Reference Manual for pprobeSup

Placed in the public domain by Burns Statistics Limited 2013

May 2, 2013

Topics documented:

`pp.TTR.multmacd` *MACD estimation for multiple assets*

Description

Creates an object like the input prices containing MACD values.

Usage

```
pp.TTR.multmacd(x, ...)
```

Arguments

`x`	a matrix or `xts` object containing prices (or volume) of multiple assets. Note: prices, not returns.
`...`	additional arguments to the `MACD` function may be given.

Details

This is a wrapper for the `MACD` function in the `TTR` package.

Value

an object like the input `x` but with values from the `signal` column from the output of `MACD`.

Testing status

Not formally tested.

Revision

This help file was last revised 2013 May 02.

See Also

`MACD`, `pp.TTR.multsymbol`.

Examples

```
require(pprobeData)
xa.macd <- pp.TTR.multmacd(xassetPrices)
```

---

pp.TTR.multsymbol *Read data from Yahoo for multiple assets*

Description

A poorly implemented function to get a specific type of data from Yahoo for a number of assets.

Usage

```
pp.TTR.multsymbol(symbols, start, end, item = "Close",
        adjust = TRUE, verbose = TRUE)
```

Arguments

`symbols`	a character vector giving the symbols of the assets of interest.
`start`	an integer in the form of yyyymmdd giving the starting date.
`end`	an integer in the form of yyyymmdd giving the ending date.
`item`	a string giving the data to be retrieved.
`adjust`	a logical value. if `TRUE`, prices are adjusted for dividends and splits.
`verbose`	a logical value. if `TRUE`, then progress information is printed.

Details

This is an unsophisticated wrapper for the `getYahooData` function in the `TTR` package.

Value

An object of class `xts` with dates along rows and assets along columns.

Testing status

Not formally tested.

Revision

This help file was last revised 2013 May 02.

See Also

`pp.TTR.multmacd`.

Examples

```
# create price matrix
sp5.close <- as.matrix(pp.TTR.multsymbol(sp5.names, 20060101,
        20120701))
```

---

pp.cacheVar *Cache a set of variance matrices*

Description

Creates and stores a series of variance matrices estimated from the given return matrix.

Usage

```
pp.cacheVar(cache, times, retmat, lookback = 250, offset = 0,
        prefix = "var", FUN = var.shrink.eqcor, verbose = TRUE,
        ...)
```

Arguments

`cache`	a character string giving the name of the directory in which to store the files.
`times`	a vector of elements of the row names of `retmat` at which estimates of the variance are desired. The `lookback` and `offset` arguments limit the earliest time that is possible.
`retmat`	a matrix of returns for the assets with rows that include the range of times for which variances are desired.
`lookback`	a single integer giving the number of rows of `retmat` to use in each estimation.
`offset`	a single integer giving the number of rows to offset in the estimation. The default of 0 means that the time on the object containing a variance uses the returns with that time. A value of 1 would mean that the latest returns used for an object are 1 time before the time on object name.
`prefix`	a character string giving the prefix to use in the object names. The rest of the name is the corresponding element of `times`.

FUN	the function to estimate the variance.
verbose	logical value; if TRUE, then the progress is reported as each evaluation of FUN is completed.
...	additional arguments may be passed to FUN.

Details

While this was written with the intention of it being used to store variance matrices, there is no reason why FUN needs to return a variance matrix. The reason to use cacheing is if either the output is large and/or the computation takes a reasonably long time and the object will be used multiple times.

Value

NULL is returned invisibly.

Side Effects

R objects are saved into the cache directory with suffix rda.

Testing status

Not formally tested.

Revision

This help file was last revised 2013 May 02.

See Also

Examples

```
require(BurStFin)
require(pprobeData)
yearend <- rownames(xassetLogReturns)[cumsum(table(
        substr(rownames(xassetLogReturns), 1, 4)))]
pp.cacheVar("VarianceStash", yearend, xassetLogReturns)
```

---

pp.date.meanvarutil *Compute utilities of random portfolios over times*

Description

Returns a matrix of utilities that is the length of the dates argument by the number of random portfolios.

Usage

```
pp.date.meanvarutil(dates, ahead, risk.aversion, pricemat,
        alphamat = NULL, vardir = NULL, varprefix = "var",
        varconstraint = NULL, number.rand = 200,
        volatility.utility = FALSE, ...)
```

Arguments

`dates`	a character vector of times that must match row names of `pricemat` (and of `alphamat` if given).
`ahead`	a single integer stating the number of rows of prices that will be used to look ahead in the evaluation of the utility.
`risk.aversion`	a single number giving the risk aversion for the utility.
`pricemat`	a numeric matrix with times on the rows and assets along columns giving the prices of the assets throughout the period of interest.
`alphamat`	a numeric matrix with times on the rows and assets along columns giving the expected returns to be used (in the constraints). If `NULL` (the default), then no constraints on expected returns may be given.
`vardir`	a character string giving the directory where variances are cached. If `NULL` (the default), then either no variance constraints are allowed or the `variance` argument can be given which will mean a static variance throughout.
`varprefix`	the prefix for the variance objects that have been cached. This is ignored if `vardir` is `NULL`.
`varconstraint`	either `NULL` or a numeric vector of the same length as `dates` giving the variance constraint to be imposed at each time.
`number.rand`	a single integer, the number of random portfolios to generate at each time point.
`volatility.utility`	a logical value; if `TRUE`, then a mean-volatility utility is computed. If `FALSE` (the default), then a mean-variance utility is computed.
`...`	constraints (arguments to `random.portfolio`) need to be given. In particular the amount of money in the portfolios needs to be specified (though that is largely immaterial to the utility).

Value

a matrix with number of rows equal to the length of `dates` and the number of columns equal to `number.rand` containing the utility computed for each random portfolio over each time frame.

Testing status

Not formally tested.

Revision

This help file was last revised 2013 May 02.

See Also

`pp.cacheVar`.

Examples

```
# attach data
require(pprobeData)

# compute random portfolio utilities
xa.util20 <- pp.date.meanvarutil(rownames(xassetPrices)[
        seq(1, 1401, by=100)], ahead=60, risk.aver=2,
        pricemat=xassetPrices, gross=1e6, long.only=TRUE,
        port.size=20, max.weight=.1)

# plot density of utilities
plot(density(xa.util20))
```

---

pp.historyScenarios *Create scenarios from historical prices*

Description

Takes a matrix of prices and produces a three-dimensional array of scenarios based on the historical behavior.

Usage

```
pp.historyScenarios(history, template, number = NULL,
        which = NULL, prices = NULL, replace = TRUE)
```

Arguments

`history`	a matrix of positive numbers with column names for the assets. Missing values are not accepted.
`template`	vector with at least two non-negative values giving the position of the rows to be put into the scenarios relative to the starting position. This is typically in increasing order and begins with 0.
`number`	the number of scenarios to be generated with random starting positions. Only one of `number` and `which` can be given.
`which`	a vector of the locations of starting positions (rows in `history`) to be used. Only one of `number` and `which` can be given.
`prices`	a named numeric vector of the prices to be used as the starting prices. If not given, then the last row of `history` is used.
`replace`	logical value: when random locations are generated, are repeated values allowed?

Value

a three-dimensional array that is `length(template)` by `length(prices)` by either `number` or `length(which)`. The first row of all slices is equal to `prices`.

Side effects

if `number` is given, then the random seed is created or modified.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`pp.priceScenarios`, `pp.normalScenarios`, `scenario.optimizer`.

Examples

```
# attach data
require(pprobeData)

xas.histscenweek <- pp.historyScenarios(xassetPrices[, 1:50],
        template=c(0, 4), number=100)
```

---

pp.meanvarutil *Compute a Utility*

Description

Takes a matrix of portfolio valuations over time and computes a utility vector.

Usage

```
pp.meanvarutil(vals, risk.aversion)
pp.meanvolutil(vals, risk.aversion)
```

Arguments

`vals`	a matrix of portfolio valuations over time.
`risk.aversion`	a number giving the risk aversion for the utility.

Value

a vector of utilities.

Testing status

Not formally tested.

Revision

This help file was last revised 2013 May 02.

See Also

`pp.date.meanvarutil`.

`pp.normalScenarios` *Generate normally distributed scenarios*

Description

Takes prices, expected returns and a variance matrix – returns a three-dimensional array of scenarios.

Usage

```
pp.normalScenarios(prices, expected.return, variance, ntimes,
        nscenarios)
```

Arguments

`prices`	named numeric vector of asset prices.
`expected.return`	numeric vector of expected returns (for each time step). If named, then these are selected and put into the same order as `prices`.
`variance`	variance matrix of the returns (for each time step). If this has `dimnames`, then the rows and columns are selected and put into the same order as `prices`.
`ntimes`	the number of time steps in the result. This must be at least 2.
`nscenarios`	the number of scenarios to produce.

Value

a `ntimes` by `length(prices)` by `nscenarios` array of prices where the first row in each slice is the input `prices`.

Side effects

The random seed is created or modified.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`pp.historyScenarios`, `pp.priceScenarios`, `scenario.optimizer`.

Examples

```
# simple case
silly <- pp.normalScenarios(prices=c(A=34.57, B=92.12),
        expected.return=c(A=0, B=0), variance=diag(2) * .01,
        ntimes=4, nscenarios=5)

# load data
require(pprobeData)

lessSilly <- pp.normalScenarios(prices=xassetPrices[500, 1:50],
        expected.return=rep(0,50),
        variance=var(xassetLogReturns[1:250,1:50]),
        ntimes=63, nscenarios=100)
```

---

pp.priceScenarios *Set the initial prices for a scenario*

Description

Takes a scenario (matrix) and a price vector; returns the scenario with the initial prices set to the price vector.

Usage

```
pp.priceScenarios(scenarios, prices)
```

Arguments

`scenarios`	a matrix or three-dimensional array representing scenarios of prices through time. Times are in the rows, assets in the columns. If there is a third dimension, then that holds different scenarios.
`prices`	a vector of prices for the same assets (in the same order).

Value

a modified version of the input `scenarios` such that all the starting prices are equal to the input `prices`.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`pp.historyScenarios`, `scenario.optimizer`.

Examples

```
origScen <-  pp.normalScenarios(prices=c(A=34.57, B=92.12),
        expected.return=c(A=0, B=0), variance=diag(2) * .01,
        ntimes=4, nscenarios=5)

# same returns, new starting prices
newPriceScen <- pp.priceScenarios(origScen,
        prices=c(A=35.71, B=91.92))
```

---

pp.smallSelect *Careful selection of active assets*

Description

Goes to some effort to select the subset of the assets of a certain size that are the best in the optimization.

Usage

```
pp.smallSelect(prices, port.size, ..., sets = 10,
        finalStringency = 2)
```

Arguments

`prices`	named numeric vector of the prices to use in the optimization.
`port.size`	the exact number of assets that are to be in the portfolio. This is presumably a small positive integer.
`...`	the rest of the arguments that define the optimization.
`sets`	the number of unique combinations of assets to create and test. The full name of this argument must be given since it is after the dot-dot-dot argument.
`finalStringency`	the value of `stringency` to use when optimizing the problem given each specific set of assets. The full name of this argument must be given since it is after the dot-dot-dot argument.

Value

a list with the following components:

`optimum`	the result of `trade.optimizer` for the best optimization found.
`sets`	a `sets` by `port.size` matrix giving the assets in each selected set.
`call`	an image of the call that created the object.

Details

There are two stages. The first stage does `sets` optimizations where each optimization will not select any of the exact subsets that have already been selected.

The second stage again does `sets` optimizations but now the assets are specified so there is no selection aspect to the optimization. These optimizations set `stringency` to `finalStringency`.

The application that led to this function was looking for 3 to 6 assets in the portfolio, and it was quite desirable to have the best selection.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`trade.optimizer`.

Examples

```
# attach data
require(pprobeData)

# select best three assets out of 100
# a real problem would hopefully have a better expected return

ones <- xassetPrices[1, 1:100]
ones[] <- 1
bestThree <- pp.smallSelect(prices=ones,
        variance=var(xassetPrices[1:500, 1:100]),
        expected.return=colMeans(xassetLogReturns[1:500,1:100]),
        gross=10000 + c(-.5, .5), long.only=TRUE, max.weight=.5,
        port.size=3)

# look at the good asset combinations
bestThree$sets
sort(table(bestThree$sets))
```

---

pputil.fourmoments *Utility function of the first four moments*

Description

Returns a utility based on the scenarios that combines the first four moments via the parameters – a value for each random portfolio.

Usage

```
pputil.fourmoments(rp, scenarios, parameters, level = NULL,
        verbose = FALSE)
```

Arguments

`rp`	a random portfolio object.
`scenarios`	a three-dimensional array: times in rows, assets in columns and scenarios in the third dimension.
`parameters`	length 3 numeric vector that is coefficients for the variance, skewness and kurtosis. Normally the first and third should be negative and the second positive.
`level`	if `NULL`, then the mean utility is returned. Otherwise, should be a number between 0 and 1 giving the quantile of the utility to return.
`verbose`	logical value, if `TRUE`, then information is printed about the result.

Value

a numeric vector as long as the number of random portfolios. The utility for each portfolio and scenario is the mean log return (across time) plus the first parameter times the variance plus the second parameter times the skewness plus the third parameter times the kurtosis.

The value for each portfolio is the mean or `level` quantile of the utilities across scenarios.

Side effects

if `verbose` is `TRUE`, then printing is done.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`scenario.optimizer`, `pputil.meanvar`, `pputil.omega`, `pputil.value`.

Examples

```
fmOpt <- scenario.optimizer(multipleTimesScen,
        utility=pputil.fourmoments,
        extraArgs=list(parameters=c(-.8, 1, -.2)),
        max.weight=.04, gross=1e7, long.only=TRUE, port.size=50,
        existing=currentHoldings)
```

`pputil.meanvar` *Mean-variance utility function*

Description

Returns a mean-variance utility based on the scenarios – a value for each random portfolio.

Usage

```
pputil.meanvar(rp, scenarios, risk.aversion = 1, level = NULL,
        verbose = FALSE)
```

Arguments

`rp`	a random portfolio object.
`scenarios`	a three-dimensional array: times in rows, assets in columns and scenarios in the third dimension.
`risk.aversion`	a (positive) number giving the amount penalize variance relative to the mean (of log returns).
`level`	if `NULL`, then the mean utility is returned. Otherwise, should be a number between 0 and 1 giving the quantile of the utility to return.
`verbose`	logical value, if `TRUE`, then information is printed about the result.

Value

a numeric vector as long as the number of random portfolios. The utility for each portfolio and scenario is the mean log return (across time) minus the risk aversion times the variance.

The value for each portfolio is the mean or `level` quantile of the utilities across scenarios.

Side effects

if `verbose` is `TRUE`, then printing is done.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`scenario.optimizer`, `pputil.fourmoments`, `pputil.omega`, `pputil.value`.

Examples

```
mvOpt <- scenario.optimizer(multipleTimesScen,
        utility=pputil.meanvar,
        extraArgs=list(risk.aversion=1.4),
        max.weight=.04, gross=1e7, long.only=TRUE, port.size=50,
        existing=currentHoldings)

sameThing <- scenario.optimizer(multipleTimesScen,
        utility=pputil.fourmoments,
        extraArgs=list(parameters=c(-1.4, 0, 0)),
        max.weight=.04, gross=1e7, long.only=TRUE, port.size=50,
        existing=currentHoldings)
```

---

pputil.omega *Utility function for the Omega ratio*

Description

Returns a utility based on the scenarios for the omega ratio (or a weighted sum of omega ratios) – a value for each random portfolio.

Usage

```
pputil.omega(rp, scenarios, target, simple = FALSE,
        verbose = FALSE)
pputil.multomega(rp, scenarios, target, weights simple = FALSE,
        verbose = FALSE)
```

Arguments

`rp`	a random portfolio object.
`scenarios`	a three-dimensional array: times in rows, assets in columns and scenarios in the third dimension. For `pputil.omega` there should be exactly two time points. For `pputil.multomega` there should be one more time point than the length of `weights`.
`target`	the target return for the omega ratio.
`weights`	vector of weights to combine omega ratios from different times. There is no assumption that the weights sum to a specific value.
`simple`	logical value: if `TRUE`, then simple returns rather than log returns (the default) are used in the calculations.
`verbose`	logical value, if `TRUE`, then information is printed about the result.

Value

a numeric vector as long as the number of random portfolios. The utility for each portfolio is the omega ratio (across scenarios) for `pputil.omega`.

For `pputil.multomega` the omega ratio at each time of the returns is computed (across scenarios) and then the weighted mean of the omega ratios is returned.

Side effects

if `verbose` is `TRUE`, then printing is done.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`scenario.optimizer`, `pputil.fourmoments`, `pputil.meanvar`, `pputil.value`.

Examples

```
omegaOpt  <- scenario.optimizer(scenWith2timepoints,
        utility=pputil.omega, extraArgs=list(target=0.04),
        max.weight=.07, gross=1e7, long.only=TRUE,
        port.size=c(20,30))

multomegaOpt  <- scenario.optimizer(scenWith4timepoints,
        utility=pputil.omega,
        extraArgs=list(target=0.04, weights=c(3, 3, 4)),
        max.weight=.07, gross=1e7, long.only=TRUE,
        port.size=c(20,30))
```

`pputil.value` *Utility function of portfolio value*

Description

Returns a "utility" based on the scenarios of the portfolio value – one number for each random portfolio.

Usage

```
pputil.value(rp, scenarios, level = NULL, verbose = FALSE)
pputil.valueWt(rp, scenarios, sweights, level = NULL, verbose = FALSE)
```

Value

a numeric vector as long as the number of random portfolios. The utility for each portfolio and scenario is the portfolio value.

For `pputil.value` the mean or `level` quantile of the portfolio values (across scenarios) is returned for each random portfolio.

For `pputil.valueWt` the weighted mean or `level` weighted quantile of the portfolio values (across scenarios) is returned for each random portfolio.

Side effects

if `verbose` is `TRUE`, then printing is done.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`scenario.optimizer`, `pputil.fourmoments`, `pputil.meanvar`, `pputil.omega`.

Examples

```
# maximize expected value
meanvalueOpt  <- scenario.optimizer(scenWith1timepoint,
        utility=pputil.value, prices=currentPrices,
        max.weight=.07, gross=1e7, long.only=TRUE,
        existing=currentPositions, port.size=c(20,30))

# maximize 20th percentile of value
q20valueOpt  <- scenario.optimizer(scenWith1timepoint,
        utility=pputil.value, extraArgs=list(level=.2),
        prices=currentPrices,
        max.weight=.07, gross=1e7, long.only=TRUE,
        existing=currentPositions, port.size=c(20,30))

# weight scenarios, maximize 30th percentile
weightValueOpt  <- scenario.optimizer(scenWith1timepoint,
        utility=pputil.valueWt,
        extraArgs=list(level=.3, sweights=vectorLengthNumScenarios),
        prices=currentPrices,
        max.weight=.07, gross=1e7, long.only=TRUE,
        existing=currentPositions, port.size=c(20,30))

# load data
require(pprobeData)

# create a scenario for monthly values
monScen <- pp.historyScenarios(xassetPrices[,1:50],
        template=c(0,21), number=100)

# use the scenario (as a matrix)
monOPt <- scenario.optimizer(monScen[2,,], utility=pputil.value,
        prices=monScen[1,,1],
        max.weight=.07, gross=1e7, long.only=TRUE,
        existing=NULL, port.size=c(20,30)

# use the scenario (as a one-row 3D array)
monOPt <- scenario.optimizer(monScen[2,,, drop=FALSE],
        utility=pputil.value, prices=monScen[1,,1],
        max.weight=.07, gross=1e7, long.only=TRUE,
        existing=NULL, port.size=c(20,30)
```

`scenario.optimizer` *Scenario optimization of a portfolio*

Description

Uses random portfolios, a utility function (written in R) and an array of scenarios to optimize a portfolio.

Usage

```
scenario.optimizer(scenarios, utility, prices = NULL,
        existing = NULL, regulate = NULL, extraArgs = NULL,
        maximize = TRUE, verbose = TRUE, start.sol = NULL,
        checkPositive = TRUE, ...)
```

Arguments

`scenarios`	either a matrix or a three-dimensional array – most likely filled with asset prices. If a matrix, then the rows correspond to assets and need to be named with the asset identifiers. The columns represent different scenarios. If a 3-D array, then the rows are times, the columns are assets and the third diminesion is scenarios.
`utility`	a function that takes a random portfolio object as its first argument, it must have an argument called 'scenarios', and it may have additional arguments as well. It needs to return a single number for each random portfolio.
`prices`	named numeric vector of prices to be used in the generation of the random portfolios. If not given, then the first scenario (at the first time if there are multiple times) is used.
`existing`	named numeric vector giving the existing portfolio. The numbers are the number of units (shares, contracts, etc.) for the assets.
`regulate`	named numeric vector giving controls for the optimization process. See the Details section for specifics.
`extraArgs`	a list with named components giving additional arguments to the `utility` function.
`maximize`	logical value: do you want to maximize or minimize the `utility` function?
`verbose`	logical value: if `TRUE`, then the progress of the optimization is printed at each iteration.
`start.sol`	named numeric vector (or a portfolio object containing a `trade` component) giving a suggestion for a starting trade.
`checkPositive`	logical value: should `scenarios` be checked to all positive values and no missing values?
`...`	the constraints to be given to `random.portfolio` from Portfolio Probe. This minimally includes a specification of the amount of money in the portfolio.

Value

a list with components:

`new.portfolio`
: named numeric vector giving the number of units for each asset in the optimized portfolio.

`trade`
: named numeric vector giving the number of units for each asset in the trade from the existing portfolio to `new.portfolio`.

`existing`
: named numeric vector giving the number of units for each asset in the existing portfolio – the same as the input `existing`.

`utility.value`
: the value of the utility for the optimized portfolio.

`optim`
: a vector giving some details of the optimization process.

`violated`
: either `NULL` or a vector of strings giving the constraints that are violated by the solution.
This is only given when the `regulate` value `returnstart` is `TRUE`.

`timestamp`
: two character strings giving the time and date of the start and finish of the optimization.

`call`
: an image of the call that created the object.

Side effects

The S language random seed is changed or created.

Dependency

This depends explicitly on the `random.portfolio` function from Portfolio Probe. Utilities are likely to depend on the `valuation` method for random portfolio objects.

Details

OVERVIEW OF OPTIMIZATION PROCESS

There is an initial set of random portfolios that satisfy the problem constraints. The utility is evaluated on each of the random portfolios, and the portfolio with the best utility is selected as the candidate for the optimized portfolio.

Then some number of iterations is done. At each iteration some number of random portfolios is generated that are within some radius of best solution as well as satisfying the other constraints of the problem. If a better solution is found among the new random portfolios, then that becomes the new candidate solution. If none of the random portfolios have a better utility, then the radius is reduced.

The optimization terminates if either the radius becomes too small, or the limit on iterations is exhausted.

REGULATE VALUES

There are a number of values that can be given in the `regulate` argument:

initbitesize the number of random portfolios to generate initially.

nbites the maximum number of iterations to do.

bitesize the number of random portfolios to generate at each iteration.

radius the initial maximum trade distance from the best solution to any random portfolio in the iterations.

reduce when an iteration fails to improve the solution, then the new radius is the old radius times `reduce`.

minradius if the new radius is less than this number, then the optimization stops.

returnstart if `TRUE`, then no optimization is done but the result of `start.sol` being the trade is returned.

This is the exception that `regulate` must be numeric.

DISCLAIMER

The only sense in which the values in `regulate` are optimal is that minimum effort was put into deciding their values.

RETURN STARTING SOLUTION

To get the utility and other information for a particular trade, then the `regulate` value of `returnstart` must be `TRUE` and `start.sol` must be given.

WRITING YOUR OWN UTILITY

The utility functions that are provided are intended merely as examples. You can easily write your own utility function to satisfy your needs.

The function must take a first argument that is expected to be a random portfolio object. It must also have an argument named `scenarios`. It can have any additional arguments that you desire.

The other key requirement is that the function needs to return a numeric vector that is as long as the number of random portfolios given it.

Most likely a call to `valuation` (from Portfolio Probe) will be made using the random portfolios and the scenarios.

Several of the utility functions show how to get either the mean utility or a quantile of the utility. `pputil.valueWt` shows how to give different weights to different scenarios. `pputil.multomega` shows how to get a single utility by doing a weighted sum of utilities at different times. It is perfectly feasible to have both weighting schemes in a single utility function.

Restrictions

Distance constraints are not allowed to be imposed (for the real problem). It would be possible to lift this restriction by adding some ugly code.

Testing status

In test suite, mildly tested.

Revision

This help file was last revised 2013 May 02.

See Also

`pp.historyScenarios`, `pp.normalScenarios`, `pp.priceScenarios`

`pputil.fourmoments`, `pputil.meanvar`, `pputil.omega`, `pputil.value`.

Examples

```
# get the utility of a particular portfolio
omegaTest  <- scenario.optimizer(scenWith2timepoints,
        utility=pputil.omega, extraArgs=list(target=0.04),
        max.weight=.07, gross=1e7, long.only=TRUE,
        existing=NULL, port.size=c(20,30),
        start.sol=myPortfolio, regulate=c(returnstart=TRUE))

# see also the examples in the pputil.* help files
```

Index