Package 'PortfolioAnalytics'

August 26, 2012

Type Package

Title Portfolio Analysis, including Numeric Methods for Optimization of Portfolios

Version 0.8.0

Date 2012-06-25

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,foreach,fGarch,Rglpk,quadprog,ROI,ROI.plugin.glpk,ROI.plugin.quadprog,pso,GenSA

License GPL

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

R topics documented:

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

Description

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

Usage

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

Arguments

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

Details

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

Author(s)

Brian G. Peterson

See Also

constraint

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

Description

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

Usage

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

Arguments

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

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

Description

classic risk return scatter of DEoptim results

Usage

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

Arguments

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

See Also

optimize.portfolio

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

Description

classic risk return scatter of random portfolios

Usage

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

Arguments

RP	set of portfolios created by optimize.portfolio
neighbors	set of 'neighbor' portfolios to overplot, see Details
return.col	string matching the objective of a 'return' objective, on vertical axis
risk.col	string matching the objective of a 'risk' objective, on horizontal axis
...	any other passthru parameters
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex
element.color	color for the default plot scatter points

See Also

optimize.portfolio

---

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

Description

boxplot of the weight distributions in the random portfolios

Usage

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

Arguments

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

See Also

optimize.portfolio

---

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

Description

boxplot of the weight distributions in the random portfolios

Usage

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

Arguments

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

See Also

optimize.portfolio

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

Description

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

Usage

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

Arguments

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

See Also

optimize.portfolio extractStats

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, risk.col, return.col, neighbors = NULL,
  main = "Random.Portfolios", ...)
```

Arguments

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

See Also

optimize.portfolio extractStats

---

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

Description

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

Usage

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

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
w	a vector of weights to test
constraints	an object of type "constraints" specifying the constraints for the optimization, see constraint
...	any other passthru parameters
trace	TRUE/FALSE whether to include debugging and additional detail in the output list
normalize	TRUE/FALSE whether to normalize results to min/max sum (TRUE), or let the optimizer penalize portfolios that do not conform (FALSE)
storage	TRUE/FALSE default TRUE for DEoptim with trace, otherwise FALSE. not typically user-called

Details

If the user has passed in either min_sum or max_sum constraints for the portfolio, or both, and are using a numerical optimization method like DEoptim, and normalize=TRUE, the default, we'll normalize the weights passed in to whichever boundary condition has been violated. If using random portfolios, all the portfolios generated will meet the constraints by construction. NOTE: this means that the weights produced by a numeric optimization algorithm like DEoptim might violate your constraints, so you'd need to renormalize them after optimizing We apply the same normalization in `optimize.portfolio` so that the weights you see have been normalized to min_sum if the generated portfolio is smaller than min_sum or max_sum if the generated portfolio is larger than max_sum. This normalization increases the speed of optimization and convergence by several orders of magnitude in many cases.

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

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

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

Additional parameters for random portfolios or `DEoptim.control` may be passed in via ...

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson

See Also

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

`constraint` *constructor for class constraint*

Description

constructor for class constraint

Usage

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

Arguments

assets	number of assets, or optionally a named vector of assets specifying seed weights
...	any other passthru parameters
min	numeric or named vector specifying minimum weight box constraints
max	numeric or named vector specifying minimum weight box constraints
min_mult	numeric or named vector specifying minimum multiplier box constraint from seed weight in assets
max_mult	numeric or named vector specifying maximum multiplier box constraint from seed weight in assets
min_sum	minimum sum of all asset weights, default .99
max_sum	maximum sum of all asset weights, default 1.01
weight_seq	seed sequence of weights, see generatesequence

Author(s)

Peter Carl and Brian G. Peterson

Examples

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

---

constraint_ROI *constructor for class constraint_ROI*

Description

constructor for class constraint_ROI

Usage

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

Arguments

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

Author(s)

Hezky Varon

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

Description

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

Usage

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

Arguments

`portfolios`	set of portfolios as generated by `extractStats`
`from`	minimum value of the sequence
`to`	maximum value of the sequence
`by`	number to increment the sequence by
`match.col`	string name of column to use for risk (horizontal axis)
`...`	any other passthru parameters
`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`constraints`	an object of type "constraints" specifying the constraints for the optimization, see `constraint`
`optimize_method`	one of "DEoptim" or "random"

Details

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

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

Description

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

Usage

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

Arguments

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

See Also

`optimize.portfolio`

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

Description

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

Usage

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

Arguments

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

See Also

`optimize.portfolio`

extractStats.optimize.portfolio.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.random

extract stats from random portfolio results

Description

This just flattens the $random_portfolio_objective_results part of the object

Usage

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

Arguments

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

See Also

`optimize.portfolio` `random_portfolios` `extractStats`

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

Description

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

Usage

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

Arguments

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

`extractWeights.rebal` *extract time series of weights from output of optimize.portfolio*

Description

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

Usage

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

Arguments

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

Details

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

See Also

`optimize.portfolio.rebalancing`

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

Description

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

Usage

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

Arguments

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

Details

The sequence created is not constrained by asset.

Author(s)

Peter Carl, Brian G. Peterson

See Also

constraint, objective

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)
```

`is.constraint` *check function for constraints*

Description

check function for constraints

Usage

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

Arguments

`x`	object to test for type `constraint`

Author(s)

bpeterson

`is.objective` *check class of an objective object*

Description

check class of an objective object

Usage

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

Arguments

`x`	an object potentially of type 'objective' to test

Author(s)

Brian G. Peterson

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

Description

utility function to replace awkward named from unlist

Usage

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

Arguments

`rnames`	character vector of names to check for cleanup

`objective` *constructor for class 'objective'*

Description

constructor for class 'objective'

Usage

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

Arguments

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

Author(s)

Brian G. Peterson

`optimize.portfolio` *wrapper for constrained optimization of portfolios*

Description

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

Usage

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

Arguments

`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`constraints`	an object of type "constraints" specifying the constraints for the optimization, see `constraint`, if using closed for solver, need to pass a `constraint_ROI` object.
`optimize_method`	one of "DEoptim", "random", "ROI","ROI_old", "pso", "GenSA". For using `ROI_old`, need to use a constraint_ROI object in constraints. For using `ROI`, pass standard `constratint` object in `constraints` argument. Presently, ROI has plugins for `quadprog` and `Rglpk`.
`search_size`	integer, how many portfolios to test, default 20,000
`trace`	TRUE/FALSE if TRUE will attempt to return additional information on the path or portfolios searched
`...`	any other passthru parameters
`rp`	matrix of random portfolio weights, default NULL, mostly for automated use by rebalancing optimization or repeated tests on same portfolios
`momentFUN`	the name of a function to call to set portfolio moments, default `set.portfolio.moments`

Details

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

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

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

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

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

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

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

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

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

Value

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

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson

`optimize.portfolio.parallel`

execute multiple optimize.portfolio calls, presumably in parallel

---

Description

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

Usage

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

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
constraints	an object of type "constraints" specifying the constraints for the optimization, see constraint
optimize_method	one of "DEoptim" or "random"
search_size	integer, how many portfolios to test, default 20,000
trace	TRUE/FALSE if TRUE will attempt to return additional information on the path or portfolios searched
...	any other passthru parameters
nodes	how many processes to run in the foreach loop, default 4

Details

This function will not speed up optimization!

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

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

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

Value

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

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson

`optimize.portfolio.rebalancing` *portfolio optimization with support for rebalancing or rolling periods*

Description

This function may eventually be wrapped into optimize.portfolio

Usage

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

Arguments

`R`	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
`constraints`	an object of type "constraints" specifying the constraints for the optimization, see `constraint`
`optimize_method`	one of "DEoptim" or "random"
`search_size`	integer, how many portfolios to test, default 20,000
`trace`	TRUE/FALSE if TRUE will attempt to return additional information on the path or portfolios searched
`...`	any other passthru parameters
`rp`	a set of random portfolios passed into the function, to prevent recalculation
`rebalance_on`	a periodicity as returned by xts function periodicity and usable by endpoints
`training_period`	period to use as training in the front of the data
`trailing_periods`	if set, an integer with the number of periods to roll over, default NULL will run from inception

Details

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

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

Value

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

Author(s)

Kris Boudt, Peter Carl, Brian G. Peterson

`plot.optimize.portfolio`

plot method for optimize.portfolio output

Description

scatter and weights chart for portfolio optimization

Usage

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

Arguments

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

Details

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

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

plot.optimize.portfolio.DEoptim

plot method for optimize.portfolio.DEoptim output

Description

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

Usage

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

Arguments

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

Details

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.random

plot method for optimize.portfolio.random output

Description

scatter and weights chart for random portfolios

Usage

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

Arguments

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

Details

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.

---

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 = FALSE, ...)
```

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

randomize_portfolio *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(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)

random_portfolios *generate an arbitary number of constrained random portfolios*

Description

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

Usage

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

Arguments

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

Value

matrix of random portfolio weights

Author(s)

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

See Also

constraint, objective, randomize_portfolio

Examples

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

---

random_walk_portfolios

deprecated random portfolios wrapper until we write a random trades function

---

Description

deprecated random portfolios wrapper until we write a random trades function

Usage

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

Arguments

...	any other passthru parameters

Author(s)

bpeterson

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 = FALSE, ...)
```

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 = FALSE, ...,
  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

---

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

Arguments

R	an xts, vector, matrix, data frame, timeSeries or zoo object of asset returns
constraints	an object of type "constraints" specifying the constraints for the optimization, see constraint
momentargs	list containing arguments to be passed down to lower level functions, default NULL
...	any other passthru parameters

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

trailingFUN *apply a function over a configurable trailing period*

Description

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

Usage

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

Arguments

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

Details

called with e.g.

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

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

Description

can we use the generic update.default function?

Usage

```
update.constraint(object, ...)
```

Arguments

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

Author(s)

bpeterson

Index