Package 'PortfolioAnalytics'

August 16, 2013

Type Package

Title Portfolio Analysis, including Numeric Methods for Optimization of Portfolios

Version 0.8.2

Date $Date: 2013-08-06 17:17:14 -0500 (Tue, 06 Aug 2013) $

Author Kris Boudt, Peter Carl, Brian G. Peterson

Contributors Hezky Varon, Guy Yollin

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

Description Portfolio optimization and analysis routines and graphics.

Depends R (>= 2.14.0),zoo,xts (>= 0.8),PerformanceAnalytics (>= 1.0.0)

Suggests
quantmod,DEoptim(>= 2.3.1),foreach,fGarch,Rglpk,quadprog,ROI,ROI.plugin.glpk,ROI.plugin.quadprog,pso,GenSA

License GPL

Collate
'charts.DE.R''charts.RP.R''constrained_objective.R''constraints.R''constraints_ROI.R''extract.efficient.frontier.R''extrac

R topics documented:

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

Description

This is the main function for adding and/or updating constraints in an object of type `portfolio`.

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', or 'factor_exposure'
`enabled`	TRUE/FALSE
`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 $objectives list to update

Details

Special cases for the weight_sum constraint are "full_investment" and "dollar_nuetral" or "active" with appropriate values set for min_sum and max_sum. see `weight_sum_constraint`

Author(s)

Ross Bennett

See Also

constraint_v2, weight_sum_constraint, box_constraint, group_constraint, turnover_constraint, diversification_constraint, position_limit_constraint

---

add.objective_v1 *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 constraint.

Usage

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

Arguments

constraints	an object of type "constraints" to add the objective to, specifying the constraints for the optimization, see constraint
type	character type of the objective to add or update, currently 'return','risk', or 'risk_budget'
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 constraint, the index number in the $objectives list to update

Details

In general, you will define your objective as one of three types: 'return', 'risk', or 'risk_budget'. 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.

Author(s)

Brian G. Peterson

See Also

constraint

`add.objective_v2` *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`.

Usage

```
add.objective_v2(portfolio, type, name, arguments = NULL,
  enabled = TRUE, ..., indexnum = NULL)
```

Arguments

`portfolio`	an object of type 'portfolio' to add the objective to, specifying the portfolio for the optimization, see `portfolio`
`type`	character type of the objective to add or update, currently 'return','risk', or 'risk_budget'
`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 constraint, the index number in the $objectives list to update

Details

In general, you will define your objective as one of three types: 'return', 'risk', or 'risk_budget'. 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.

Author(s)

Brian G. Peterson and Ross Bennett

See Also

`objective`

add.objective_v2 *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.

Usage

```
add.objective_v2(portfolio, type, name, arguments = NULL,
  enabled = TRUE, ..., indexnum = NULL)
```

Arguments

portfolio	an object of type 'portfolio' to add the objective to, specifying the portfolio for the optimization, see portfolio
type	character type of the objective to add or update, currently 'return','risk', or 'risk_budget'
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 constraint, the index number in the $objectives list to update

Details

In general, you will define your objective as one of three types: 'return', 'risk', or 'risk_budget'. 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.

Author(s)

Brian G. Peterson and Ross Bennett

See Also

objective

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

R	
weights	a matrix of weights generated from random_portfolios or optimize.portfolio
FUN	
...	any passthrough arguments to FUN

Author(s)

Ross Bennett

box_constraint *constructor for box_constraint.*

Description

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 seed 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 seed weight in assets

max_mult	numeric or named vector specifying maximum multiplier box constraint from seed 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

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

---

chart.Scatter.DE *classic risk return scatter of DEoptim results*

Description

classic risk return scatter of DEoptim results

Usage

```
chart.Scatter.DE(DE, R = NULL, portfolio = NULL,
  neighbors = NULL, return.col = "mean", risk.col = "ES",
  ..., element.color = "darkgray", cex.axis = 0.8)
```

Arguments

DE	set of portfolios created by optimize.portfolio
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the objective function where required
portfolio	an object of type "portfolio" specifying the constraints and objectives for the optimization
neighbors	set of 'neighbor' portfolios to overplot, see Details in charts.DE
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points

See Also

optimize.portfolio

`chart.Scatter.GenSA` *classic risk return scatter of random portfolios*

Description

The GenSA optimizer does not store the portfolio weights like DEoptim or random portfolios so we will generate random portfolios for the scatter plot.

Usage

```
chart.Scatter.GenSA(GenSA, R, rp = NULL,
  return.col = "mean", risk.col = "StdDev", ...,
  element.color = "darkgray", cex.axis = 0.8, main = "")
```

Arguments

`ROI`	object created by `optimize.portfolio`
`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
`rp`	set of weights generated by `random_portfolio`
`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
`...`	any other passthru parameters
`cex.axis`	The magnification to be used for axis annotation relative to the current setting of `cex`
`element.color`	color for the default plot scatter points

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

Author(s)

Ross Bennett

See Also

`optimize.portfolio`

chart.Scatter.pso *classic risk return scatter of random portfolios*

Description

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

Usage

```
chart.Scatter.pso(pso, R, return.col = "mean",
  risk.col = "StdDev", ..., element.color = "darkgray",
  cex.axis = 0.8, main = "")
```

Arguments

pso	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points

Author(s)

Ross Bennett

See Also

optimize.portfolio

chart.Scatter.ROI *classic risk return scatter of random portfolios*

Description

The ROI optimizers do not store the portfolio weights like DEoptim or random portfolios so we will generate random portfolios for the scatter plot.

Usage

```
chart.Scatter.ROI(ROI, R, rp = NULL, portfolio = NULL,
  return.col = "mean", risk.col = "StdDev", ...,
  element.color = "darkgray", cex.axis = 0.8, main = "")
```

Arguments

ROI	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
rp	set of weights generated by random_portfolio
portfolio	pass in a different portfolio object used in set.portfolio.moments
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points

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

Author(s)

Ross Bennett

See Also

optimize.portfolio

chart.Scatter.RP *classic risk return scatter of random portfolios*

Description

classic risk return scatter of random portfolios

Usage

```
chart.Scatter.RP(RP, R = NULL, neighbors = NULL,
  return.col = "mean", risk.col = "ES", ...,
  element.color = "darkgray", cex.axis = 0.8)
```

Arguments

RP	set of portfolios created by optimize.portfolio
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the objective function when return.col or risk.col is not part of the extractStats output.
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points

See Also

optimize.portfolio

chart.Weights.DE *boxplot of the weight distributions in the random portfolios*

Description

boxplot of the weight distributions in the random portfolios

Usage

```
chart.Weights.DE(DE, neighbors = NULL, ...,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8)
```

Arguments

DE	set of random portfolios created by optimize.portfolio
neighbors	set of 'neighbor' portfolios to overplot
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
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot lines
...	any other passthru parameters
main	an overall title for the plot: see title

See Also

optimize.portfolio

---

chart.Weights.GenSA *boxplot of the weights in the portfolio*

Description

boxplot of the weights in the portfolio

Usage

```
chart.Weights.GenSA(GenSA, neighbors = NULL, ...,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8)
```

Arguments

GenSA	object created by optimize.portfolio
neighbors	set of 'neighbor' portfolios to overplot
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
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot lines
...	any other passthru parameters
main	an overall title for the plot: see title

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

chart.Weights.pso *boxplot of the weights in the portfolio*

Description

boxplot of the weights in the portfolio

Usage

```
chart.Weights.pso(pso, neighbors = NULL, ...,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8)
```

Arguments

pso	object created by optimize.portfolio
neighbors	set of 'neighbor' portfolios to overplot
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

cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot lines
...	any other passthru parameters
main	an overall title for the plot: see title

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

chart.Weights.ROI *boxplot of the weights in the portfolio*

Description

boxplot of the weights in the portfolio

Usage

```
chart.Weights.ROI(ROI, neighbors = NULL, ...,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8)
```

Arguments

ROI	object created by optimize.portfolio
neighbors	set of 'neighbor' portfolios to overplot
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
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot lines
...	any other passthru parameters
main	an overall title for the plot: see title

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

chart.Weights.RP *boxplot of the weight distributions in the random portfolios*

Description

boxplot of the weight distributions in the random portfolios

Usage

```
chart.Weights.RP(RP, neighbors = NULL, ...,
  main = "Weights", las = 3, xlab = NULL, cex.lab = 1,
  element.color = "darkgray", cex.axis = 0.8)
```

Arguments

RP	set of random portfolios created by optimize.portfolio
neighbors	set of 'neighbor' portfolios to overplot
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
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot lines
...	any other passthru parameters
main	an overall title for the plot: see title

See Also

optimize.portfolio

charts.DE *scatter and weights chart for random portfolios*

Description

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.

Usage

```
charts.DE(DE, R = NULL, risk.col, return.col,
  neighbors = NULL, main = "DEoptim.Portfolios", ...)
```

Arguments

DE	set of random portfolios created by optimize.portfolio
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the objective function where required
...	any other passthru parameters
risk.col	string name of column to use for risk (horizontal axis)
return.col	string name of column to use for returns (vertical axis)
neighbors	set of 'neighbor portfolios to overplot
main	an overall title for the plot: see title

See Also

optimize.portfolio extractStats

charts.GenSA *scatter and weights chart for portfolios*

Description

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

Usage

```
charts.GenSA(GenSA, R, rp = NULL, return.col = "mean",
  risk.col = "StdDev", cex.axis = 0.8,
  element.color = "darkgray", neighbors = NULL,
  main = "GenSA.Portfolios", ...)
```

Arguments

GenSA	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
rp	set of weights generated by random_portfolio
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points
neighbors	set of 'neighbor' portfolios to overplot
main	an overall title for the plot: see title

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

charts.pso *scatter and weights chart for portfolios*

Description

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

Usage

```
charts.pso(pso, R, return.col = "mean",
  risk.col = "StdDev", cex.axis = 0.8,
  element.color = "darkgray", neighbors = NULL,
  main = "PSO.Portfolios", ...)
```

Arguments

pso	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points
neighbors	set of 'neighbor' portfolios to overplot
main	an overall title for the plot: see title

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

charts.ROI *scatter and weights chart for portfolios*

Description

The ROI optimizers do not store the portfolio weights like DEoptim or random portfolios so we will generate random portfolios for the scatter plot.

Usage

```
charts.ROI(ROI, R, rp = NULL, portfolio = NULL,
  risk.col = "StdDev", return.col = "mean",
  cex.axis = 0.8, element.color = "darkgray",
  neighbors = NULL, main = "ROI.Portfolios", ...)
```

Arguments

ROI	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
rp	set of weights generated by random_portfolio
portfolio	pass in a different portfolio object used in set.portfolio.moments

risk.col	string matching the objective of a 'risk' objective, on horizontal axis
return.col	string matching the objective of a 'return' objective, on vertical axis
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points
neighbors	set of 'neighbor' portfolios to overplot
main	an overall title for the plot: see title

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

Author(s)

Ross Bennett

See Also

optimize.portfolio

---

charts.RP *scatter and weights chart for random portfolios*

Description

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.

Usage

```
charts.RP(RP, R = NULL, risk.col, return.col,
  neighbors = NULL, main = "Random.Portfolios", ...)
```

Arguments

RP	set of random portfolios created by optimize.portfolio
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the objective function when return.col or risk.col is not part of the extractStats output.
...	any other passthru parameters
risk.col	string name of column to use for risk (horizontal axis)
return.col	string name of column to use for returns (vertical axis)
neighbors	set of 'neighbor portfolios to overplot
main	an overall title for the plot: see title

See Also

optimize.portfolio extractStats

---

constrained_group_tmp *Generic function to impose group constraints on a vector of weights*

Description

This function gets group subsets of the weights vector and checks if the sum of the weights in that group violates the minimum or maximum value. If the sum of weights in a given group violates its maximum or minimum value, the group of weights is normalized to be equal to the minimum or maximum value. This group normalization causes the sum of weights to change. The weights vector is then normalized so that the min_sum and max_sum constraints are satisfied. This "re-normalization" of the weights vector may then cause the group constraints to not be satisfied.

Usage

```
constrained_group_tmp(groups, cLO, cUP, weights, min_sum,
  max_sum, normalize = TRUE)
```

Arguments

groups	vector to group assets
cLO	vector of group weight minimums
cUP	vector of group weight maximums
weights	vector of weights
min_sum	minimum sum of weights
max_sum	maximum sum of weights
normalize	TRUE/FALSE to normalize the weights vector to satisfy the min_sum and max_sum constraints

Details

Group constraints are implemented in ROI solvers, but this function could be used in constrained_objective for random portfolios, DEoptim, pso, or gensa solvers.

Author(s)

Ross Bennett

`constrained_objective_v1` *function to calculate a numeric return value for a portfolio based on a set of constraints*

Description

function to calculate a numeric return value for a portfolio based on a set of constraints, we'll try to make as few assumptions as possible, and only run objectives that are required by the user

Usage

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

Arguments

`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`w`	a vector of weights to test
`constraints`	an object of type "constraints" specifying the constraints for the optimization, see `constraint`
`...`	any other passthru parameters
`trace`	TRUE/FALSE whether to include debugging and additional detail in the output list
`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

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, the default, 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 might violate

your constraints, so you'd need to renormalize them 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 less than 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 random portfolios or DEoptim.control may be passed in via ...

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson

See Also

constraint, objective, DEoptim.control

`constrained_objective_v2`

constrained_objective_v2 function to 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, we'll try to make as few assumptions as possible, and only run objectives that are required by the user

Usage

```
constrained_objective_v2(w, R, portfolio, ...,
  trace = FALSE, normalize = TRUE, storage = FALSE)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
w	a vector of weights to test
portfolio	an object of type "portfolio" specifying the constraints and objectives for the optimization, see constraint
...	any other passthru parameters
trace	TRUE/FALSE whether to include debugging and additional detail in the output list
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

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, the default, 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 might violate your constraints, so you'd need to renormalize them 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 less than 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 random portfolios or DEoptim.control may be passed in via ...

Author(s)

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

See Also

constraint, objective, DEoptim.control

---

constrained_objective_v2

constrained_objective_v2 2 function to 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, we'll try to make as few assumptions as possible, and only run objectives that are required by the user

Usage

```
constrained_objective_v2(w, R, portfolio, ...,
  trace = FALSE, normalize = TRUE, storage = FALSE)
```

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
w	a vector of weights to test
portfolio	an object of type "portfolio" specifying the constraints and objectives for the optimization, see constraint
...	any other passthru parameters
trace	TRUE/FALSE whether to include debugging and additional detail in the output list
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

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, the default, 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 might violate your constraints, so you'd need to renormalize them 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 less than 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 random portfolios or `DEoptim.control` may be passed in via ...

Author(s)

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

See Also

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

`constraint` *constructor for class constraint*

Description

constructor for class constraint

constructor for class v2_constraint

Usage

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

constraint_v2(type, enabled = TRUE, ...,
  constrclass = "v2_constraint")
```

Arguments

assets	number of assets, or optionally a named vector of assets specifying seed 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 seed weight in assets
max_mult	numeric or named vector specifying maximum multiplier box constraint from seed 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, currently 'weight_sum', 'box', or 'group'
assets	number of assets, or optionally a named vector of assets specifying seed weights
...	any other passthru parameters
constrclass	character to name the constraint class

Author(s)

Peter Carl and Brian G. Peterson

Ross Bennett

Examples

```
exconstr <- constraint(assets=10, min_sum=1, max_sum=1, min=.01, max=.35, weight_seq=generatesequence())
```

---

constraint_fnMap *Constraint mapping function*

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. I think our first step should be to test each constraint type, in some sort of hierarchy, starting with box constraints (almost all solvers support box constraints, of course), since some of the other transformations will violate the box constraints, and we'll need to transform back again.

Usage

```
constraint_fnMap(weights, portfolio)
```

Arguments

weights	vector of weights
portfolio	object of class portfolio

Author(s)

Ross Bennett

---

constraint_fn_map *Constraint mapping function*

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. I think our first step should be to test each constraint type, in some sort of hierarchy, starting with box constraints (almost all solvers support box constraints, of course), since some of the other transformations will violate the box constraints, and we'll need to transform back again.

Usage

```
constraint_fn_map(weights, portfolio)
```

Arguments

weights	vector of weights
portfolio	object of class portfolio

Author(s)

Ross Bennett

---

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 *constructor for class constraint*

Description

constructor for class constraint

Usage

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

Arguments

assets	number of assets, or optionally a named vector of assets specifying seed 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 seed weight in assets
max_mult	numeric or named vector specifying maximum multiplier box constraint from seed 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

Author(s)

Peter Carl and Brian G. Peterson

Examples

```
exconstr <- constraint(assets=10, min_sum=1, max_sum=1, min=.01, max=.35, weight_seq=generatesequence())
```

constraint_v2 *constructor for class v2_constraint*

Description

constructor for class v2_constraint

Usage

```
constraint_v2(type, enabled = TRUE, ...,
  constrclass = "v2_constraint")
```

Arguments

type	character type of the constraint to add or update, currently 'weight_sum', 'box', or 'group'
assets	number of assets, or optionally a named vector of assets specifying seed weights
...	any other passthru parameters
constrclass	character to name the constraint class

Author(s)

Ross Bennett

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

This function is called by add.constraint when type="diversification" is specified, add.constraint

Usage

```
diversification_constraint(type = "diversification",
  div_target, 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 box and/or group constraints

Author(s)

Ross Bennett

Examples

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

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

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

extract.efficient.frontier

extract the efficient frontier of portfolios that meet your objectives over a range of risks

Description

note that this function will be extremely sensitive to the objectives in your constraint object. It will be especially obvious if you are looking at a risk budget objective and your return preference is not set high enough.

Usage

```
extract.efficient.frontier(portfolios = NULL,
  match.col = "ES", from = 0, to = 1, by = 0.005, ...,
  R = NULL, constraints = NULL,
  optimize_method = "random")
```

Arguments

portfolios	set of portfolios as generated by extractStats
from	minimum value of the sequence
to	maximum value of the sequence
by	number to increment the sequence by
match.col	string name of column to use for risk (horizontal axis)
...	any other passthru parameters
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
optimize_method	one of "DEoptim" or "random"

Details

If you do not have a set of portfolios to extract from, portfolios may be generated automatically, which would take a very long time.

---

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

See Also

optimize.portfolio

extractStats.optimize.portfolio.DEoptim

extract some stats from a portfolio list run with DEoptim via optimize.portfolio

Description

This function will take everything in the objective_measures slot and `unlist` it. This may produce a very large number of columns or strange column names.

Usage

```
extractStats.optimize.portfolio.DEoptim(object,
  prefix = NULL, ...)
```

Arguments

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

See Also

optimize.portfolio

extractStats.optimize.portfolio.GenSA

extract some stats from a portfolio list run with GenSA via optimize.portfolio

Description

This function will extract the optimal portfolio weights and objective measures The GenSA output does not store weights evaluated at each iteration The GenSA output for trace.mat contains nb.steps, temperature, function.value, and current.minimum

Usage

```
extractStats.optimize.portfolio.GenSA(object,
  prefix = NULL, ...)
```

Arguments

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

`extractStats.optimize.portfolio.parallel`

extract some stats from a portfolio list run via foreach in optimize.portfolio.parallel

---

Description

This function will take everything in the objective_measures slot and `unlist` it. This may produce a very large number of columns or strange column names.

Usage

```
extractStats.optimize.portfolio.parallel(object,
  prefix = NULL, ...)
```

Arguments

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

See Also

`optimize.portfolio` `optimize.portfolio.parallel` `extractStats`

`extractStats.optimize.portfolio.pso`

extract some stats from a portfolio list run with pso via `optimize.portfolio`

Description

This function will extract the weights (swarm positions) from the PSO output and the out value (swarm fitness values) for each iteration of the optimization.

Usage

```
extractStats.optimize.portfolio.pso(object,
  prefix = NULL, ...)
```

Arguments

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

Author(s)

Ross Bennett

`extractStats.optimize.portfolio.random`
extract stats from random portfolio results

Description

This just flattens the $random_portfolio_objective_results part of the object

Usage

```
extractStats.optimize.portfolio.random(object,
  prefix = NULL, ...)
```

Arguments

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

See Also

`optimize.portfolio` `random_portfolios` `extractStats`

`extractStats.optimize.portfolio.ROI`
extract some stats from a portfolio list run with ROI via `optimize.portfolio`

Description

This function will take everything in the objective_measures slot and `unlist` it. This may produce a very large number of columns or strange column names.

Usage

```
extractStats.optimize.portfolio.ROI(object,
  prefix = NULL, ...)
```

Arguments

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

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

extractWeights.optimize.portfolio *extract weights from output of optimize.portfolio*

Description

extract weights from output of optimize.portfolio

Usage

```
extractWeights.optimize.portfolio(object)
```

Arguments

object	object of type optimize.portfolio to extract weights from

Author(s)

Ross Bennett

See Also

optimize.portfolio

extractWeights.optimize.portfolio.rebalancing

extract time series of weights from output of optimize.portfolio.rebalancing

Description

optimize.portfolio.rebalancing outputs a list of optimize.portfolio objects, one for each rebalancing period

Usage

```
extractWeights.optimize.portfolio.rebalancing(RebalResults,
  ...)
```

Arguments

RebalResults	object of type optimize.portfolio.rebalancing to extract weights from
...	any other passthru parameters

Details

The output list is indexed by the dates of the rebalancing periods, as determined by endpoints

See Also

optimize.portfolio.rebalancing

extractWeights.rebal

extract time series of weights from output of optimize.portfolio

Description

optimize.portfolio.rebalancing outputs a list of optimize.portfolio objects, one for each rebalancing period

Usage

```
extractWeights.rebal(RebalResults, ...)
```

Arguments

RebalResults	object of type optimize.portfolio.rebalancing to extract weights from
...	any other passthru parameters

Details

The output list is indexed by the dates of the rebalancing periods, as determined by `endpoints`

See Also

`optimize.portfolio.rebalancing`

`factor_exposure_constraint` *Constructor for factor exposure constraint*

Description

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 seed 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.

Author(s)

Ross Bennett

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, ...)
```

Arguments

weights	vector of weights
portfolio	object of class portfolio
relax	TRUE/FALSE, default FALSE. Enable constraints to be relaxed.

Details

I think our first step should be to test each constraint type, in some sort of hierarchy, starting with box constraints (almost all solvers support box constraints, of course), since some of the other transformations will violate the box constraints, and we'll need to transform back again.

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.

Leverage, box, group, and position limit constraints are transformed. Diversification and turnover constraints are penalized

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, see* portfolio.spec

Description

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. This function will check that these variables are in the portfolio object in the constraints list. This function could be used at the beginning of optimize.portfolio to check the constraints in the portfolio object.

Usage

```
get.constraints(portfolio)
```

Arguments

portfolio	an object of class 'portfolio'

Details

Returns an object of class constraint which is a flat list of weight_sum, box, and group constraints. Uses the same naming as the v1_constraint object which may be useful when passed to other functions.

Author(s)

Ross Bennett

See Also

portfolio.spec, constraint_v2

---

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

Description

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 . 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. Uses the same naming as the v1_constraint object which may be useful when passed to other functions.

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

---

group_constraint *constructor for group_constraint*

Description

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 seed weights
groups	vector 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

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="group",
                        groups=c(2, 2),
                        group_labels=c("Style A", "Style B"),
                        group_min=c(0.15, 0.25),
                        group_max=c(0.65, 0.55))
```

`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`	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
`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

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

Insert a list of constraints into the constraints slot of a 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

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

Usage

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

Arguments

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

Author(s)

Ross Bennett

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)

bpeterson

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

`minmax_objective` *constructor for class tmp_minmax_objective*

Description

I am add this as a temporary objective allowing for a min and max to be specified. Testing to understand how the objective function responds to a range of allowable values. I will likely add this to the turnover, diversification, and volatility constraints allowing the user to specify a range of values.

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
`min`	minimum value
`max`	maximum value
`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 null, we'll try to minimize the metric

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

If max is violated to the upside, penalize the metric If min is violated to the downside, penalize the metric Try to meet the range between min and max

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

constructor for class 'objective'

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

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

Description

TODO write function to check sensitivity of optimal results by using optimize.portfolio.parallel results

Usage

```
optimize.portfolio.parallel(R, constraints,
  optimize_method = c("DEoptim", "random"),
  search_size = 20000, trace = FALSE, ..., nodes = 4)
```

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`
`optimize_method`	one of "DEoptim" or "random"
`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
`nodes`	how many processes to run in the foreach loop, default 4

Details

This function will not speed up optimization!

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 support for rebalancing or rolling periods*

Description

This function may eventually be wrapped into optimize.portfolio

Usage

```
optimize.portfolio.rebalancing(R, portfolio,
  constraints = NULL, objectives = NULL,
  optimize_method = c("DEoptim", "random", "ROI"),
  search_size = 20000, trace = FALSE, ..., rp = NULL,
  rebalance_on = NULL, training_period = NULL,
  trailing_periods = 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
`constraints`	default=NULL, a list of constraint objects
`objectives`	default=NULL, a list of objective objects
`optimize_method`	one of "DEoptim", "random", 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
`rp`	a set of random portfolios passed into the function, to prevent recalculation
`rebalance_on`	a periodicity as returned by xts function periodicity and usable by endpoints
`training_period`	period to use as training in the front of the data
`trailing_periods`	if set, an integer with the number of periods to roll over, default NULL will run from inception

Details

For now, we'll set the rebalancing periods here, though I think they should eventually be part of the constraints object

This function is massively parallel, and will require 'foreach' and we suggest that you register a parallel backend.

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_v1`
version 1 portfolio optimization with support for rebalancing or rolling periods

---

Description

This function may eventually be wrapped into optimize.portfolio

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,
  trailing_periods = 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`
`optimize_method`	one of "DEoptim" or "random"
`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`	a set of random portfolios passed into the function, to prevent recalculation
`rebalance_on`	a periodicity as returned by xts function periodicity and usable by endpoints
`training_period`	period to use as training in the front of the data
`trailing_periods`	if set, an integer with the number of periods to roll over, default NULL will run from inception

Details

For now, we'll set the rebalancing periods here, though I think they should eventually be part of the constraints object

This function is massively parallel, and will require 'foreach' and we suggest that you register a parallel backend.

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_v1 *wrapper for constrained optimization of portfolios*

Description

This function aims to provide a wrapper for constrained optimization of portfolios that allows the user to specify box constraints and business 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")
```

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, if using closed for solver, need to pass a constraint_ROI object.
optimize_method	one of "DEoptim", "random", "ROI","ROI_old", "pso", "GenSA". For using ROI_old, need to use a constraint_ROI object in constraints. For using ROI, pass standard constratint object in constraints argument. Presently, ROI has plugins for quadprog and Rglpk.
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

Details

This function currently supports DEoptim and random portfolios 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.

The extension to ROI solves a limit type of convex optimization problems: 1) Maxmimize portfolio return subject box constraints on weights 2) Minimize portfolio variance subject to box constraints (otherwise known as global minimum variance portfolio) 3) Minimize portfolio variance subject to box constraints and a desired portfolio return 4) Maximize quadratic utility subject to box constraints and risk aversion parameter (this is passed into optimize.portfolio as as added argument to the constraints object) 5) Mean CVaR optiimization subject to box constraints and target portfolio return Lastly, because these convex optimization problem are standardized, there is no need for a penalty term. Therefore, the multiplier argument in add.objective passed into the complete constraint object are ingnored by the solver. ROI also can solve quadratic and linear problems with group constraints by added a groups argument into the constraints object. This argument is a vector with each of its elements the number of assets per group. The group constraints, cLO and cUP, are also added to the constraints object.

For example, if you have 9 assets, and would like to require that the the first 3 assets are in one group, the second 3 are in another, and the third are in another, then you add the grouping by constraints$groups <- c(3,3,3). To apply the constraints that the first group must compose of at least 20 group 15 group should compose of more that 50 you would add the lower group constraint as constraints$cLO <- c(0.20, 0.15, 0.10) and the upper constraints as constraints$cUP <- rep(0.5,3). These group constraint can be set for all five optimization problems listed above.

If you would like to interface with optimize.portfolio using matrix formulations, then use ROI_old.

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_v2` *version 2 wrapper for constrained optimization of portfolios*

Description

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

Usage

```
optimize.portfolio_v2(R, portfolio, constraints = NULL,
  objectives = NULL,
  optimize_method = c("DEoptim", "random", "ROI", "ROI_old", "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
`portfolio`	an object of type "portfolio" specifying the constraints and objectives for the optimization
`constraints`	default=NULL, a list of constraint objects
`objectives`	default=NULL, a list of objective objects
`optimize_method`	one of "DEoptim", "random", "ROI","ROI_old", "pso", "GenSA". For using `ROI_old`, need to use a constraint_ROI object in constraints. For using `ROI`, pass standard `constratint` object in `constraints` argument. Presently, ROI has plugins for `quadprog` and `Rglpk`.
`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.

Details

This function currently supports DEoptim, random portfolios, ROI, pso, and GenSA 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`.

The extension to ROI solves a limited type of convex optimization problems: 1) Maxmimize portfolio return subject leverage, box, and/or constraints on weights 2) Minimize portfolio variance subject to leverage, box, and/or group constraints (otherwise known as global minimum variance portfolio) 3) Minimize portfolio variance subject to leverage, box, and/or group constraints and a desired portfolio return 4) Maximize quadratic utility subject to leverage, box, and/or group constraints and risk aversion parameter (this is passed into `optimize.portfolio` as as added argument to the `constraints` object) 5) Mean CVaR optimization subject to leverage, box, and/or group constraints and target portfolio return Lastly, because these convex optimization problem are standardized, there is no need for a penalty term. Therefore, the `multiplier` argument in `add.objective` passed into the complete constraint object are ingnored by the solver.

If you would like to interface with `optimize.portfolio` using matrix formulations, then use `ROI_old`.

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_v2` *version 2 wrapper for constrained optimization of portfolios*

Description

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

Usage

```
optimize.portfolio_v2(R, portfolio,
  optimize_method = c("DEoptim", "random", "ROI", "ROI_old", "pso", "GenSA"),
  search_size = 20000, trace = FALSE, ..., rp = NULL,
  momentFUN = "set.portfolio.moments_v2",
  message = FALSE)
```

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 constraint, if using closed for solver, need to pass a constraint_ROI object.
optimize_method	one of "DEoptim", "random", "ROI","ROI_old", "pso", "GenSA". For using ROI_old, need to use a constraint_ROI object in constraints. For using ROI, pass standard constratint object in constraints argument. Presently, ROI has plugins for quadprog and Rglpk.
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.

Details

This function currently supports DEoptim, random portfolios, ROI, pso, and GenSA 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.

The extension to ROI solves a limited type of convex optimization problems: 1) Maxmimize portfolio return subject leverage, box, and/or constraints on weights 2) Minimize portfolio variance subject to leverage, box, and/or group constraints (otherwise known as global minimum variance

portfolio) 3) Minimize portfolio variance subject to leverage, box, and/or group constraints and a desired portfolio return 4) Maximize quadratic utility subject to leverage, box, and/or group constraints and risk aversion parameter (this is passed into `optimize.portfolio` as as added argument to the `constraints` object) 5) Mean CVaR optimization subject to leverage, box, and/or group constraints and target portfolio return Lastly, because these convex optimization problem are standardized, there is no need for a penalty term. Therefore, the `multiplier` argument in `add.objective` passed into the complete constraint object are ingnored by the solver.

If you would like to interface with `optimize.portfolio` using matrix formulations, then use `ROI_old`.

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

`plot.optimize.portfolio` *plot method for optimize.portfolio output*

Description

scatter and weights chart for portfolio optimization

Usage

```
plot.optimize.portfolio(x, ..., return.col = "mean",
  risk.col = "ES", neighbors = NULL,
  main = "optimized portfolio plot")
```

Arguments

`x`	set of portfolios created by `optimize.portfolio`
`...`	any other passthru parameters
`risk.col`	string name of column to use for risk (horizontal axis)
`return.col`	string name of column to use for returns (vertical axis)
`neighbors`	set of 'neighbor portfolios to overplot
`main`	an overall title for the plot: see `title`

Details

this is a fallback that will be called for classes of portfolio that do not have specific pre-existing plot methods.

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.

---

plot.optimize.portfolio.DEoptim

plot method for optimize.portfolio.DEoptim output

Description

scatter and weights chart for DEoptim portfolio optimizations run with trace=TRUE

Usage

```
plot.optimize.portfolio.DEoptim(x, ..., R = NULL,
  return.col = "mean", risk.col = "ES", neighbors = NULL,
  main = "optimized portfolio plot")
```

Arguments

x	set of portfolios created by optimize.portfolio
...	any other passthru parameters
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the objective function where required
risk.col	string name of column to use for risk (horizontal axis)
return.col	string name of column to use for returns (vertical axis)
neighbors	set of 'neighbor portfolios to overplot
main	an overall title for the plot: see title

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.

plot.optimize.portfolio.GenSA *scatter and weights chart for portfolios*

Description

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

Usage

```
plot.optimize.portfolio.GenSA(GenSA, R, rp = NULL,
  return.col = "mean", risk.col = "StdDev",
  cex.axis = 0.8, element.color = "darkgray",
  neighbors = NULL, main = "GenSA.Portfolios", ...)
```

Arguments

GenSA	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
rp	set of weights generated by random_portfolio
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points
neighbors	set of 'neighbor' portfolios to overplot
main	an overall title for the plot: see title

Author(s)

Ross Bennett

See Also

optimize.portfolio

plot.optimize.portfolio.pso *scatter and weights chart for portfolios*

Description

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

Usage

```
plot.optimize.portfolio.pso(pso, R, return.col = "mean",
  risk.col = "StdDev", cex.axis = 0.8,
  element.color = "darkgray", neighbors = NULL,
  main = "PSO.Portfolios", ...)
```

Arguments

pso	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
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
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points
neighbors	set of 'neighbor' portfolios to overplot
main	an overall title for the plot: see title

Author(s)

Ross Bennett

See Also

optimize.portfolio

plot.optimize.portfolio.random

plot method for optimize.portfolio.random output

Description

scatter and weights chart for random portfolios

Usage

```
plot.optimize.portfolio.random(x, ..., R = NULL,
  return.col = "mean", risk.col = "ES", neighbors = NULL,
  main = "optimized portfolio plot")
```

Arguments

x	set of portfolios created by optimize.portfolio
...	any other passthru parameters
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the objective function when return.col or risk.col is not part of the extractStats output.
risk.col	string name of column to use for risk (horizontal axis)
return.col	string name of column to use for returns (vertical axis)
neighbors	set of 'neighbor portfolios to overplot
main	an overall title for the plot: see title

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.

plot.optimize.portfolio.ROI

scatter and weights chart for portfolios

Description

The ROI optimizers do not store the portfolio weights like DEoptim or random portfolios so we will generate random portfolios for the scatter plot.

Usage

```
plot.optimize.portfolio.ROI(ROI, R, rp = NULL,
  portfolio = NULL, risk.col = "StdDev",
  return.col = "mean", element.color = "darkgray",
  neighbors = NULL, main = "ROI.Portfolios", ...)
```

Arguments

ROI	object created by optimize.portfolio
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalulate the risk and return metric
rp	set of weights generated by random_portfolio
portfolio	pass in a different portfolio object used in set.portfolio.moments
risk.col	string matching the objective of a 'risk' objective, on horizontal axis
return.col	string matching the objective of a 'return' objective, on vertical axis
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points
neighbors	set of 'neighbor' portfolios to overplot
main	an overall title for the plot: see title

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

Author(s)

Ross Bennett

See Also

optimize.portfolio

portfolio.spec *constructor for class portfolio*

Description

constructor for class portfolio

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.
weight_seq	seed sequence of weights, see generatesequence
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.

Author(s)

Ross Bennett

Examples

```
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

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)

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
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

Author(s)

Ross Bennett #'

Examples

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

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

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

---

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 objects of class 'constraint'*

Description

print method for objects of class 'constraint'

Usage

```
print.constraint(obj)
```

Arguments

portfolio	object of class constraint

Author(s)

Ross Bennett

`print.optimize.portfolio.DEoptim`

Printing Output of optimize.portfolio

Description

print method for optimize.portfolio.DEoptim

Usage

```
print.optimize.portfolio.DEoptim(object,
  digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`object`	an object of class "optimize.portfolio.DEoptim" resulting from a call to optimize.portfolio
`digits`	the number of significant digits to use when printing.
`...`	any other passthru parameters

`print.optimize.portfolio.GenSA`

Printing Output of optimize.portfolio

Description

print method for optimize.portfolio.GenSA

Usage

```
print.optimize.portfolio.GenSA(object,
  digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`object`	an object of class "optimize.portfolio.GenSA" resulting from a call to optimize.portfolio
`digits`	the number of significant digits to use when printing
`...`	any other passthru parameters

print.optimize.portfolio.pso

Printing Output of optimize.portfolio

Description

print method for optimize.portfolio.pso

Usage

```
print.optimize.portfolio.pso(object,
  digits = max(3, getOption("digits") - 3), ...)
```

Arguments

object	an object of class "optimize.portfolio.pso" resulting from a call to optimize.portfolio
digits	the number of significant digits to use when printing.
...	any other passthru parameters

print.optimize.portfolio.random

Printing Output of optimize.portfolio

Description

print method for optimize.portfolio.random

Usage

```
print.optimize.portfolio.random(object,
  digits = max(3, getOption("digits") - 3), ...)
```

Arguments

object	an object of class "optimize.portfolio.random" resulting from a call to optimize.portfolio
digits	the number of significant digits to use when printing.
...	any other passthru parameters

print.optimize.portfolio.ROI *Printing Output of optimize.portfolio*

Description

print method for optimize.portfolio.ROI

Usage

```
print.optimize.portfolio.ROI(object,
  digits = max(3, getOption("digits") - 3), ...)
```

Arguments

object	an object of class "optimize.portfolio.ROI" resulting from a call to optimize.portfolio
digits	the number of significant digits to use when printing.
...	any other passthru parameters

print.portfolio *Printing Portfolio Specification Objects*

Description

Print method for class "portfolio"

Usage

```
print.portfolio(portfolio)
```

Arguments

portfolio	object of class portfolio

Author(s)

Ross Bennett

randomize_portfolio_v1

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

Description

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

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 weighting vector

Author(s)

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

randomize_portfolio_v2

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_v2(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
rounding	integer how many decimals should we round to

Value

named weighting vector

Author(s)

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

---

randomize_portfolio_v2

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_v2(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
rounding	integer how many decimals should we round to

Value

named weighting vector

Author(s)

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

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(assets=10, min_mult=-Inf, max_mult=Inf, min_sum=.99, max_sum=1.01, min=.01, max=.4, w
rp<- random_portfolios(rpconstraints=rpconstraint,permutations=1000)
head(rp)
```

`random_portfolios_v2` *version 2 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_v2(portfolio, permutations = 100, ...)
```

Arguments

`portfolio`	an object of type "portfolio" 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

`portfolio.spec`, `objective`, `randomize_portfolio_v2`

`random_portfolios_v2` *version 2 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_v2(portfolio, permutations = 100, ...)
```

Arguments

portfolio	an object of type "portfolio" 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

portfolio.spec, objective, randomize_portfolio_v2

---

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

return_constraint *constructor for return_constraint*

Description

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

Author(s)

Ross Bennett

Examples

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

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

pspec <- add.constraint(portfolio=pspec, type="return", div_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

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

Author(s)

Brian G. Peterson

`rp_transform` *Transform a weights vector to satisfy leverage, box, group, and position_limit constraints using logic from* `randomize_portfolio`

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, or position_limit constraints are violated. The resulting weights vector might be quite different from the original weights vector.

Usage

```
rp_transform(w, min_sum = 0.99, max_sum = 1.01, min, max,
  groups, cLO, cUP, max_pos = NULL, group_pos = NULL,
  max_pos_long = NULL, max_pos_short = 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
: numeric or named vector specifying minimum weight box constraints

max
: 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

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)

---

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

---

set.portfolio.moments_v2

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_v2(R, portfolio, momentargs = 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
...	any other passthru parameters

---

set.portfolio.moments_v2

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_v2(R, portfolio, momentargs = 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
...	any other passthru parameters

---

summary.optimize.portfolio

Summarizing Output of optimize.portfolio

Description

summary method for class "optimize.portfolio"

Usage

```
summary.optimize.portfolio(object, ...)
```

Arguments

object	an object of class "optimize.portfolio.pso" resulting from a call to optimize.portfolio
...	any other passthru parameters. Currently not used.

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

Description

summary method for optimize.portfolio.rebalancing

Usage

```
summary.optimize.portfolio.rebalancing(object, ...)
```

Arguments

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

summary.portfolio *Summarizing Portfolio Specification Objects*

Description

summary method for class "portfolio"

Usage

```
summary.portfolio(portfolio)
```

Arguments

portfolio	object of class portfolio

Author(s)

Ross Bennett

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 Exmample 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
...	any other passthru parameters
n	numeric number of trailing periods
FUN	string describing the function to be called
FUNargs	list describing any additional arguments

Details

called with e.g.

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

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

This function is called by add.constraint when type="turnover" is specified. see `add.constraint` This function allows the user to specify a target turnover value

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 turnover constraint is currently only supported for global minimum variance problem with ROI quadprog plugin

Author(s)

Ross Bennett

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

Author(s)

Ross Bennett

txfrm_box_constraint *Transform weights that violate min or max box constraints*

Description

This is a helper function called inside constraint_fnMap to transform the weights vector to satisfy box constraints.

Usage

```
txfrm_box_constraint(weights, min, max)
```

Arguments

weights	vector of weights
min	vector of minimum asset weights from box constraints
max	vector of maximum asset weights from box constraints

Author(s)

Ross Bennett

---

txfrm_group_constraint

Transform weights that violate group constraints

Description

This is a helper function called inside constraint_fnMap to transform the weights vector to satisfy group constraints.

Usage

```
txfrm_group_constraint(weights, groups, cLO, cUP)
```

Arguments

weights	vector of weights
groups	vector of groups
cLO	vector of minimum group weights from group constraints
cUP	vector of maximum group weights from group constraints

Author(s)

Ross Bennett

txfrm_position_limit_constraint

Transform weights for position_limit constraints

Description

This is a helper function called inside constraint_fnMap to transform the weights vector to satisfy position_limit constraints. This function sets the minimum nassets-max_pos assets equal to 0 such that the max_pos number of assets will have non-zero weights.

Usage

```
txfrm_position_limit_constraint(weights, max_pos,
  nassets, tolerance = .Machine$double.eps^0.5)
```

Arguments

weights	vector of weights
max_pos	maximum position of assets with non_zero weights
nassets	number of assets

Author(s)

Ross Bennett

txfrm_weight_sum_constraint

Transform weights that violate weight_sum constraints

Description

This is a helper function called inside constraint_fnMap to transform the weights vector to satisfy weight_sum constraints.

Usage

```
txfrm_weight_sum_constraint(weights, min_sum, max_sum)
```

Arguments

weights	vector of weights
min_sum	minimum sum of asset weights
max_sum	maximum sum of asset weights

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

```
update.constraint(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

---

volatility_constraint *constructor for volatility_constraint*

Description

This function is called by add.constraint when type="volatility" is specified, add.constraint Penalize if portfolio standard deviation deviates from volatility target

Usage

```
volatility_constraint(type, vol_target, enabled = FALSE,
  ...)
```

Arguments

type	character type of the constraint
vol_target	target volatilty constraint
enabled	TRUE/FALSE
...	any other passthru parameters to specify box and/or group constraints

Author(s)

Ross Bennett

---

weight_sum_constraint *constructor for weight_sum_constraint*

Description

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 This function allows the user to specify the minimum and maximum that the weights sum to

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
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	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

Author(s)

Ross Bennett

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