Package 'PortfolioAnalytics'

May 18, 2018

Type Package

Title Portfolio Analysis, Including Numerical Methods for Optimization of Portfolios

Version 1.1.0

Date 2018-05-17

Maintainer Brian G. Peterson <brian@braverock.com>

Description Portfolio optimization and analysis routines and graphics.

Depends R (>= 3.3.0), zoo, xts (>= 0.10-1), foreach, PerformanceAnalytics (>= 1.5.1)

Suggests quantmod, DEoptim(>= 2.2.1), iterators, fGarch, Rglpk, quadprog, ROI (>= 0.1.0), ROI.plugin.glpk (>= 0.0.2), ROI.plugin.quadprog (>= 0.0.2), ROI.plugin.symphony (>= 0.0.2), pso, GenSA, corpcor, testthat, nloptr (>= 1.0.0), MASS, robustbase

Imports methods

License GPL-2 | GPL-3

URL https://github.com/braverock/PortfolioAnalytics

RoxygenNote 6.0.1.9000

NeedsCompilation yes

Author Brian G. Peterson [cre, aut, cph],
Peter Carl [aut, cph],
Kris Boudt [ctb, cph],
Ross Bennett [ctb, cph],
Hezky Varon [ctb],
Guy Yollin [ctb],
R. Douglas Martin [ctb]

Repository CRAN

Date/Publication 2018-05-17 22:48:29 UTC

R topics documented:

`PortfolioAnalytics-package`

Numeric methods for optimization of portfolios

Description

PortfolioAnalytics is an R package to provide numerical solutions for portfolio problems with complex constraints and objective sets. The goal of the package is to aid practicioners and researchers in solving portfolio optimization problems with complex constraints and objectives that mirror real-world applications.

One of the goals of the packages is to provide a common interface to specify constraints and objectives that can be solved by any supported solver (i.e. optimization method). Currently supported optimization methods include

- random portfolios
- differential evolution
- particle swarm optimization
- generalized simulated annealing
- linear and quadratic programming routines

The solver can be specified with the optimize_method argument in optimize.portfolio and optimize.portfolio.rebalancing. The optimize_method argument must be one of "random", "DEoptim", "pso", "GenSA", "ROI", "quadprog", "glpk", or "symphony".

Additional information on random portfolios is provided below. The differential evolution algorithm is implemented via the DEoptim package, the particle swarm optimization algorithm via the pso package, the generalized simulated annealing via the GenSA package, and linear and quadratic programming are implemented via the ROI package which acts as an interface to the Rglpk, Rsymphony, and quadprog packages.

A key strength of PortfolioAnalytics is the generalization of constraints and objectives that can be solved.

If optimize_method="ROI" is specified, a default solver will be selected based on the optimization problem. The glpk solver is the default solver for LP and MILP optimization problems. The quadprog solver is the default solver for QP optimization problems. For example, optimize_method = "quadprog" can be specified and the optimization problem will be solved via ROI using the quadprog plugin package.

The extension to ROI solves a limited type of convex optimization problems:

- Maxmimize portfolio return subject leverage, box, group, position limit, target mean return, and/or factor exposure constraints on weights.
- Minimize portfolio variance subject to leverage, box, group, turnover, and/or factor exposure constraints (otherwise known as global minimum variance portfolio).
- Minimize portfolio variance subject to leverage, box, group, and/or factor exposure constraints and a desired portfolio return.
- Maximize quadratic utility subject to leverage, box, group, target mean return, turnover, and/or factor exposure constraints and risk aversion parameter. (The risk aversion parameter is passed into optimize.portfolio as an added argument to the portfolio object).
- Maximize portfolio mean return per unit standard deviation (i.e. the Sharpe Ratio) can be done by specifying maxSR=TRUE in optimize.portfolio. If both mean and StdDev are specified as objective names, the default action is to maximize quadratic utility, therefore maxSR=TRUE must be specified to maximize Sharpe Ratio.

- Minimize portfolio ES/ETL/CVaR optimization subject to leverage, box, group, position limit, target mean return, and/or factor exposure constraints and target portfolio return.
- Maximize portfolio mean return per unit ES/ETL/CVaR (i.e. the STARR Ratio) can be done by specifying `maxSTARR=TRUE` in `optimize.portfolio`. If both mean and ES/ETL/CVaR are specified as objective names, the default action is to maximize mean return per unit ES/ETL/CVaR.

These problems also support a weight_concentration objective where concentration of weights as measured by HHI is added as a penalty term to the quadratic objective.

Because these convex optimization problem are standardized, there is no need for a penalty term. The `multiplier` argument in `add.objective` passed into the complete constraint object are ingnored by the ROI solver.

Many real-world portfolio optimization problems are global optimization problems, and therefore are not suitable for linear or quadratic programming routines. `PortfolioAnalytics` provides a random portfolio optimization method and also utilizes the R packages DEoptim, pso, and GenSA for solving non-convex global optimization problems.

`PortfolioAnalytics` supports three methods of generating random portfolios.

- The sample method to generate random portfolios is based on an idea by Pat Burns. This is the most flexible method, but also the slowest, and can generate portfolios to satisfy leverage, box, group, position limit, and leverage constraints.
- The simplex method to generate random portfolios is based on a paper by W. T. Shaw. The simplex method is useful to generate random portfolios with the full investment constraint (where the sum of the weights is equal to 1) and min box constraints. Values for min_sum and max_sum of the leverage constraint will be ignored, the sum of weights will equal 1. All other constraints such as the box constraint max, group and position limit constraints will be handled by elimination. If the constraints are very restrictive, this may result in very few feasible portfolios remaining. Another key point to note is that the solution may not be along the vertexes depending on the objective. For example, a risk budget objective will likely place the portfolio somewhere on the interior.
- The grid method to generate random portfolios is based on the `gridSearch` function in package `NMOF`. The grid search method only satisfies the min and max box constraints. The min_sum and max_sum leverage constraint will likely be violated and the weights in the random portfolios should be normalized. Normalization may cause the box constraints to be violated and will be penalized in `constrained_objective`.

`PortfolioAnalytics` leverages the `PerformanceAnalytics` package for many common objective functions. The objective types in `PortfolioAnalytics` are designed to be used with `PerformanceAnalytics` functions, but any user supplied valid R function can be used as an objective.

Optimization

This summary attempts to provide an overview of how to construct a portfolio object with constraints and objectives, run the optimization, and chart the results.

The portfolio object is initialized with the `portfolio.spec` function. The main argument to `portfolio.spec` is `assets`. The `assets` argument can be a scalar value for the number of assets, a character vector of fund names, or a named vector of initial weights.

Adding constraints to the portfolio object is done with add.constraint. The add.constraint function is the main interface for adding and/or updating constraints to the portfolio object. This function allows the user to specify the portfolio to add the constraints to, the type of constraints, arguments for the constraint, and whether or not to enable the constraint. If updating an existing constraint, the indexnum argument can be specified.

Objectives can be added to the portfolio object with add.objective. The add.objective function is the main function for adding and/or updating objectives to the portfolio object. This function allows the user to specify the portfolio to add the objectives to, the type, name of the objective function, arguments to the objective function, and whether or not to enable the objective. If updating an existing objective, the indexnum argument can be specified.

With the constraints and objectives specified in the portfolio object, the portfolio object can be passed to optimize.portfolio or optimize.portfolio.rebalancing to run the optimization. Arguments to optimize.portfolio include asset returns, the portfolio obect specifying constraints and objectives, optimization method, and other parameters specific to the solver. optimize.portfolio.rebalancing adds support for backtesting portfolio optimization through time with rebalancing or rolling periods.

Advanced Optimization

In addition to the more standard optimizations described above, PortfolioAnalytics also supports multi-layer optimization and regime switching optimization.

Support for multi-layer optimization allows one to construct a top level portfolio and several sub-portfolios with potentially different assets, constraints, and objectives. First, each sub-portfolio is optimized out-of-sample which creates a time series of returns. One can think of the out of sample returns for each sub-portfolio as the returns for a synthetic instrument. Finally, the out-of-sample returns of each sub-portfolio are then used as inputs for the top level optimization. The top level portfolio and sub-portfolios are created as normal using portfolio.spec, add.constraint, and add.objective. The multi-layer portfolio specification object is first initialized by passing the top level portfolio to mult.portfolio.spec. Sub-portfolios are then added with add.sub.portfolio. The multi-layer portfolio specification object can then be passed to optimize.portfolio and optimize.portfolio.rebalancing. See demo(multi_layer_optimization).

Support for regime switching models allows one to change constraints and objectives depending on the current regime. Portfolios are created as normal with portfolio.spec, add.constraint, and add.objective. The portfolios are then combined with a regime object using regime.portfolios to create a regime portfolio specification which can then be passed to optimize.portfolio and optimize.portfolio.rebalancing. Regime switching optimization is implemented in such a way that any arbitrary regime model can be used. See demo(regime_switching).

Portfolio Moments

The PortfolioAnalytics framework to estimate solutions to constrained optimization problems is implemented in such a way that the moments of the returns are set once for use in lower level optimization functions. The set.portfolio.moments function computes the first, second, third, and fourth moments depending on the objective function(s) in the portfolio object. For example, if the third and fourth moments do not need to be calculated for a given objective, then set.portfolio.moments will try to detect this and not compute those moments. Currently, set.portfolio.moments implements methods to compute moments based on sample estimates, higher moments from fitting a statistical factor model based on the work of Kris Boudt, the Black Litterman model, and the Fully

Flexible Framework based on the work of Attilio Meucci (NEED REFERENCE HERE). See the Custom Moment and Objective Functions vignette for a more detailed description and examples.

Charts and Graphs

Intuition into the optimization can be aided through visualization. The goal of creating the charts is to provide visualization tools for optimal portfolios regardless of the chosen optimization method.

`chart.Weights` plots the weights of the optimal portfolio. `chart.RiskReward` plots the optimal portfolio in risk-reward space. The random portfolios, DEoptim, and pso solvers will return trace portfolio information at each iteration when `optimize.portfolio` is run with `trace=TRUE`. If this is the case, `chart.RiskReward` will plot these portfolios so that the feasible space can be easily visualized. Although the GenSA and ROI solvers do not return trace portfolio information, random portfolios can be be generated with the argument `rp=TRUE` in `chart.RiskReward`. A `plot` function is provided that will plot the weights and risk-reward scatter chart. The component risk contribution can be charted for portfolio optimization problems with risk budget objectives with `chart.RiskBudget`. Neighbor portfolios can be plotted in `chart.RiskBudget`, `chart.Weights`, and `chart.RiskReward`.

Efficient frontiers can be extracted from `optimize.portfolio` objects or created from a `portfolio` object. The efficient frontier can be charted in risk-reward space with `chart.EfficientFrontier`. The weights along the efficient frontier can be charted with `chart.EF.Weights`.

Multiple objects created via `optimize.portfolio` can be combined with `combine.optimizations` for visual comparison. The weights of the optimal portfolios can be plotted with `chart.Weights`. The optimal portfolios can be compared in risk-reward space with `chart.RiskReward`. The portfolio component risk contributions of the multiple optimal portfolios can be plotted with `chart.RiskBudget`.

Demos

`PortfolioAnalytics` contains a comprehensive collection of demos to demonstrate the functionality from very basic optimization problems such as estimating the solution to a minimum variance portfolio to more complex optimization problems with custom moment and objective functions.

Vignettes

TODO

Package Dependencies

Several of the functions in the `PortfolioAnalytics` package require time series data of returns and the `xts` package is used for working with time series data.

The `PerformanceAnalytics` package is used for many common objective functions. The objective types in `PortfolioAnalytics` are designed to be used with `PerformanceAnalytics` functions such as `StdDev`, `VaR`, and `ES`.

The `foreach` and `iterators` packages are used extensively throughout the package to support parallel programming. The primary functions where `foreach` loops are used is `optimize.portfolio`, `optimize.portfolio.rebalancing`, and `create.EfficientFrontier`.

In addition to a random portfolios optimzation method, `PortfolioAnalytics` supports backend solvers by leveraging the following packages: `DEoptim`, `pso`, `GenSA`, `ROI` and associated ROI plugin packages.

Further Work

Continued work to improved charts and graphs.

Continued work to improve features to combine and compare multiple optimal portfolio objects.

Support for more solvers.

Comments, suggestions, and/or code patches are welcome.

Acknowledgements

TODO

Author(s)

Ross Bennett
Kris Boudt
Peter Carl
Brian G. Peterson

Maintainer: Brian G. Peterson <brian@braverock.com>

References

Boudt, Kris and Lu, Wanbo and Peeters, Benedict, *Higher Order Comoments of Multifactor Models and Asset Allocation* (June 16, 2014). Available at SSRN: http://ssrn.com/abstract=2409603 or http://dx.doi.org/10.2139/ssrn.2409603

Chriss, Neil A and Almgren, Robert, *Portfolios from Sorts* (April 27, 2005). Available at SSRN: http://ssrn.com/abstract=720041 or http://dx.doi.org/10.2139/ssrn.720041

Meucci, Attilio, *The Black-Litterman Approach: Original Model and Extensions* (August 1, 2008). Shorter version in, THE ENCYCLOPEDIA OF QUANTITATIVE FINANCE, Wiley, 2010. Available at SSRN: http://ssrn.com/abstract=1117574 or http://dx.doi.org/10.2139/ssrn.1117574

Meucci, Attilio, *Fully Flexible Views: Theory and Practice* (August 8, 2008). Fully Flexible Views: Theory and Practice, Risk, Vol. 21, No. 10, pp. 97-102, October 2008. Available at SSRN: http://ssrn.com/abstract=1213325

Scherer, Bernd and Martin, Doug, *Modern Portfolio Optimization*. Springer. 2005.

Shaw, William Thornton, *Portfolio Optimization for VAR, CVaR, Omega and Utility with General Return Distributions: A Monte Carlo Approach for Long-Only and Bounded Short Portfolios with Optional Robustness and a Simplified Approach to Covariance Matching* (June 1, 2011). Available at SSRN: http://ssrn.com/abstract=1856476 or http://dx.doi.org/10.2139/ssrn.1856476

See Also

CRAN task view on Empirical Finance
https://cran.r-project.org/view=Econometrics

CRAN task view on Optimization
https://cran.r-project.org/view=Optimization

Large-scale portfolio optimization with DEoptim
https://cran.r-project.org/package=DEoptim

`ac.ranking` *Asset Ranking*

Description

Compute the first moment from a single complete sort

Usage

```
ac.ranking(R, order, ...)
```

Arguments

`R`	xts object of asset returns
`order`	a vector of indexes of the relative ranking of expected asset returns in ascending order. For example, `order = c(2, 3, 1, 4)` means that the expected returns of `R[,2] < R[,3], < R[,1] < R[,4]`.
`...`	any other passthrough parameters

Details

This function computes the estimated centroid vector from a single complete sort using the analytical approximation as described in R. Almgren and N. Chriss, "Portfolios from Sorts". The centroid is estimated and then scaled such that it is on a scale similar to the asset returns. By default, the centroid vector is scaled according to the median of the asset mean returns.

Value

The estimated first moments based on ranking views

References

R. Almgren and N. Chriss, "Portfolios from Sorts" http://papers.ssrn.com/sol3/papers.cfm?abstract_id=720041

See Also

`centroid.complete.mc` `centroid.sectors` `centroid.sign` `centroid.buckets`

Examples

```
data(edhec)
R <- edhec[,1:4]
ac.ranking(R, c(2, 3, 1, 4))
```

`add.constraint` *General interface for adding and/or updating optimization constraints.*

Description

This is the main function for adding and/or updating constraints to the `portfolio.spec` object.

Usage

```
add.constraint(portfolio, type, enabled = TRUE, message = FALSE, ...,
  indexnum = NULL)
```

Arguments

`portfolio`	an object of class 'portfolio' to add the constraint to, specifying the constraints for the optimization, see `portfolio.spec`
`type`	character type of the constraint to add or update, currently 'weight_sum' (also 'leverage' or 'weight'), 'box', 'group', 'turnover', 'diversification', 'position_limit', 'return', 'factor_exposure', or 'leverage_exposure'
`enabled`	TRUE/FALSE. The default is enabled=TRUE.
`message`	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
`...`	any other passthru parameters to specify constraints
`indexnum`	if you are updating a specific constraint, the index number in the $constraints list to update

Details

The following constraint types may be specified:

- `weight_sum`, `weight`, `leverage` Specify constraint on the sum of the weights, see `weight_sum_constraint`
- `full_investment` Special case to set `min_sum=1` and `max_sum=1` of weight sum constraints
- `dollar_neutral`, `active` Special case to set `min_sum=0` and `max_sum=0` of weight sum constraints
- `box` box constraints for the individual asset weights, see `box_constraint`
- `long_only` Special case to set `min=0` and `max=1` of box constraints
- `group` specify the sum of weights within groups and the number of assets with non-zero weights in groups, see `group_constraint`

- `turnover` Specify a constraint for target turnover. Turnover is calculated from a set of initial weights, see `turnover_constraint`
- `diversification` target diversification of a set of weights, see `diversification_constraint`
- `position_limit` Specify the number of non-zero, long, and/or short positions, see `position_limit_constraint`
- `return` Specify the target mean return, see `return_constraint`
- `factor_exposure` Specify risk factor exposures, see `factor_exposure_constraint`
- `leverage_exposure` Specify a maximum leverage exposure, see `leverage_exposure_constraint`

Author(s)

Ross Bennett

See Also

`portfolio.spec` `weight_sum_constraint`, `box_constraint`, `group_constraint`, `turnover_constraint`, `diversification_constraint`, `position_limit_constraint`, `return_constraint`, `factor_exposure_constraint`, `leverage_exposure_constraint`

Examples

```
data(edhec)
returns <- edhec[, 1:4]
fund.names <- colnames(returns)
pspec <- portfolio.spec(assets=fund.names)

# Add the full investment constraint that specifies the weights must sum to 1.
pspec <- add.constraint(portfolio=pspec, type="weight_sum", min_sum=1, max_sum=1)

# The full investment constraint can also be specified with type="full_investment"
pspec <- add.constraint(portfolio=pspec, type="full_investment")

# Another common constraint is that portfolio weights sum to 0.
pspec <- add.constraint(portfolio=pspec, type="weight_sum", min_sum=0, max_sum=0)
pspec <- add.constraint(portfolio=pspec, type="dollar_neutral")
pspec <- add.constraint(portfolio=pspec, type="active")

# Add box constraints
pspec <- add.constraint(portfolio=pspec, type="box", min=0.05, max=0.4)

# min and max can also be specified per asset
pspec <- add.constraint(portfolio=pspec,
                        type="box",
                        min=c(0.05, 0, 0.08, 0.1),
                        max=c(0.4, 0.3, 0.7, 0.55))

# A special case of box constraints is long only where min=0 and max=1
# The default action is long only if min and max are not specified
pspec <- add.constraint(portfolio=pspec, type="box")
pspec <- add.constraint(portfolio=pspec, type="long_only")
```

```
# Add group constraints
pspec <- add.constraint(portfolio=pspec,
                        type="group",
                        groups=list(c(1, 2, 1), 4),
                        group_min=c(0.1, 0.15),
                        group_max=c(0.85, 0.55),
                        group_labels=c("GroupA", "GroupB"),
                        group_pos=c(2, 1))

# Add position limit constraint such that we have a maximum number
# of three assets with non-zero weights.
pspec <- add.constraint(portfolio=pspec, type="position_limit", max_pos=3)

# Add diversification constraint
pspec <- add.constraint(portfolio=pspec, type="diversification", div_target=0.7)

# Add turnover constraint
pspec <- add.constraint(portfolio=pspec, type="turnover", turnover_target=0.2)

# Add target mean return constraint
pspec <- add.constraint(portfolio=pspec, type="return", return_target=0.007)

# Example using the indexnum argument
portf <- portfolio.spec(assets=fund.names)
portf <- add.constraint(portf, type="full_investment")
portf <- add.constraint(portf, type="long_only")

# indexnum corresponds to the index number of the constraint
# The full_investment constraint was the first constraint added and has
# indexnum=1
portf$constraints[[1]]

# View the constraint with indexnum=2
portf$constraints[[2]]

# Update the constraint to relax the sum of weights constraint
portf <- add.constraint(portf, type="weight_sum",
min_sum=0.99, max_sum=1.01,
indexnum=1)

# Update the constraint to modify the box constraint
portf <- add.constraint(portf, type="box",
min=0.1, max=0.8,
indexnum=2)
```

---

add.objective *General interface for adding optimization objectives, including risk, return, and risk budget*

Description

This function is the main function for adding and updating business objectives in an object of type portfolio.spec.

Usage

```
add.objective_v1(constraints, type, name, arguments = NULL, enabled = TRUE,
  ..., indexnum = NULL)

add.objective(portfolio, constraints = NULL, type, name, arguments = NULL,
  enabled = TRUE, ..., indexnum = NULL)
```

Arguments

constraints	a 'v1_constraint' object for backwards compatibility, see constraint
type	character type of the objective to add or update, currently 'return','risk', 'risk_budget', 'quadratic_utility', or 'weight_concentration'
name	name of the objective, should correspond to a function, though we will try to make allowances
arguments	default arguments to be passed to an objective function when executed
enabled	TRUE/FALSE
...	any other passthru parameters
indexnum	if you are updating a specific objective, the index number in the $objectives list to update
portfolio	an object of type 'portfolio' to add the objective to, specifying the portfolio for the optimization, see portfolio

Details

In general, you will define your objective as one of the following types: 'return', 'risk', 'risk_budget', 'quadratic utility', or 'weight_concentration'. These have special handling and intelligent defaults for dealing with the function most likely to be used as objectives, including mean, median, VaR, ES, etc.

Objectives of type 'turnover' and 'minmax' are also supported.

Author(s)

Brian G. Peterson and Ross Bennett

See Also

objective, portfolio.spec

Examples

```
data(edhec)
returns <- edhec[,1:4]
fund.names <- colnames(returns)
portf <- portfolio.spec(assets=fund.names)
# Add some basic constraints
portf <- add.constraint(portf, type="full_investment")
portf <- add.constraint(portf, type="long_only")

# Creates a new portfolio object using portf and adds a quadratic utility
# objective. This will add two objectives to the portfolio object; 1) mean and
# 2) var. The risk aversion parameter is commonly referred to as lambda in the
# quadratic utility formulation that controls how much the portfolio variance
# is penalized.
portf.maxQU <- add.objective(portf, type="quadratic_utility",
                             risk_aversion=0.25)

# Creates a new portfolio object using portf and adds mean as an objective
portf.maxMean <- add.objective(portf, type="return", name="mean")

# Creates a new portfolio object using portf and adds StdDev as an objective
portf.minStdDev <- add.objective(portf, type="risk", name="StdDev")

# Creates a new portfolio object using portf and adds ES as an objective.
# Note that arguments to ES are passed in as a named list.
portf.minES <- add.objective(portf, type="risk", name="ES",
                             arguments=list(p=0.925, clean="boudt"))

# Creates a new portfolio object using portf.minES and adds a risk budget
# objective with limits on component risk contribution.
# Note that arguments to ES are passed in as a named list.
portf.RiskBudgetES <- add.objective(portf.minES, type="risk_budget", name="ES",
                             arguments=list(p=0.925, clean="boudt"),
                             min_prisk=0, max_prisk=0.6)

# Creates a new portfolio object using portf.minES and adds a risk budget
# objective with equal component risk contribution.
# Note that arguments to ES are passed in as a named list.
portf.EqRiskES <- add.objective(portf.minES, type="risk_budget", name="ES",
                                  arguments=list(p=0.925, clean="boudt"),
                                  min_concentration=TRUE)

# Creates a new portfolio object using portf and adds a weight_concentration
# objective. The conc_aversion parameter controls how much concentration is
# penalized. The portfolio concentration is defined as the Herfindahl Hirschman
# Index of the weights.
portf.conc <- add.objective(portf, type="weight_concentration",
                            name="HHI", conc_aversion=0.01)
```

`add.sub.portfolio` *Add sub-portfolio*

Description

Add a sub-portfolio to a multiple layer portfolio specification object

Usage

```
add.sub.portfolio(mult.portfolio, portfolio, optimize_method = c("DEoptim",
  "random", "ROI", "pso", "GenSA"), search_size = 20000, rp = NULL,
  rebalance_on = NULL, training_period = NULL, trailing_periods = NULL,
  ..., indexnum = NULL)
```

Arguments

mult.portfolio	a mult.portfolio.spec object
portfolio	a portfolio object to add as a sub portfolio.
optimize_method	optimization method for the sub portfolio
search_size	integer, how many portfolios to test, default 20,000
rp	matrix of random portfolio weights, default NULL, mostly for automated use by rebalancing optimization or repeated tests on same portfolios
rebalance_on	haracter string of period to rebalance on. See endpoints for valid names.
training_period	an integer of the number of periods to use as a training data in the front of the returns data
trailing_periods	an integer with the number of periods to roll over (i.e. width of the moving or rolling window), the default is NULL will run using the returns data from inception
...	additonal passthrough parameters to optimize.portfolio.rebalancing
indexnum	the index number of the sub portfolio. If indexnum=NULL (the default), then the sub portfolio object is appended to the list of sub portfolios in the mult.portfolio object. If indexnum is specified, the portfolio in that index number is overwritten.

Author(s)

Ross Bennett

See Also

mult.portfolio.spec portfolio.spec optimize.portfolio optimize.portfolio.rebalancing

applyFUN *Apply a risk or return function to a set of weights*

Description

This function is used to calculate risk or return metrics given a matrix of weights and is primarily used as a convenience function used in chart.Scatter functions

Usage

```
applyFUN(R, weights, FUN = "mean", arguments)
```

Arguments

R	xts object of asset returns
weights	a matrix of weights generated from random_portfolios or optimize.portfolio
FUN	name of a function
arguments	named list of arguments to FUN

Author(s)

Ross Bennett

barplotGroupWeights *barplot of group weights by group or category*

Description

This function is called by chart.GroupWeights function if chart.type="barplot"

Usage

```
barplotGroupWeights(object, ..., grouping = c("groups", "category"),
  main = "Group Weights", las = 3, xlab = NULL, cex.lab = 0.8,
  element.color = "darkgray", cex.axis = 0.8)
```

Arguments

object	object of class optimize.portfolio
...	passthrough parameters to plot
grouping	• groups: group the weights by group constraints • category_labels: group the weights by category_labels in portfolio object
main	an overall title for the plot: see title

las	numeric in {0,1,2,3}; the style of axis labels **0:** always parallel to the axis [*default*], **1:** always horizontal, **2:** always perpendicular to the axis, **3:** always vertical.
xlab	a title for the x axis: see title
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex
element.color	color for the default border and axis
cex.axis	The magnification to be used for x and y axis relative to the current setting of cex

Author(s)

Ross Bennett

`black.litterman` *Black Litterman Estimates*

Description

Compute the Black Litterman estimate of moments for the posterior normal.

Usage

```
black.litterman(R, P, Mu = NULL, Sigma = NULL, Views = NULL)
```

Arguments

R	returns
P	a K x N pick matrix
Mu	vector of length N of the prior expected values. The sample mean is used if Mu=NULL.
Sigma	an N x N matrix of the prior covariance matrix. The sample covariance is used if Sigma=NULL.
Views	a vector of length K of the views

Value

- BLMu: posterior expected values
- BLSigma: posterior covariance matrix

Note

This function is largely based on the work of Xavier Valls to port the matlab code of Attilio Meucci to R as documented in the Meucci package.

Author(s)

Ross Bennett, Xavier Valls

References

A. Meucci - "Exercises in Advanced Risk and Portfolio Management" http://symmys.com/node/170.

See Also

BlackLittermanFormula

BlackLittermanFormula *Computes the Black-Litterman formula for the moments of the posterior normal.*

Description

This function computes the Black-Litterman formula for the moments of the posterior normal, as described in A. Meucci, "Risk and Asset Allocation", Springer, 2005.

Usage

```
BlackLittermanFormula(Mu, Sigma, P, v, Omega)
```

Arguments

Mu	[vector] (N x 1) prior expected values.
Sigma	[matrix] (N x N) prior covariance matrix.
P	[matrix] (K x N) pick matrix.
v	[vector] (K x 1) vector of views.
Omega	[matrix] (K x K) matrix of confidence.

Value

BLMu [vector] (N x 1) posterior expected values.

BLSigma [matrix] (N x N) posterior covariance matrix.

Author(s)

Xavier Valls <flamejat@gmail.com>

References

A. Meucci - "Exercises in Advanced Risk and Portfolio Management" http://symmys.com/node/170.

See Meucci's script for "BlackLittermanFormula.m"

---

box_constraint *constructor for box_constraint.*

Description

Box constraints specify the upper and lower bounds on the weights of the assets. This function is called by add.constraint when type="box" is specified. See add.constraint.

Usage

```
box_constraint(type = "box", assets, min, max, min_mult, max_mult,
  enabled = TRUE, message = FALSE, ...)
```

Arguments

type	character type of the constraint
assets	number of assets, or optionally a named vector of assets specifying initial weights
min	numeric or named vector specifying minimum weight box constraints
max	numeric or named vector specifying minimum weight box constraints
min_mult	numeric or named vector specifying minimum multiplier box constraint from initial weight in assets
max_mult	numeric or named vector specifying maximum multiplier box constraint from initial weight in assets
enabled	TRUE/FALSE
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters to specify box constraints

Value

an object of class 'box_constraint'

Author(s)

Ross Bennett

See Also

add.constraint

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

# defaults to min=0 and max=1
pspec <- add.constraint(pspec, type="box")

# specify box constraints as a scalar
pspec <- add.constraint(pspec, type="box", min=0.05, max=0.45)

# specify box constraints per asset
pspec <- add.constraint(pspec,
                        type="box",
                        min=c(0.05, 0.10, 0.08, 0.06),
                        max=c(0.45, 0.55, 0.35, 0.65))
```

CCCgarch.MM *compute comoments for use by lower level optimization functions when the conditional covariance matrix is a CCC GARCH model*

Description

it first estimates the conditional GARCH variances, then filters out the time-varying volatility and estimates the higher order comoments on the innovations rescaled such that their unconditional covariance matrix is the conditional covariance matrix forecast

Usage

```
CCCgarch.MM(R, momentargs = NULL, ...)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
momentargs	list containing arguments to be passed down to lower level functions, default NULL
...	any other passthru parameters

center *Center*

Description

Center a matrix

Usage

```
center(x)
```

Arguments

x	matrix

Details

This function is used primarily to center a time series of asset returns or factors. Each column should represent the returns of an asset or factor realizations. The expected value is taken as the sample mean.

x.centered = x - mean(x)

Value

matrix of centered data

centroid.buckets *Buckets Centroid*

Description

Compute the centroid for buckets of assets

Usage

```
centroid.buckets(buckets, simulations = 1000)
```

Arguments

buckets	a list where each element contains the index of the assets in the respective bucket. The assets within each bucket have no order. The bucket elements are in ascending order such that R_bucket_1 < ... < R_bucket_n
simulations	number of simulations

Details

A common use of buckets is to divide the assets into quartiles or deciles, but is generalized here for an arbitrary number of buckets and arbitrary number of assets in each bucket.

Value

the centroid vector

Author(s)

Ross Bennett

---

centroid.complete.mc *Complete Cases Centroid*

Description

Numerical method to estimate complete cases centroid

Usage

```
centroid.complete.mc(order, simulations = 1000)
```

Arguments

`order`	a vector of indexes of the relative ranking of expected asset returns in ascending order. For example, `order = c(2, 3, 1, 4)` expresses a view on the expected returns such that R_2 < R_3 < R_1 < R_4
`simulations`	number of simulations

Value

the centroid vector

Author(s)

Ross Bennett

Examples

```
# Express a view on the assets such that
# R_2 < R_1 < R_3 < R_4
centroid.complete.mc(c(2, 1, 3, 4))
```

centroid.sectors *Multiple Sectors Centroid*

Description

Compute the centroid for expressing views on the relative ranking of assets within sectors.

Usage

```
centroid.sectors(sectors, simulations = 1000)
```

Arguments

sectors	a list where each list element contains the order of each asset in the given sector
simulations	number of simulations

Value

the centroid vector

Author(s)

Ross Bennett

Examples

```
# Express a view on the assets in two sectors
# Sector 1 View: R_2 < R_1 < R_3
# Sector 2 View: R_5 < R_4
x <- list()
x[[1]] <- c(2, 1, 3)
x[[2]] <- c(5, 4)
centroid.sectors(x)
```

centroid.sign *Positive and Negative View Centroid*

Description

Compute the centroid for expressing a view on assets with positive or negative expected returns

Usage

```
centroid.sign(positive, negative, simulations = 1000)
```

Arguments

positive	a vector of the index of assets with positive expected return in ascending order
negative	a vector of the index of assets with negative expected return in ascending order.
simulations	number of simulations

Value

the centroid vector

Author(s)

Ross Bennett

Examples

```
# Express a view that
# R_1 < R_2 < 0 < R_3 < R_4
centroid.sign(c(1, 2), c(4, 3))
```

---

chart.Concentration *Classic risk reward scatter and concentration*

Description

This function charts the optimize.portfolio object in risk-return space and the degree of concentration based on the weights or percentage component contribution to risk.

Usage

```
chart.Concentration(object, ..., return.col = "mean", risk.col = "ES",
  chart.assets = FALSE, conc.type = c("weights", "pct_contrib"),
  col = heat.colors(20), element.color = "darkgray", cex.axis = 0.8,
  xlim = NULL, ylim = NULL)
```

Arguments

object	optimal portfolio created by optimize.portfolio.
...	any other passthru parameters.
return.col	string matching the objective of a 'return' objective, on vertical axis.
risk.col	string matching the objective of a 'risk' objective, on horizontal axis.
chart.assets	TRUE/FALSE. Includes a risk reward scatter of the assets in the chart.
conc.type	concentration type can be based on the concentration of weights or concentration of percentage component contribution to risk (only works with risk budget objective for the optimization).
col	color palette or vector of colors to use.

element.color	color for the border and axes.
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex.
xlim	set the x-axis limit, same as in plot.
ylim	set the y-axis limit, same as in plot.

Author(s)

Peter Carl and Ross Bennett

See Also

optimize.portfolio

---

chart.EF.Weights *Chart weights along an efficient frontier*

Description

This function produces a stacked barplot of weights along an efficient frontier.

Usage

```
chart.EF.Weights(object, ...)

## S3 method for class 'efficient.frontier'
chart.EF.Weights(object, ..., colorset = NULL,
  n.portfolios = 25, by.groups = FALSE, match.col = "ES", main = "",
  cex.lab = 0.8, cex.axis = 0.8, cex.legend = 0.8, legend.labels = NULL,
  element.color = "darkgray", legend.loc = "topright")

## S3 method for class 'optimize.portfolio'
chart.EF.Weights(object, ..., colorset = NULL,
  n.portfolios = 25, by.groups = FALSE, match.col = "ES", main = "",
  cex.lab = 0.8, cex.axis = 0.8, cex.legend = 0.8, legend.labels = NULL,
  element.color = "darkgray", legend.loc = "topright")
```

Arguments

object	object of class efficient.frontier or optimize.portfolio.
...	passthru parameters to barplot.
colorset	color palette or vector of colors to use.
n.portfolios	number of portfolios to extract along the efficient frontier.
by.groups	TRUE/FALSE. If TRUE, the group weights are charted.

match.col	string name of column to use for risk (horizontal axis). Must match the name of an objective.
main	title used in the plot.
cex.lab	the magnification to be used for x-axis and y-axis labels relative to the current setting of 'cex'.
cex.axis	the magnification to be used for sizing the axis text relative to the current setting of 'cex', similar to plot.
cex.legend	the magnification to be used for sizing the legend relative to the current setting of 'cex', similar to plot.
legend.labels	character vector to use for the legend labels.
element.color	provides the color for drawing less-important chart elements, such as the box lines, axis lines, etc.
legend.loc	NULL, "topright", "right", or "bottomright". If legend.loc is NULL, the legend will not be plotted.

Author(s)

Ross Bennett

---

chart.EfficientFrontier

Chart the efficient frontier and risk-return scatter

Description

Chart the efficient frontier and risk-return scatter of the assets for optimize.portfolio or efficient.frontier objects

Usage

```
chart.EfficientFrontier(object, ...)

## S3 method for class 'optimize.portfolio.ROI'
chart.EfficientFrontier(object, ...,
  match.col = "ES", n.portfolios = 25, xlim = NULL, ylim = NULL,
  cex.axis = 0.8, element.color = "darkgray", main = "Efficient Frontier",
  RAR.text = "SR", rf = 0, tangent.line = TRUE, cex.legend = 0.8,
  chart.assets = TRUE, labels.assets = TRUE, pch.assets = 21,
  cex.assets = 0.8)

## S3 method for class 'optimize.portfolio'
chart.EfficientFrontier(object, ...,
  match.col = "ES", n.portfolios = 25, xlim = NULL, ylim = NULL,
  cex.axis = 0.8, element.color = "darkgray", main = "Efficient Frontier",
  RAR.text = "SR", rf = 0, tangent.line = TRUE, cex.legend = 0.8,
```

```
  chart.assets = TRUE, labels.assets = TRUE, pch.assets = 21,
  cex.assets = 0.8)

## S3 method for class 'efficient.frontier'
chart.EfficientFrontier(object, ...,
  match.col = "ES", n.portfolios = NULL, xlim = NULL, ylim = NULL,
  cex.axis = 0.8, element.color = "darkgray", main = "Efficient Frontier",
  RAR.text = "SR", rf = 0, tangent.line = TRUE, cex.legend = 0.8,
  chart.assets = TRUE, labels.assets = TRUE, pch.assets = 21,
  cex.assets = 0.8)
```

Arguments

object	object to chart.
...	passthru parameters to plot
match.col	string name of column to use for risk (horizontal axis). match.col must match the name of an objective measure in the objective_measures or opt_values slot in the object created by optimize.portfolio.
n.portfolios	number of portfolios to use to plot the efficient frontier.
xlim	set the x-axis limit, same as in plot.
ylim	set the y-axis limit, same as in plot.
cex.axis	numerical value giving the amount by which the axis should be magnified relative to the default.
element.color	provides the color for drawing less-important chart elements, such as the box lines, axis lines, etc.
main	a main title for the plot.
RAR.text	string name for risk adjusted return text to plot in the legend.
rf	risk free rate. If rf is not null, the maximum Sharpe Ratio or modified Sharpe Ratio tangency portfolio will be plotted.
tangent.line	TRUE/FALSE to plot the tangent line.
cex.legend	numerical value giving the amount by which the legend should be magnified relative to the default.
chart.assets	TRUE/FALSE to include the assets.
labels.assets	TRUE/FALSE to include the asset names in the plot. chart.assets must be TRUE to plot asset names.
pch.assets	plotting character of the assets, same as in plot.
cex.assets	numerical value giving the amount by which the asset points and labels should be magnified relative to the default.

Details

For objects created by optimize.portfolio with 'DEoptim', 'random', or 'pso' specified as the optimize_method:

- The efficient frontier plotted is based on the the trace information (sets of portfolios tested by the solver at each iteration) in objects created by `optimize.portfolio`.

For objects created by optimize.portfolio with 'ROI' specified as the optimize_method:

- The mean-StdDev or mean-ETL efficient frontier can be plotted for optimal portfolio objects created by `optimize.portfolio`.
- If `match.col="StdDev"`, the mean-StdDev efficient frontier is plotted.
- If `match.col="ETL"` (also "ES" or "CVaR"), the mean-ETL efficient frontier is plotted.

Note that `trace=TRUE` must be specified in `optimize.portfolio`

GenSA does not return any useable trace information for portfolios tested at each iteration, therfore we cannot extract and chart an efficient frontier.

By default, the tangency portfolio (maximum Sharpe Ratio or modified Sharpe Ratio) will be plotted using a risk free rate of 0. Set `rf=NULL` to omit this from the plot.

Author(s)

Ross Bennett

---

chart.EfficientFrontierOverlay

Plot multiple efficient frontiers

Description

Overlay the efficient frontiers of multiple portfolio objects on a single plot.

Usage

```
chart.EfficientFrontierOverlay(R, portfolio_list, type, n.portfolios = 25,
  match.col = "ES", search_size = 2000, main = "Efficient Frontiers",
  cex.axis = 0.8, element.color = "darkgray", legend.loc = NULL,
  legend.labels = NULL, cex.legend = 0.8, xlim = NULL, ylim = NULL, ...,
  chart.assets = TRUE, labels.assets = TRUE, pch.assets = 21,
  cex.assets = 0.8, col = NULL, lty = NULL, lwd = NULL)
```

Arguments

`R`	an xts object of asset returns
`portfolio_list`	list of portfolio objects created by `portfolio.spec` and combined with `combine.portfolios`
`type`	type of efficient frontier, see `create.EfficientFrontier`
`n.portfolios`	number of portfolios to extract along the efficient frontier. This is only used for objects of class `optimize.portfolio`
`match.col`	string name of column to use for risk (horizontal axis). Must match the name of an objective.

search_size	passed to optimize.portfolio for type="DEoptim" or type="random".
main	title used in the plot.
cex.axis	the magnification to be used for sizing the axis text relative to the current setting of 'cex', similar to plot.
element.color	provides the color for drawing less-important chart elements, such as the box lines, axis lines, etc.
legend.loc	location of the legend; NULL, "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center".
legend.labels	character vector to use for the legend labels.
cex.legend	The magnification to be used for sizing the legend relative to the current setting of 'cex', similar to plot.
xlim	set the x-axis limit, same as in plot.
ylim	set the y-axis limit, same as in plot.
...	passthrough parameters to plot.
chart.assets	TRUE/FALSE to include the assets.
labels.assets	TRUE/FALSE to include the asset names in the plot.
pch.assets	plotting character of the assets, same as in plot.
cex.assets	A numerical value giving the amount by which the asset points and labels should be magnified relative to the default.
col	vector of colors with length equal to the number of portfolios in portfolio_list.
lty	vector of line types with length equal to the number of portfolios in portfolio_list.
lwd	vector of line widths with length equal to the number of portfolios in portfolio_list.

Author(s)

Ross Bennett

---

chart.GroupWeights *Chart weights by group or category*

Description

Chart weights by group or category

Usage

```
chart.GroupWeights(object, ..., grouping = c("groups", "category"),
  plot.type = "line", main = "Group Weights", las = 3, xlab = NULL,
  cex.lab = 0.8, element.color = "darkgray", cex.axis = 0.8)
```

Arguments

object	object of class optimize.portfolio.
...	passthrough parameters to plot.
grouping	• groups: group the weights by group constraints. • category_labels: group the weights by category_labels in the portfolio object.
plot.type	"line" or "barplot".
main	an overall title for the plot: see title.
las	numeric in {0,1,2,3}; the style of axis labels **0:** always parallel to the axis, **1:** always horizontal, **2:** always perpendicular to the axis, **3:** always vertical[*default*].
xlab	a title for the x axis: see title.
cex.lab	the magnification to be used for x and y labels relative to the current setting of cex.
element.color	color for the default border and axis.
cex.axis	the magnification to be used for x and y axis relative to the current setting of cex.

Author(s)

Ross Bennett

---

chart.RiskBudget *Generic method to chart risk contribution*

Description

This function is the generic method to chart risk budget objectives for optimize.portfolio, optimize.portfolio.rebalancing, and opt.list objects. This function charts the contribution or percent contribution of the resulting objective measures of a risk_budget_objective. The risk contributions for optimize.portfolio.rebalancing objects are plotted through time with chart.StackedBar.

Usage

```
chart.RiskBudget(object, ...)

## S3 method for class 'optimize.portfolio'
chart.RiskBudget(object, ..., neighbors = NULL,
  risk.type = "absolute", main = "Risk Contribution", ylab = "",
  xlab = NULL, cex.axis = 0.8, cex.lab = 0.8,
```

```
  element.color = "darkgray", las = 3, ylim = NULL)

## S3 method for class 'optimize.portfolio.rebalancing'
chart.RiskBudget(object, ...,
  match.col = "ES", risk.type = "absolute", regime = NULL,
  main = "Risk Contribution")

## S3 method for class 'opt.list'
chart.RiskBudget(object, ..., match.col = "ES",
  risk.type = "absolute", main = "Risk Budget", plot.type = "line",
  cex.axis = 0.8, cex.lab = 0.8, element.color = "darkgray", las = 3,
  ylim = NULL, colorset = NULL, legend.loc = NULL, cex.legend = 0.8)
```

Arguments

object	optimal portfolio object created by optimize.portfolio or optimize.portfolio.rebalancing
...	any other passthru parameters to plot
neighbors	risk contribution or pct_contrib of neighbor portfolios to be plotted, see Details.
risk.type	"absolute" or "percentage" to plot risk contribution in absolute terms or percentage contribution.
main	main title for the chart.
ylab	label for the y-axis.
xlab	label for the x-axis.
cex.axis	the magnification to be used for axis annotation relative to the current setting of cex.
cex.lab	the magnification to be used for axis annotation relative to the current setting of cex.
element.color	provides the color for drawing less-important chart elements, such as the box lines, axis lines, etc.
las	numeric in {0,1,2,3}; the style of axis labels **0:** always parallel to the axis [*default*], **1:** always horizontal, **2:** always perpendicular to the axis, **3:** always vertical.
ylim	set the y-axis limit, same as in plot
match.col	string of risk column to match. The opt.list object may contain risk budgets for ES or StdDev and this will match the proper column names of the objectives list outp (e.g. ES.contribution).
regime	integer of the regime number. For use with optimize.portfolio.rebalancing run with regime switching portfolios.
plot.type	"line" or "barplot".
colorset	color palette or vector of colors to use
legend.loc	legend.loc NULL, "topright", "right", or "bottomright". If legend.loc is NULL, the legend will not be plotted
cex.legend	The magnification to be used for the legend relative to the current setting of cex

Details

neighbors may be specified in three ways. The first is as a single number of neighbors. This will extract the neighbors closest to the portfolios in terms of the out numerical statistic. The second method consists of a numeric vector for neighbors. This will extract the neighbors with portfolio index numbers that correspond to the vector contents. The third method for specifying neighbors is to pass in a matrix. This matrix should look like the output of extractStats, and should contain properly named contribution and pct_contrib columns.

See Also

optimize.portfolio optimize.portfolio.rebalancing chart.StackedBar

---

chart.RiskReward *classic risk reward scatter*

Description

This function charts the optimize.portfolio object in risk-return space.

Usage

```
chart.RiskReward(object, ...)

## S3 method for class 'optimize.portfolio.DEoptim'
chart.RiskReward(object, ...,
  neighbors = NULL, return.col = "mean", risk.col = "ES",
  chart.assets = FALSE, element.color = "darkgray", cex.axis = 0.8,
  xlim = NULL, ylim = NULL)

## S3 method for class 'optimize.portfolio.GenSA'
chart.RiskReward(object, ...,
  neighbors = NULL, return.col = "mean", risk.col = "ES",
  chart.assets = FALSE, element.color = "darkgray", cex.axis = 0.8,
  ylim = NULL, xlim = NULL, rp = FALSE)

## S3 method for class 'optimize.portfolio.pso'
chart.RiskReward(object, ...,
  neighbors = NULL, return.col = "mean", risk.col = "ES",
  chart.assets = FALSE, element.color = "darkgray", cex.axis = 0.8,
  xlim = NULL, ylim = NULL)

## S3 method for class 'optimize.portfolio.ROI'
chart.RiskReward(object, ...,
  neighbors = NULL, return.col = "mean", risk.col = "ES",
  chart.assets = FALSE, element.color = "darkgray", cex.axis = 0.8,
  xlim = NULL, ylim = NULL, rp = FALSE)
```

```
## S3 method for class 'optimize.portfolio.random'
chart.RiskReward(object, ...,
  neighbors = NULL, return.col = "mean", risk.col = "ES",
  chart.assets = FALSE, element.color = "darkgray", cex.axis = 0.8,
  xlim = NULL, ylim = NULL)

## S3 method for class 'opt.list'
chart.RiskReward(object, ..., risk.col = "ES",
  return.col = "mean", main = "", ylim = NULL, xlim = NULL,
  labels.assets = TRUE, chart.assets = FALSE, pch.assets = 1,
  cex.assets = 0.8, cex.axis = 0.8, cex.lab = 0.8, colorset = NULL,
  element.color = "darkgray")
```

Arguments

object	optimal portfolio created by optimize.portfolio.
...	any other passthru parameters.
neighbors	set of 'neighbor' portfolios to overplot, see Details.
return.col	string matching the objective of a 'return' objective, on vertical axis.
risk.col	string matching the objective of a 'risk' objective, on horizontal axis.
chart.assets	TRUE/FALSE. Includes a risk reward scatter of the assets in the chart.
element.color	color for the default plot scatter points.
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex.
xlim	set the x-axis limit, same as in plot.
ylim	set the y-axis limit, same as in plot.
rp	TRUE/FALSE to generate random portfolios to plot the feasible space
main	a main title for the plot.
labels.assets	TRUE/FALSE to include the names in the plot.
pch.assets	plotting character of the assets, same as in plot
cex.assets	numerical value giving the amount by which the asset points should be magnified relative to the default.
cex.lab	numerical value giving the amount by which the labels should be magnified relative to the default.
colorset	color palette or vector of colors to use.

Details

neighbors may be specified in three ways. The first is as a single number of neighbors. This will extract the neighbors closest portfolios in terms of the out numerical statistic. The second method consists of a numeric vector for neighbors. This will extract the neighbors with portfolio index numbers that correspond to the vector contents. The third method for specifying neighbors is to pass in a matrix. This matrix should look like the output of extractStats, and should contain risk.col,return.col, and weights columns all properly named.

See Also

optimize.portfolio

---

chart.Weights *boxplot of the weights of the optimal portfolios*

Description

This function charts the optimal weights of a portfolio run via optimize.portfolio or optimize.portfolio.rebalancing The upper and lower bounds on weights can be plotted for single period optimizations. The optimal weights will be charted through time for optimize.portfolio.rebalancing objects. For optimize.portfolio.rebalancing objects, the weights are plotted with chart.StackedBar.

Usage

```
chart.Weights(object, ...)

## S3 method for class 'optimize.portfolio.rebalancing'
chart.Weights(object, ...,
  main = "Weights")

## S3 method for class 'optimize.portfolio.DEoptim'
chart.Weights(object, ...,
  neighbors = NULL, main = "Weights", las = 3, xlab = NULL,
  cex.lab = 1, element.color = "darkgray", cex.axis = 0.8,
  colorset = NULL, legend.loc = "topright", cex.legend = 0.8,
  plot.type = "line")

## S3 method for class 'optimize.portfolio.GenSA'
chart.Weights(object, ...,
  neighbors = NULL, main = "Weights", las = 3, xlab = NULL,
  cex.lab = 1, element.color = "darkgray", cex.axis = 0.8,
  colorset = NULL, legend.loc = "topright", cex.legend = 0.8,
  plot.type = "line")

## S3 method for class 'optimize.portfolio.pso'
chart.Weights(object, ..., neighbors = NULL,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8, colorset = NULL,
  legend.loc = "topright", cex.legend = 0.8, plot.type = "line")

## S3 method for class 'optimize.portfolio.ROI'
chart.Weights(object, ..., neighbors = NULL,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8, colorset = NULL,
  legend.loc = "topright", cex.legend = 0.8, plot.type = "line")
```

```
## S3 method for class 'optimize.portfolio.random'
chart.Weights(object, ...,
  neighbors = NULL, main = "Weights", las = 3, xlab = NULL,
  cex.lab = 1, element.color = "darkgray", cex.axis = 0.8,
  colorset = NULL, legend.loc = "topright", cex.legend = 0.8,
  plot.type = "line")

## S3 method for class 'opt.list'
chart.Weights(object, neighbors = NULL, ...,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8, colorset = NULL,
  legend.loc = "topright", cex.legend = 0.8, plot.type = "line")
```

Arguments

object	optimal portfolio object created by optimize.portfolio.
...	any other passthru parameters .
main	an overall title for the plot: see title
neighbors	set of 'neighbor' portfolios to overplot. See Details.
las	numeric in {0,1,2,3}; the style of axis labels **0:** always parallel to the axis, **1:** always horizontal, **2:** always perpendicular to the axis, **3:** always vertical [*default*].
xlab	a title for the x axis: see title
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex
element.color	provides the color for drawing less-important chart elements, such as the box lines, axis lines, etc.
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex.
colorset	color palette or vector of colors to use.
legend.loc	location of the legend. If NULL, the legend will not be plotted.
cex.legend	The magnification to be used for legend annotation relative to the current setting of cex.
plot.type	"line" or "barplot" to plot.

See Also

optimize.portfolio optimize.portfolio.rebalancing chart.StackedBar

check_constraints *check if a set of weights satisfies the constraints*

Description

This function checks if a set of weights satisfies all constraints. This is used as a helper function for random portfolios created with `rp_simplex` and `rp_grid` to eliminate portfolios that do not satisfy the constraints.

Usage

```
check_constraints(weights, portfolio)
```

Arguments

`weights`	vector of weights
`portfolio`	object of class 'portfolio'

Value

TRUE if all constraints are satisfied, FALSE if any constraint is violated

Author(s)

Ross Bennett

cokurtosisMF *Cokurtosis Matrix Estimate*

Description

Estimate cokurtosis matrix using a statistical factor model

Usage

```
cokurtosisMF(beta, stockM2, stockM4, factorM2, factorM4)
```

Arguments

`beta`	(N x k) matrix of factor loadings (i.e. the betas) from a statistical factor model
`stockM2`	vector of length N of the 2nd moment of the model residuals
`stockM4`	vector of length N of the 4th moment of the model residuals
`factorM2`	(k x k) matrix of the 2nd moment of the factor realizations from a statistical factor model
`factorM4`	(k x k^3) matrix of the 4th moment of the factor realizations from a statistical factor model

Details

This function estimates an (N x N^3) cokurtosis matrix from a statistical factor model with k factors, where N is the number of assets.

Value

(N x N^3) cokurtosis matrix

`cokurtosisSF` *Cokurtosis Matrix Estimate*

Description

Estimate cokurtosis matrix using a single factor statistical factor model

Usage

```
cokurtosisSF(beta, stockM2, stockM4, factorM2, factorM4)
```

Arguments

`beta`	vector of length N or (N x 1) matrix of factor loadings (i.e. the betas) from a single factor statistical factor model
`stockM2`	vector of length N of the 2nd moment of the model residuals
`stockM4`	vector of length N of the 4th moment of the model residuals
`factorM2`	scalar of the 2nd moment of the factor realizations from a single factor statistical factor model
`factorM4`	scalar of the 4th moment of the factor realizations from a single factor statistical factor model

Details

This function estimates an (N x N^3) cokurtosis matrix from a statistical factor model with k factors, where N is the number of assets.

Value

(N x N^3) cokurtosis matrix

combine.optimizations *Combine objects created by optimize.portfolio*

Description

This function takes a list of objects created by optimize.portfolio and sets the class name attribute to 'opt.list' for use in generic functions

Usage

```
combine.optimizations(x)
```

Arguments

x	a list of objects created by optimize.portfolio

Value

an opt.list object

combine.portfolios *Combine a list of portfolio objects*

Description

This function takes a list of objects created by portfolio.spec and sets the class name attribute to 'portfolio.list' for use in generic functions

Usage

```
combine.portfolios(x)
```

Arguments

x	a list of objects created by portfolio.spec

Value

a portfolio.list object

constrained_objective *calculate a numeric return value for a portfolio based on a set of constraints and objectives*

Description

Function to calculate a numeric return value for a portfolio based on a set of constraints and objectives. We'll try to make as few assumptions as possible and only run objectives that are enabled by the user.

Usage

```
constrained_objective_v1(w, R, constraints, ..., trace = FALSE,
  normalize = TRUE, storage = FALSE)

constrained_objective(w, R, portfolio, ..., trace = FALSE, normalize = TRUE,
  storage = FALSE, env = NULL)
```

Arguments

w	a vector of weights to test.
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns.
constraints	a v1_constraint object for backwards compatibility with constrained_objective_v1.
...	any other passthru parameters.
trace	TRUE/FALSE whether to include debugging and additional detail in the output list. The default is FALSE. Several charting functions require that trace=TRUE.
normalize	TRUE/FALSE whether to normalize results to min/max sum (TRUE), or let the optimizer penalize portfolios that do not conform (FALSE)
storage	TRUE/FALSE default TRUE for DEoptim with trace, otherwise FALSE. not typically user-called.
portfolio	an object of class portfolio specifying the constraints and objectives for the optimization, see portfolio.
env	environment of moments calculated in optimize.portfolio

Details

If the user has passed in either min_sum or max_sum constraints for the portfolio, or both, and are using a numerical optimization method like DEoptim, and normalize=TRUE, we'll normalize the weights passed in to whichever boundary condition has been violated. If using random portfolios, all the portfolios generated will meet the constraints by construction. NOTE: this means that the weights produced by a numeric optimization algorithm like DEoptim, pso, or GenSA might violate constraints, and will need to be renormalized after optimizing. We apply the same normalization in optimize.portfolio so that the weights you see have been normalized to min_sum if the generated portfolio is smaller than min_sum or max_sum if the generated portfolio is larger

than max_sum. This normalization increases the speed of optimization and convergence by several orders of magnitude in many cases.

You may find that for some portfolios, normalization is not desirable, if the algorithm cannot find a direction in which to move to head towards an optimal portfolio. In these cases, it may be best to set normalize=FALSE, and penalize the portfolios if the sum of the weighting vector lies outside the min_sum and/or max_sum.

Whether or not we normalize the weights using min_sum and max_sum, and are using a numerical optimization engine like DEoptim, we will penalize portfolios that violate weight constraints in much the same way we penalize other constraints. If a min_sum/max_sum normalization has not occurred, convergence can take a very long time. We currently do not allow for a non-normalized full investment constraint. Future version of this function could include this additional constraint penalty.

When you are optimizing a return objective, you must specify a negative multiplier for the return objective so that the function will maximize return. If you specify a target return, any return that deviates from your target will be penalized. If you do not specify a target return, you may need to specify a negative VTR (value to reach) , or the function will not converge. Try the maximum expected return times the multiplier (e.g. -1 or -10). Adding a return objective defaults the multiplier to -1.

Additional parameters for other solvers (e.g. random portfolios or `DEoptim.control` or pso or GenSA may be passed in via ...

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson, Ross Bennett

See Also

`constraint`, `objective`, `DEoptim.control`

`constraint_ROI` *constructor for class constraint_ROI*

Description

constructor for class constraint_ROI

Usage

```
constraint_ROI(assets = NULL, op.problem, solver = c("glpk", "quadprog"),
  weight_seq = NULL)
```

Arguments

assets	number of assets, or optionally a named vector of assets specifying seed weights
op.problem	an object of type "OP" (optimization problem, of ROI) specifying the complete optimization problem, see ROI help pages for proper construction of OP object.
solver	string argument for what solver package to use, must have ROI plugin installed for that solver. Currently support is for glpk and quadprog.
weight_seq	seed sequence of weights, see generatesequence

Author(s)

Hezky Varon

---

constraint_v1 *constructors for class constraint*

Description

See main documentation entry in add.constraint.

Usage

```
constraint_v1(assets = NULL, ..., min, max, min_mult, max_mult,
  min_sum = 0.99, max_sum = 1.01, weight_seq = NULL)

constraint(type, enabled = TRUE, ..., constrclass = "v2_constraint")
```

Arguments

assets	number of assets, or optionally a named vector of assets specifying initial weights
...	any other passthru parameters
min	numeric or named vector specifying minimum weight box constraints
max	numeric or named vector specifying minimum weight box constraints
min_mult	numeric or named vector specifying minimum multiplier box constraint from initial weight in assets
max_mult	numeric or named vector specifying maximum multiplier box constraint from initial weight in assets
min_sum	minimum sum of all asset weights, default .99
max_sum	maximum sum of all asset weights, default 1.01
weight_seq	seed sequence of weights, see generatesequence
type	character type of the constraint to add or update
enabled	TRUE/FALSE to enabled the constraint
constrclass	name of class for the constraint

Details

This includes the deprecated constructor for the `v1_constraint` object for backwards compatibility.

Author(s)

Peter Carl, Brian G. Peterson, Ross Bennett

See Also

`add.constraint`

`coskewnessMF` *Coskewness Matrix Estimate*

Description

Estimate coskewness matrix using a statistical factor model

Usage

```
coskewnessMF(beta, stockM3, factorM3)
```

Arguments

`beta`	(N x k) matrix of factor loadings (i.e. the betas) from a statistical factor model
`stockM3`	vector of length N of the 3rd moment of the model residuals
`factorM3`	(k x k^2) matrix of the 3rd moment of the factor realizations from a statistical factor model

Details

This function estimates an (N x N^2) coskewness matrix from a statistical factor model with k factors, where N is the number of assets.

Value

(N x N^2) coskewness matrix

coskewnessSF *Coskewness Matrix Estimate*

Description

Estimate coskewness matrix using a single factor statistical factor model

Usage

```
coskewnessSF(beta, stockM3, factorM3)
```

Arguments

beta	vector of length N or (N x 1) matrix of factor loadings (i.e. the betas) from a single factor statistical factor model
stockM3	vector of length N of the 3rd moment of the model residuals
factorM3	scalar of the 3rd moment of the factor realizations from a single factor statistical factor model

Details

This function estimates an (N x N^2) coskewness matrix from a single factor statistical factor model with k=1 factors, where N is the number of assets.

Value

(N x N^2) coskewness matrix

covarianceMF *Covariance Matrix Estimate*

Description

Estimate covariance matrix using a statistical factor model

Usage

```
covarianceMF(beta, stockM2, factorM2)
```

Arguments

beta	(N x k) matrix of factor loadings (i.e. the betas) from a statistical factor model
stockM2	vector of length N of the variance (2nd moment) of the model residuals (i.e. idiosyncratic variance of the stock)
factorM2	(k x k) matrix of the covariance (2nd moment) of the factor realizations from a statistical factor model

Details

This function estimates an (N x N) covariance matrix from a statistical factor model with k factors, where N is the number of assets.

Value

(N x N) covariance matrix

`covarianceSF` *Covariance Matrix Estimate*

Description

Estimate covariance matrix using a single factor statistical factor model

Usage

```
covarianceSF(beta, stockM2, factorM2)
```

Arguments

`beta`	vector of length N or (N x 1) matrix of factor loadings (i.e. the betas) from a single factor statistical factor model
`stockM2`	vector of length N of the variance (2nd moment) of the model residuals (i.e. idiosyncratic variance of the stock)
`factorM2`	scalar value of the 2nd moment of the factor realizations from a single factor statistical factor model

Details

This function estimates an (N x N) covariance matrix from a single factor statistical factor model with k=1 factors, where N is the number of assets.

Value

(N x N) covariance matrix

`create.EfficientFrontier` *create an efficient frontier*

Description

create an efficient frontier

Usage

```
create.EfficientFrontier(R, portfolio, type, n.portfolios = 25,
  risk_aversion = NULL, match.col = "ES", search_size = 2000, ...)
```

Arguments

`R`	xts object of asset returns
`portfolio`	object of class 'portfolio' specifying the constraints and objectives, see `portfolio.spec`.
`type`	type of efficient frontier, see Details.
`n.portfolios`	number of portfolios to calculate along the efficient frontier
`risk_aversion`	vector of risk_aversion values to construct the efficient frontier. `n.portfolios` is ignored if `risk_aversion` is specified and the number of points along the efficient frontier will be equal to the length of `risk_aversion`.
`match.col`	column to match when extracting the efficient frontier from an objected created by `optimize.portfolio`.
`search_size`	passed to `optimize.portfolio` for type="DEoptim" or type="random".
`...`	passthrough parameters to `optimize.portfolio`.

Details

Currently there are 4 'types' supported to create an efficient frontier:

- "mean-var", "mean-sd", or "mean-StdDev": This is a special case for an efficient frontier that can be created by a QP solver. The `portfolio` object should have two objectives: 1) mean and 2) var. If the portfolio object does not contain these objectives, they will be added using default parameters. The efficient frontier will be created via `meanvar.efficient.frontier`.
- "mean-ETL", "mean-ES", "mean-CVaR", "mean-etl": This is a special case for an efficient frontier that can be created by an LP solver. The `portfolio` object should have two objectives: 1) mean and 2) ETL/ES/CVaR. If the portfolio object does not contain these objectives, they will be added using default parameters. The efficient frontier is created via `meanetl.efficient.frontier`.
- "DEoptim": This can handle more complex constraints and objectives than the simple mean-var and mean-ETL cases. For this type, we actually call `optimize.portfolio` with `optimize_method="DEoptim"` and then extract the efficient frontier with `extract.efficient.frontier`.
- "random": This can handle more complex constraints and objectives than the simple mean-var and mean-ETL cases. For this type, we actually call `optimize.portfolio` with `optimize_method="random"` and then extract the efficient frontier with `extract.efficient.frontier`.

Value

an object of class 'efficient.frontier' with the objective measures and weights of portfolios along the efficient frontier.

Author(s)

Ross Bennett

See Also

optimize.portfolio, portfolio.spec, meanvar.efficient.frontier, meanetl.efficient.frontier

---

diversification *Function to compute diversification as a constraint*

Description

Diversification is defined as 1 minus the sum of the squared weights

$$diversification = 1 - sum(w^2)$$

Usage

```
diversification(weights)
```

Arguments

weights	vector of asset weights

Author(s)

Ross Bennett

---

diversification_constraint *constructor for diversification_constraint*

Description

The diversification constraint specifies a target diversification value. This function is called by add.constraint when type="diversification" is specified, see add.constraint. Diversification is computed as `1 - sum(weights^2)`.

Usage

```
diversification_constraint(type = "diversification", div_target = NULL,
  enabled = TRUE, message = FALSE, ...)
```

Arguments

type	character type of the constraint
div_target	diversification target value
enabled	TRUE/FALSE
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters to specify diversification constraint an object of class 'diversification_constraint'

Author(s)

Ross Bennett

See Also

add.constraint

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

pspec <- add.constraint(portfolio=pspec, type="diversification", div_target=0.7)
```

EntropyProg *Entropy pooling program for blending views on scenarios with a prior scenario-probability distribution*

Description

Entropy program will change the initial predictive distribution 'p' to a new set 'p_' that satisfies specified moment conditions but changes other propoerties of the new distribution the least by minimizing the relative entropy between the two distributions. Theoretical note: Relative Entropy (Kullback-Leibler information criterion KLIC) is an asymmetric measure.

Usage

```
EntropyProg(p, A = NULL, b = NULL, Aeq, beq, verbose = FALSE)
```

Arguments

p	a vector of initial probabilities based on prior (reference model, empirical distribution, etc.). Sum of 'p' must be 1
A	matrix consisting of inequality constraints (paired with argument 'b'). Denoted as 'F' in the Meucci paper
b	vector consisting of inequality constraints (paired with matrix A). Denoted as 'f' in the Meucci paper
Aeq	matrix consisting of equality constraints (paired with argument 'beq'). Denoted as 'H' in the Meucci paper. (denoted as 'H' in the "Meucci - Flexible Views Theory & Practice" paper formlua 86 on page 22)
beq	vector corresponding to the matrix of equality constraints (paired with argument 'Aeq'). Denoted as 'h' in the Meucci paper
verbose	If TRUE, prints out additional information. Default FALSE.

'

$$\tilde{p} \equiv argmin_{Fx \leq f, Hx \equiv h} \{ \sum_{1}^{J} x_j \big(ln\big(x_j\big) - ln\big(p_j\big)\big) \} \ell\big(x, \lambda, \nu\big) \equiv x' \big(ln\big(x\big) - ln\big(p\big)\big) + \lambda' \big(Fx - f\big) + \nu' \big(H$$

Details

We retrieve a new set of probabilities for the joint-scenarios using the Entropy pooling method Of the many choices of 'p' that satisfy the views, we choose 'p' that minimize the entropy or distance of the new probability distribution to the prior joint-scenario probabilities.

We use Kullback-Leibler divergence or relative entropy dist(p,q): Sum across all scenarios [p-t * ln(p-t / q-t)] Therefore we define solution as p* = argmin (choice of p) [sum across all scenarios: p-t * ln(p-t / q-t)], such that 'p' satisfies views. The views modify the prior in a cohrent manner (minimizing distortion) We forumulate the stress tests of the baseline scenarios as linear constraints on yet-to-be defined probabilities Note that the numerical optimization acts on a very limited number of variables equal to the number of views. It does not act directly on the very large number of variables of interest, namely the probabilities of the Monte Carlo scenarios. This feature guarantees the numerical feasability of entropy optimization.

Note that new probabilities are generated in much the same way that the state-price density modifies objective probabilities of pay-offs to risk-neutral probabilities in contingent-claims asset pricing

Compute posterior (=change of measure) with Entropy Pooling, as described in

Value

a list with

- p_: revised probabilities based on entropy pooling
- optimizationPerformance: a list with status of optimization, value, number of iterations, and sum of probabilities

Author(s)

Ram Ahluwalia <ram@wingedfootcapital.com>

References

A. Meucci - "Fully Flexible Views: Theory and Practice". See page 22 for illustration of numerical implementation Symmys site containing original MATLAB source code http://www.symmys.com NLOPT open-source optimization site containing background on algorithms http://ab-initio.mit.edu/wiki/index.php/NLopt We use the information-theoretic estimator of Kitamur and Stutzer (1997). Reversing 'p' and 'p_' leads to the empirical likelihood" estimator of Qin and Lawless (1994). See Robertson et al, "Forecasting Using Relative Entropy" (2002) for more theory

---

`equal.weight` *Create an equal weight portfolio*

Description

This function calculates objective measures for an equal weight portfolio.

Usage

```
equal.weight(R, portfolio, ...)
```

Arguments

`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`portfolio`	an object of type "portfolio" specifying the constraints and objectives for the optimization
`...`	any other passthru parameters to `constrained_objective`

Details

This function is simply a wrapper around `constrained_objective` to calculate the objective measures in the given `portfolio` object of an equal weight portfolio. The portfolio object should include all objectives to be calculated.

Value

a list containing the returns, weights, objective measures, call, and portfolio object

Author(s)

Ross Bennett

etl_milp_opt *Minimum ETL MILP Optimization*

Description

This function is called by optimize.portfolio to solve minimum ETL problems via mixed integer linear programming.

Usage

```
etl_milp_opt(R, constraints, moments, target, alpha, solver = "glpk",
  control = NULL)
```

Arguments

R	xts object of asset returns
constraints	object of constraints in the portfolio object extracted with get_constraints
moments	object of moments computed based on objective functions
target	target return value
alpha	alpha value for ETL/ES/CVaR
solver	solver to use
control	list of solver control parameters

Author(s)

Ross Bennett

etl_opt *Minimum ETL LP Optimization*

Description

This function is called by optimize.portfolio to solve minimum ETL problems.

Usage

```
etl_opt(R, constraints, moments, target, alpha, solver = "glpk",
  control = NULL)
```

Arguments

R	xts object of asset returns
constraints	object of constraints in the portfolio object extracted with get_constraints
moments	object of moments computed based on objective functions
target	target return value
alpha	alpha value for ETL/ES/CVaR
solver	solver to use
control	list of solver control parameters

Author(s)

Ross Bennett

---

extractCokurtosis *Cokurtosis Estimate*

Description

Extract the cokurtosis matrix estimate from a statistical factor model

Usage

```
extractCokurtosis(model, ...)
```

Arguments

model	statistical factor model estimated via statistical.factor.model
...	not currently used

Value

cokurtosis matrix estimate

Author(s)

Ross Bennett

See Also

statistical.factor.model

extractCoskewness *Coskewness Estimate*

Description

Extract the coskewness matrix estimate from a statistical factor model

Usage

```
extractCoskewness(model, ...)
```

Arguments

model	statistical factor model estimated via statistical.factor.model
...	not currently used

Value

coskewness matrix estimate

Author(s)

Ross Bennett

See Also

statistical.factor.model

extractCovariance *Covariance Estimate*

Description

Extract the covariance matrix estimate from a statistical factor model

Usage

```
extractCovariance(model, ...)
```

Arguments

model	statistical factor model estimated via statistical.factor.model
...	not currently used

Value

covariance matrix estimate

Author(s)

Ross Bennett

See Also

statistical.factor.model

---

extractEfficientFrontier

Extract the efficient frontier data points

Description

This function extracts the efficient frontier from an object created by optimize.portfolio.

Usage

```
extractEfficientFrontier(object, match.col = "ES", n.portfolios = 25,
  risk_aversion = NULL)
```

Arguments

object	an optimal portfolio object created by optimize.portfolio
match.col	string name of column to use for risk (horizontal axis). match.col must match the name of an objective measure in the objective_measures or opt_values slot in the object created by optimize.portfolio.
n.portfolios	number of portfolios to use to plot the efficient frontier
risk_aversion	vector of risk_aversion values to construct the efficient frontier. n.portfolios is ignored if risk_aversion is specified and the number of points along the efficient frontier is equal to the length of risk_aversion.

Details

If the object is an optimize.portfolio.ROI object and match.col is "ES", "ETL", or "CVaR", then the mean-ETL efficient frontier will be created via meanetl.efficient.frontier.

If the object is an optimize.portfolio.ROI object and match.col is "StdDev", then the mean-StdDev efficient frontier will be created via meanvar.efficient.frontier. Note that if 'var' is specified as the name of an objective, the value returned will be 'StdDev'.

For objects created by optimize.portfolo with the DEoptim, random, or pso solvers, the efficient frontier will be extracted from the object via extract.efficient.frontier. This means that optimize.portfolio must be run with trace=TRUE.

Value

an efficient.frontier object with weights and other metrics along the efficient frontier

Author(s)

Ross Bennett

`extractGroups` *Extract the group and/or category weights*

Description

This function extracts the weights by group and/or category from an object of class `optimize.portfolio`. Group constraints or category_labels must be specified for this to return group constraints.

Usage

```
extractGroups(object, ...)
```

Arguments

`object`	object of class `optimize.portfolio`
`...`	passthrough parameters. Not currently used

Value

a list with two elements

- weights: Optimal set of weights from the `optimize.portfolio` object
- category_weights: Weights by category if category_labels are supplied in the `portfolio` object
- group_weights: Weights by group if group is a constraint type

Author(s)

Ross Bennett

`extractObjectiveMeasures` *Extract the objective measures*

Description

This function will extract the objective measures from the optimal portfolio run via `optimize.portfolio`

Usage

```
extractObjectiveMeasures(object)
```

Arguments

object	list returned by optimize.portfolio

Value

list of objective measures

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

extractStats *extract some stats and weights from a portfolio run via optimize.portfolio*

Description

This function will dispatch to the appropriate class handler based on the input class of the optimize.portfolio output object.

Usage

```
extractStats(object, prefix = NULL, ...)
```

Arguments

object	list returned by optimize.portfolio
prefix	prefix to add to output row names
...	any other passthru parameters

Details

For optimize.portfolio objects:

In general, extractStats will extract the values objective measures and weights at each iteration of a set of weights. This is the case for the DEoptim, random portfolios, and pso solvers that return trace information. Note that trace=TRUE must be specified in optimize.portfolio to return the trace information.

For optimize.portfolio.pso objects, this function will extract the weights (swarm positions) from the PSO output and the out values (swarm fitness values) for each iteration of the optimization. This function can be slow because we need to run constrained_objective to calculate the objective measures on the transformed weights.

For optimize.portfolio.rebalancing objects:

The extractStats function will return a list of the objective measures and weights at each rebalance date for optimize.portfolio.rebalancing objects. The objective measures and weights of each iteration or permutation will be returned if the optimization was done with DEoptim, random portfolios, or pso. This could potentially result in a very large list object where each list element has thousands of rows of at each rebalance period.

The output from the GenSA solver does not store weights evaluated at each iteration The GenSA output for trace.mat contains nb.steps, temperature, function.value, and current.minimum

See Also

optimize.portfolio

---

extractWeights *Extract weights from a portfolio run via* optimize.portfolio *or* optimize.portfolio.rebalancing

---

Description

This function will dispatch to the appropriate class handler based on the input class of the optimize.portfolio or optimize.portfolio.rebalancing output object

Usage

```
extractWeights(object, ...)
```

Arguments

object	list returned by optimize.portfolio
...	any other passthru parameters

See Also

optimize.portfolio, optimize.portfolio.rebalancing

---

factor_exposure_constraint *Constructor for factor exposure constraint*

Description

The factor exposure constraint sets upper and lower bounds on exposures to risk factors. This function is called by add.constraint when type="factor_exposure" is specified, see add.constraint

Usage

```
factor_exposure_constraint(type = "factor_exposure", assets, B, lower, upper,
  enabled = TRUE, message = FALSE, ...)
```

Arguments

type	character type of the constraint
assets	named vector of assets specifying initial weights
B	vector or matrix of risk factor exposures
lower	vector of lower bounds of constraints for risk factor exposures
upper	vector of upper bounds of constraints for risk factor exposures
enabled	TRUE/FALSE
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters to specify risk factor exposure constraints

Details

B can be either a vector or matrix of risk factor exposures (i.e. betas). If B is a vector, the length of B must be equal to the number of assets and lower and upper must be scalars. If B is passed in as a vector, it will be converted to a matrix with one column.

If B is a matrix, the number of rows must be equal to the number of assets and the number of columns represent the number of factors. The length of lower and upper must be equal to the number of factors. The B matrix should have column names specifying the factors and row names specifying the assets. Default column names and row names will be assigned if the user passes in a B matrix without column names or row names.

Value

an object of class 'factor_exposure_constraint'

Author(s)

Ross Bennett

See Also

add.constraint

fn_map *mapping function to transform or penalize weights that violate constraints*

Description

The purpose of the mapping function is to transform a weights vector that does not meet all the constraints into a weights vector that does meet the constraints, if one exists, hopefully with a minimum of transformation.

Usage

```
fn_map(weights, portfolio, relax = FALSE, verbose = FALSE, ...)
```

Arguments

weights	vector of weights
portfolio	object of class portfolio
relax	TRUE/FALSE, default FALSE. Enable constraints to be relaxed.
verbose	print error messages for debuggin purposes
...	any other passthru parameters

Details

The first step is to test for violation of the constraint. If the constraint is violated, we will apply a transformation such that the weights vector satisfies the constraints. The following constraint types are tested in the mapping function: leverage, box, group, and position limit. The transformation logic is based on code from the random portfolio sample method.

If relax=TRUE, we will attempt to relax the constraints if a feasible portfolio could not be formed with an initial call to rp_transform. We will attempt to relax the constraints up to 5 times. If we do not have a feasible portfolio after attempting to relax the constraints, then we will default to returning the weights vector that violates the constraints.

Value

- weights: vector of transformed weights meeting constraints.
- min: vector of min box constraints that may have been modified if relax=TRUE.
- max: vector of max box constraints that may have been modified if relax=TRUE.
- cLO: vector of lower bound group constraints that may have been modified if relax=TRUE.
- cUP: vector of upper bound group constraints that may have been modified if relax=TRUE.

Author(s)

Ross Bennett

`generatesequence` *create a sequence of possible weights for random or brute force portfolios*

Description

This function creates the sequence of min<->max weights for use by random or brute force optimization engines.

Usage

```
generatesequence(min = 0.01, max = 1, by = min/max, rounding = 3)
```

Arguments

`min`	minimum value of the sequence
`max`	maximum value of the sequence
`by`	number to increment the sequence by
`rounding`	integrer how many decimals should we round to

Details

The sequence created is not constrained by asset.

Author(s)

Peter Carl, Brian G. Peterson

See Also

`constraint`, `objective`

`get_constraints` *Helper function to get the enabled constraints out of the portfolio object When the v1_constraint object is instantiated via constraint, the arguments min_sum, max_sum, min, and max are either specified by the user or default values are assigned. These are required by other functions such as* `optimize.portfolio` *and* `constrained_objective` *. This function will check that these variables are in the portfolio object in the constraints list. We will default to* `min_sum=1` *and* `max_sum=1` *if leverage constraints are not specified. We will default to* `min=-Inf` *and* `max=Inf` *if box constraints are not specified. This function is used at the beginning of optimize.portfolio and other functions to extract the constraints from the portfolio object. We Use the same naming as the v1_constraint object.*

Description

Helper function to get the enabled constraints out of the portfolio object

When the v1_constraint object is instantiated via constraint, the arguments min_sum, max_sum, min, and max are either specified by the user or default values are assigned. These are required by other functions such as `optimize.portfolio` and `constrained_objective` . This function will check that these variables are in the portfolio object in the constraints list. We will default to `min_sum=1` and `max_sum=1` if leverage constraints are not specified. We will default to `min=-Inf` and `max=Inf` if box constraints are not specified. This function is used at the beginning of optimize.portfolio and other functions to extract the constraints from the portfolio object. We Use the same naming as the v1_constraint object.

Usage

```
get_constraints(portfolio)
```

Arguments

`portfolio`	an object of class 'portfolio'

Value

an object of class 'constraint' which is a flattened list of enabled constraints

Author(s)

Ross Bennett

See Also

`portfolio.spec`

---

gmv_opt *GMV/QU QP Optimization*

Description

This function is called by optimize.portfolio to solve minimum variance or maximum quadratic utility problems

Usage

```
gmv_opt(R, constraints, moments, lambda, target, lambda_hhi, conc_groups,
  solver = "quadprog", control = NULL)
```

Arguments

R	xts object of asset returns
constraints	object of constraints in the portfolio object extracted with get_constraints
moments	object of moments computed based on objective functions
lambda	risk_aversion parameter
target	target return value
lambda_hhi	concentration aversion parameter
conc_groups	list of vectors specifying the groups of the assets.
solver	solver to use
control	list of solver control parameters

Author(s)

Ross Bennett

---

gmv_opt_leverage *GMV/QU QP Optimization with Turnover Constraint*

Description

This function is called by optimize.portfolio to solve minimum variance or maximum quadratic utility problems with a leverage constraint

Usage

```
gmv_opt_leverage(R, constraints, moments, lambda, target, solver = "quadprog",
  control = NULL)
```

Arguments

R	xts object of asset returns
constraints	object of constraints in the portfolio object extracted with get_constraints
moments	object of moments computed based on objective functions
lambda	risk_aversion parameter
target	target return value
solver	solver to use
control	list of solver control parameters

Author(s)

Ross Bennett

gmv_opt_ptc *GMV/QU QP Optimization with Proportional Transaction Cost Constraint*

Description

This function is called by optimize.portfolio to solve minimum variance or maximum quadratic utility problems with proportional transaction cost constraint

Usage

```
gmv_opt_ptc(R, constraints, moments, lambda, target, init_weights,
  solver = "quadprog", control = NULL)
```

Arguments

R	xts object of asset returns
constraints	object of constraints in the portfolio object extracted with get_constraints
moments	object of moments computed based on objective functions
lambda	risk_aversion parameter
target	target return value
init_weights	initial weights to compute turnover
solver	solver to use
control	list of solver control parameters

Author(s)

Ross Bennett

gmv_opt_toc *GMV/QU QP Optimization with Turnover Constraint*

Description

This function is called by optimize.portfolio to solve minimum variance or maximum quadratic utility problems with turnover constraint

Usage

```
gmv_opt_toc(R, constraints, moments, lambda, target, init_weights,
  solver = "quadprog", control = NULL)
```

Arguments

R	xts object of asset returns
constraints	object of constraints in the portfolio object extracted with get_constraints
moments	object of moments computed based on objective functions
lambda	risk_aversion parameter
target	target return value
init_weights	initial weights to compute turnover
solver	solver to use
control	list of solver control parameters

Author(s)

Ross Bennett

---

group_constraint *constructor for group_constraint*

Description

Group constraints specify the grouping of the assets, weights of the groups, and number of postions (i.e. non-zero weights) iof the groups. This function is called by add.constraint when type="group" is specified. see add.constraint

Usage

```
group_constraint(type = "group", assets, groups, group_labels = NULL,
  group_min, group_max, group_pos = NULL, enabled = TRUE, message = FALSE,
  ...)
```

Arguments

type	character type of the constraint
assets	number of assets, or optionally a named vector of assets specifying initial weights
groups	list of vectors specifying the groups of the assets
group_labels	character vector to label the groups (e.g. size, asset class, style, etc.)
group_min	numeric or vector specifying minimum weight group constraints
group_max	numeric or vector specifying minimum weight group constraints
group_pos	vector specifying the number of non-zero weights per group
enabled	TRUE/FALSE
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters to specify group constraints

Value

an object of class 'group_constraint'

Author(s)

Ross Bennett

See Also

add.constraint

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

# Assets 1 and 3 are groupA
# Assets 2 and 4 are groupB
pspec <- add.constraint(portfolio=pspec,
                        type="group",
                        groups=list(groupA=c(1, 3),
                                    groupB=c(2, 4)),
                        group_min=c(0.15, 0.25),
                        group_max=c(0.65, 0.55))

# 2 levels of grouping (e.g. by sector and geography)
pspec <- portfolio.spec(assets=5)
# Assets 1, 3, and 5 are Tech
# Assets 2 and 4 are Oil
# Assets 2, 4, and 5 are UK
# Assets 1 and are are US
group_list <- list(group1=c(1, 3, 5),
                   group2=c(2, 4),
                   groupA=c(2, 4, 5),
                   groupB=c(1, 3))

pspec <- add.constraint(portfolio=pspec,
                        type="group",
                        groups=group_list,
                        group_min=c(0.15, 0.25, 0.2, 0.1),
                        group_max=c(0.65, 0.55, 0.5, 0.4))
```

---

group_fail *Test if group constraints have been violated*

Description

The function loops through each group and tests if cLO or cUP have been violated for the given group. This is a helper function for rp_transform.

Usage

```
group_fail(weights, groups, cLO, cUP, group_pos = NULL)
```

Arguments

weights	weights vector to test
groups	list of vectors specifying the groups of the assets
cLO	numeric or vector specifying minimum weight group constraints
cUP	numeric or vector specifying minimum weight group constraints
group_pos	vector specifying the number of non-zero weights per group

Value

logical vector: TRUE if group constraints are violated for a given group

Author(s)

Ross Bennett

HHI *Concentration of weights*

Description

This function computes the concentration of weights using the Herfindahl Hirschman Index

Usage

```
HHI(weights, groups = NULL)
```

Arguments

weights	set of portfolio weights
groups	list of vectors of grouping

Author(s)

Ross Bennett

indexes *Six Major Economic Indexes*

Description

Monthly data of five indexes beginning on 2000-01-31 and ending 2009-12-31. The indexes are: US Bonds, US Equities, International Equities, Commodities, US T-Bills, and Inflation

Usage

```
data(indexes)
```

Format

CSV converted into xts object with montly observations

Examples

```
data(indexes)

#preview the data
head(indexes)

#summary period statistics
summary(indexes)
```

insert_constraints *Insert a list of constraints into the constraints slot of a portfolio object*

Description

This is a helper function primarily for backwards compatibility to insert constraints from a 'v1_constraint' object into the v2 'portfolio' object.

Usage

```
insert_constraints(portfolio, constraints)
```

Arguments

portfolio	object of class 'portfolio'
constraints	list of constraint objects

Author(s)

Ross Bennett

insert_objectives *Insert a list of objectives into the objectives slot of a portfolio object*

Description

This is a helper function primarily for backwards compatibility to insert objectives from a 'v1_constraint' object into the v2 'portfolio' object.

Usage

```
insert_objectives(portfolio, objectives)
```

Arguments

portfolio	object of class 'portfolio'
objectives	list of objective objects

Author(s)

Ross Bennett

inverse.volatility.weight *Create an inverse volatility weighted portfolio*

Description

This function calculates objective measures for an equal weight portfolio.

Usage

```
inverse.volatility.weight(R, portfolio, ...)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
portfolio	an object of type "portfolio" specifying the constraints and objectives for the optimization
...	any other passthru parameters to constrained_objective

Details

This function is simply a wrapper around constrained_objective to calculate the objective measures in the given portfolio object of an inverse volatility weight portfolio. The portfolio object should include all objectives to be calculated.

Value

a list containing the returns, weights, objective measures, call, and portfolio object

Author(s)

Peter Carl

---

is.constraint *check function for constraints*

Description

check function for constraints

Usage

```
is.constraint(x)
```

Arguments

x	object to test for type constraint

Author(s)

Brian G. Peterson

---

is.objective *check class of an objective object*

Description

check class of an objective object

Usage

```
is.objective(x)
```

Arguments

x	an object potentially of type 'objective' to test

Author(s)

Brian G. Peterson

is.portfolio *check function for portfolio*

Description

check function for portfolio

Usage

```
is.portfolio(x)
```

Arguments

x	object to test for type portfolio

Author(s)

Ross Bennett

leverage_exposure_constraint *constructor for leverage_exposure_constraint*

Description

The leverage_exposure constraint specifies a maximum leverage where leverage is defined as the sum of the absolute value of the weights. Leverage exposure is computed as the sum of the absolute value of the weights, sum(abs(weights)).

Usage

```
leverage_exposure_constraint(type = "leverage_exposure", leverage = NULL,
  enabled = TRUE, message = FALSE, ...)
```

Arguments

type	character type of the constraint
leverage	maximum leverage value
enabled	TRUE/FALSE
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters to specify diversification constraint an object of class 'diversification_constraint'

Details

This should be used for constructing, for example, 130/30 portfolios or dollar neutral portfolios with 2:1 leverage. For the ROI solvers, this is implemented as a MILP problem and is not supported for problems formulated as a quadratic programming problem. This may change in the future if a MIQP solver is added.

This function is called by add.constraint when type="leverage_exposure" is specified, see `add.constraint`.

Author(s)

Ross Bennett

See Also

`add.constraint`

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

pspec <- add.constraint(portfolio=pspec, type="leverage_exposure", leverage=1.6)
```

---

maxret_milp_opt *Maximum Return MILP Optimization*

Description

This function is called by optimize.portfolio to solve maximum return problems via mixed integer linear programming.

Usage

```
maxret_milp_opt(R, constraints, moments, target, solver = "glpk",
  control = NULL)
```

Arguments

R	xts object of asset returns
constraints	object of constraints in the portfolio object extracted with get_constraints
moments	object of moments computed based on objective functions
target	target return value
solver	solver to use
control	list of solver control parameters

Author(s)

Ross Bennett

`maxret_opt` *Maximum Return LP Optimization*

Description

This function is called by optimize.portfolio to solve maximum return

Usage

```
maxret_opt(R, moments, constraints, target, solver = "glpk", control = NULL)
```

Arguments

`R`	xts object of asset returns
`moments`	object of moments computed based on objective functions
`constraints`	object of constraints in the portfolio object extracted with `get_constraints`
`target`	target return value
`solver`	solver to use
`control`	list of solver control parameters

Author(s)

Ross Bennett

`meanetl.efficient.frontier`
Generate the efficient frontier for a mean-etl portfolio

Description

This function generates the mean-ETL efficient frontier of a portfolio specifying the constraints and objectives. The `portfolio` object should have two objectives: 1) mean and 2) ES (or ETL or cVaR). If the portfolio object does not contain these objectives, they will be added using default parameters.

Usage

```
meanetl.efficient.frontier(portfolio, R, n.portfolios = 25, ...)
```

Arguments

portfolio	a portfolio object with constraints and objectives created via portfolio.spec
R	an xts or matrix of asset returns
n.portfolios	number of portfolios to generate the efficient frontier
...	passthru parameters to optimize.portfolio

Value

a matrix of objective measure values and weights along the efficient frontier

Author(s)

Ross Bennett

---

meanvar.efficient.frontier

Generate the efficient frontier for a mean-variance portfolio

---

Description

This function generates the mean-variance efficient frontier of a portfolio specifying the constraints and objectives. The portfolio object should have two objectives: 1) mean and 2) var (or sd or StdDev). If the portfolio object does not contain these objectives, they will be added using default parameters.

Usage

```
meanvar.efficient.frontier(portfolio, R, n.portfolios = 25,
  risk_aversion = NULL, ...)
```

Arguments

portfolio	a portfolio object with constraints created via portfolio.spec
R	an xts or matrix of asset returns
n.portfolios	number of portfolios to plot along the efficient frontier
risk_aversion	vector of risk_aversion values to construct the efficient frontier. n.portfolios is ignored if risk_aversion is specified and the number of points along the efficient frontier is equal to the length of risk_aversion.
...	passthru parameters to optimize.portfolio

Value

a matrix of objective measure values and weights along the efficient frontier

Author(s)

Ross Bennett

meucci.moments *Compute moments*

Description

Compute the first and second moments using the Fully Flexible Views framework as described in A. Meucci - "Fully Flexible Views: Theory and Practice".

Usage

```
meucci.moments(R, posterior_p)
```

Arguments

R	xts object of asset returns
posterior_p	vector of posterior probabilities

Value

a list with the first and second moments

- mu: vector of expected returns
- sigma: covariance matrix

Author(s)

Ross Bennett

References

A. Meucci - "Fully Flexible Views: Theory and Practice".

meucci.ranking *Asset Ranking*

Description

Express views on the relative expected asset returns as in A. Meucci, "Fully Flexible Views: Theory and Practice" and compute the first and second moments.

Usage

```
meucci.ranking(R, p, order)
```

Arguments

R	xts object of asset returns
p	a vector of the prior probability values
order	a vector of indexes of the relative ranking of expected asset returns in ascending order. For example, `order = c(2, 3, 1, 4)` means that the expected returns of `R[,2] < R[,3], < R[,1] < R[,4]`.

Value

The estimated moments based on ranking views

Note

This function is based on the `ViewRanking` function written by Ram Ahluwalia in the Meucci package.

References

A. Meucci, "Fully Flexible Views: Theory and Practice" http://www.symmys.com/node/158 See Meucci script for "RankingInformation/ViewRanking.m"

See Also

meucci.moments

Examples

```
data(edhec)
R <- edhec[,1:4]
p <- rep(1 / nrow(R), nrow(R))
meucci.ranking(R, p, c(2, 3, 1, 4))
```

---

minmax_objective *constructor for class tmp_minmax_objective*

Description

This objective allows for min and max targets to be specified.

Usage

```
minmax_objective(name, target = NULL, arguments = NULL, multiplier = 1,
  enabled = TRUE, ..., min, max)
```

Arguments

name	name of the objective, should correspond to a function, though we will try to make allowances
target	univariate target for the objective
arguments	default arguments to be passed to an objective function when executed
multiplier	multiplier to apply to the objective, usually 1 or -1
enabled	TRUE/FALSE
...	any other passthru parameters
min	minimum value
max	maximum value

Details

If target is set, we'll try to meet the metric

If target is NULL and min and max are specified, then do the following:

If max is violated to the upside, penalize the metric. If min is violated to the downside, penalize the metric. The purpose of this objective is to try to meet the range between min and max

Value

object of class 'minmax_objective'

Author(s)

Ross Bennett

---

mult.portfolio.spec *Multple Layer Portfolio Specification*

Description

Create and specify a multiple layer portfolio

Usage

```
mult.portfolio.spec(portfolio, levels = 2, ...)
```

Arguments

portfolio	the "top level" portfolio
levels	number of levels of sub-portfolios
...	any additional parameters

Details

The sub.portfolios slot is a list where each element contains the portfolio object and rebalancing parameters for the optimization of the sub portfolio. This allows, for example, each sub portfolio to have different rebalancing frequencies (i.e. monthly or quarterly), optimization methods, etc.

Each sub portfolio is optimized with optimize.portfolio.rebalancing to create a time series of proxy returns.

The "top level" portfolio is used to specify the constraints and objectives to control the optimization given the proxy returns of each sub portfolio.

Value

a mult.portfolio.spec object with the top level portfolio and sub portfolios with optimization parameters for each sub portfolio

Author(s)

Ross Bennett

---

name.replace *utility function to replace awkward named from unlist*

Description

utility function to replace awkward named from unlist

Usage

```
name.replace(rnames)
```

Arguments

rnames	character vector of names to check for cleanup

---

objective *constructor for class 'objective'*

Description

Typically called as a sub-function by the user function add.objective. See main documentation there.

Usage

```
objective(name, target = NULL, arguments, enabled = TRUE, ...,
  multiplier = 1, objclass = "objective")
```

Arguments

name	name of the objective which will be used to call a function, like 'ES', 'VaR', 'mean'
target	univariate target for the objective, default NULL
arguments	default arguments to be passed to an objective function when executed
enabled	TRUE/FALSE
...	any other passthrough parameters
multiplier	multiplier to apply to the objective, usually 1 or -1
objclass	string class to apply, default 'objective'

Author(s)

Brian G. Peterson

See Also

add.objective, portfolio.spec

---

optimize.portfolio *Constrained optimization of portfolios*

Description

This function aims to provide a wrapper for constrained optimization of portfolios that specify constraints and objectives.

Usage

```
optimize.portfolio_v1(R, constraints, optimize_method = c("DEoptim", "random",
  "ROI", "ROI_old", "pso", "GenSA"), search_size = 20000, trace = FALSE,
  ..., rp = NULL, momentFUN = "set.portfolio.moments_v1")

optimize.portfolio(R, portfolio = NULL, constraints = NULL,
  objectives = NULL, optimize_method = c("DEoptim", "random", "ROI", "pso",
  "GenSA"), search_size = 20000, trace = FALSE, ..., rp = NULL,
  momentFUN = "set.portfolio.moments", message = FALSE)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
constraints	default=NULL, a list of constraint objects. An object of class 'v1_constraint' can be passed in here.
optimize_method	one of "DEoptim", "random", "ROI", "pso", "GenSA". A solver for ROI can also be specified and will be solved using ROI. See Details.

search_size	integer, how many portfolios to test, default 20,000
trace	TRUE/FALSE if TRUE will attempt to return additional information on the path or portfolios searched
...	any other passthru parameters
rp	matrix of random portfolio weights, default NULL, mostly for automated use by rebalancing optimization or repeated tests on same portfolios
momentFUN	the name of a function to call to set portfolio moments, default set.portfolio.moments_v2
portfolio	an object of type "portfolio" specifying the constraints and objectives for the optimization
objectives	default=NULL, a list of objective objects.
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.

Details

This function currently supports DEoptim, random portfolios, pso, GenSA, and ROI as back ends. Additional back end contributions for Rmetrics, ghyp, etc. would be welcome.

When using random portfolios, search_size is precisely that, how many portfolios to test. You need to make sure to set your feasible weights in generatesequence to make sure you have search_size unique portfolios to test, typically by manipulating the 'by' parameter to select something smaller than .01 (I often use .002, as .001 seems like overkill)

When using DE, search_size is decomposed into two other parameters which it interacts with, NP and itermax.

NP, the number of members in each population, is set to cap at 2000 in DEoptim, and by default is the number of parameters (assets/weights) * 10.

itermax, if not passed in dots, defaults to the number of parameters (assets/weights) * 50.

When using GenSA and want to set verbose=TRUE, instead use trace.

If optimize_method="ROI" is specified, a default solver will be selected based on the optimization problem. The glpk solver is the default solver for LP and MILP optimization problems. The quadprog solver is the default solver for QP optimization problems. For example, optimize_method = "quadprog" can be specified and the optimization problem will be solved via ROI using the quadprog solver.

The extension to ROI solves a limited type of convex optimization problems:

- Maxmimize portfolio return subject leverage, box, group, position limit, target mean return, and/or factor exposure constraints on weights.
- Minimize portfolio variance subject to leverage, box, group, turnover, and/or factor exposure constraints (otherwise known as global minimum variance portfolio).
- Minimize portfolio variance subject to leverage, box, group, and/or factor exposure constraints and a desired portfolio return.
- Maximize quadratic utility subject to leverage, box, group, target mean return, turnover, and/or factor exposure constraints and risk aversion parameter. (The risk aversion parameter is passed into optimize.portfolio as an added argument to the portfolio object).
- Maximize portfolio mean return per unit standard deviation (i.e. the Sharpe Ratio) can be done by specifying maxSR=TRUE in optimize.portfolio. If both mean and StdDev are specified as objective names, the default action is to maximize quadratic utility, therefore maxSR=TRUE must be specified to maximize Sharpe Ratio.

- Minimize portfolio ES/ETL/CVaR optimization subject to leverage, box, group, position limit, target mean return, and/or factor exposure constraints and target portfolio return.
- Maximize portfolio mean return per unit ES/ETL/CVaR (i.e. the STARR Ratio) can be done by specifying `maxSTARR=TRUE` in `optimize.portfolio`. If both mean and ES/ETL/CVaR are specified as objective names, the default action is to maximize mean return per unit ES/ETL/CVaR.

These problems also support a weight_concentration objective where concentration of weights as measured by HHI is added as a penalty term to the quadratic objective.

Because these convex optimization problem are standardized, there is no need for a penalty term. The `multiplier` argument in `add.objective` passed into the complete constraint object are ingnored by the ROI solver.

Value

a list containing the following elements

- `weights`: The optimal set weights.
- `objective_measures`: A list containing the value of each objective corresponding to the optimal weights.
- `opt_values`: A list containing the value of each objective corresponding to the optimal weights.
- `out`: The output of the solver.
- `call`: The function call.
- `portfolio`: The portfolio object.
- `R`: The asset returns.
- `data summary`: The first row and last row of `R`.
- `elapsed_time`: The amount of time that elapses while the optimization is run.
- `end_t`: The date and time the optimization completed.

When Trace=TRUE is specified, the following elements will be returned in addition to the elements above. The output depends on the optimization method and is specific to each solver. Refer to the documentation of the desired solver for more information.

`optimize_method="random"`

- `random_portfolios`: A matrix of the random portfolios.
- `random_portfolio_objective_results`: A list of the following elements for each random portfolio.
  - `out`: The output value of the solver corresponding to the random portfolio weights.
  - `weights`: The weights of the random portfolio.
  - `objective_measures`: A list of each objective measure corresponding to the random portfolio weights.

`optimize_method="DEoptim"`

- `DEoutput`: A list (of length 2) containing the following elements:

  - optim
  - member
- DEoptim_objective_results: A list containing the following elements for each intermediate population.
  - out: The output of the solver.
  - weights: Population weights.
  - init_weights: Initial population weights.
  - objective_measures: A list of each objective measure corresponding to the weights

optimize_method="pso"

- PSOoutput: A list containing the following elements:
  - par
  - value
  - counts
  - convergence
  - message
  - stats

optimize_method="GenSA"

- GenSAoutput: A list containing the following elements:
  - value
  - par
  - trace.mat
  - counts

Note

An object of class v1_constraint can be passed in for the constraints argument. The v1_constraint object was used in the previous 'v1' specification to specify the constraints and objectives for the optimization problem, see constraint. We will attempt to detect if the object passed into the constraints argument is a v1_constraint object and update to the 'v2' specification by adding the constraints and objectives to the portfolio object.

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson, Ross Bennett

See Also

portfolio.spec

`optimize.portfolio.parallel` *Execute multiple optimize.portfolio calls, presumably in parallel*

Description

This function will not speed up optimization!

Usage

```
optimize.portfolio.parallel(R, portfolio, optimize_method = c("DEoptim",
  "random", "ROI", "pso", "GenSA"), search_size = 20000, trace = FALSE, ...,
  rp = NULL, momentFUN = "set.portfolio.moments", message = FALSE,
  nodes = 4)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
portfolio	an object of type "portfolio" specifying the constraints and objectives for the optimization
optimize_method	one of "DEoptim", "random", "pso", "GenSA".
search_size	integer, how many portfolios to test, default 20,000
trace	TRUE/FALSE if TRUE will attempt to return additional information on the path or portfolios searched
...	any other passthru parameters
rp	matrix of random portfolio weights, default NULL, mostly for automated use by rebalancing optimization or repeated tests on same portfolios
momentFUN	the name of a function to call to set portfolio moments, default set.portfolio.moments_v2
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
nodes	how many processes to run in the foreach loop, default 4

Details

This function exists to run multiple copies of optimize.portfolio, presumabley in parallel using foreach.

This is typically done to test your parameter settings, specifically total population size, but also possibly to help tune your convergence settings, number of generations, stopping criteria, etc.

If you want to use all the cores on your multi-core computer, use the parallel version of the apppropriate optimization engine, not this function.

Value

a list containing the optimal weights, some summary statistics, the function call, and optionally trace information

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson

`optimize.portfolio.rebalancing`
Portfolio Optimization with Rebalancing Periods

Description

Portfolio optimization with support for rebalancing periods for out-of-sample testing (i.e. backtesting)

Usage

```
optimize.portfolio.rebalancing_v1(R, constraints,
  optimize_method = c("DEoptim", "random", "ROI"), search_size = 20000,
  trace = FALSE, ..., rp = NULL, rebalance_on = NULL,
  training_period = NULL, rolling_window = NULL)

optimize.portfolio.rebalancing(R, portfolio = NULL, constraints = NULL,
  objectives = NULL, optimize_method = c("DEoptim", "random", "ROI"),
  search_size = 20000, trace = FALSE, ..., rp = NULL,
  rebalance_on = NULL, training_period = NULL, rolling_window = NULL)
```

Arguments

`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`constraints`	default NULL, a list of constraint objects
`optimize_method`	one of "DEoptim", "random", "pso", "GenSA", or "ROI"
`search_size`	integer, how many portfolios to test, default 20,000
`trace`	TRUE/FALSE if TRUE will attempt to return additional information on the path or portfolios searched
`...`	any other passthru parameters to `optimize.portfolio`
`rp`	a set of random portfolios passed into the function to prevent recalculation
`rebalance_on`	character string of period to rebalance on. See `endpoints` for valid names.
`training_period`	an integer of the number of periods to use as a training data in the front of the returns data
`rolling_window`	an integer of the width (i.e. number of periods) of the rolling window, the default of NULL will run the optimization using the data from inception.
`portfolio`	an object of type "portfolio" specifying the constraints and objectives for the optimization
`objectives`	default NULL, a list of objective objects

Details

Run portfolio optimization with periodic rebalancing at specified time periods. Running the portfolio optimization with periodic rebalancing can help refine the constraints and objectives by evaluating the out of sample performance of the portfolio based on historical data.

If both `training_period` and `rolling_window` are `NULL`, then `training_period` is set to a default value of 36.

If `training_period` is `NULL` and a `rolling_window` is specified, then `training_period` is set to the value of `rolling_window`.

The user should be aware of the following behavior when both `training_period` and `rolling_window` are specified and have different values

- `training_period < rolling_window`: For example, if you have `rolling_window=60`, `training_period=50`, and the periodicity of the data is the same as the rebalance frequency (i.e. monthly data with `rebalance_on="months"`) then the returns data used in the optimization at each iteration are as follows:
  - 1: R[1:50,]
  - 2: R[1:51,]
  - ...
  - 11: R[1:60,]
  - 12: R[1:61,]
  - 13: R[2:62,]
  - ...

  This results in a growing window for several optimizations initially while the endpoint iterator (i.e. `[50, 51, ...]`) is less than the rolling window width.
- `training_period > rolling_window`: The data used in the initial optimization is `R[(training_period - rolling_`
  This results in some of the data being "thrown away", i.e. periods 1 to `(training_period - rolling_window - 1)` are not used in the optimization.

This function is a essentially a wrapper around `optimize.portfolio` and thus the discussion in the Details section of the `optimize.portfolio` help file is valid here as well.

This function is massively parallel and requires the 'foreach' package. It is suggested to register a parallel backend.

Value

a list containing the following elements

- `portfolio`: The portfolio object.
- `R`: The asset returns.
- `call`: The function call.
- `elapsed_time`: The amount of time that elapses while the optimization is run.
- `opt_rebalancing`: A list of `optimize.portfolio` objects computed at each rebalancing period.

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson

See Also

portfolio.spec optimize.portfolio

Examples

```
## Not run:
data(edhec)
R <- edhec[,1:4]
funds <- colnames(R)

portf <- portfolio.spec(funds)
portf <- add.constraint(portf, type="full_investment")
portf <- add.constraint(portf, type="long_only")
portf <- add.objective(portf, type="risk", name="StdDev")

# Quarterly rebalancing with 5 year training period
bt.opt1 <- optimize.portfolio.rebalancing(R, portf,
optimize_method="ROI",
rebalance_on="quarters",
training_period=60)

# Monthly rebalancing with 5 year training period and 4 year rolling window
bt.opt2 <- optimize.portfolio.rebalancing(R, portf,
optimize_method="ROI",
rebalance_on="months",
training_period=60,
rolling_window=48)

## End(Not run)
```

---

pHist *Generates histogram*

Description

Generates histogram

Usage

```
pHist(X, p, nBins, freq = FALSE)
```

Arguments

X	a vector containing the data points
p	a vector containing the probabilities for each of the data points in X
nBins	expected number of Bins the data set is to be broken down into
freq	a boolean variable to indicate whether the graphic is a representation of frequencies

Value

a list with f the frequency for each midpoint x the midpoints of the nBins intervals

Author(s)

Ram Ahluwalia <ram@wingedfootcapital.com> and Xavier Valls <flamejat@gmail.com>

References

http://www.symmys.com See Meucci script pHist.m used for plotting

---

plot.optimize.portfolio.DEoptim

plot method for objects of class optimize.portfolio

Description

Scatter and weights chart for portfolio optimizations run with trace=TRUE

Usage

```
## S3 method for class 'optimize.portfolio.DEoptim'
plot(x, ..., return.col = "mean",
  risk.col = "ES", chart.assets = FALSE, neighbors = NULL,
  main = "optimized portfolio plot", xlim = NULL, ylim = NULL)

## S3 method for class 'optimize.portfolio.GenSA'
plot(x, ..., rp = FALSE,
  return.col = "mean", risk.col = "ES", chart.assets = FALSE,
  cex.axis = 0.8, element.color = "darkgray", neighbors = NULL,
  main = "GenSA.Portfolios", xlim = NULL, ylim = NULL)

## S3 method for class 'optimize.portfolio.pso'
plot(x, ..., return.col = "mean",
  risk.col = "ES", chart.assets = FALSE, cex.axis = 0.8,
  element.color = "darkgray", neighbors = NULL, main = "PSO.Portfolios",
  xlim = NULL, ylim = NULL)
```

```
## S3 method for class 'optimize.portfolio.ROI'
plot(x, ..., rp = FALSE, risk.col = "ES",
  return.col = "mean", chart.assets = FALSE, element.color = "darkgray",
  neighbors = NULL, main = "ROI.Portfolios", xlim = NULL, ylim = NULL)

## S3 method for class 'optimize.portfolio.random'
plot(x, ..., return.col = "mean",
  risk.col = "ES", chart.assets = FALSE, neighbors = NULL, xlim = NULL,
  ylim = NULL, main = "optimized portfolio plot")

## S3 method for class 'optimize.portfolio'
plot(x, ..., return.col = "mean",
  risk.col = "ES", chart.assets = FALSE, neighbors = NULL, xlim = NULL,
  ylim = NULL, main = "optimized portfolio plot")
```

Arguments

x	set of portfolios created by optimize.portfolio
...	any other passthru parameters
return.col	string name of column to use for returns (vertical axis)
risk.col	string name of column to use for risk (horizontal axis)
chart.assets	TRUE/FALSE to include risk-return scatter of assets
neighbors	set of 'neighbor portfolios to overplot
main	an overall title for the plot: see title
xlim	set the limit on coordinates for the x-axis
ylim	set the limit on coordinates for the y-axis
rp	TRUE/FALSE to plot feasible portfolios generated by random_portfolios
cex.axis	the magnification to be used for axis annotation relative to the current setting of cex.
element.color	provides the color for drawing less-important chart elements, such as the box lines, axis lines, etc.

Details

return.col must be the name of a function used to compute the return metric on the random portfolio weights risk.col must be the name of a function used to compute the risk metric on the random portfolio weights

neighbors may be specified in three ways. The first is as a single number of neighbors. This will extract the neighbors closest portfolios in terms of the out numerical statistic. The second method consists of a numeric vector for neighbors. This will extract the neighbors with portfolio index numbers that correspond to the vector contents. The third method for specifying neighbors is to pass in a matrix. This matrix should look like the output of extractStats, and should contain risk.col,return.col, and weights columns all properly named.

The ROI and GenSA solvers do not store the portfolio weights like DEoptim or random portfolios, random portfolios can be generated for the scatter plot with the rp argument.

portfolio.moments.bl *Portfolio Moments*

Description

Set portfolio moments for use by lower level optimization functions using a basic Black Litterman model.

Usage

```
portfolio.moments.bl(R, portfolio, momentargs = NULL, P, Mu = NULL,
  Sigma = NULL, ...)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
portfolio	an object of type portfolio specifying the constraints and objectives for the optimization, see portfolio.spec
momentargs	list containing arguments to be passed down to lower level functions, default NULL
P	a K x N pick matrix representing views
Mu	vector of length N of the prior expected values. The sample mean is used if Mu=NULL.
Sigma	an N x N matrix of the prior covariance matrix. The sample covariance is used if Sigma=NULL.
...	any other passthru parameters

Note

If any of the objectives in the portfolio object have clean as an argument, the cleaned returns are used to fit the model.

portfolio.moments.boudt *Portfolio Moments*

Description

Set portfolio moments for use by lower level optimization functions using a statistical factor model based on the work of Kris Boudt.

Usage

```
portfolio.moments.boudt(R, portfolio, momentargs = NULL, k = 1, ...)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
portfolio	an object of type portfolio specifying the constraints and objectives for the optimization, see portfolio.spec
momentargs	list containing arguments to be passed down to lower level functions, default NULL
k	number of factors used for fitting statistical factor model
...	any other passthru parameters

Note

If any of the objectives in the portfolio object have clean as an argument, the cleaned returns are used to fit the model.

---

portfolio.spec *constructor for class portfolio*

Description

The portfolio object is created with portfolio.spec. The portfolio object is an S3 object of class 'portfolio' used to hold the initial asset weights, constraints, objectives, and other information about the portfolio. The only required argument to portfolio.spec is assets.

Usage

```
portfolio.spec(assets = NULL, category_labels = NULL, weight_seq = NULL,
  message = FALSE)
```

Arguments

assets	number of assets, or optionally a named vector of assets specifying seed weights. If seed weights are not specified, an equal weight portfolio will be assumed.
category_labels	character vector to categorize assets by sector, industry, geography, market-cap, currency, etc. Default NULL
weight_seq	seed sequence of weights, see generatesequence Default NULL
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.

Details

The portfolio object contains the following elements:

- assets named vector of the seed weights
- category_labels character vector to categorize the assets by sector, geography, etc.
- weight_seq sequence of weights used by random_portfolios. See generatesequence

- constraints a list of constraints added to the portfolio object with add.constraint
- objectives a list of objectives added to the portfolio object with add.objective
- call the call to portfolio.spec with all of the specified arguments

Value

an object of class portfolio

Author(s)

Ross Bennett, Brian G. Peterson

See Also

add.constraint, add.objective, optimize.portfolio

Examples

```
data(edhec)
pspec <- portfolio.spec(assets=colnames(edhec))
pspec <- portfolio.spec(assets=10, weight_seq=generatesequence())
```

---

portfolio_risk_objective

constructor for class portfolio_risk_objective

Description

if target is null, we'll try to minimize the risk metric

Usage

```
portfolio_risk_objective(name, target = NULL, arguments = NULL,
  multiplier = 1, enabled = TRUE, ...)
```

Arguments

name	name of the objective, should correspond to a function, though we will try to make allowances
target	univariate target for the objective
arguments	default arguments to be passed to an objective function when executed
multiplier	multiplier to apply to the objective, usually 1 or -1
enabled	TRUE/FALSE
...	any other passthru parameters

Value

object of class 'portfolio_risk_objective'

Author(s)

Brian G. Peterson

`position_limit_constraint` *constructor for position_limit_constraint*

Description

This function is called by add.constraint when type="position_limit" is specified, `add.constraint` Allows the user to specify the maximum number of positions (i.e. number of assets with non-zero weights) as well as the maximum number of long and short positions.

Usage

```
position_limit_constraint(type = "position_limit", assets, max_pos = NULL,
  max_pos_long = NULL, max_pos_short = NULL, enabled = TRUE,
  message = FALSE, ...)
```

Arguments

`type`	character type of the constraint
`assets`	named vector of assets specifying initial weights
`max_pos`	maximum number of assets with non-zero weights
`max_pos_long`	maximum number of assets with long (i.e. buy) positions
`max_pos_short`	maximum number of assets with short (i.e. sell) positions
`enabled`	TRUE/FALSE
`message`	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
`...`	any other passthru parameters to specify position limit constraints

Value

an object of class 'position_limit_constraint'

Author(s)

Ross Bennett

See Also

`add.constraint`

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

pspec <- add.constraint(portfolio=pspec, type="position_limit", max_pos=3)
pspec <- add.constraint(portfolio=pspec, type="position_limit", max_pos_long=3, max_pos_short=1)
```

---

pos_limit_fail *function to check for violation of position limits constraints*

Description

This is used as a helper function for rp_transform to check for violation of position limit constraints. The position limit constraints checked are max_pos, max_pos_long, and max_pos_short.

Usage

```
pos_limit_fail(weights, max_pos, max_pos_long, max_pos_short)
```

Arguments

weights	vector of weights to test
max_pos	maximum number of assets with non-zero weights
max_pos_long	maximum number of assets with long (i.e. buy) positions
max_pos_short	maximum number of assets with short (i.e. sell) positions

Value

TRUE if any position_limit is violated. FALSE if all position limits are satisfied

---

print.constraint *print method for constraint objects*

Description

print method for constraint objects

Usage

```
## S3 method for class 'constraint'
print(x, ...)
```

Arguments

x	object of class constraint
...	any other passthru parameters

Author(s)

Ross Bennett

---

print.efficient.frontier

Print an efficient frontier object

Description

Print method for efficient frontier objects. Display the call to create or extract the efficient frontier object and the portfolio from which the efficient frontier was created or extracted.

Usage

```
## S3 method for class 'efficient.frontier'
print(x, ...)
```

Arguments

x	objective of class efficient.frontier
...	any other passthru parameters

Author(s)

Ross Bennett

See Also

create.EfficientFrontier

`print.optimize.portfolio.rebalancing` *Printing output of optimize.portfolio.rebalancing*

Description

print method for `optimize.portfolio.rebalancing` objects

Usage

```
## S3 method for class 'optimize.portfolio.rebalancing'
print(x, ..., digits = 4)
```

Arguments

`x`	an object used to select a method
`...`	any other passthru parameters
`digits`	the number of significant digits to use when printing.

Author(s)

Ross Bennett

See Also

`optimize.portfolio.rebalancing`

`print.optimize.portfolio.ROI` *Printing output of optimize.portfolio*

Description

print method for `optimize.portfolio` objects

Usage

```
## S3 method for class 'optimize.portfolio.ROI'
print(x, ..., digits = 4)

## S3 method for class 'optimize.portfolio.random'
print(x, ..., digits = 4)

## S3 method for class 'optimize.portfolio.DEoptim'
print(x, ..., digits = 4)
```

```
## S3 method for class 'optimize.portfolio.GenSA'
print(x, ..., digits = 4)

## S3 method for class 'optimize.portfolio.pso'
print(x, ..., digits = 4)
```

Arguments

x	an object used to select a method
...	any other passthru parameters
digits	the number of significant digits to use when printing.

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

print.portfolio *Printing Portfolio Specification Objects*

Description

Print method for objects of class portfolio created with portfolio.spec

Usage

```
## S3 method for class 'portfolio'
print(x, ...)
```

Arguments

x	an object of class portfolio
...	any other passthru parameters

Author(s)

Ross Bennett

See Also

portfolio.spec

print.summary.optimize.portfolio

Printing summary output of optimize.portfolio

Description

print method for objects of class summary.optimize.portfolio

Usage

```
## S3 method for class 'summary.optimize.portfolio'
print(x, ...)
```

Arguments

x	an object of class summary.optimize.portfolio.
...	any other passthru parameters. Currently not used.

Author(s)

Ross Bennett

See Also

summary.optimize.portfolio

print.summary.optimize.portfolio.rebalancing

Printing summary output of optimize.portfolio.rebalancing

Description

print method for objects of class summary.optimize.portfolio.rebalancing

Usage

```
## S3 method for class 'summary.optimize.portfolio.rebalancing'
print(x, ..., digits = 4)
```

Arguments

x	an object of class summary.optimize.portfolio.rebalancing.
...	any other passthru parameters
digits	number of digits used for printing

Author(s)

Ross Bennett

See Also

summary.optimize.portfolio.rebalancing

---

quadratic_utility_objective

constructor for quadratic utility objective

Description

This function calls return_objective and portfolio_risk_objective to create a list of the objectives to be added to the portfolio.

Usage

```
quadratic_utility_objective(risk_aversion = 1, target = NULL,
  enabled = TRUE)
```

Arguments

risk_aversion	risk_aversion (i.e. lambda) parameter to penalize variance
target	target mean return value
enabled	TRUE/FALSE, default enabled=TRUE

Value

a list of two elements

- return_objective
- portfolio_risk_objective

Author(s)

Ross Bennett

randomize_portfolio *version 2 generate random permutations of a portfolio seed meeting your constraints on the weights of each asset*

Description

version 2 generate random permutations of a portfolio seed meeting your constraints on the weights of each asset

Usage

```
randomize_portfolio(portfolio, max_permutations = 200)
```

Arguments

portfolio	an object of type "portfolio" specifying the constraints for the optimization, see portfolio.spec
max_permutations	integer: maximum number of iterations to try for a valid portfolio, default 200

Value

named weighting vector

Author(s)

Peter Carl, Brian G. Peterson, (based on an idea by Pat Burns)

randomize_portfolio_v1 *Random portfolio sample method*

Description

This function generates random permutations of a portfolio seed meeting leverage and box constraints. The final step is to run fn_map on the random portfolio weights to transform the weights so they satisfy other constraints such as group or position limit constraints. This is the 'sample' method for random portfolios and is based on an idea by Pat Burns.

Usage

```
randomize_portfolio_v1(rpconstraints, max_permutations = 200, rounding = 3)
```

Arguments

rpconstraints	an object of type "constraints" specifying the constraints for the optimization, see constraint
max_permutations	integer: maximum number of iterations to try for a valid portfolio, default 200
rounding	integer how many decimals should we round to

Value

named weights vector

Author(s)

Peter Carl, Brian G. Peterson, (based on an idea by Pat Burns)

---

random_portfolios *version 2 generate an arbitary number of constrained random portfolios*

Description

Generate random portfolios using the 'sample', 'simplex', or 'grid' method. See details.

Usage

```
random_portfolios(portfolio, permutations = 100, rp_method = "sample",
  eliminate = TRUE, ...)
```

Arguments

portfolio	an object of class 'portfolio' specifying the constraints for the optimization, see portfolio.spec
permutations	integer: number of unique constrained random portfolios to generate
rp_method	method to generate random portfolios. Currently "sample", "simplex", or "grid". See Details.
eliminate	TRUE/FALSE, eliminate portfolios that do not satisfy constraints
...	any other passthru parameters

Details

Random portfolios can be generate using one of three methods.

- sample: The 'sample' method to generate random portfolios is based on an idea pioneerd by Pat Burns. This is the most flexible method, but also the slowest, and can generate portfolios to satisfy leverage, box, group, position limit, and leverage exposure constraints.

- simplex: The 'simplex' method to generate random portfolios is based on a paper by W. T. Shaw. The simplex method is useful to generate random portfolios with the full investment constraint, where the sum of the weights is equal to 1, and min box constraints. Values for `min_sum` and `max_sum` of the leverage constraint will be ignored, the sum of weights will equal 1. All other constraints such as group and position limit constraints will be handled by elimination. If the constraints are very restrictive, this may result in very few feasible portfolios remaining.
- grid: The 'grid' method to generate random portfolios is based on the `gridSearch` function in package 'NMOF'. The grid search method only satisfies the `min` and `max` box constraints. The `min_sum` and `max_sum` leverage constraints will likely be violated and the weights in the random portfolios should be normalized. Normalization may cause the box constraints to be violated and will be penalized in `constrained_objective`.

The constraint types checked are leverage, box, group, position limit, and leverage exposure. Any portfolio that does not satisfy all these constraints will be eliminated. This function is particularly sensitive to `min_sum` and `max_sum` leverage constraints. For the sample method, there should be some "wiggle room" between `min_sum` and `max_sum` in order to generate a sufficient number of feasible portfolios. For example, `min_sum=0.99` and `max_sum=1.01` is recommended instead of `min_sum=1` and `max_sum=1`. If `min_sum=1` and `max_sum=1`, the number of feasible portfolios may be 1/3 or less depending on the other constraints.

Value

matrix of random portfolio weights

Author(s)

Peter Carl, Brian G. Peterson, Ross Bennett

See Also

`portfolio.spec`, `objective`, `rp_sample`, `rp_simplex`, `rp_grid`

`random_portfolios_v1` *generate an arbitary number of constrained random portfolios*

Description

repeatedly calls `randomize_portfolio` to generate an arbitrary number of constrained random portfolios.

Usage

```
random_portfolios_v1(rpconstraints, permutations = 100, ...)
```

Arguments

rpconstraints	an object of type "constraints" specifying the constraints for the optimization, see constraint
permutations	integer: number of unique constrained random portfolios to generate
...	any other passthru parameters

Value

matrix of random portfolio weights

Author(s)

Peter Carl, Brian G. Peterson, (based on an idea by Pat Burns)

See Also

constraint, objective, randomize_portfolio

Examples

```
rpconstraint<-constraint_v1(assets=10,
                        min_mult=-Inf,
                        max_mult=Inf,
                        min_sum=.99,
                        max_sum=1.01,
                        min=.01,
                        max=.4,
                        weight_seq=generatesequence())

rp<- random_portfolios_v1(rpconstraints=rpconstraint,permutations=1000)
head(rp)
```

---

random_walk_portfolios

deprecated random portfolios wrapper until we write a random trades function

---

Description

deprecated random portfolios wrapper until we write a random trades function

Usage

```
random_walk_portfolios(...)
```

Arguments

...	any other passthru parameters

Author(s)

bpeterson

`regime.portfolios` *Regime Portfolios*

Description

Construct a `regime.portfolios` object that contains a time series of regimes and portfolios corresponding to the regimes.

Usage

```
regime.portfolios(regime, portfolios)
```

Arguments

`regime`	xts or zoo object specifying the regime
`portfolios`	list of portfolios created by `combine.portfolios` with corresponding regimes

Details

Create a `regime.portfolios` object to support regime switching optimization. This object is then passed in as the `portfolio` argument in `optimize.portfolio`. The regime is detected and the corresponding portfolio is selected. For example, if the current regime is 1, then portfolio 1 will be selected and used in the optimization.

Value

a `regime.portfolios` object with the following elements

- regime: An xts object of the regime
- portfolio: List of portfolios corresponding to the regime

Author(s)

Ross Bennett

return_constraint *constructor for return_constraint*

Description

The return constraint specifes a target mean return value. This function is called by add.constraint when type="return" is specified, add.constraint

Usage

```
return_constraint(type = "return", return_target, enabled = TRUE,
  message = FALSE, ...)
```

Arguments

type	character type of the constraint
return_target	return target value
enabled	TRUE/FALSE
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters

Value

an object of class 'return_constraint'

Author(s)

Ross Bennett

See Also

add.constraint

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

pspec <- add.constraint(portfolio=pspec, type="return", return_target=mean(colMeans(ret)))
```

return_objective *constructor for class return_objective*

Description

if target is null, we'll try to maximize the return metric

Usage

```
return_objective(name, target = NULL, arguments = NULL, multiplier = -1,
  enabled = TRUE, ...)
```

Arguments

name	name of the objective, should correspond to a function, though we will try to make allowances
target	univariate target for the objective
arguments	default arguments to be passed to an objective function when executed
multiplier	multiplier to apply to the objective, usually 1 or -1
enabled	TRUE/FALSE
...	any other passthru parameters

Details

if target is set, we'll try to meet or exceed the metric, penalizing a shortfall

Value

object of class 'return_objective'

Author(s)

Brian G. Peterson

risk_budget_objective *constructor for class risk_budget_objective*

Description

constructor for class risk_budget_objective

Usage

```
risk_budget_objective(assets, name, target = NULL, arguments = NULL,
  multiplier = 1, enabled = TRUE, ..., min_prisk, max_prisk,
  min_concentration = FALSE, min_difference = FALSE)
```

Arguments

assets	vector of assets to use, should come from constraints object
name	name of the objective, should correspond to a function, though we will try to make allowances
target	univariate target for the objective
arguments	default arguments to be passed to an objective function when executed
multiplier	multiplier to apply to the objective, usually 1 or -1
enabled	TRUE/FALSE
...	any other passthru parameters
min_prisk	minimum percentage contribution to risk
max_prisk	maximum percentage contribution to risk
min_concentration	TRUE/FALSE whether to minimize concentration, default FALSE, always TRUE if min_prisk and max_prisk are NULL
min_difference	TRUE/FALSE whether to minimize difference between concentration, default FALSE

Value

object of class 'risk_budget_objective'

Author(s)

Brian G. Peterson

---

rp_grid *Generate random portfolios based on grid search method*

Description

This function generates random portfolios based on the `gridSearch` function from the 'NMOF' package.

Usage

```
rp_grid(portfolio, permutations = 2000, normalize = TRUE)
```

Arguments

portfolio	an object of class 'portfolio' specifying the constraints for the optimization, see `portfolio.spec`
permutations	integer: number of unique constrained random portfolios to generate
normalize	TRUE/FALSE to normalize the weghts to satisfy min_sum or max_sum

Details

The number of levels is calculated based on permutations and number of assets. The number of levels must be an integer and may not result in the exact number of permutations. We round up to the nearest integer for the levels so the number of portfolios generated will be greater than or equal to permutations.

The grid search method only satisfies the `min` and `max` box constraints. The `min_sum` and `max_sum` leverage constraints will likely be violated and the weights in the random portfolios should be normalized. Normalization may cause the box constraints to be violated and will be penalized in `constrained_objective`.

Value

matrix of random portfolio weights

`rp_sample` *Generate random portfolios using the sample method*

Description

This function generates random portfolios based on an idea by Pat Burns.

Usage

```
rp_sample(portfolio, permutations, max_permutations = 200)
```

Arguments

`portfolio`	an object of type "portfolio" specifying the constraints for the optimization, see `portfolio.spec`
`permutations`	integer: number of unique constrained random portfolios to generate
`max_permutations`	integer: maximum number of iterations to try for a valid portfolio, default 200

Details

The 'sample' method to generate random portfolios is based on an idea pioneerd by Pat Burns. This is the most flexible method, but also the slowest, and can generate portfolios to satisfy leverage, box, group, and position limit constraints.

Value

a matrix of random portfolio weights

`rp_simplex` *Generate random portfolios using the simplex method*

Description

This function generates random portfolios based on the method outlined in the Shaw paper. Need to add reference.

Usage

```
rp_simplex(portfolio, permutations, fev = 0:5)
```

Arguments

`portfolio`	an object of class 'portfolio' specifying the constraints for the optimization, see `portfolio.spec`
`permutations`	integer: number of unique constrained random portfolios to generate
`fev`	scalar or vector for FEV biasing

Details

The simplex method is useful to generate random portfolios with the full investment constraint where the sum of the weights is equal to 1 and min box constraints with no upper bound on max constraints. Values for min_sum and max_sum will be ignored, the sum of weights will equal 1. All other constraints such as group and position limit constraints will be handled by elimination. If the constraints are very restrictive, this may result in very few feasible portfolios remaining.

The random portfolios are created by first generating a set of uniform random numbers.

$$U \sim [0, 1]$$

The portfolio weights are then transformed to satisfy the min of the box constraints.

$$w_i = min_i + (1 - \sum_{j=1}^{N} min_j) \frac{log(U_i^q}{\sum_{k=1}^{N} log(U_k^q}$$

`fev` controls the Face-Edge-Vertex (FEV) biasing where

$$q = 2^{fev}$$

As q approaches infinity, the set of weights will be concentrated in a single asset. To sample the interior and exterior, `fev` can be passed in as a vector. The number of portfolios, `permutations`, and the length of `fev` affect how the random portfolios are generated. For example, if `permutations=10000` and `fev=0:4`, 2000 portfolios will be generated for each value of `fev`.

Value

a matrix of random portfolio weights

`rp_transform` *Transform a weights vector to satisfy constraints*

Description

This function uses a block of code from `randomize_portfolio` to transform the weight vector if either the weight_sum (leverage) constraints, box constraints, group constraints, position_limit constraints, or leverage exposure constraints are violated. The logic from `randomize_portfolio` is heavily utilized here with extensions to handle more complex constraints. The resulting weights vector might be quite different from the original weights vector.

Usage

```
rp_transform(w, min_sum, max_sum, min_box, max_box, groups = NULL,
  cLO = NULL, cUP = NULL, max_pos = NULL, group_pos = NULL,
  max_pos_long = NULL, max_pos_short = NULL, leverage = NULL,
  weight_seq = NULL, max_permutations = 200)
```

Arguments

`w`	weights vector to be transformed
`min_sum`	minimum sum of all asset weights, default 0.99
`max_sum`	maximum sum of all asset weights, default 1.01
`min_box`	numeric or named vector specifying minimum weight box constraints
`max_box`	numeric or named vector specifying maximum weight box constraints
`groups`	vector specifying the groups of the assets
`cLO`	numeric or vector specifying minimum weight group constraints
`cUP`	numeric or vector specifying minimum weight group constraints
`max_pos`	maximum assets with non-zero weights
`group_pos`	vector specifying maximum number assets with non-zero weights per group
`max_pos_long`	maximum number of assets with long (i.e. buy) positions
`max_pos_short`	maximum number of assets with short (i.e. sell) positions
`leverage`	maximum leverage exposure where leverage is defined as `sum(abs(weights))`
`weight_seq`	vector of seed sequence of weights
`max_permutations`	integer: maximum number of iterations to try for a valid portfolio, default 200

Value

named weighting vector

Author(s)

Peter Carl, Brian G. Peterson, Ross Bennett (based on an idea by Pat Burns)

scatterFUN *Apply a risk or return function to asset returns*

Description

This function is used to calculate risk or return metrics given a matrix of asset returns and will be used for a risk-reward scatter plot of the assets

Usage

```
scatterFUN(R, FUN, arguments = NULL)
```

Arguments

R	xts object of asset returns
FUN	name of function
arguments	named list of arguments to FUN

Author(s)

Ross Bennett

set.portfolio.moments *Portfolio Moments*

Description

Set portfolio moments for use by lower level optimization functions. Currently three methods for setting the moments are available

Usage

```
set.portfolio.moments(R, portfolio, momentargs = NULL, method = c("sample",
  "boudt", "black_litterman", "meucci"), ...)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
portfolio	an object of type "portfolio" specifying the constraints and objectives for the optimization, see portfolio.spec
momentargs	list containing arguments to be passed down to lower level functions, default NULL
method	the method used to estimate portfolio moments. Valid choices include "sample", "boudt", and "black_litterman".
...	any other passthru parameters

Details

- sample: sample estimates are used for the moments
- boudt: estimate the second, third, and fourth moments using a statistical factor model based on the work of Kris Boudt. See `statistical.factor.model`
- black_litterman: estimate the first and second moments using the Black Litterman Formula. See `black.litterman`.

`set.portfolio.moments_v1`

set portfolio moments for use by lower level optimization functions

---

Description

set portfolio moments for use by lower level optimization functions

Usage

```
set.portfolio.moments_v1(R, constraints, momentargs = NULL, ...)
```

Arguments

`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`constraints`	an object of type "constraints" specifying the constraints for the optimization, see `constraint`
`momentargs`	list containing arguments to be passed down to lower level functions, default NULL
`...`	any other passthru parameters FIXME NOTE: this isn't perfect as it overwrites the moments for all objectives, not just one with clean='boudt'

`statistical.factor.model`

Statistical Factor Model

Description

Fit a statistical factor model using Principal Component Analysis (PCA)

Usage

```
statistical.factor.model(R, k = 1, ...)
```

Arguments

R	xts of asset returns
k	number of factors to use
...	additional arguments passed to prcomp

Details

The statistical factor model is fitted using prcomp. The factor loadings, factor realizations, and residuals are computed and returned given the number of factors used for the model.

Value

#'

- factor_loadings N x k matrix of factor loadings (i.e. betas)
- factor_realizations m x k matrix of factor realizations
- residuals m x N matrix of model residuals representing idiosyncratic risk factors

Where N is the number of assets, k is the number of factors, and m is the number of observations.

---

summary.efficient.frontier

Summarize an efficient frontier object

Description

Summary method for efficient frontier objects. Display the call to create or extract the efficient frontier object as well as the weights and risk and return metrics along the efficient frontier.

Usage

```
## S3 method for class 'efficient.frontier'
summary(object, ..., digits = 3)
```

Arguments

object	object of class efficient.frontier
...	passthrough parameters
digits	number of digits to round to

Author(s)

Ross Bennett

summary.optimize.portfolio *Summarizing output of optimize.portfolio*

Description

summary method for class optimize.portfolio

Usage

```
## S3 method for class 'optimize.portfolio'
summary(object, ...)
```

Arguments

object	an object of class optimize.portfolio.
...	any other passthru parameters. Currently not used.

Author(s)

Ross Bennett

See Also

optimize.portfolio

summary.optimize.portfolio.rebalancing *summary method for optimize.portfolio.rebalancing*

Description

summary method for optimize.portfolio.rebalancing

Usage

```
## S3 method for class 'optimize.portfolio.rebalancing'
summary(object, ...)
```

Arguments

object	object of type optimize.portfolio.rebalancing
...	any other passthru parameters

summary.portfolio *Summarize Portfolio Specification Objects*

Description

summary method for class `portfolio` created with `portfolio.spec`

Usage

```
## S3 method for class 'portfolio'
summary(object, ...)
```

Arguments

`object`	an object of class `portfolio`
`...`	any other passthru parameters

Author(s)

Ross Bennett

See Also

`portfolio.spec`

trailingFUN *apply a function over a configurable trailing period*

Description

this function is primarily designed for use with portfolio functions passing 'x' or 'R' and weights, but may be usable for other things as well, see Example for a vector example.

Usage

```
trailingFUN(R, weights, n = 0, FUN, FUNargs = NULL, ...)
```

Arguments

`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`weights`	a vector of weights to test
`n`	numeric number of trailing periods
`FUN`	string describing the function to be called
`FUNargs`	list describing any additional arguments
`...`	any other passthru parameters

Details

called with e.g.

trailingFUN(seq(1:100), weights=NULL, n=12, FUN='mean',FUNargs=list())

`transaction_cost_constraint` *constructor for transaction_cost_constraint*

Description

The transaction cost constraint specifies a proportional cost value. This function is called by add.constraint when type="transaction_cost" is specified, see `add.constraint`.

Usage

```
transaction_cost_constraint(type = "transaction_cost", assets, ptc,
  enabled = TRUE, message = FALSE, ...)
```

Arguments

`type`	character type of the constraint
`assets`	number of assets, or optionally a named vector of assets specifying initial weights
`ptc`	proportional transaction cost value
`enabled`	TRUE/FALSE
`message`	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
`...`	any other passthru parameters to specify box and/or group constraints

Details

Note that with the ROI solvers, proportional transaction cost constraint is currently only supported for the global minimum variance and quadratic utility problems with ROI quadprog plugin.

Value

an object of class 'transaction_cost_constraint'

Author(s)

Ross Bennett

See Also

`add.constraint`

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

pspec <- add.constraint(portfolio=pspec, type="transaction_cost", ptc=0.01)
```

---

turnover *Calculates turnover given two vectors of weights. This is used as an objective function and is called when the user adds an objective of type turnover with* add.objective

---

Description

Calculates turnover given two vectors of weights. This is used as an objective function and is called when the user adds an objective of type turnover with add.objective

Usage

```
turnover(weights, wts.init = NULL)
```

Arguments

weights	vector of weights from optimization
wts.init	vector of initial weights used to calculate turnover from

Author(s)

Ross Bennett

---

turnover_constraint *constructor for turnover_constraint*

Description

The turnover constraint specifies a target turnover value. This function is called by add.constraint when type="turnover" is specified, see add.constraint. Turnover is calculated from a set of initial weights. Turnover is computed as sum(abs(initial_weights - weights)) / N where N is the number of assets.

Usage

```
turnover_constraint(type = "turnover", turnover_target, enabled = TRUE,
  message = FALSE, ...)
```

Arguments

type	character type of the constraint
turnover_target	target turnover value
enabled	TRUE/FALSE
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters to specify box and/or group constraints

Details

Note that with the ROI solvers, turnover constraint is currently only supported for the global minimum variance and quadratic utility problems with ROI quadprog plugin.

Value

an object of class 'turnover_constraint'

Author(s)

Ross Bennett

See Also

add.constraint

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

pspec <- add.constraint(portfolio=pspec, type="turnover", turnover_target=0.6)
```

---

turnover_objective *constructor for class turnover_objective*

Description

if target is null, we'll try to minimize the turnover metric

Usage

```
turnover_objective(name, target = NULL, arguments = NULL, multiplier = 1,
  enabled = TRUE, ...)
```

Arguments

name	name of the objective, should correspond to a function, though we will try to make allowances
target	univariate target for the objective
arguments	default arguments to be passed to an objective function when executed
multiplier	multiplier to apply to the objective, usually 1 or -1
enabled	TRUE/FALSE
...	any other passthru parameters

Details

if target is set, we'll try to meet the metric

Value

an objective of class 'turnover_objective'

Author(s)

Ross Bennett

---

update.constraint *function for updating constrints, not well tested, may be broken*

Description

can we use the generic update.default function?

Usage

```
## S3 method for class 'constraint'
update(object, ...)
```

Arguments

object	object of type constraint to update
...	any other passthru parameters, used to call constraint

Author(s)

bpeterson

update_constraint_v1tov2

Helper function to update v1_constraint objects to v2 specification in the portfolio object

Description

The function takes the constraints and objectives specified in the v1_constraint object and updates the portfolio object with those constraints and objectives. This function is used inside optimize.portfolio to maintain backwards compatibility if the user passes in a v1_constraint object for the constraint arg in optimize.portfolio.

Usage

```
update_constraint_v1tov2(portfolio, v1_constraint)
```

Arguments

portfolio	portfolio object passed into optimize.portfolio
v1_constraint	object of type v1_constraint passed into optimize.portfolio

Value

portfolio object containing constraints and objectives from v1_constraint

Author(s)

Ross Bennett

See Also

portfolio.spec, add.constraint

var.portfolio

Calculate portfolio variance

Description

This function is used to calculate the portfolio variance via a call to constrained_objective when var is an object for mean variance or quadratic utility optimization.

Usage

```
var.portfolio(R, weights)
```

Arguments

R	xts object of asset returns
weights	vector of asset weights

Value

numeric value of the portfolio variance

Author(s)

Ross Bennett

---

weight_concentration_objective

Constructor for weight concentration objective

Description

This function penalizes weight concentration using the Herfindahl-Hirschman Index as a measure of concentration.

Usage

```
weight_concentration_objective(name, conc_aversion, conc_groups = NULL,
  arguments = NULL, enabled = TRUE, ...)
```

Arguments

name	name of concentration measure, currently only "HHI" is supported.
conc_aversion	concentration aversion value(s)
conc_groups	list of vectors specifying the groups of the assets. Similar to groups in group_constraint
arguments	default arguments to be passed to an objective function when executed
enabled	TRUE/FALSE
...	any other passthru parameters

Details

The conc_aversion argument can be a scalar or vector of concentration aversion values. If conc_aversion is a scalar and conc_groups is NULL, then the concentration aversion value will be applied to the overall weights.

If conc_groups is specified as an argument, then the concentration aversion value(s) will be applied to each group.

Value

an object of class 'weight_concentration_objective'

Author(s)

Ross Bennett

`weight_sum_constraint` *constructor for weight_sum_constraint*

Description

The constraint specifies the upper and lower bound on the sum of the weights. This function is called by add.constraint when "weight_sum", "leverage", "full_investment", "dollar_neutral", or "active" is specified as the type. see `add.constraint`

Usage

```
weight_sum_constraint(type = "weight_sum", min_sum = 0.99, max_sum = 1.01,
  enabled = TRUE, ...)
```

Arguments

`type`	character type of the constraint
`min_sum`	minimum sum of all asset weights, default 0.99
`max_sum`	maximum sum of all asset weights, default 1.01
`enabled`	TRUE/FALSE
`...`	any other passthru parameters to specify weight_sum constraints

Details

Special cases for the weight_sum constraint are "full_investment" and "dollar_nuetral" or "active"

If `type="full_investment"`, `min_sum=1` and `max_sum=1`

If `type="dollar_neutral"` or `type="active"`, `min_sum=0`, and `max_sum=0`

Value

an object of class 'weight_sum_constraint'

Author(s)

Ross Bennett

See Also

`add.constraint`

Examples

```
data(edhec)
ret <- edhec[, 1:4]

pspec <- portfolio.spec(assets=colnames(ret))

# min_sum and max_sum can be specified with type="weight_sum" or type="leverage"
pspec <- add.constraint(pspec, type="weight_sum", min_sum=1, max_sum=1)

# Specify type="full_investment" to set min_sum=1 and max_sum=1
pspec <- add.constraint(pspec, type="full_investment")

# Specify type="dollar_neutral" or type="active" to set min_sum=0 and max_sum=0
pspec <- add.constraint(pspec, type="dollar_neutral")
pspec <- add.constraint(pspec, type="active")
```

Index