+,Expression,missing-method), 11
name,Atom-method
(Atom-class), 41
name,BinaryOperator-method
(BinaryOperator-class), 44
name,CBC_CONIC-method
(CBC_CONIC-class), 49
name,Constant-method
(Constant-class), 74
name,ConstantSolver-method
(ConstantSolver-class), 76
name,CPLEX_CONIC-method
(CPLEX_CONIC-class), 83
name,CPLEX_QP-method
(CPLEX_QP-class), 85
name,CVXOPT-method
(CVXOPT-class), 95
name,ECOS-method
(ECOS-class), 130
name,ECOS_BB-method
(ECOS_BB-class), 132
name,EqConstraint-method
(==,Expression,Expression-method), 35
name,Expression-method
(Expression-class), 145
name,EyeMinusInv-method
(EyeMinusInv-class), 151
name,GeoMean-method
(GeoMean-class), 155
name,GLPK-method
(GLPK-class), 161
name,GLPK_MI-method
(GLPK_MI-class), 163
name,GUROBI_CONIC-method
(GUROBI_CONIC-class), 166
name,GUROBI_QP-method
(GUROBI_QP-class), 168
name,IneqConstraint-method
(<=,Expression,Expression-method), 33
name,MOSEK-method
(MOSEK-class), 229
name,NonP0sConstraint-method
(NonPosConstraint-class), 236
name,Norm1-method
(Norm1-class), 239
name,NormInf-method
(NormInf-class), 243
name,OneMinusPos-method
(OneMinusPos-class), 250
name,OSQP-method
(OSQP-class), 252
name,Parameter-method
(Parameter-class), 254
name,PfEigenvalue-method
(PfEigenvalue-class), 256
INDEX

name, Pnorm-method (Pnorm-class), 259
name, Power-method (Power-class), 262
name, PSDConstraint-method (%>>%), 345
name, QuadForm-method (QuadForm-class), 281
name, ReductionSolver-method (ReductionSolver-class), 291
name, SCS-method (SCS-class), 298
name, SpecialIndex-method ([, Expression, index, missing, ANY-method), 340
name, UnaryOperator-method (UnaryOperator-class), 328
name, Variable-method (Variable-class), 333
name, ZeroConstant-method (ZeroConstraint-class), 339
names, DgpCanonMethods-method (DgpCanonMethods-class), 121
ncol, Atom-method (Atom-class), 41
ncol, Expression-method (Expression-class), 145
ndim, Expression-method (Expression-class), 145
Neg, 234
neg, 235
NegExpression, 14
NegExpression (^, Expression, missing-method), 13
NegExpression-class (^, Expression, missing-method), 13
NonlinearConstraint (NonlinearConstraint-class), 235
NonlinearConstraint-class, 235
NonPosConstraint, 236
NonPosConstraint-class, 236
Norm, 237
norm, 97
norm (norm, Expression, character-method), 237
norm, Expression, character-method, 237
Norm1, 240
Norm1 (Norm1-class), 239
norm1, 238
Norm2, 241
norm2, 242
norm_inf, 246
norm_nuc, 247
NormInf, 244
NormInf (NormInf-class), 243
NormInf-class, 243
NormNuc, 245
NormNuc (NormNuc-class), 245
NormNuc-class, 245
nrow, Atom-method (Atom-class), 41
nrow, Expression-method (Expression-class), 145
num_cones (cone-methods), 69
num_cones, ExpCone-method (ExpCone-class), 144
num_cones, SOC-method (SOC-class), 307
num_cones, SOCAxis-method (SOCAxis-class), 309
Objective, 182, 250
Objective (Objective-class), 249
objective (problem-parts), 270
objective, Problem-method (Problem-class), 266
Objective-arith, 248
Objective-class, 249
objective<-(problem-parts), 270
objective<-, Problem-method (Problem-class), 266
one_minus_pos, 252
OneMinusPos, 251
OneMinusPos (OneMinusPos-class), 250
OneMinusPos-class, 250
OSQP, 253
OSQP (OSQP-class), 252
OSQP-class, 252
p_norm, 238, 278
Parameter, 47, 150, 196, 234, 255, 269, 333
Parameter (Parameter-class), 254
Parameter-class, 254
parameters (expression-parts), 149
parameters, Canonical-method (Canonical-class), 46
parameters, Leaf-method (Leaf-class), 193
parameters, Parameter-method (Parameter-class), 254
parameters, Problem-method (Problem-class), 266
perform, 256
perform, Canonicalization, Problem-method (Canonicalization-class), 48
perform, CBC_CONIC, Problem-method (CBC_CONIC-class), 49
perform, Chain, Problem-method (Chain-class), 52
perform, Complex2Real, Problem-method (Complex2Real-class), 54
perform, ConstantSolver, Problem-method (ConstantSolver-class), 76
perform, CBC_CONIC, Problem-method (CBC_CONIC-class), 49
perform, CVXOPT, Problem-method (CVXOPT-class), 95
perform, Dcp2Cone, Problem-method (Dcp2Cone-class), 98
perform, Dgp2Dcp, Problem-method (Dgp2Dcp-class), 108
perform, ECOS, Problem-method (ECOS-class), 130
perform, ECOS_BB, Problem-method (ECOS_BB-class), 132
perform, EvalParams, Problem-method (EvalParams-class), 141
perform, FlipObjective, Problem-method (FlipObjective-class), 153
perform, GUROBI_CONIC, Problem-method (GUROBI_CONIC-class), 166
perform, MatrixStuffing, Problem-method (MatrixStuffing-class), 212
perform, MOSEK, Problem-method (MOSEK-class), 229
perform, QpSolver, Problem-method (QpSolver-class), 280
perform, Reduction, Problem-method (Reduction-class), 289
perform, SCS, Problem-method (SCS-class), 298
pf_eigenvalue, 258
PFEigenvalue, 257
PFEigenvalue (PFEigenvalue-class), 256
PFEigenvalue-class, 256
Pnorm, 260
Pnorm (Pnorm-class), 259
Pnorm-class, 259
Pos, 261
pos, 262
Power, 264
Power (Power-class), 262
power (^, Expression, numeric-method), 346
Power-class, 262
prepend, SolvingChain, Chain-method (SolvingChain-class), 311
Problem (Problem-class), 266
Problem-arith, 265
Problem-class, 266
problem-parts, 270
prod (prod_entries), 272
prod_entries, 272
ProdEntries, 271
ProdEntries (ProdEntries-class), 270
ProdEntries-class, 270
project (project-methods), 273
project, Leaf-method (Leaf-class), 193
project-methods, 273
project_and_assign (project-methods), 273
project_and_assign, Leaf-method (Leaf-class), 193
Promote, 274
Promote (Promote-class), 274
Promote-class, 274
psd_coeff_offset, 276
PSDConstraint, 346
PSDConstraint (%>>%), 345
PSDConstraint-class (%>>%), 345
PSDWrap, 275
PSDWrap (PSDWrap-class), 275
PSDWrap-class, 275
psolve, 276
psolve, Problem-method (psolve), 276
Qp2SymbolicQp-class, 280
QpMatrixStuffing (QpMatrixStuffing-class), 280
QpMatrixStuffing-class, 280
QpSolver, 280
QpSolver-class, 280
quad_form, 285
quad_over_lin, 285
QuadForm, 282
QuadForm (QuadForm-class), 281
QuadForm-class, 281
QuadOverLin, 284
QuadOverLin (QuadOverLin-class), 283
QuadOverLin-class, 283

Rdict, 287, 288
Rdict (Rdict-class), 286
Rdict-class, 286
Rdictdefault, 287
Rdictdefault (Rdictdefault-class), 287
Rdictdefault-class, 287
Real, 288
Real (Real-class), 288
Real-class, 288
reduce, 289, 297
reduce, Reduction-method
   (Reduction-class), 289
Reduction, 38, 178, 256, 289, 290, 296
Reduction-class, 289
reduction_format_constr, ConicSolver-method
   (ConicSolver-class), 71
reduction_format_constr, SCS-method
   (SCS-class), 298
reduction_solve, ConstantSolver, ANY-method
   (ConstantSolver-class), 76
reduction_solve, ReductionSolver, ANY-method
   (ReductionSolver-class), 291
reduction_solve, SolvingChain, Problem-method
   (SolvingChain-class), 311
reduction_solve via_data, SolvingChain-method
   (SolvingChain-class), 311
ReductionSolver, 177, 227, 292
ReductionSolver-class, 291
remove_from_solver_blacklist
   (installed_solvers), 177
resetOptions, 293
Reshape, 294
Reshape (Reshape-class), 293
reshape (reshape_expr), 294
Reshape-class, 293
reshape_expr, 294
residual (residual-methods), 296
residual, Constraint-method
   (Constraint-class), 78
residual, EqConstraint-method
   (==, Expression, Expression-method), 35
residual, ExpCone-method
   (ExpCone-class), 144
residual, IneqConstraint-method
   (<=, Expression, Expression-method), 33
residual, NonPosConstraint-method
   (NonPosConstraint-class), 236
residual, PSDConstraint-method (%>>%), 345
residual, SOC-method (SOC-class), 307
residual, ZeroConstraint-method
   (ZeroConstraint-class), 339
residual-methods, 296
retrieve, 296
retrieve, Reduction, Solution-method
   (Reduction-class), 289
scaled_lower_tri, 297
scalene, 297
SCS, 299
SCS (SCS-class), 298
SCS-class, 298
SCS.dims_to_solver_dict, 300
SCS.extract_dual_value, 300
set_solver_blacklist
   (installed_solvers), 177
setIdCounter, 175, 301
show, Constant-method (Constant-class), 74
sigma_max, 303
SigmaMax, 302
SigmaMax (SigmaMax-class), 301
SigmaMax-class, 301
sign, Expression-method, 303
sign-methods, 304
sign_from_args, 305
sign_from_args, Abs-method (Abs-class), 37
sign_from_args, AffAtom-method
   (AffAtom-class), 39
sign_from_args, Atom-method
   (sign_from_args), 305
sign_from_args, BinaryOperator-method
   (BinaryOperator-class), 44
INDEX

sign_from_args, Conv-method (Conv-class), 82
sign_from_args, CumMax-method (CumMax-class), 87
sign_from_args, Entr-method (Entr-class), 140
sign_from_args, Exp-method (Exp-class), 142
sign_from_args, EyeMinusInv-method (EyeMinusInv-class), 151
sign_from_args, GeoMean-method (GeoMean-class), 155
sign_from_args, Huber-method (Huber-class), 173
sign_from_args, KLDiv-method (KLDiv-class), 183
sign_from_args, Kron-method (Kron-class), 185
sign_from_args, LambdaMax-method (LambdaMax-class), 187
sign_from_args, Log-method (Log-class), 199
sign_from_args, Log1p-method (Log1p-class), 201
sign_from_args, LogDet-method (LogDet-class), 202
sign_from_args, Logistic-method (Logistic-class), 204
sign_from_args, LogSumExp-method (LogSumExp-class), 205
sign_from_args, MatrixFrac-method (MatrixFrac-class), 210
sign_from_args, MaxElemwise-method (MaxElemwise-class), 215
sign_from_args, MaxEntries-method (MaxEntries-class), 216
sign_from_args, MinElemwise-method (MinElemwise-class), 222
sign_from_args, MinEntries-method (MinEntries-class), 223
sign_from_args, NegExpression-method (-, Expression, missing-method), 13
sign_from_args, Norm1-method (Norm1-class), 239
sign_from_args, NormInf-method (NormInf-class), 243
sign_from_args, NormNuc-method (NormNuc-class), 245
sign_from_args, OneMinusPos-method (OneMinusPos-class), 250
sign_from_args, PFEigenvalue-method (PFEigenvalue-class), 256
sign_from_args, Pnorm-method (Pnorm-class), 259
sign_from_args, Power-method (Power-class), 262
sign_from_args, ProdEntries-method (ProdEntries-class), 270
sign_from_args, QuadForm-method (QuadForm-class), 281
sign_from_args, QuadOverLin-method (QuadOverLin-class), 283
sign_from_args, SigmaMax-method (SigmaMax-class), 291
sign_from_args, SumLargest-method (SumLargest-class), 315
sign_from_args, SymbolicQuadForm-method (SymbolicQuadForm-class), 321
size, 305
size, Constraint-method (Constraint-class), 78
size, EqConstraint-method (==, Expression, Expression-method), 35
size, ExpCone-method (ExpCone-class), 144
size, Expression-method (Expression-class), 145
size, IneqConstraint-method (<=, Expression, Expression-method), 33
size, ListORExpr-method (size), 305
size, SOC-method (SOC-class), 307
size, SOCAxis-method (SOCAxis-class), 309
size, ZeroConstraint-method (Constraint-class), 78
size-methods, 306
size_metrics (problem-parts), 270
size_metrics, Problem-method (Problem-class), 266
SizeMetrics (SizeMetrics-class), 307
SOC, 308
SOC (SOC-class), 307
SOC-class, 307
SOCAxis, 69, 310
INDEX

SOCAxis (SOCAxis-class), 309
SOCAxis-class, 309
Solution, 48, 52, 54, 71, 77, 78, 95, 109, 141, 154, 178, 212, 268, 290, 296, 297, 310, 329
Solution-class, 310
solve (psolve), 276
solve, Problem, ANY-method (psolve), 276
solve_via_data, CBC_CONIC-method (CBC_CONIC-class), 49
solve_via_data, ConstantSolver-method (ConstantSolver-class), 76
solve_via_data, CPLEX_CONIC-method (CPLEX_CONIC-class), 83
solve_via_data, CPLEX_QP-method (CPLEX_QP-class), 85
solve_via_data, CVXOPT-method (CVXOPT-class), 95
solve_via_data, ECOS-method (ECOS-class), 130
solve_via_data, ECOS_BB-method (ECOS_BB-class), 132
solve_via_data, GLPK-method (GLPK-class), 161
solve_via_data, GLPK_MI-method (GLPK_MI-class), 163
solve_via_data, MOSEK-method (MOSEK-class), 229
solve_via_data, OSQP-method (OSQP-class), 252
solve_via_data, ReductionSolver-method (ReductionSolver-class), 291
solve_via_data, SCS-method (SCS-class), 298
solver_stats, Problem-method (Problem-class), 266
solver_stats<-, Problem-method (Problem-class), 266
SolverStats (SolverStats-class), 311
SolverStats-class, 311
SolvingChain, 81, 312
SolvingChain-class, 311
SpecialIndex

SpecialIndex-class

(sqrt, sqrt, Expression-method), 313
sqrt, Expression-method, 313
square (square, Expression-method), 313
square, Expression-method, 313
status, Problem-method (Problem-class), 266
status_map, CBC_CONIC-method (CBC_CONIC-class), 49
status_map, CPLEX_CONIC-method (CPLEX_CONIC-class), 83
status_map, CPLEX_QP-method (CPLEX_QP-class), 85
status_map, CVXOPT-method (CVXOPT-class), 95
status_map, ECOS-method (ECOS-class), 130
status_map, GLPK-method (GLPK-class), 161
status_map, GLPK_MI-method (GLPK_MI-class), 163
status_map, GUROBI_CONIC-method (GUROBI_CONIC-class), 166
status_map, GUROBI_QP-method (GUROBI_QP-class), 168
status_map, OSQP-method (OSQP-class), 252
status_map, SCS-method (SCS-class), 298
status_map_lp, CBC_CONIC-method (CBC_CONIC-class), 49
status_map_mip, CBC_CONIC-method (CBC_CONIC-class), 49
stuffed_objective, ConeMatrixStuffing, Problem, CoeffExtractor-class

SumEntries-class, 314
SumEntries-class, 314
SumLargest, 316
SumLargest (SumLargest-class), 315
SumLargest-class, 315
SumSmallest, 317
SumSquares, 317
symbolic_quad_form, 322
| SymbolicQuadForm | (SymbolicQuadForm-class), 321 |
| SymbolicQuadForm-class | 321 |
| t(t.Expression), 323 |
| t,Expression-method(t.Expression), 323 |
| t.Expression, 323 |
| to_numeric, 324 |
| to_numeric,Abs-method(Abs-class), 37 |
| to_numeric,AddExpression-method (+,Expression,missing-method), 11 |
| to_numeric,BinaryOperator-method (BinaryOperator-class), 44 |
| to_numeric,Conjugate-method (Conjugate-class), 73 |
| to_numeric,Conv-method(Conv-class), 82 |
| to_numeric,CumMax-method (CumMax-class), 87 |
| to_numeric,CumSum-method (CumSum-class), 89 |
| to_numeric,DiagMat-method (DiagMat-class), 122 |
| to_numeric,DiagVec-method (DiagVec-class), 124 |
| to_numeric,DivExpression-method (/,Expression,Expression-method), 32 |
| to_numeric,Entr-method(Entr-class), 140 |
| to_numeric,Exp-method(Exp-class), 142 |
| to_numeric,EyeMinusInv-method (EyeMinusInv-class), 151 |
| to_numeric,GeoMean-method (GeoMean-class), 155 |
| to_numeric,HStack-method (HStack-class), 171 |
| to_numeric,Huber-method(Huber-class), 173 |
| to_numeric,Imag-method(Imag-class), 176 |
| to_numeric,Index-method ([,Expression,missing,missing,ANY-method), 341 |
| to_numeric,KLDiv-method(KLDiv-class), 183 |
| to_numeric,Kron-method(Kron-class), 185 |
| to_numeric,LambdaMax-method (LambdaMax-class), 187 |
| to_numeric,LambdaSumLargest-method (LambdaSumLargest-class), 189 |
| to_numeric,Log-method(Log-class), 199 |
| to_numeric,Log1p-method(Log1p-class), 201 |
| to_numeric,LogDet-method (LogDet-class), 202 |
| to_numeric,Logistic-method (Logistic-class), 204 |
| to_numeric,LogSumExp-method (LogSumExp-class), 205 |
| to_numeric,MatrixFrac-method (MatrixFrac-class), 210 |
| to_numeric,MaxElemwise-method (MaxElemwise-class), 215 |
| to_numeric,MaxEntries-method (MaxEntries-class), 216 |
| to_numeric,MinElemwise-method (MinElemwise-class), 222 |
| to_numeric,MinEntries-method (MinEntries-class), 223 |
| to_numeric,MulExpression-method (%*,Expression,Expression-method), 343 |
| to_numeric,Multiply-method (Multiply-class), 232 |
| to_numeric,Norm1-method(Norm1-class), 239 |
| to_numeric,NormInf-method (NormInf-class), 243 |
| to_numeric,NormNuc-method (NormNuc-class), 245 |
| to_numeric,OneMinusPos-method (OneMinusPos-class), 250 |
| to_numeric,PFEigenvalue-method (PFEigenvalue-class), 256 |
| to_numeric,Pnorm-method(Pnorm-class), 259 |
| to_numeric,Power-method(Power-class), 262 |
| to_numeric,ProdEntries-method (ProdEntries-class), 270 |
| to_numeric,Promote-method (Promote-class), 274 |
| to_numeric,QuadForm-method (QuadForm-class), 281 |
| to_numeric,QuadOverLin-method (QuadOverLin-class), 283 |
| to_numeric,Real-method(Real-class), 288 |
| to_numeric,Reshape-method |
Index

(Reshape-class), 293

to_numeric, SigmaMax-method
(SigmaMax-class), 301
to_numeric, SpecialIndex-method
([,Expression,missing,missing,ANY-method), 341
to_numeric, SumEntries-method
(SumEntries-class), 314
to_numeric, SumLargest-method
(SumLargest-class), 315
to_numeric, Trace-method (Trace-class), 324
to_numeric, Transpose-method
(Transpose-class), 325
to_numeric, UnaryOperator-method
(UnaryOperator-class), 328
to_numeric, UpperTri-method
(UpperTri-class), 330
to_numeric, VStack-method
(VStack-class), 337
to_numeric, Wrap-method (Wrap-class), 338
total_variation (tv), 327
TotalVariation, 323
tr (matrix_trace), 214
Trace, 325
Trace (Trace-class), 324
trace (matrix_trace), 214
Trace-class, 324
Transpose, 326
Transpose (Transpose-class), 325
Transpose-class, 325
tri_to_full, 326
tv, 327
UnaryOperator, 328
UnaryOperator (UnaryOperator-class), 328
UnaryOperator-class, 328
unpack_results, 328
unpack_results, Problem-method
(Problem-class), 266
upper_tri, 331
UpperTri, 330
UpperTri (UpperTri-class), 330
UpperTri-class, 330

validate_args, 332
validate_args, Atom-method (Atom-class), 41
validate_args, AxisAtom-method
(AxisAtom-class), 43
validate_args, Conv-method (Conv-class), 82
validate_args, Elementwise-method
(Elementwise-class), 133
validate_args, HStack-method
(HStack-class), 171
validate_args, Huber-method
(Huber-class), 173
validate_args, Kron-method (Kron-class), 185
validate_args, LambdaMax-method
(LambdaMax-class), 187
validate_args, LambdaSumLargest-method
(LambdaSumLargest-class), 189
validate_args, LogDet-method
(LogDet-class), 202
validate_args, MatrixFrac-method
(MatrixFrac-class), 210
validate_args, Pnorm-method
(Pnorm-class), 259
validate_args, QuadForm-method
(QuadForm-class), 281
validate_args, QuadOverLin-method
(QuadOverLin-class), 283
validate_args, Reshape-method
(Reshape-class), 293
validate_args, SumLargest-method
(SumLargest-class), 315
validate_args, Trace-method
(Trace-class), 324
validate_args, UpperTri-method
(UpperTri-class), 330
validate_args, VStack-method
(VStack-class), 337
validate_val, 332
validate_val, Leaf-method (Leaf-class), 193
value (value-methods), 333
value, Atom-method (Atom-class), 41
value, CallbackParam-method
(CallbackParam-class), 46
value, Constant-method (Constant-class), 74
value, Expression-method
(Expression-class), 145
value, Leaf-method (Leaf-class), 193
value, Objective-method (Objective-class), 249
value, Parameter-method (Parameter-class), 254
value, Problem-method (Problem-class), 266
value, Variable-method (Variable-class), 333
value-methods, 333
value<-, (value-methods), 333
value<-, Leaf-method (Leaf-class), 193
value<-, Parameter-method (Parameter-class), 254
value<-, Problem-method (Problem-class), 266
value_impl, Atom-method (Atom-class), 41
Variable, 47, 150, 151, 175, 195, 234, 269, 278, 329, 333, 334
Variable (Variable-class), 333
Variable-class, 333
variables (expression-parts), 149
variables, Canonical-method (Canonical-class), 46
variables, Leaf-method (Leaf-class), 193
variables, Problem-method (Problem-class), 266
variables, Variable-method (Variable-class), 333
vec, 335
vectorized_lower_tri_to_mat, 335
violation (residual-methods), 296
violation, Constraint-method (Constraint-class), 78
VStack, 337
VStack (VStack-class), 337
vstack, 336
VStack-class, 337
Wrap, 338
Wrap (Wrap-class), 338
Wrap-class, 338
ZeroConstraint, 339
ZeroConstraint-class, 339