

# Package ‘PortfolioAnalytics’

August 18, 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

**Copyright** (c) 2004-2012

**Collate**

'charts.DE.R''charts.RP.R''constrained\_objective.R''constraints.R''constraints\_ROI.R''extract.efficient.frontier.R''extract

## R topics documented:

add.constraint . . . . .	4
add.objective . . . . .	6
applyFUN . . . . .	7
box_constraint . . . . .	7
CCCgarch.MM . . . . .	8
chart.Scatter.DE . . . . .	9
chart.Scatter.GenSA . . . . .	10
chart.Scatter.pso . . . . .	11
chart.Scatter.ROI . . . . .	12

chart.Scatter.RP . . . . .	13
chart.Weights.DE . . . . .	13
chart.Weights.GenSA . . . . .	14
chart.Weights.pso . . . . .	15
chart.Weights.ROI . . . . .	16
chart.Weights.RP . . . . .	17
charts.DE . . . . .	18
charts.GenSA . . . . .	18
charts.pso . . . . .	19
charts.ROI . . . . .	20
charts.RP . . . . .	21
constrained_group_tmp . . . . .	22
constrained_objective . . . . .	23
constraint . . . . .	24
constraint_ROI . . . . .	26
diversification . . . . .	26
diversification_constraint . . . . .	27
extract.efficient.frontier . . . . .	28
extractObjectiveMeasures . . . . .	29
extractStats . . . . .	29
extractStats.optimize.portfolio.DEoptim . . . . .	30
extractStats.optimize.portfolio.GenSA . . . . .	30
extractStats.optimize.portfolio.parallel . . . . .	31
extractStats.optimize.portfolio.pso . . . . .	31
extractStats.optimize.portfolio.random . . . . .	32
extractStats.optimize.portfolio.ROI . . . . .	32
extractWeights . . . . .	33
extractWeights.optimize.portfolio . . . . .	33
extractWeights.optimize.portfolio.rebalancing . . . . .	34
factor_exposure_constraint . . . . .	35
fn_map . . . . .	36
generatesequence . . . . .	37
get_constraints . . . . .	37
group_constraint . . . . .	38
group_fail . . . . .	39
indexes . . . . .	40
insert_constraints . . . . .	40
insert_objectives . . . . .	41
is.constraint . . . . .	41
is.objective . . . . .	42
is.portfolio . . . . .	42
minmax_objective . . . . .	43
name.replace . . . . .	44
objective . . . . .	44
optimize.portfolio . . . . .	45
optimize.portfolio.parallel . . . . .	47
optimize.portfolio.rebalancing . . . . .	48
plot.optimize.portfolio . . . . .	50

plot.optimize.portfolio.DEoptim	51
plot.optimize.portfolio.GenSA	51
plot.optimize.portfolio.pso	52
plot.optimize.portfolio.random	53
plot.optimize.portfolio.ROI	54
portfolio.spec	55
portfolio_risk_objective	56
position_limit_constraint	57
pos_limit_fail	58
print.constraint	58
print.optimize.portfolio.DEoptim	59
print.optimize.portfolio.GenSA	59
print.optimize.portfolio.pso	60
print.optimize.portfolio.random	60
print.optimize.portfolio.ROI	61
print.portfolio	61
randomize_portfolio_v1	62
randomize_portfolio_v2	62
random_portfolios_v1	63
random_portfolios_v2	64
random_walk_portfolios	65
return_constraint	65
return_objective	66
risk_budget_objective	67
rp_transform	68
set.portfolio.moments_v1	69
set.portfolio.moments_v2	69
summary.optimize.portfolio	70
summary.optimize.portfolio.rebalancing	70
summary.portfolio	71
trailingFUN	71
turnover	72
turnover_constraint	72
turnover_objective	73
txfrm_box_constraint	74
txfrm_group_constraint	74
txfrm_position_limit_constraint	75
txfrm_weight_sum_constraint	75
update.constraint	76
update_constraint_v1tov2	76
weight_sum_constraint	77

---

add.constraint	<i>General interface for adding and/or updating optimization constraints.</i>
----------------	---

---

## Description

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

## Usage

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

## Arguments

portfolio	an object of class 'portfolio' to add the constraint to, specifying the constraints for the optimization, see <a href="#">portfolio.spec</a>
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. The default is enabled=TRUE.
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
...	any other passthru parameters to specify constraints
indexnum	if you are updating a specific constraint, the index number in the \$constraints list to update

## Details

The following constraint types may be specified:

- `weight_sum`, `weight`, `leverage` Specify constraint on the sum of the weights, see [weight\\_sum\\_constraint](#)
- `full_investment` Special case to set `min_sum=1` and `max_sum=1` of weight sum constraints
- `dollar_neutral`, `active` Special case to set `min_sum=0` and `max_sum=0` of weight sum constraints
- `box` box constraints for the individual asset weights, see [box\\_constraint](#)
- `long_only` Special case to set `min=0` and `max=1` of box constraints
- `group` specify the sum of weights within groups and the number of assets with non-zero weights in groups, see [group\\_constraint](#)
- `turnover` Specify a constraint for target turnover. Turnover is calculated from a set of initial weights, see [turnover\\_constraint](#)
- `diversification` target diversification of a set of weights, see [diversification\\_constraint](#)
- `position_limit` Specify the number of non-zero positions, see [position\\_limit\\_constraint](#)
- `return` Specify the target mean return, see [return\\_constraint](#)
- `factor_exposure` Specify risk factor exposures, see [factor\\_exposure\\_constraint](#)

**Author(s)**

Ross Bennett

**See Also**

[portfolio.spec](#), [weight\\_sum\\_constraint](#), [box\\_constraint](#), [group\\_constraint](#), [turnover\\_constraint](#), [diversification\\_constraint](#), [position\\_limit\\_constraint](#), [return\\_constraint](#), [factor\\_exposure\\_constraint](#)

**Examples**

```

data(edhec)
returns <- edhec[, 1:4]
fund.names <- colnames(returns)
pspec <- portfolio.spec(assets=fund.names)
# Add the full investment constraint that specifies the weights must sum to 1.
pspec <- add.constraint(portfolio=pspec, type="weight_sum", min_sum=1, max_sum=1)
# The full investment constraint can also be specified with type="full_investment"
pspec <- add.constraint(portfolio=pspec, type="full_investment")

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

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

min and max can also be specified per asset
pspec <- add.constraint(portfolio=pspec, type="box", min=c(0.05, 0, 0.08, 0.1), max=c(0.4, 0.3, 0.7, 0.55))
# A special case of box constraints is long only where min=0 and max=1
# The default action is long only if min and max are not specified
pspec <- add.constraint(portfolio=pspec, type="box")
pspec <- add.constraint(portfolio=pspec, type="long_only")

# Add group constraints
pspec <- add.constraint(portfolio=pspec, type="group", groups=c(3, 1), group_min=c(0.1, 0.15), group_max=c(0.8, 0.8))

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

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

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

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

```

---

add.objective	<i>General interface for adding optimization objectives, including risk, return, and risk budget</i>
---------------	--

---

### Description

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

### Usage

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

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

add.objective(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 <a href="#">portfolio</a>
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
constraints	an object of type "constraints" to add the objective to, specifying the constraints for the optimization, see <a href="#">constraint</a> (for _v1 objectives only)

### 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.

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

### Author(s)

Brian G. Peterson and Ross Bennett

**See Also**

[objective](#), [portfolio.spec](#)

---

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**

Box constraints specify the upper and lower bounds on the weights of the assets. This function is called by add.constraint when type="box" is specified. see [add.constraint](#)

**Usage**

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

**Arguments**

type	character type of the constraint
assets	number of assets, or optionally a named vector of assets specifying 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	<i>classic risk return scatter of DEoptim results</i>
------------------	---

---

**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 <a href="#">optimize.portfolio</a>
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">charts.DE</a>
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 <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate the risk and return metric
rp	set of weights generated by <a href="#">random_portfolio</a>
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 <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate the risk and return metric
rp	set of weights generated by <a href="#">random_portfolio</a>
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	<i>classic risk return scatter of random portfolios</i>
------------------	---

---

**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 <a href="#">optimize.portfolio</a>
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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	<i>boxplot of the weight distributions in the random portfolios</i>
------------------	---

---

**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 <a href="#">optimize.portfolio</a>
neighbors	set of 'neighbor' portfolios to overplot
las	numeric in {0,1,2,3}; the style of axis labels <b>0:</b> always parallel to the axis [ <i>default</i> ], <b>1:</b> always horizontal, <b>2:</b> always perpendicular to the axis, <b>3:</b> always vertical.
xlab	a title for the x axis: see <a href="#">title</a>
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 <a href="#">title</a>

**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 <a href="#">optimize.portfolio</a>
neighbors	set of 'neighbor' portfolios to overplot
las	numeric in {0,1,2,3}; the style of axis labels <b>0:</b> always parallel to the axis [ <i>default</i> ], <b>1:</b> always horizontal, <b>2:</b> always perpendicular to the axis,

	<b>3:</b> always vertical.
xlab	a title for the x axis: see <a href="#">title</a>
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 <a href="#">title</a>

**Author(s)**

Ross Bennett

**See Also**

[optimize.portfolio](#)

---

chart.Weights.pso	<i>boxplot of the weights in the portfolio</i>
-------------------	--

---

**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 <a href="#">optimize.portfolio</a>
neighbors	set of 'neighbor' portfolios to overplot
las	numeric in {0,1,2,3}; the style of axis labels <b>0:</b> always parallel to the axis [ <i>default</i> ], <b>1:</b> always horizontal, <b>2:</b> always perpendicular to the axis, <b>3:</b> always vertical.
xlab	a title for the x axis: see <a href="#">title</a>
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex

<code>cex.axis</code>	The magnification to be used for axis annotation relative to the current setting of <code>cex</code>
<code>element.color</code>	color for the default plot lines
<code>...</code>	any other passthru parameters
<code>main</code>	an overall title for the plot: see <a href="#">title</a>

**Author(s)**

Ross Bennett

**See Also**

[optimize.portfolio](#)

---

<code>chart.Weights.ROI</code>	<i>boxplot of the weights in the portfolio</i>
--------------------------------	--

---

**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**

<code>ROI</code>	object created by <a href="#">optimize.portfolio</a>
<code>neighbors</code>	set of 'neighbor' portfolios to overplot
<code>las</code>	numeric in {0,1,2,3}; the style of axis labels <b>0:</b> always parallel to the axis [ <i>default</i> ], <b>1:</b> always horizontal, <b>2:</b> always perpendicular to the axis, <b>3:</b> always vertical.
<code>xlab</code>	a title for the x axis: see <a href="#">title</a>
<code>cex.lab</code>	The magnification to be used for x and y labels relative to the current setting of <code>cex</code>
<code>cex.axis</code>	The magnification to be used for axis annotation relative to the current setting of <code>cex</code>
<code>element.color</code>	color for the default plot lines
<code>...</code>	any other passthru parameters
<code>main</code>	an overall title for the plot: see <a href="#">title</a>



**Author(s)**

Ross Bennett

**See Also**[optimize.portfolio](#)


---

chart.Weights.RP	<i>boxplot of the weight distributions in the random portfolios</i>
------------------	---

---

**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 <a href="#">optimize.portfolio</a>
neighbors	set of 'neighbor' portfolios to overplot
las	numeric in {0,1,2,3}; the style of axis labels <b>0:</b> always parallel to the axis [ <i>default</i> ], <b>1:</b> always horizontal, <b>2:</b> always perpendicular to the axis, <b>3:</b> always vertical.
xlab	a title for the x axis: see <a href="#">title</a>
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 <a href="#">title</a>

**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 <a href="#">optimize.portfolio</a>
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">title</a>

### 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 <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate the risk and return metric
rp	set of weights generated by <a href="#">random_portfolio</a>
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 <a href="#">title</a>

**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 <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">title</a>

**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 <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate the risk and return metric
rp	set of weights generated by <a href="#">random_portfolio</a>
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 <a href="#">title</a>

### 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 <a href="#">optimize.portfolio</a>
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">title</a>

**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` *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_v1(w, R, constraints, ...,
  trace = FALSE, normalize = TRUE, storage = FALSE)
```

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

```
constrained_objective(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 <a href="#">constraint</a>
...	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 other solvers (e.g. random portfolios or `DEoptim.control` or `pso` or `GenSA`) may be passed in via ...

## Author(s)

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

## See Also

[constraint](#), [objective](#), [DEoptim.control](#)

---

constraint

*constructor for class constraint*

---

## Description

This function is the constructor for the constraint object stored in the `portfolio.spec` object.



**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 <a href="#">generatesequence</a>
type	character type of the constraint to add or update
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

**Details**

See main documentation in [add.constraint](#)

**Author(s)**

Peter Carl, Brian G. Peterson, Ross Bennett

**See Also**

[add.constraint](#)

**Examples**

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

---

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 <a href="#">generatesequence</a>

**Author(s)**

Hezky Varon

---

diversification      *Function to compute diversification as a constraint*

---

**Description**

Diversification is defined as 1 minus the sum of the squared weights  $\text{diversification} \leftarrow 1 - \sum(w^2)$

**Usage**

```
diversification(weights)
```

**Arguments**

weights	vector of asset weights
---------	-------------------------

**Author(s)**

Ross Bennett

---

diversification\_constraint  
*constructor for diversification\_constraint*

---

### Description

The diversification constraint specifies a target diversification value. This function is called by `add.constraint` when `type="diversification"` is specified, see [add.constraint](#).

### Usage

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

### Arguments

<code>type</code>	character type of the constraint
<code>div_target</code>	diversification target value
<code>enabled</code>	TRUE/FALSE
<code>message</code>	TRUE/FALSE. The default is <code>message=FALSE</code> . Display messages if TRUE.
<code>...</code>	any other passthru parameters to specify box and/or 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="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 <a href="#">extractStats</a>
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 <a href="#">constraint</a>
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.

---

extractObjectiveMeasures  
*Extract the objective measures*

---

**Description**

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

**Usage**

```
extractObjectiveMeasures(object)
```

**Arguments**

object            list returned by `optimize.portfolio`

**Value**

list of objective measures

**Author(s)**

Ross Bennett

**See Also**

[optimize.portfolio](#)

---

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

---

**Description**

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

**Usage**

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

**Arguments**

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

**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 <code>optimize.portfolio</code>
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 <code>optimize.portfolio</code>
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	<i>extract weights from a portfolio run via optimize.portfolio or optimize.portfolio.rebalancing</i>
----------------	--

---

**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	<i>extract weights from output of optimize.portfolio</i>
-----------------------------------	--

---

**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](#)

---

`factor_exposure_constraint`*Constructor for factor exposure constraint*

---

### Description

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

### Usage

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

### Arguments

<code>type</code>	character type of the constraint
<code>assets</code>	named vector of assets specifying seed weights
<code>B</code>	vector or matrix of risk factor exposures
<code>lower</code>	vector of lower bounds of constraints for risk factor exposures
<code>upper</code>	vector of upper bounds of constraints for risk factor exposures
<code>enabled</code>	TRUE/FALSE
<code>message</code>	TRUE/FALSE. The default is <code>message=FALSE</code> . Display messages if TRUE.
<code>...</code>	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

### See Also

[add.constraint](#)

---

fn_map	<i>mapping function to transform or penalize weights that violate constraints</i>
--------	---

---

### 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	<i>create a sequence of possible weights for random or brute force portfolios</i>
------------------	---

---

### 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	integer 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	<i>Helper function to get the enabled constraints out of the portfolio object</i>
-----------------	---

---

### 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**

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

**Usage**

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

**Arguments**

type	character type of the constraint
assets	number of assets, or optionally a named vector of assets specifying 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)

**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 adding 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

---

<code>name.replace</code>	<i>utility function to replace awkward named from unlist</i>
---------------------------	--

---

**Description**

utility function to replace awkward named from unlist

**Usage**

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

**Arguments**

<code>rnames</code>	character vector of names to check for cleanup
---------------------	--

---

<code>objective</code>	<i>constructor for class 'objective'</i>
------------------------	--

---

**Description**

Typically called as a sub-function by the user function [add.objective](#). See main documentation there.

**Usage**

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

**Arguments**

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

**Author(s)**

Brian G. Peterson

**See Also**

[add.objective](#), [portfolio.spec](#)

---

optimize.portfolio      *constrained optimization of portfolios*

---

## Description

This function aims to provide a wrapper for constrained optimization of portfolios that allows the user to specify box constraints and business objectives. It will be the objective functionFUN passed to any supported R optimization solver.

## Usage

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

```
optimize.portfolio_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)
```

```
optimize.portfolio(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. An object of class ]v1_constraint' can be passed in here.
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 constraint 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 <a href="#">set.portfolio.moments_v2</a>
message	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.

## 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 limited type of convex optimization problems:

- Maximize portfolio return subject leverage, box, group, position limit, target mean return, and/or factor exposure constraints on weights
- Minimize portfolio variance subject to leverage, box, group, and/or factor exposure constraints (otherwise known as global minimum variance portfolio)
- Minimize portfolio variance subject to leverage, box, group, and/or factor exposure constraints and a desired portfolio return
- Maximize quadratic utility subject to leverage, box, group, target mean return, and/or factor exposure constraints and risk aversion parameter. (The risk aversion parameter is passed into optimize.portfolio as an added argument to the portfolio object)
- Mean CVaR optimization subject to leverage, box, group, position limit, target mean return, and/or factor exposure constraints and target portfolio return

Because these convex optimization problem are standardized, there is no need for a penalty term. The multiplier argument in [add.objective](#) passed into the complete constraint object are ignored by the ROI 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 15 should compose of more than 50 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 convex optimization problems listed above, as well as for the global stochastic solvers DEoptim, random, pso, and GenSA.

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

### Note

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

### Author(s)

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

### See Also

[portfolio.spec](#)

---

optimize.portfolio.parallel

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

---

### Description

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 <a href="#">constraint</a>
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, presumably 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 appropriate 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_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)
```

```
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

---

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 <a href="#">optimize.portfolio</a>
...	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 <a href="#">title</a>

**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 <a href="#">optimize.portfolio</a>
...	any other passthru parameters
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">title</a>

### 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 <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate the risk and return metric
rp	set of weights generated by <a href="#">random_portfolio</a>
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 <a href="#">title</a>

**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**

psd	object created by <a href="#">optimize.portfolio</a>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">title</a>

**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 <a href="#">optimize.portfolio</a>
...	any other passthru parameters
R	an optional an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate 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 <a href="#">title</a>

**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 <code>optimize.portfolio</code>
R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns, used to recalculate the risk and return metric
rp	set of weights generated by <code>random_portfolio</code>
portfolio	pass in a different portfolio object used in <code>set.portfolio.moments</code>
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 <code>title</code>

**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	<i>constructor for class portfolio</i>
----------------	--

---

**Description**

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

**Usage**

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

**Arguments**

<code>assets</code>	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.
<code>category_labels</code>	character vector to categorize assets by sector, industry, geography, market-cap, currency, etc. Default NULL
<code>weight_seq</code>	seed sequence of weights, see <a href="#">generatesequence</a> Default NULL
<code>message</code>	TRUE/FALSE. The default is <code>message=FALSE</code> . Display messages if TRUE.

**Details**

The portfolio object contains the following elements:

- `assets` named vector of the seed weights
- `category_labels` character vector to categorize the assets by sector, geography, etc.
- `weight_seq` sequence of weights used by [random\\_portfolios](#). See [generatesequence](#)
- `constraints` a list of constraints added to the portfolio object with [add.constraint](#)
- `objectives` a list of objectives added to the portfolio object with [add.objective](#)
- call the call to `portfolio.spec` with all of the specified arguments

**Value**

an object of class `portfolio`

**Author(s)**

Ross Bennett, Brian G. Peterson

**See Also**

[add.constraint](#), [add.objective](#), [optimize.portfolio](#)

**Examples**

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

---

portfolio\_risk\_objective

*constructor for class portfolio\_risk\_objective*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Author(s)**

Brian G. Peterson



---

position\_limit\_constraint  
*constructor for position\_limit\_constraint*

---

## Description

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

## Usage

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

## Arguments

<code>type</code>	character type of the constraint
<code>max_pos</code>	maximum number of assets with non-zero weights
<code>max_pos_long</code>	maximum number of assets with long (i.e. buy) positions
<code>max_pos_short</code>	maximum number of assets with short (i.e. sell) positions
<code>enabled</code>	TRUE/FALSE
<code>message</code>	TRUE/FALSE. The default is <code>message=FALSE</code> . Display messages if TRUE.
<code>...</code>	any other passthru parameters to specify position limit 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="position_limit", max_pos=3)  
pspec <- add.constraint(portfolio=pspec, type="position_limit", max_pos_long=3, max_pos_short=1)
```

pos\_limit\_fail      *function to check for violation of position limits constraints*

---

### Description

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

### Usage

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

### Arguments

<code>weights</code>	vector of weights to test
<code>max_pos</code>	maximum number of assets with non-zero weights
<code>max_pos_long</code>	maximum number of assets with long (i.e. buy) positions
<code>max_pos_short</code>	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

<code>portfolio</code>	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**

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

---

`print.portfolio`      *Printing Portfolio Specification Objects*

---

**Description**

Print method for class "portfolio"

**Usage**

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

**Arguments**

<code>portfolio</code>	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 <a href="#">portfolio.spec</a>
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 arbitrary 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 <a href="#">constraint</a>
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 arbitrary 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 <a href="#">constraint</a>
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

The return constraint specifies a target mean return value. This function is called by `add.constraint` when `type="return"` is specified, [add.constraint](#)

### Usage

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

### Arguments

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

### Author(s)

Ross Bennett

**See Also**

[add.constraint](#)

**Examples**

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

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

pspec <- add.constraint(portfolio=pspec, type="return", 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	<i>Transform a weights vector to satisfy leverage, box, group, and position_limit constraints using logic from randomize_portfolio</i>
--------------	--

---

### 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, cL0, 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
cL0	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 <a href="#">constraint</a>
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 <a href="#">portfolio.spec</a>
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	<i>Summarizing Portfolio Specification Objects</i>
-------------------	--

---

**Description**

summary method for class "portfolio"

**Usage**

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

**Arguments**

portfolio      object of class portfolio

**Author(s)**

Ross Bennett

---

trailingFUN	<i>apply a function over a configurable trailing period</i>
-------------	---

---

**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	<i>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 <a href="#">add.objective</a></i>
----------	--

---

**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	<i>constructor for turnover_constraint</i>
---------------------	--

---

**Description**

The turnover constraint specifies a target turnover value. This function is called by `add.constraint` when `type="turnover"` is specified, see [add.constraint](#). Turnover is calculated from a set of initial weights.

**Usage**

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

**Arguments**

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



**Details**

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

**Author(s)**

Ross Bennett

**See Also**

[add.constraint](#)

**Examples**

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

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

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

---

turnover\_objective      *constructor for class turnover\_objective*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**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**

<code>weights</code>	vector of weights
<code>min</code>	vector of minimum asset weights from box constraints
<code>max</code>	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**

<code>weights</code>	vector of weights
<code>groups</code>	vector of groups
<code>cLO</code>	vector of minimum group weights from group constraints
<code>cUP</code>	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**

<code>weights</code>	vector of weights
<code>max_pos</code>	maximum position of assets with non_zero weights
<code>nassets</code>	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**

<code>weights</code>	vector of weights
<code>min_sum</code>	minimum sum of asset weights
<code>max_sum</code>	maximum sum of asset weights

**Author(s)**

Ross Bennett

---

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

---

**Description**can we use the generic `update.default` function?**Usage**`update.constraint(object, ...)`**Arguments**

<code>object</code>	object of type <code>constraint</code> to update
<code>...</code>	any other passthru parameters, used to call <code>constraint</code>

**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**

<code>portfolio</code>	portfolio object passed into <code>optimize.portfolio</code>
<code>v1_constraint</code>	object of type <code>v1_constraint</code> passed into <code>optimize.portfolio</code>

**Value**

portfolio object containing constraints and objectives from v1\_constraint

**Author(s)**

Ross Bennett

**See Also**

[portfolio.spec](#), [add.constraint](#)

---

`weight_sum_constraint` *constructor for weight\_sum\_constraint*

---

**Description**

The constraint specifies the upper and lower bound that the weights sum to. This function is called by `add.constraint` when "weight\_sum", "leverage", "full\_investment", "dollar\_neutral", or "active" is specified as the type. see [add.constraint](#)

**Usage**

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

**Arguments**

<code>type</code>	character type of the constraint
<code>min_sum</code>	minimum sum of all asset weights, default 0.99
<code>max_sum</code>	maximum sum of all asset weights, default 1.01
<code>enabled</code>	TRUE/FALSE
<code>message</code>	TRUE/FALSE. The default is message=FALSE. Display messages if TRUE.
<code>...</code>	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

**See Also**[add.constraint](#)**Examples**

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

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

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

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

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

# Index

## \*Topic **datasets**

- indexes, 40
- add.constraint, 4, 7, 8, 25, 27, 35, 38, 39, 55–57, 65, 66, 72, 73, 77, 78
- add.objective, 6, 44, 46, 55, 56, 72
- add.objective\_v1 (add.objective), 6
- add.objective\_v2, (add.objective), 6
- applyFUN, 7
- box\_constraint, 4, 5, 7
- CCCGarch.MM, 8
- chart.Scatter.DE, 9
- chart.Scatter.GenSA, 10
- chart.Scatter.pso, 11
- chart.Scatter.ROI, 12
- chart.Scatter.RP, 13
- chart.Weights.DE, 13
- chart.Weights.GenSA, 14
- chart.Weights.pso, 15
- chart.Weights.ROI, 16
- chart.Weights.RP, 17
- charts.DE, 9, 18
- charts.GenSA, 18
- charts.pso, 19
- charts.ROI, 20
- charts.RP, 21
- constrained\_group\_tmp, 22
- constrained\_objective, 23
- constrained\_objective\_v2 (constrained\_objective), 23
- constraint, 6, 23, 24, 24, 28, 37, 47, 48, 62–64, 69, 76
- constraint\_ROI, 26
- constraint\_v2 (constraint), 24
- constraint\_v1, (constraint), 24
- DEoptim.control, 24
- diversification, 26
- diversification\_constraint, 4, 5, 27
- extract.efficient.frontier, 28
- extractObjectiveMeasures, 29
- extractStats, 18, 21, 22, 28, 29, 31, 32, 50, 51, 54
- extractStats.optimize.portfolio.DEoptim, 30
- extractStats.optimize.portfolio.GenSA, 30
- extractStats.optimize.portfolio.parallel, 31
- extractStats.optimize.portfolio.pso, 31
- extractStats.optimize.portfolio.random, 32
- extractStats.optimize.portfolio.ROI, 32
- extractWeights, 33
- extractWeights.optimize.portfolio, 33
- extractWeights.optimize.portfolio.rebalancing, 34
- factor\_exposure\_constraint, 4, 5, 35
- fn\_map, 36
- generatesequence, 25, 26, 37, 55
- get\_constraints, 37
- group\_constraint, 4, 5, 38
- group\_fail, 39
- indexes, 40
- insert\_constraints, 40
- insert\_objectives, 41
- is.constraint, 41
- is.objective, 42
- is.portfolio, 42
- minmax\_objective, 43
- name.replace, 44

objective, [7](#), [24](#), [37](#), [44](#), [63](#), [64](#)  
 optimize.portfolio, [9–22](#), [24](#), [29–34](#), [45](#),  
     [50–56](#)  
 optimize.portfolio.parallel, [31](#), [47](#)  
 optimize.portfolio.rebalancing, [33](#), [34](#),  
     [48](#)  
 optimize.portfolio\_v2  
     (optimize.portfolio), [45](#)  
 optimize.portfolio\_v2,  
     (optimize.portfolio), [45](#)  
 optimize\_portfolio\_v1  
     (optimize.portfolio), [45](#)  
  
 plot.optimize.portfolio, [50](#)  
 plot.optimize.portfolio.DEoptim, [51](#)  
 plot.optimize.portfolio.GenSA, [51](#)  
 plot.optimize.portfolio.pso, [52](#)  
 plot.optimize.portfolio.random, [53](#)  
 plot.optimize.portfolio.ROI, [54](#)  
 portfolio, [6](#)  
 portfolio (portfolio.spec), [55](#)  
 portfolio.spec, [4–7](#), [24](#), [38](#), [44](#), [47](#), [55](#), [63](#),  
     [64](#), [69](#), [77](#)  
 portfolio\_risk\_objective, [56](#)  
 pos\_limit\_fail, [58](#)  
 position\_limit\_constraint, [4](#), [5](#), [57](#)  
 print.constraint, [58](#)  
 print.optimize.portfolio.DEoptim, [59](#)  
 print.optimize.portfolio.GenSA, [59](#)  
 print.optimize.portfolio.pso, [60](#)  
 print.optimize.portfolio.random, [60](#)  
 print.optimize.portfolio.ROI, [61](#)  
 print.portfolio, [61](#)  
  
 random\_portfolio, [10](#), [12](#), [19](#), [20](#), [52](#), [54](#)  
 random\_portfolios, [32](#), [55](#)  
 random\_portfolios  
     (random\_portfolios\_v2), [64](#)  
 random\_portfolios\_v1, [63](#)  
 random\_portfolios\_v2, [64](#)  
 random\_walk\_portfolios, [65](#)  
 randomize\_portfolio, [63](#), [64](#), [68](#)  
 randomize\_portfolio  
     (randomize\_portfolio\_v2), [62](#)  
 randomize\_portfolio\_v1, [62](#)  
 randomize\_portfolio\_v2, [62](#), [64](#)  
 return\_constraint, [4](#), [5](#), [65](#)  
 return\_objective, [66](#)  
 risk\_budget\_objective, [67](#)  
  
 rp\_transform, [39](#), [58](#), [68](#)  
  
 set.portfolio.moments  
     (set.portfolio.moments\_v2), [69](#)  
 set.portfolio.moments\_v1, [69](#)  
 set.portfolio.moments\_v2, [46](#), [69](#)  
 summary.optimize.portfolio, [70](#)  
 summary.optimize.portfolio.rebalancing,  
     [70](#)  
 summary.portfolio, [71](#)  
  
 title, [14–22](#), [50–54](#)  
 trailingFUN, [71](#)  
 turnover, [72](#)  
 turnover\_constraint, [4](#), [5](#), [72](#)  
 turnover\_objective, [73](#)  
 txfrm\_box\_constraint, [74](#)  
 txfrm\_group\_constraint, [74](#)  
 txfrm\_position\_limit\_constraint, [75](#)  
 txfrm\_weight\_sum\_constraint, [75](#)  
  
 update.constraint, [76](#)  
 update\_constraint\_v1tov2, [76](#)  
  
 weight\_sum\_constraint, [4](#), [5](#), [77](#)