

# Package ‘parma’

July 2, 2014

**Type** Package

**Title** Portfolio Allocation and Risk Management Applications

**Version** 1.5-1

**Date** 2014-02-23

**Author** Alexios Ghalanos and Bernhard Pfaff

**Maintainer** alexios ghalanos <alexios@4dscape.com>

**Description** Portfolio Optimization

**Collate** p-cmaes.R p-classes.R p-constraints.R p-timeseries.R p-fun.R  
p-Utility.R p-MILP.R p-NLP.R p-GNLP.R p-LP.R p-QP.R p-SOCP.R  
p-main.R p-methods.R zzz.R

**Depends** R (>= 2.10), methods, nloptr

**Imports** slam, Rglpk, quadprog, FRAPO, corpcor, parallel

**Suggests** truncnorm, xts, Rsymphony

**LazyLoad** yes

**LazyData** yes

**License** GPL-3

**Repository** CRAN

**Repository/R-Forge/Project** parma2

**Repository/R-Forge/Revision** 18

**Repository/R-Forge/DateTimeStamp** 2014-02-23 13:27:46

**Date/Publication** 2014-02-23 22:41:41

**NeedsCompilation** no

## R topics documented:

parma-package . . . . .	2
cmaes . . . . .	4
constraints . . . . .	6
etfdata . . . . .	7
parmafrontier-methods . . . . .	8
parmaPort-class . . . . .	9
parmasolve-methods . . . . .	10
parmaSpec-class . . . . .	11
parmaspec-methods . . . . .	12
parmautility-methods . . . . .	14
riskfun . . . . .	15
<b>Index</b>	<b>17</b>

---

parma-package	<i>The parma package</i>
---------------	--------------------------

---

### Description

Portfolio Allocation and Risk Management. Models and Methods for scenario and moment based optimization of portfolios.

### Details

Package:	parma
Type:	Package
Version:	1.5-1
Date:	2014-02-23
License:	GPL
LazyLoad:	yes
Depends:	methods
Imports:	nloptr, Rglpk, quadprog
Suggests:	Rsymphony, truncnorm, timeSeries

The portfolio allocation and risk management applications (parma) package contains a unique set of methods and models for the optimal allocation of capital in financial portfolios. It uniquely represents certain discontinuous problems using their smooth approximation counterparts and implements fractional based programming for the direct optimization of risk-to-reward ratios. In combination with the rmgarch package, it enables the confident solution to scenario based optimization problems using such risk and deviation measures as Mean Absolute Deviation (MAD), Variance (EV), Minimax, Conditional Value at Risk (CVaR), Conditional Drawdown at Risk (CDaR) and Lower Partial Moments (LPM). In addition, it implements moment based optimization for use with the quadratic EV problem, and a higher moment CARA utility expansion using the coskewness and cokurtosis matrices generated from the GO-GARCH with affine GH or NIG distributions. Bench-

mark relative optimization (tracking error) is also implemented as are basic mixed integer cardinality constraints. Finally, for non-convex problem formulations such as the upper to lower partial moments function, global optimization methods using a penalty based method are available. The key functions in the package are `parmaspec` which defines the optimization setup, and `parmasolve` which solves the problem given a chosen representation and solver. A portfolio frontier function is implemented in `parmafrontier`, utility optimization in `parmautility` and a custom translation of the cmaes global optimization solver of Hansen (2006) with full features is implemented in `cmaes`.

### How to cite this package

Whenever using this package, please cite as

```
@Manual{Ghalanos_2014,
  author      = {Alexios Ghalanos and Bernhard Pfaff},
  title       = {{parma}: Portfolio Allocation and Risk Management Applications.},
  year        = {2014},
  note        = {R package version 1.5-1.},}
```

### License

The releases of this package is licensed under GPL version 3.

### Author(s)

Alexios Ghalanos and Bernhard Pfaff

### References

- Charnes, A. and Cooper, W. 1962, Programming with linear fractional functionals, *Naval Research Logistics Quarterly*, **9**, 181–186.
- Dinkelbach, W. 1967, On nonlinear fractional programming, *Management Science*, **13(7)**, 492–498.
- Fishburn, P.C. 1977, Mean-risk analysis with risk associated with below-target returns, *The American Economic Review*, **67(2)**, 116–126.
- Ghalanos, A. 2012, Higher Moment Models for Risk and Portfolio Management, Thesis (submitted) *Cass Business School*.
- Hansen, N. 2006, The CMA Evolution Strategy: A Comparing Review, *Towards a New Evolutionary Computation (Studies in Fuzziness and Soft Computing)*, **192**, 75–102.
- Holthausen, D. 1981, A risk-return model with risk and return measured as deviations from a target return, *The American Economic Review*, **71**, 182–188.
- Konno, H. and Yamazaki, H. 1991, Mean-absolute deviation portfolio optimization model and its applications to Tokyo stock market, *Management Science*, **37(5)**, 519–531.
- Markowitz, H. 1952, Portfolio selection, *The Journal of Finance*, **7(1)**, 77–91.
- Rockafellar, R.T. and Uryasev, S. and Zabarankin, M., 2006, Generalized deviations in risk analysis, *Finance and Stochastics*, **10(1)**, 51–74.
- Stoyanov, S.V. and Rachev, S.T. and Fabozzi, F.J. 2007, Optimal financial portfolios, *Applied Mathematical Finance*, **14(5)**, 401–436.

**Description**

The direct translation of the Hansen's cmaes matlab code v3.60.

**Usage**

```
cmaes(pars, fun, lower = rep(0, length(pars)), upper = rep(1, length(pars)),
      insigma = 1, ctrl = cmaes.control(), ...)
```

```
cmaes.control(
  options = list(StopFitness = -Inf, MaxFunEvals = Inf,
    MaxIter = '1e3*(N+5)^2/sqrt(popsize)', StopFunEvals = Inf,
    StopIter = Inf, TolX = '1e-11*max(insigma)', TolUpX = '1e3*max(insigma)',
    TolFun = 1e-12, TolHistFun = 1e-13, StopOnStagnation = TRUE,
    StopOnWarnings = TRUE, StopOnEqualFunctionValues = '2 + N/3',
    DiffMaxChange = Inf, DiffMinChange = 0, WarnOnEqualFunctionValues = FALSE,
    EvalParallel = FALSE, EvalInitialX = TRUE, Restarts = 0,
    IncPopSize = 2, PopSize = '4 + floor(3*log(N))', ParentNumber = 'floor(popsize/2)',
    RecombinationWeights = c("superlinear", "linear", "constant"),
    DiagonalOnly = '0*(1+100*N/sqrt(popsize))+(N>=1000)',
    CMA = TRUE, Seed = 'as.integer(Sys.time())', DispFinal = TRUE,
    DispModulo = 100, Warnings = FALSE),
  CMA = list(cs = '(mueff+2)/(N+mueff+3)',
    damps = '1 + 2*max(0, sqrt((mueff-1)/(N+1))-1) + cs',
    ccum = '(4 + mueff/N) / (N+4 + 2*mueff/N)', ccov1 = '2 / ((N+1.3)^2+mueff)',
    ccovmu = '2 * (mueff-2+1/mueff) / ((N+2)^2+mueff)', active = 0))
```

**Arguments**

pars	A numeric vector of starting parameters.
fun	The user function to be minimized.
lower	A vector the lower parameter bounds.
upper	A vector with the upper parameter bounds.
insigma	The initial coordinate wise standard deviations for the search.
ctrl	A list with control parameters as returned from calling the 'cmaes.control' function.
...	Additional arguments passed to the user function.
options	The main options in the cmaes.control which may be optionally strings which are evaluated on initialization of the solver.
CMA	The options for the active CMA.

## Details

This solver has been translated from the matlab version created by Nikolaus Hansen and available on his website [http://www.lri.fr/~hansen/cmaes\\_inmatlab.html](http://www.lri.fr/~hansen/cmaes_inmatlab.html). There is also a **cmaes** on CRAN but this does not offer the same level of options and flexibility that the matlab version offers. For more details on what the options mean and generally how the cmaes solver works, consult the relevant website and literature.

## Author(s)

Alexios Ghalanos

## References

Hansen, N. 2006, The CMA Evolution Strategy: A Comparing Review, *Towards a New Evolutionary Computation (Studies in Fuzziness and Soft Computing)*, **192**, 75–102.

## Examples

```
## Not run:
ctrl = cmaes.control()
ctrl$options$StopOnWarnings = FALSE
ctrl$cma$active = 1
ctrl$options$TolFun = 1e-12
ctrl$options$DispModulo=100
ctrl$options$Restarts = 0
ctrl$options$MaxIter = 3000
ctrl$options$TolUpX = 5
ctrl$options$PopSize = 300
test1 = cmaes(rnorm(10), fun = parma:::fsphere,
lower = -Inf, upper = Inf, insigma = 1, ctrl = ctrl)
test2 = cmaes(rnorm(10), fun = parma:::frosenbrock,
lower = -Inf, upper = Inf, insigma = 1, ctrl = ctrl)

ctrl = cmaes.control()
ctrl$options$StopOnWarnings = FALSE
ctrl$cma$active = 1
ctrl$options$TolFun = 1e-12
ctrl$options$DispModulo=100
ctrl$options$Restarts = 0
ctrl$options$MaxIter = 3000
ctrl$options$PopSize = 400
test3 = cmaes(rep(1, 10), fun = parma:::frastrigin10,
lower = -50, upper = 50, insigma = 1, ctrl = ctrl)

## End(Not run)
```

---

 constraints

*NLP custom constraint functions*


---

### Description

Provides a number of custom constraints and their jacobians for use with the NLP representation (both minimum risk and the fractional problem).

### Usage

```

ineqfun.turnover.min(w, optvars, usersvars)
ineqjac.turnover.min(w, optvars, usersvars)
ineqfun.bsturnover.min(w, optvars, usersvars)
ineqjac.bsturnover.min(w, optvars, usersvars)
ineqfun.turnover.opt(w, optvars, usersvars)
ineqjac.turnover.opt(w, optvars, usersvars)
ineqfun.bsturnover.opt(w, optvars, usersvars)
ineqjac.bsturnover.opt(w, optvars, usersvars)
ineqfun.variance.opt(w, optvars, usersvars)
ineqjac.variance.opt(w, optvars, usersvars)
ineqfun.variance.min(w, optvars, usersvars)
ineqjac.variance.min(w, optvars, usersvars)

```

### Arguments

w	The decision weight vector.
optvars	Problem specific list (not for use by user).
usersvars	User specific list with some required parameters to be set (see details).

### Details

Functions preceded by “ineqfun” denote the functions which may be passed to the ineqfun option in the parmaspec function, while “eqfun” to the eqfun option. Functions which include “jac” in the first part of the name denote the jacobians of the equivalent constraints and should be passed to the ineqgrad or eqgrad options in parmaspec. Functions ending with “min” denote formulations for use with the minrisk type problems while functions ending with “opt” denote the fractional risk formulation.

For the simple turnover constraint, the usersvars list must contain an entry called ‘wold’ denoting the previous vector of weights with which the comparison will be made. Additionally, an entry called ‘turnover’ is required which denotes the (positive) value for the maximum turnover.

For the buy and sell turnover constraint (bsturnover), there should instead be (positive) ‘buyturnover’ and ‘sellturnover’ entries in the usersvars list. Finally note that when using this type of constraints in a fractional programming setup, care should be taken that the combination of bounds, turnover limits and the forecast return vector do not result in a negative expected return in which case the problem is not solvable.

The variance constraint allows the targeting of a maximum acceptable variance. The extra arguments which must be passed to the usersvars list are 'Cov' for the asset covariance matrix and 'varbound' representing the maximum acceptable upper variance.

**Value**

Used internally by the NLP solver. The fun return a scalar, while the jac return a matrix with n.cols equal to the length of the decision vector (which may be greater than the length of the weights as in the fractional problem which contains the fractional multiplier as well as other formulations which have additional decision variables).

**Author(s)**

Alexios Ghalanos

---

etfdata

*15 Exchange Traded Funds (ETFs)*

---

**Description**

The xts dataset consists of the adjusted daily closing prices of 15 iShare ETFs for the period 2003-05-28 to 2012-06-01 (2272 periods) representing a selection of US style and international equity benchmark. The iShare series are IWF (Russell 1000 Growth Index), IWD (Russell 1000 Value Index), IWO (Russell 2000 Growth Index), IWN (Russell 2000 Value Index), EEM (MSCI Emerging Markets Index), TLT (Barclays 20+ Year T-Bond), EWC (MSCI Canada Index), EWA (MSCI Australia Index), EWJ (MSCI Japan Index), EWG (MSCI Germany Index), EWL (MSCI Switzerland Index), EWQ (MSCI France Index), EWU (MSCI UK Index), EPP (MSCI Pacific ex-Japan), EZA (MSCI South Africa Index).

**Usage**

```
data(etfdata)
```

**Format**

An xts matrix containing 2272x15 observations.

**Source**

Yahoo Finance

---

 parmafrontier-methods *Efficient Frontier Generator*


---

## Description

Solves for the portfolios on the efficient frontier given a specification object.

## Usage

```
parmafrontier(spec, n.points = 100, miny = NULL, maxy = NULL, type = NULL,
  solver = NULL, solver.control = list(), parma.control = list(ubounds = 10000,
  mbounds = 1e+05, penalty = 10000), cluster = NULL)
```

## Arguments

<code>spec</code>	A <a href="#">parmaSpec</a> object.
<code>n.points</code>	The number of portfolios to solve for along the frontier.
<code>miny</code>	(Optional) Minimum return from which to calculate the frontier. If not provided, will be calculated by a first pass optimization.
<code>maxy</code>	(Optional) Maximum return for which to calculate the frontier. If not provided, will be based on the maximum forecast.
<code>type</code>	The problem type to use (the <code>show</code> method on <a href="#">parmaSpec</a> will indicate the available options).
<code>solver</code>	For a GNLP type problem the choice of global solver, either “cmaes” or “crs” from the <code>nloptr</code> package.
<code>solver.control</code>	A list with optional control parameters passed to the <code>nloptr</code> or <a href="#">cmaes</a> solver.
<code>parma.control</code>	Internal NLP tuning parameters, where ‘ubounds’ represents the symmetric unconstrained parameter bounds in the fractional problem, ‘mbounds’ the multiplier upper bounds in the fractional formulation, ‘penalty’ the GNLP penalty parameter.
<code>cluster</code>	A precreated cluster object from the parallel package for the parallel evaluation of the frontier portfolios (see note).

## Details

If using a cluster object, care should be taken since some of the LP based problems are quite memory intensive. Though some care some gone into using slam matrices and forcing garbage collection in intermediate setup steps of these problems, there is no guarantee that memory will not be quickly depleted for large problems using many cores/sockets.

## Value

A [matrix](#) object with columns for the weights, risk, reward and termination status of the solver (not available for the QP solver), and `n.points` rows. The user should investigate the non-converged solutions as well as remove any NA based rows (non-converged with error).



**Author(s)**

Alexios Ghalanos

---

parmaPort-class	Class "parmaPort"
-----------------	-------------------

---

**Description**

The parma optimal portfolio class.

**Objects from the Class**

Objects can be created by calls to [parmasolve..](#)

**Slots**

**solution:** Object of class "vector" The list with the optimal values.

**model:** Object of class "vector" A list with details of the risk model.

**Methods**

**show** signature(object = "parmaPort"): Summary.

**weights** signature(object = "parmaPort"): Extracts the optimal weights.

**tictoc** signature(object = "parmaPort"): Extracts the time elapsed to solve the problem.

**checkarbitrage** signature(object = "parmaPort"): Extracts the arbitrage check on the scenario.

**parmarisk** signature(object = "parmaPort"): Extracts the expected risk of the optimized portfolio.

**parmareward** signature(object = "parmaPort"): Extracts the expected reward of the optimized portfolio.

**parmastatus** signature(object = "parmaPort"): Solver termination code for the LP and NLP solvers.

**Author(s)**

Alexios Ghalanos

**Examples**

```
showClass("parmaPort")
```

**Description**

Solves for the optimal weights given parmaSpec defined model.

**Usage**

```
parmasolve(spec, type = NULL, solver = NULL, solver.control = list(), x0 = NULL,
w0 = NULL, parma.control = list(ubounds = 1e4, mbounds = 1e5,
penalty = 1e4, eqSlack = 1e-05), ...)
```

**Arguments**

spec	A <a href="#">parmaSpec</a> object.
type	The problem type to use (the show method on <a href="#">parmaSpec</a> will indicate the available options).
solver	For a GNLP type problem the choice of global solver, either “cmaes” or “crs” from the <a href="#">nloptr</a> package. For LP type problems, either <a href="#">glpk</a> or <a href="#">symphony</a> .
solver.control	A list with optional control parameters passed to the <a href="#">nloptr</a> or <a href="#">cmaes</a> solver.
x0	Optional starting parameters for the NLP type problems. This may be of size greater than the number of assets depending on whether it takes extra optimization parameters (CVaR has VaR in position 1, whilst the fractional formulation has the multiplier in the last position).
w0	Optional starting parameters for only the asset weights.
parma.control	Internal NLP tuning parameters, where ‘ubounds’ represents the symmetric unconstrained parameter bounds in the fractional problem, ‘mbounds’ the multiplier upper bounds in the fractional formulation, ‘penalty’ the GNLP penalty parameter and ‘eqSlack’ the slack value to use for converting equalities to inequalities in the SOCP formulation.
...	Currently only the ‘verbose’ argument (logical) which is used and passed to the <a href="#">GLPK</a> solver (similar to ‘trace’).

**Details**

In most of the cases, the intersection of objective and constraints will define whether a problem is LP, MILP, QP, MIQP, QCQP, NLP, MINLP or GNLP. However, there are cases when the problem can be solved by more than one type of solver so that the ‘type’ option allows the user to choose between the options. For the GNLP type, the ‘solver’ allows for a choice of “cmaes” or “crs”, while the LP type there is a choice of “glpk” or “symphony”. Future development will likely expand on these choices.

**Value**

A [parmaPort](#) object containing details of the PARMA optimized portfolio.

**Author(s)**

Alexios Ghalanos

---

parmaSpec-class      *Class "parmaSpec"*

---

**Description**

Object returned from calling [parmaspec](#).

**Objects from the Class**

Objects can be created by calls of the form `new("parmaSpec")`.

**Slots**

**model:** A list with details of the risk and optimization model.

**modeldata:** A list with the data.

**constraints:** A list with details on the optimization constraints.

**Methods**

**parmasolve** signature(spec = "parmaSpec"): Solves for the optimal weights.

**show** signature(object = "parmaSpec"): Summary method.

**parmaset<-** signature(object = "parmaSpec"): Set a specified 'arg' from the list of arguments defined in the [parmaspec](#) to some 'value' and return a modified specification.

**parmaget** signature(object = "parmaSpec"): Get a specified 'arg' from the list of arguments defined in the [parmaspec](#).

**Author(s)**

Alexios Ghalanos

**Examples**

```
showClass("parmaSpec")
```

## Description

Defines the type model and method for optimization using either a scenario or covariance matrix.

## Usage

```
parmaspec(scenario = NULL, probability = NULL, S = NULL, Q = NULL, qB = NULL,
benchmark = NULL, benchmarkS = NULL, forecast = NULL, target = NULL,
targetType = c("inequality", "equality"),
risk = c("MAD", "MiniMax", "CVaR", "CDaR", "EV", "LPM", "LPMUPM"),
riskType = c("minrisk", "optimal", "maxreward"), riskB = NULL,
options = list(alpha = 0.05, threshold = 999, moment = 1, lmoment=1,
umoment=1, lthreshold = -0.01, uthreshold = 0.01),
LB = NULL, UB = NULL, budget = 1, leverage = NULL,
ineqfun = NULL, ineqgrad = NULL, eqfun = NULL, eqgrad = NULL,
uservars = list(), ineq.mat = NULL, ineq.LB = NULL,
ineq.UB = NULL, eq.mat = NULL, eqB = NULL, max.pos = NULL,
asset.names = NULL, ...)
```

## Arguments

scenario	An n-by-m scenario matrix.
probability	An optional n-by-1 vector of scenario probabilities which must sum to 1 (only currently used in LP problems). Default is to assign equal weights to each row of the scenario.
S	An m-by-m positive definite covariance matrix.
Q	A list of m-by-m positive definite matrices for QCQP type problems.
qB	a vector of the same length as Q denoting the upper bound on Q.
benchmark	A n-by-1 scenario benchmark matrix, used when scenario is not NULL.
benchmarkS	An m+1 vector consisting of the benchmark variance (1) and covariances (m) with the other m assets, used when S is not NULL.
forecast	A vector (m) of forecast values for the assets. If a benchmark is included, this should then be the active forecast over the benchmark.
target	The target return required when riskType is minrisk. If a benchmark is used, then this is the active target over the benchmark (given the active forecasts above).
targetType	Whether the target should be a hard equality or inequality.
risk	The risk measure.
riskType	The type of optimization to use, with a choice of minimizing the risk given the relevant constraints, on optimizing directly the risk to reward ratio using established fractional programming methods or maximizing the reward subject to a risk upper bound and other constraints (only currently supported by the SOCP solver for covariance matrix type problems).

riskB	For the case that riskType is “maxreward”, then riskB is the upper bound for the risk constraint.
options	A vector of optional parameters related to the tail risk measures CVaR, CDaR and LPM.
LB	The lower bounds for the asset weights. If using a benchmark, this should be the maximum deviation below the benchmark weights.
UB	The upper bounds for the asset weights. If using a benchmark, this should be the maximum deviation above the benchmark weights.
budget	The investment constraint. If using a benchmark, this is usually set to zero so that weights represent the active bets on the benchmark.
leverage	The leverage constraint for Long/Short optimization.
ineqfun	A list of user inequality functions for use in an NLP type setup (see details).
ineqgrad	A list of user inequality gradients if ineqfun was provided.
eqfun	A list of user equality functions for use in an NLP type setup (see details).
eqgrad	A list of user equality gradients if eqfun was provided.
uservars	A list of any additional user required values to be used with user defined inequality and equality functions (for NLP).
ineq.mat	A k-by-m inequality matrix (for LP and QP problems).
ineq.LB	A vector (k) of the lower bounds for the ineq.mat.
ineq.UB	A vector (k) of the upper bounds for the ineq.mat.
eq.mat	A l-by-m equality matrix (for LP and QP problems).
eqB	A vector (l) of equalities for the eq.mat.
max.pos	Cardinality Constraints. The maximum assets to include in the solution, effectively making this a MILP, MIQP or MINLP problem.
asset.names	An optional character vector of asset names.
...	Not used.

## Details

The parmaspec method is the entry point for specifying and solving portfolio problems in the parma package. Currently 7 measures of risk are supported, 3 based on tail measures: Conditional Value at Risk (CVaR), Conditional Drawdown at Risk (CDaR) and Lower Partial Moments (LPM), and 3 based on the Lp-Norm: Mean Absolute Deviation (L\_1, MAD), Mean Variance (L\_2, EV) and MiniMax (L\_inf, Minimax). The LPMUPM measure is the ratio of lower to upper partial moments, a non convex measure discussed in Holthausen (1981). Additionally, the problems may be solved based on minimization of risk subject to a target return, else on the optimal risk-reward ratio using fractional programming (see references), thus avoiding the estimation of the entire frontier. Problems are classified and solved according to whether they can be formulated as Linear (LP), Mixed Integer LP (MILP), Quadratic (QP), Mixed Integer Quadratic (MIQP), Second Order Cone Programming (SOCP), Non-Linear (NLP), Mixed Integer NLP (MINLP) and Global NLP (GNLP). This in turn depends on the intersection of objectives and constraints. It is possible that a problem may be solved both as an LP and NLP (or QP and NLP), and this can be defined during the solver stage ([parmasolve](#)). Because all NLP models, make use of analytical derivatives, the results should

be the same for any formulation chosen, and considerations such as memory usage should guide the choice of formulation (with some LP models being particularly expensive). Not all problem types are supported, but this might change subject to the availability of solvers in R which can deal with these specific types e.g. MINLP and MIQP. The `parmaspec` also allows the input of a benchmark so that benchmark relative optimization is carried out. User defined equality and inequality functions for NLP problems need to be properly defined to be accepted by the model, and their analytic gradients also provided, unless the problem is solved as a GNLP in which case a derivative free penalty function is used. These custom constraint functions should be provided in a list, and should take as arguments the vector of decision variables ‘w’, an argument called ‘optvars’ which is used by the program internally, and an argument called ‘uservars’ which is a list with optional user defined values for the constraints. The examples in the `inst` folder provide some guidance, and the user is left to his own devices to study the underlying workings of the program to understand how to supply these. Finally, the NLP functions which are known to be discontinuous because of the presence of functions such as the min and max, have been re-written to take advantage of smooth approximations to such functions, details of which may be found in the vignette. The package support for GNLP is based on a choice of the `cmaes` solver of the `cmaes` package (which is not production level) or the `crs` solver of the `nloptr` package which may be defined in `solver.control` option of the `parmasolve` method with named argument ‘solver’. High quality GNLP solvers are not available in R and as such support for these types of problems is experimental at best and your mileage will vary. The problems which must be solved as GNLP include the ‘LPMUPM’ measure, all problems with risk type ‘optimal’ AND cardinality constraints (‘max.pos’), and all problems with custom NLP constraints without derivatives, non-convex inequalities or non-affine equalities.

### Value

A `parmaSpec` object containing details of the PARMA specification.

### Author(s)

Alexios Ghalanos

---

parmautility-methods *Utility Based Optimization*

---

### Description

Utility based portfolio optimization using either Taylor series expansion of utility function with moments or scenario based.

### Usage

```
parmautility(U = c("CARA", "Power"), method = c("moment", "scenario"),
  scenario = NULL, M1 = NULL, M2 = NULL, M3 = NULL, M4 = NULL, RA = 1,
  budget = 1, LB = rep(0, length(M1)), UB = rep(1, length(M1)))
```

**Arguments**

U	The utility function (only CARA currently implemented).
method	Whether to use moment or scenario based optimization (only moment currently implemented).
scenario	A n-by-m scenario matrix.
M1	A vector (m) of forecasts.
M2	An m-by-m positive definite covariance matrix.
M3	An m-by-m <sup>2</sup> third co-moment matrix.
M4	An m-by-m <sup>3</sup> fourth co-moment matrix.
RA	Risk Aversion Coefficient for CARA.
budget	The investment constraint.
LB	The lower bounds for the asset weights (positive).
UB	The upper bounds for the asset weights.

**Details**

The function currently only implements the CARA moment based approach, but will be expanded in the future. The moment approach can take as inputs either M1 and M2 (2-moment approximation), or M1, M2, M3 and M4 (4-moment approximation). Not many models generate M3 and M4, but the “gogarch” model with manig or magh distribution will.

**Value**

A [parmaPort](#) object containing details of the PARMA optimized portfolio.

**Author(s)**

Alexios Ghalanos

**References**

Ghalanos, A. and Rossi, E. and Urga, G. 2012, Independent Factor Autoregressive Conditional Density Model *submitted-TBA*

**Description**

Calculates a given portfolio risk/deviation measure given a set of weights and matrix of returns, possible representing a forecast scenario.

**Usage**

```
riskfun(weights, Data, risk = c("mad", "ev", "minimax", "cvar", "cdar", "lpm"),  
benchmark = NULL, alpha = 0.05, moment = 1, threshold = 0, VaR = NULL, DaR = NULL)
```

**Arguments**

weights	vector of weights.
Data	Matrix of returns.
risk	Choice of measure.
benchmark	(Optional) vector of benchmark returns with same number of rows as Data.
alpha	The lower quantile for the “cvar” and “cdar” measures.
moment	The “lpm” measure moment.
threshold	The “lpm” measure threshold. A value of 999 will subtract the portfolio mean.
VaR	(Optional) The pre-calculated VaR for the “cvar” measure.
DaR	(Optional) The pre-calculated DaR for the “cdar” measure.

**Details**

A simple utility function for the calculation and understanding of some of the risk and deviation measures implemented in the package.

**Value**

A numeric value representing the risk/deviation measure.

**Author(s)**

Alexios Ghalanos



# Index

- \*Topic **classes**
  - parmaPort-class, 9
  - parmaSpec-class, 11
- \*Topic **datasets**
  - etfdata, 7
- \*Topic **methods**
  - parmafrontier-methods, 8
  - parmasolve-methods, 10
  - parmaspec-methods, 12
  - parmautility-methods, 14
- \*Topic **optimize**
  - constraints, 6
  - riskfun, 15
- checkarbitrage (parmaPort-class), 9
- checkarbitrage, ANY-method (parmaPort-class), 9
- checkarbitrage, parmaPort-method (parmaPort-class), 9
- cmaes, 3, 4, 8, 10
- constraints, 6
- etfdata, 7
- ineqfun.bsturnover.min (constraints), 6
- ineqfun.bsturnover.opt (constraints), 6
- ineqfun.turnover.min (constraints), 6
- ineqfun.turnover.opt (constraints), 6
- ineqfun.variance.min (constraints), 6
- ineqfun.variance.opt (constraints), 6
- ineqjac.bsturnover.min (constraints), 6
- ineqjac.bsturnover.opt (constraints), 6
- ineqjac.turnover.min (constraints), 6
- ineqjac.turnover.opt (constraints), 6
- ineqjac.variance.min (constraints), 6
- ineqjac.variance.opt (constraints), 6
- matrix, 8
- parma (parma-package), 2
- parma-package, 2
- parmafrontier, 3
- parmafrontier (parmafrontier-methods), 8
- parmafrontier, ANY-method (parmafrontier-methods), 8
- parmafrontier, parmaSpec-method (parmafrontier-methods), 8
- parmafrontier-methods, 8
- parmaget (parmaSpec-class), 11
- parmaget, ANY-method (parmaSpec-class), 11
- parmaget, parmaSpec-method (parmaSpec-class), 11
- parmaPort, 10, 15
- parmaPort-class, 9
- parmareward (parmaPort-class), 9
- parmareward, ANY-method (parmaPort-class), 9
- parmareward, parmaPort-method (parmaPort-class), 9
- parmarisk (parmaPort-class), 9
- parmarisk, ANY-method (parmaPort-class), 9
- parmarisk, parmaPort-method (parmaPort-class), 9
- parmaset<- (parmaSpec-class), 11
- parmaset<-, ANY, ANY-method (parmaSpec-class), 11
- parmaset<-, parmaSpec, vector-method (parmaSpec-class), 11
- parmasolve, 3, 9, 13, 14
- parmasolve (parmasolve-methods), 10
- parmasolve, ANY-method (parmasolve-methods), 10
- parmasolve, parmaSpec-method (parmasolve-methods), 10
- parmasolve-methods, 10
- parmaSpec, 8, 10, 14
- parmaspec, 3, 11
- parmaspec (parmaspec-methods), 12

parmaspec, ANY-method  
    (parmaspec-methods), 12

parmaSpec-class, 11

parmaspec-methods, 12

parmastatus (parmaPort-class), 9

parmastatus, ANY-method  
    (parmaPort-class), 9

parmastatus, parmaPort-method  
    (parmaPort-class), 9

parmautility, 3

parmautility (parmautility-methods), 14

parmautility, ANY-method  
    (parmautility-methods), 14

parmautility-methods, 14

riskfun, 15

show, parmaPort-method  
    (parmaPort-class), 9

show, parmaSpec-method  
    (parmaSpec-class), 11

tictoc (parmaPort-class), 9

tictoc, ANY-method (parmaPort-class), 9

tictoc, parmaPort-method  
    (parmaPort-class), 9

weights, parmaPort-method  
    (parmaPort-class), 9