Package ‘rportfolios’

August 19, 2016

Version 1.0-1

Date 2016-08-18

Title Random Portfolio Generation

Author Frederick Novomestky <fn334@nyu.edu>

Maintainer Frederick Novomestky <fn334@nyu.edu>

Depends R (>= 2.0.1), truncdist

Description A collection of tools used to generate
various types of random portfolios. The weights of these
portfolios are random variables derived from truncated
continuous random variables.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2016-08-19 13:37:41

R topics documented:

`collapse.segments` *Collapse a list or vectors of portfolio segments*

Description

This function returns a vector of investment indices from the given segments vector or list of vectors.

Usage

```
collapse.segments(segments)
```

Arguments

`segments`	A vector or list of vectors that defint the portfolio segments

Value

A vector of investment indices.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

Examples

```
###
### define the segments
###
    I <- list()
    I[[1]] <- c( 1, 2, 3 )
    I[[2]] <- c( 4, 5 )
    I[[3]] <- c( 6, 7 )
    I[[4]] <- c( 8, 9, 10 )
    collapse.segments( I )
    collapse.segments( I[[1]] )
    collapse.segments( I[[2]] )
    collapse.segments( I[[3]] )
    collapse.segments( I[[4]] )
```

---

extract.segments *Extract Investment Segment Exposures*

Description

This function extracts the investment exposures from one or more portfolios for the specified investment segments.

Usage

```
extract.segments(portfolios, segments, collapse = FALSE)
```

Arguments

`portfolios`	A vector or matrix that defines the portfolios
`segments`	A vector or list of vectors that defines the investment segments
`collapse`	A logical value. If TRUE, only the investment segment exposures are returned

Details

If the `collapse` argument is FALSE, the segment complement exposures are zero and the investment segment exposures are taken from the portfolios. If the `collapse` argument is TRUE, then only the investment segment exposures are returned. The private function `vector.extract.segments` is used to perform the extraction. For matrices of investment weights, the apply function is used with `vector.extract.segments` to obtain a matrix of extracted segment weights. The transpose of this matrix is returned.

Value

A vector for one portfolio or a matrix for multiple portfolios.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

Examples

```
onePortfolio <- random.longonly( 10 )
I <- list()
I[[1]] <- c( 1, 2, 3 )
I[[2]] <- c( 4, 5 )
I[[3]] <- c( 6, 7 )
I[[4]] <- c( 8, 9, 10 )
extract.segments( onePortfolio, I[[1]], FALSE )
extract.segments( onePortfolio, I[[1]], TRUE )
```

---

overweight.segments *Overweight Active Investment Segment Exposures*

Description

This function overweights the investment exposures of the given portfolios in the given investment segments by the proportion x_o of the total exposure in the segment complement.

Usage

```
overweight.segments(portfolios, segments, x.o)
```

Arguments

portfolios	A vector or matrix that defines the portfolios
segments	A vector or list of vectors that defines the investment segments
x.o	A positive real value for the proportion of total passive exposure allocated to the active exposures

Details

if $x_o = 0$, then the original portfolios are returned. If $x_o = 1$, then the total exposure of the segment complement, or passive segment, is allocated to the active investment segment of all the portfolios. The private function `vector.overweight.segments` does the actual work. If the argument `portfolios` is a matrix, then the apply function is used with private function to obtain a matrix of weights. The transpose of this matrix is returned.

Value

A vector of adjusted investment exposures for one portfolio or a matrix for more than one portfolio.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Grinold, R. C. and R. H. Kahn, 1999. *Active Portfolio Management: Quantitative Approach for Providing Superior Returns and Controlling Risk*, Second Edition, McGraw-Hill, New York, NY.

See Also

`segment.complement`

Examples

```
onePortfolio <- random.longonly( 10 )
I <- list()
I[[1]] <- c( 1, 2, 3 )
I[[2]] <- c( 4, 5 )
I[[3]] <- c( 6, 7 )
I[[4]] <- c( 8, 9, 10 )
overweight.segments( onePortfolio, I[[1]], 0 )
overweight.segments( onePortfolio, I[[1]], .1 )
```

`portfolio.composite` *Merge portfolios into a composite*

Description

This function merges a list of portfolios using a specified set of weights. The components in the list can be single portfolio vectors a matrix of portfolios.

Usage

```
portfolio.composite(portfolios, weights = NULL)
```

Arguments

`portfolios`	A list of vectors or matrices corresponding to portfolios of investments
`weights`	A numeric vector of weights for the components

Details

The private function `vector.composite` is used to create a single portfolio from a list of portfolio weight vectors. The private function `matrix.composite` generates the weighted composite matrix from a list of portfolio weight matrices.

Value

A vector or matrix.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

Examples

```
segments <- list()
segments[[1]] <- c( 1, 2, 3 )
segments[[2]] <- c( 4, 5 )
segments[[3]] <- c( 6, 7 )
segments[[4]] <- c( 8, 9, 10 )
weights <- c( .3, .2, .2, .3 )
vectors <- list()
matrices <- list()
for ( i in 1:4 ) {
vectors[[i]] <- random.longonly( 10, segments=segments[[i]] )
matrices[[i]] <- rlongonly( 200, 10, segments=segments[[i]] )
}
combined.vectors <- portfolio.composite( vectors, weights )
combined.matrices <- portfolio.composite( matrices, weights )
```

---

portfolio.difference *Portfolio Difference Measure*

Description

This function computes a measure of the difference between one or more portfolios and a benchmark portfolio.

Usage

```
portfolio.difference(portfolios, x.b, method = c("relative", "absolute"))
```

Arguments

portfolios	A numeric vector or matrix that defines the portfolio exposures
x.b	A numeric vector that defines the benchmark exposures
method	A character value that defines the difference measure

Details

The absolute deviation between a portfolio $\mathbf{x}$ and a benchmark $\mathbf{x_b}$ is denoted by $D_a\left(\mathbf{x},\mathbf{x}_b\right)$ and is computed as $D_a\left(\mathbf{x},\mathbf{x}_b\right) = \frac{1}{2}\sum_{i=1}^{n}\left|x_i - x_{b,i}\right|$.

The relative deviation between a portfolio and a benchmark is denoted by $D_r\left(\mathbf{x},\mathbf{x}_b\right)$ and is computed as $D_r\left(\mathbf{x},\mathbf{x}_b\right) = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|x_i - x_{b,i}\right|}{x_i + x_{b,i}}$.

The private function `vector.difference` performs the actual calculation of the difference based on the given method.

Value

A single numeric measure for one portfolio or a numeric vector for a collection of portfolios

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Worthington, A. C., 2009. Household Asset Portfolio Diversification: Evidence from the Household, Income and Labour Dynamics in Australia (Hilda) Survey, Working Paper, Available at SSRN: http://ssrn.com/abstract=1421567.

Examples

```
onePortfolio <- random.longonly( 100, 75 )
aBenchmark <- rep( 0.01, 100 )
portfolio.difference( onePortfolio, aBenchmark, method="absolute" )
portfolio.difference( onePortfolio, aBenchmark, method="relative" )
```

`portfolio.diversification` *Portfolio Diversification Measure*

Description

This function computes one of several portfolio diversification measures for a single portfolio or a collection of portfolios.

Usage

```
portfolio.diversification(portfolios, method = c("naive", "herfindahl",
"herfindahl-hirschman", "hannah-kay", "shannon"), alpha = 2)
```

Arguments

`portfolios`	a vector or matrix of portfolio exposures
`method`	a character value for the method used to compute the measure
`alpha`	a numeric value for parameter required for the Hannah-Kay measure

Details

The function ocmputes a portfolio diversification measure for a single portfolio or for a collection of portfolios organized as a matrix.

Value

A vector with one or more values.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Worthington, A. C., 2009. Household Asset Portfolio Diversification: Evidence from the Household, Income and Labour Dynamics in Australia (Hilda) Survey, Working Paper, Available at SSRN: http:////ssrn.com//abstract=1421567.

Examples

```
onePortfolio <- random.longonly( 100, 75 )
naive <- portfolio.diversification( onePortfolio, method = "naive" )
herfindahl <- portfolio.diversification( onePortfolio, method = "herfindahl" )
herfindahl.hirschman <- portfolio.diversification( onePortfolio, method = "herfindahl-hirschman" )
hannah.kay <- portfolio.diversification( onePortfolio, method = "hannah-kay" )
shannon <- portfolio.diversification( onePortfolio, method = "shannon" )
```

---

ractive — *Generate random active portfolios*

Description

This function generates m random actively managed portfolios relative to a given benchmark portfolio. Each portfolio is the combination of a benchmark portfolio and a notional neutral long short portfolio with given gross notional exposure. The number of non zero positions in the long short portfolios is k.

Usage

```
ractive(m, x.b, x.g, k = length(x.b), segments = NULL, max.iter = 2000,
eps = 0.001)
```

Arguments

`m`	A positive integer value for the number of portfolios in the sample
`x.b`	A numeric vector with the investment weights in the benchmark portfolio
`x.g`	A positive numeric value for the gross notional exposure in the long short portfolio
`k`	A positive integer value for the number of non zero positions in the long short portfolio
`segments`	A vector or list of vectors that defines the portfolio segments
`max.iter`	A positive integer value for the maximum iterations for the long short portfolio
`eps`	A small positive real value for the convergence criteria for the gross notional exposure

Details

The function executes the function `random.active` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step.

Value

A numeric $m \times n$ matrix. The rows are the portfolios and the columns are the investment weights for each portfolio

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Grinold, R. C. and R. H. Kahn, 1999. *Active Portfolio Management: Quantitative Approach for Providing Superior Returns and Controlling Risk*, Second Edition, McGraw-Hill, New York, NY.

Qian, E. E., R. H. Hua and E. H. Sorensen, 2007. *Quantitative Equity Portfolio Management*, Chapman \& Hall, London, UK.

Scherer, B., 2007. *Portfolio Construction and Risk Budgeting*, Third Edition, Risk Books, London, UK.

See Also

`random.active`

Examples

```
###
### benchmark consists of 20 equally weighted investments
###
x.b <- rep( 1, 30 ) / 30
###
### the gross notional exposure of the long short portfolio is a benchmark weight
###
x.g <- 1 / 30
###
### generate 100 random active portfolios with 30 non zero positions in the long short portfolios
###
x.matrix <- ractive( 100, x.b, x.g )
###
### generate 100 random active portfolios with 10 non zero positions in the long short portfolios
###
y.matrix <- ractive( 100, x.b, x.g, 10 )
```

ractive.test *Generate random active portfolios*

Description

This function generates m random actively managed portfolios relative to a given benchmark portfolio. Each portfolio is the combination of a benchmark portfolio and a notional neutral long short portfolio with given gross notional exposure. The number of non zero positions in the long short portfolios is k. The function is used to evaluate the computational performance of the portfolio generation algorithm.

Usage

```
ractive.test(m, x.b, x.g, k = length(x.b), segments = NULL, max.iter = 2000, eps = 0.001)
```

Arguments

m	A positive integer value for the number of portfolios in the sample
x.b	A numeric vector with the investment weights in the benchmark portfolio
x.g	A positive numeric value for the gross notional exposure in the long short portfolio
k	A positive integer value for the number of non zero positions in the long short portfolio
segments	A vector or list of vectors that defines the portfolio segments
max.iter	A positive integer value for the maximum iterations for the long short portfolio
eps	A small positive real value for the convergence criteria for the gross notional exposure

Details

The function executes the function `random.active.test` using the R function `lapply`. The result is a list containing the investment weight vector and the number of iterations. These data are stored in a matrix of investment weights and a vector of iterations. This list is returned.

Value

A list with two named components.

xmatrix	An $m \times n$ matrix of investment weights
iters	An $m \times 1$ vector with the number of iterations used to obtain the portfolios

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Grinold, R. C. and R. H. Kahn, 1999. *Active Portfolio Management: Quantitative Approach for Providing Superior Returns and Controlling Risk*, Second Edition, McGraw-Hill, New York, NY.

Qian, E. E., R. H. Hua and E. H. Sorensen, 2007. *Quantitative Equity Portfolio Management*, Chapman \& Hall, London, UK.

Scherer, B., 2007. *Portfolio Construction and Risk Budgeting*, Third Edition, Risk Books, London, UK.

See Also

random.active.test

Examples

```
###
### benchmark consists of 20 equally weighted investments
###
x.b <- rep( 1, 30 ) / 30
###
### the gross notional exposure of the long short portfolio is a benchmark weight
###
x.g <- 1 / 30
###
### generate 100 random active portfolios with 30 non zero positions in the long short portfolios
###
x.matrix <- ractive.test( 100, x.b, x.g )
###
### generate 100 random active portfolios with 10 non zero positions in the long short portfolios
###
y.matrix <- ractive.test( 100, x.b, x.g, 10 )
```

---

random.active *Random actively managed portfolio*

Description

This function generates an actively managed random portfolio relative to a given benchmark portfolio. The active portfolio is the sum of the benchmark portfolio and a notional neutral long short portfolio with given gross notional exposure. The number of non zero positions in the long short portfolio is k.

Usage

```
random.active(x.b, x.g, k = length( x.b ), segments = NULL, max.iter = 2000,
              eps = 0.001)
```

Arguments

`x.b`	A numeric vector with the investment weights in the benchmark portfolio
`x.g`	A positive numeric value for the gross notional exposure in the long short portfolio
`k`	A positive integer value for the number of non zero positions in the long short portfolio
`segments`	A vector or list of vectors that defines the portfolio segments
`max.iter`	A positive integer value for the maximum iterations for the long short portfolio
`eps`	A small positive real value for the convergence criteria for the gross notional exposure

Details

The algorithm uses the function `random.longshort` to generate long portfolios that have identical total long and short exposures equal to one half the given gross notional exposure `x.g`. The resultant portfolio `x.ls` is algebraically added to the benchmark portfolio `x.b`.

Value

An $n \times 1$ numeric vector with the investment weights.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Jacobs, B. I. and K. N. Levy, 1997. The Long and Short of Long-Short Investing, *Journal of Investing*, Spring 1997, 73-86.

Jacobs, B. I., K. N. Levy and H. M. Markowitz, 2005. Portfolio Optimization with Factors, Scenarios and Realist Short Positions, *Operations Research*, July/August 2005, 586-599.

See Also

`random.longshort`

Examples

```
###
### the benchmark portfolios consists of 30 equally weighted investments
###
x.b <- rep( 1, 30 ) / 30
###
### the gross notional exposure of the long short portfolio is a benchmark weight
###
x.g <- 1 / 30
###
### generate 100 active portfolios with 30 non zero positions in the long short portolios
```

```
###
x <- random.active( x.b, x.g )
###
### generate 100 active portfolios with 10 non zero positions in the long short portolios
###
y <- random.active( x.b, x.g, 10 )
```

`random.active.test` *Random actively managed portfolio*

Description

This function generates an actively managed random portfolio relative to a given benchmark portfolio. The active portfolio is the sum of the benchmark portfolio and a notional neutral long short portfolio with given gross notional exposure. The number of non zero positions in the long short portfolio is k. The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
random.active.test(x.b, x.g, k = length(x.b), segments = NULL, max.iter = 2000,
eps = 0.001)
```

Arguments

`x.b`	A numeric vector with the investment weights in the benchmark portfolio
`x.g`	A positive numeric value for the gross notional exposure in the long short portfolio
`k`	A positive integer value for the number of non zero weights in the long short portfolio
`segments`	A vector or list of vectors that defines the portfolio segments
`max.iter`	A positive integer value for the maximum iterations for generating the long short portfolio
`eps`	A small positive real value for the convergence criteria for the gross notional exposure

Details

The algorithm uses the function `random.longshort.test` to generate long portfolios that have identical total long and short exposures equal to one half the given gross notional exposure `x.g`. The resultant portfolio `x.ls` is algebraically added to the benchmark portfolio `x.b`.

Value

A list with two named components.

`x`	An $n \times 1$ numerical vector of investment weights
`iter`	An integer value for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Grinold, R. C. and R. H. Kahn, 1999. *Active Portfolio Management: Quantitative Approach for Providing Superior Returns and Controlling Risk*, Second Edition, McGraw-Hill, New York, NY.

Qian, E. E., R. H. Hua and E. H. Sorensen, 2007. *Quantitative Equity Portfolio Management*, Chapman \& Hall, London, UK.

Scherer, B., 2007. *Portfolio Construction and Risk Budgeting*, Third Edition, Risk Books, London, UK.

See Also

random.longshort

Examples

```
###
### benchmark consists of 20 equally weighted investments
###
x.b <- rep( 1, 30 ) / 30
###
### gross notion exposure is one of the investment weights
###
x.g <- 1 / 30
###
### generate 100 active portfolios with 30 non zero positions in the long short portfolio
###
x.result <- random.active.test( x.b, x.g )
###
### generate 100 active portfolios with 10 non zero positions in the long short portfolio
###
y.result <- random.active.test( x.b, x.g, 10 )
```

---

random.benchmark *Random Naive Benchmark Portfolios*

Description

This function generates a vector of investment weights for a benchmark portfolio where the weights are non-negative and the sume of the weights is a given total. The weights are naively derived from an i.i.d. sample of positively truncated random variables.

Usage

```
random.benchmark(n = 2, k = n, segments = NULL, x.t = 1,
margins = c("unif", "beta", "exp", "frechet",
"gamma", "gev", "gpd", "gumbel", "lnorm", "logis", "norm",
"weibull"), ...)
```

Arguments

`n`	A positive integer for the number of investments in the portfolio
`k`	A positive integer for the number of non-zero exposures or cardinality
`segments`	A vector or list of vectors that defines the investment segments
`x.t`	A positive real value for the sum of the investment exposures
`margins`	A character value for the underlying distribution of the truncated random variable. The default is a uniform distribution
`...`	Other arguments passed to the random variate simulation function

Details

If the `segments` argument is a NULL value, then the benchmark has full cardinality, $k = n$, or partial cardinality, $k < n$. In the case of partial cardinality, an investment segment is defined by a simple random sample without replacement of k investment indices from the n investments. When the `segments` argument is not NULL, the investment segment is constructed from the argument. The investment segment is represented by the set A with cardinality k. If argument `k` and `segments` are not specified, then then $A = \{i|1 \leq i \leq n\}$ For the k non-zero investment exposures, a random sample of size k is drawn from the truncated random variable, $S_i \quad i \in A$. The non-zero investment exposures are given by $x_i = S_i \Bigg/ \sum_{j \in A} S_j, \, i \in A$.

Currently, there are twelve truncated distributions available. They are the uniforn (the default), beta, exponential, Frechet, gamma, generalized extreme value (gev), generalized Pareto (gpd), Gumbel, log normal, logistic, normal and Weibull distributions. Random samples are truncated to the positive half of the real line.

Value

A numeric $n \times 1$ vector of investment exposures.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Qian, E. E., R. H. Hua and E. H. Sorensen, 2007. *Quantitative Equity Portfolio Management*, Chapman \& Hall, London, UK.

See Also

`random.benchmark.test`

Examples

```
###
### long only portfolio of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are uniform
###
p.1 <- random.benchmark( 30 )
###
### long only portfolio of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are uniform
###
p.2 <- random.benchmark( 30, 10 )
###
### long only portfolio of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are log normal
### with zero log mean and unit log standard deviation
###
p.3 <- random.benchmark( 30, margins="lnorm", meanlog=0, sdlog=1 )
###
### long only portfolio of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are log normal
### with zero log mean and unit log standard deviation
###
p.4 <- random.benchmark( 30, 10, margins="lnorm", meanlog=0, sdlog=1 )
```

`random.benchmark.test` *Random Naive Benchmark Portfolio*

Description

This function generates a vector of investment weights for a benchmark portfolio where the weights are non-negative and the sume of the weights is a given total. The weights are naively derived from an i.i.d. sample of truncated random variables. This function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
random.benchmark.test(n = 2, k = n, segments = NULL, x.t = 1,
margins = c("unif", "beta", "exp", "frechet",
"gamma", "gev", "gpd", "gumbel", "lnorm", "logis", "norm",
"weibull"), ...)
```

Arguments

`n`	A positive integer for the number of investments in the portfolio
`k`	A positive integer for the number of non-zero exposures or cardinality
`segments`	A vector or list of vectors that defines the investment segments
`x.t`	A positive real value for the sum of the investment exposures

`margins`	A character value for the underlying distribution of the truncated random variable. The default is a uniform distribution
`...`	Other arguments passed to the random variate simulation function

Details

The details are described in the function `random.benchmark`.

Value

A list with two named components.

`x`	An $m \times n$ numerical vector of investment weights
`iter`	An integer value for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

`random.benchmark`

Examples

```
###
### long only portfolio of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are uniform
###
p.1 <- random.benchmark.test( 30 )
###
### long only portfolio of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are uniform
###
p.2 <- random.benchmark.test( 30, 10 )
###
### long only portfolio of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are log normal
### with zero log mean and unit log standard deviation
###
p.3 <- random.benchmark.test( 30, margins="lnorm", meanlog=0, sdlog=1 )
###
### long only portfolio of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are log normal
### with zero log mean and unit log standard deviation
###
p.4 <- random.benchmark.test( 30, 10, margins="lnorm", meanlog=0, sdlog=1 )
```

random.bounded *Random bounded portfolio*

Description

This function generates a portfolio of n investments where the weights are constrained to be within investment specific lower and upper bounds.

Usage

```
random.bounded(n = 2, x.t = 1, x.l = rep(0, n), x.u = rep(x.t, n), max.iter = 1000)
```

Arguments

`n`	An integer value for the number of investments in the portfolio
`x.t`	Numeric value for the sum of the investment weights
`x.l`	Numeric vector for the lower bounds on the investment weights
`x.u`	Numeric vector for the upper bound on the investment weights
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The simulation method is an extension the method in the function `random.longonly`. The desired portfolio $\mathbf{x} = \left[\begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} \right]'$ is defined $\mathbf{x} = \mathbf{x}_l + \mathbf{z}$, that is, the sum of a portfolio of lower bounds with total allocation $\mathbf{1}'\,\mathbf{x}_l$ and the portfolio $\mathbf{z}$ with total allocation $x_t - \mathbf{1}'\,x_l$. This second portfolio has non-negative weights and upper bounds equal to the range vector $\mathbf{x}_r = \mathbf{x}_u - \mathbf{x}_l$.

In the function `random.longonly`, all investment weights have the same lower and upper bounds. In `random.bounded` investment weights can have different bounds. Therefore, rather than performing a random sampling without replacement of the weights, `random.bounded` begins with the selection of the indices $i_1, i_2, \ldots, i_n$ as a random sample without replacement of the set of investment weight subscripts. The subscript of the index sample defines the order in which the random weights are generated. The allocations in $\mathbf{z}$ are scaled uniform random variables.

After completing this acceptance rejection procedure, the function determines any unallocated surplus which is the total allocation minus the sum of the lower bounds. If there is any surplus, then the allocation gap is computed as the difference of the upper bounds and the current investment allocations. Investments are chosen at random and minimum of the surplus and the gap is added to the allocation. The surplus is reduced by this amount and the adjustment is performed for each investment.

Value

A numeric vector with investment weights.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Cheng, R. C. H., 1977. The Generation of Gamma Variables with Non-integral Sape Parameter, *Journal of the Royal Statistical Society*, Series C (Applied Statistics), 26(1), 71.

Kinderman, A. J. and J. G. Ramage, 1976. Computer Generation of Normal Random Variables, *Journal of the American Statistical Association*, December 1976, 71(356), 893.

Marsaglia, G. and T. A. Bray, 1964. A Convenient method for generating normal variables, *SIAM Review*, 6(3), July 1964, 260-264.

Ross, S. M. (2006). *Simulation*, Fourth Edition, Academic Press, New York NY.

Tadikamalla, P. R., (1978). Computer generation of gamma random variables - II, Communications of the ACM, 21 (11), November 1978, 925-928.

See Also

random.longonly

Examples

```
###
### standard long only portfolio
###
p.1 <- random.bounded( 30, 1 )
###
### 3% lower bound for all investments
### 100% upper bound for all investments
###
x.lb.all.3 <- rep( 0.03, 30 )
x.ub.all.100 <- rep( 1, 30 )
p.2 <- random.bounded( 30, 1, x.l = x.lb.all.3, x.u= x.ub.all.100 )
###
### 4% upper bound for all investments
### 3% lower bound for all investments
x.ub.all.4 <- rep( 0.04, 30 )
p.3 <- random.bounded( 30, 1, x.l = x.lb.all.3, x.u = x.ub.all.4 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
p.4 <- random.bounded( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.100 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.5 <- random.bounded( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.100 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
```

```
p.6 <- random.bounded( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.4 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.7 <- random.bounded( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.4 )
###
### 3% lower bound for all investments
### 4% upper bound for 1-10, 5% upper bound for 11-20 and 4% upper boundfor 21-30
###
x.ub.4.5.4 <- c( rep( 0.04, 10 ), rep( 0.05, 10 ), rep( 0.04, 10 ) )
p.8 <- random.bounded( 30, 1, x.l = x.lb.all.3, x.u= x.ub.4.5.4 )
###
### 3% lower bound for all investments
### 5% upper bound for 1-10, 4% upper bound for 11-20 and 5% upper bound for 21-30
###
x.ub.5.4.5 <- c( rep( 0.05, 10 ), rep( 0.04, 10 ), rep( 0.05, 10 ) )
p.9 <- random.bounded( 30, 1, x.l = x.lb.all.3, x.u= x.ub.5.4.5 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.10 <- random.bounded( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.4.5.4 )
###
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.11 <- random.bounded( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.4.5.4 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.12 <- random.bounded( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.5.4.5 )
###
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.13 <- random.bounded( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.5.4.5 )
```

`random.bounded.test` *Random bounded portfolio*

Description

This function generates a portfolio of n investments where the weights are constrained to be within investment specific lower and upper bounds.

Usage

```
random.bounded.test(n = 2, x.t = 1, x.l = rep(0, n), x.u = rep(x.t, n), max.iter = 1000)
```

Arguments

`n`	An integer value for the number of investments in the portfolio
`x.t`	Numeric value for the sum of the investment weights
`x.l`	Numeric vector for the lower bounds on the investment weights
`x.u`	Numeric vector for the upper bound on the investment weights
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The simulation method is an extension the method in the function `random.longonly`. The desired portfolio $\mathbf{x} = \left[\begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array}\right]'$ is defined $\mathbf{x} = \mathbf{x}_l + \mathbf{z}$, that is, the sum of a portfolio of lower bounds with total allocation $\mathbf{1}'\,\mathbf{x}_l$ and the portfolio $\mathbf{z}$ with total allocation $x_t - \mathbf{1}'\,x_l$. This second portfolio has non-negative weights and upper bounds equal to the range vector $\mathbf{x}_r = \mathbf{x}_u - \mathbf{x}_l$.

In the function `random.longonly`, all investment weights have the same lower and upper bounds. In `random.bounded` investment weights can have different bounds. Therefore, rather than performing a random sampling without replacement of the weights, `random.bounded` begins with the selection of the indices $i_1, i_2, \ldots, i_n$ as a random sample without replacement of the set of investment weight subscripts. The subscript of the index sample defines the order in which the random weights are generated. The allocations in $\mathbf{z}$ are scaled uniform random variables.

After completing this acceptance rejection procedure, the function determines any unallocated surplus which is the total allocation minus the sum of the lower bounds. If there is any surplus, then the allocation gap is computed as the difference of the upper bounds and the current investment allocations. Investments are chosen at random and minimum of the surplus and the gap is added to the allocation. The surplus is reduced by this amount and the adjustment is performed for each investment.

Value

A numeric vector with investment weights.

Author(s)

Frederick Novomestky `<fn334@nyu.edu>`

References

Cheng, R. C. H., 1977. The Generation of Gamma Variables with Non-integral Sape Parameter, *Journal of the Royal Statistical Society*, Series C (Applied Statistics), 26(1), 71.

Kinderman, A. J. and J. G. Ramage, 1976. Computer Generation of Normal Random Variables, *Journal of the American Statistical Association*, December 1976, 71(356), 893.

Marsaglia, G. and T. A. Bray, 1964. A Convenient method for generating normal variables, *SIAM Review*, 6(3), July 1964, 260-264.

Ross, S. M. (2006). *Simulation*, Fourth Edition, Academic Press, New York NY.

Tadikamalla, P. R., (1978). Computer generation of gamma random variables - II, Communications of the ACM, 21 (11), November 1978, 925-928.

See Also

random.longonly

Examples

```
###
### standard long only portfolio
###
p.1 <- random.bounded.test( 30, 1 )
###
### 3% lower bound for all investments
### 100% upper bound for all investments
###
x.lb.all.3 <- rep( 0.03, 30 )
x.ub.all.100 <- rep( 1, 30 )
p.2 <- random.bounded.test( 30, 1, x.l = x.lb.all.3, x.u= x.ub.all.100 )
###
### 4% upper bound for all investments
### 3% lower bound for all investments
x.ub.all.4 <- rep( 0.04, 30 )
p.3 <- random.bounded.test( 30, 1, x.l = x.lb.all.3, x.u = x.ub.all.4 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
p.4 <- random.bounded.test( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.100 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.5 <- random.bounded.test( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.100 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
p.6 <- random.bounded.test( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.4 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.7 <- random.bounded.test( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.4 )
###
### 3% lower bound for all investments
### 4% upper bound for 1-10, 5% upper bound for 11-20 and 4% upper boundfor 21-30
###
x.ub.4.5.4 <- c( rep( 0.04, 10 ), rep( 0.05, 10 ), rep( 0.04, 10 ) )
p.8 <- random.bounded.test( 30, 1, x.l = x.lb.all.3, x.u= x.ub.4.5.4 )
###
```

```
### 3% lower bound for all investments
### 5% upper bound for 1-10, 4% upper bound for 11-20 and 5% upper bound for 21-30
###
x.ub.5.4.5 <- c( rep( 0.05, 10 ), rep( 0.04, 10 ), rep( 0.05, 10 ) )
p.9 <- random.bounded.test( 30, 1, x.l = x.lb.all.3, x.u= x.ub.5.4.5 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.10 <- random.bounded.test( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.4.5.4 )
###
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.11 <- random.bounded.test( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.4.5.4 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.12 <- random.bounded.test( 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.5.4.5 )
###
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.13 <- random.bounded.test( 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.5.4.5 )
```

---

random.equal *Random equal weighted portfolios*

Description

This function generates a random portfolio of n investments in which there are only k positive equal weights. The weights sum to the given value x_t.

Usage

```
random.equal(n = 2, k = n, segments = NULL, x.t = 1)
```

Arguments

n	A positive integer for the number of investments in the portfolio
k	A positive integer for the number of investments with positive equal weights
segments	A vector or list of vectors that defines the portfolio segments
x.t	A positive numeric value for the sum of weights

Details

The R function `sample` is used to generate a simple random sample without replacement of k values from the integers $1, 2, \ldots, n$. These are the subscripts into an $n \times 1$ zero vector to assign the equal weight x_t/k.

Value

An $n \times 1$ numeric vector of investment weights for the equal weighted portfolio. The weights are proportions of invested capital.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

Examples

```
x <- random.equal( 30, 5 )
```

---

random.equal.test *Random equal weighted portfolios*

Description

This function generates a random portfolio of n investments in which there are only k positive equal weights. The sum of the weights is x_t. The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
random.equal.test(n = 2, k = n, segments = NULL, x.t = 1)
```

Arguments

`n`	A positive integer for the number of investments in the portfolio
`k`	A positive integer for the number of investments with positive equal weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t`	The sum of the investment weights

Details

The R function `sample` is used to generate a simple random sample without replacement of k values from the integers $1, 2, \ldots, n$. These are the subscripts into an $n \times 1$ zero vector to assign the equal weight x_t/k.

Value

A list with two named components.

`x`	An $n \times 1$ numerical vector of investment weights
`iter`	An integer value for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Evans, J. and S. Archer, 1968. Diversification and the Reduction of Risk: An Empirical Analysis, *Journal of Finance*, 23, 761-767.

Upson, R. B., P. F. Jessup and K. Matsumoto, 1975. Portfolio Diversification Strategies, *Financial Analysts Journal*, 31(3), 86-88.

Elton, E. J. and M. J. Gruber, 1977. Risk Reduction and Portfolio Size: An Analytical Solution, *Journal of Business*, 50(4), 415-437.

Bird, R. and M. Tippett, 1986. Naive Diversification and Portfolio Risk - A Note, *Management Science*, 32(2), 244-251.

Statman, M., 1987. How many stocks make a diversified portfolio, *Journal of Financial and Quantitative Analysis*, 22, 353-363.

Newbould, G. D. and P. S. Poon, 1993. The minimum number of stocks needed for diversification, *Financial Practice and Education*, 3, 85-87.

O'Neal, E. S., 1997. How Many Mutual Funds Constitute a Diversified Mutual Fund Portfolio, *Financial Analysts Journal*, 53(2), 37-46.

Statman, M., 2004. The diversification puzzle, *Financial Analysts Journal*, 60, 48-53.

Benjelloun, H. and Siddiqi, 2006. Direct diversification with small stock portfolios. *Advances in Investment Analysis and Portfolio Management*, 2, 227-252.

Benjelloun, H., 2010. Evans and Archer - forty years later, *Investment Management and Financial Innovation*, 7(1), 98-104.

Examples

```
###
### equally weighted portfolio of 30 investments of which 5 are non-zero and
### the rest are zero.  the weights sum to 1.
###
result <- random.equal.test( 30, 5 )
```

---

random.general — *Random general portfolio*

Description

This function generates a general random portfolio of n investments with k long or short positiobs, The probability that a a non-zero investment weight is positive is p. The maximum absolute exposure for any investment is x.u. The default value is 1.

Usage

```
random.general(n = 2, k=n, segments=NULL, p = 0.5, x.u = 1)
```

Arguments

`n`	A positive integer value for the number of investments in the portfolio
`k`	A positive integer value for the number of non-zero positions
`segments`	A vector or list of vectors that defines the portfolio segments
`p`	A positive numeric value for the probability that an investment weight is positive
`x.u`	A positive numeric value for the maximum absolute exposure to an investment

Details

If $k < n$ the function `random.general` is recursively called with n set equal to k to obtain a $k \times 1$ vector of non-zero long and short weights. The R function `sample` is used to generate a simple random sample without replacement of k values from the integers $1, 2, \ldots, n$. These are the subscripts into an $n \times 1$ zero vector to assign the k non-zero weights. This vector is returned.

If $k = n$, the R function `rbinom` is used to generate a vector of plus and minus ones corresponding to the long and short positions. The R function `runif` is used to generate uniformly distributed values between 0 and 1. These are scaled by x.u and then multiplied by the signs. The sum of the investment weights is not restricted.

Value

An $n \times 1$ numeric vector of investment weights for the equal weighted portfolio.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

Examples

```
###
### long only portfolio of 30 investments with 30 non zero positions
###
x.long <- random.general( 30, p=1.0 )
###
### long only portfolio of 30 investments with 10 non zero positions
###
y.long <- random.general( 30, 10, p=1.0 )
###
### short only portfolio of 30 investments with 30 non zero positions
###
x.short <- random.general( 30, p=0.0 )
###
### short only portfolio of 30 investments with 10 non zero positions
###
y.short <- random.general( 30, 10, p=1.0 )
###
### long short portfolio of 30 investments with 30 non zero positions
###
x.long.short <- random.general( 30, p=0.5 )
###
```

```
### long short portfolio of 30 investments with 10 non zero positions
###
y.long.short <- random.general( 30, 10, p=0.5 )
###
### long bias portfolio of 30 investments with 30 non zero positions
###
x.long.bias <- random.general( 30, p=0.7 )
###
### long bias portfolio of 30 investments with 10 non zero positions
###
y.long.bias <- random.general( 30, 10, p=0.7 )
###
### short bias portfolio of 30 investments with 30 non zero positions
###
x.short.bias <- random.general( 30, p=0.3 )
###
### short bias portfolio of 30 investments with 10 non zero positions
###
y.short.bias <- random.general( 30, 10, p=0.3 )
```

`random.general.test` *Random general portfolio*

Description

This function generates a general random portfolio of n investments with k long and short positions. The probability that a non-zero investment weight is positive is p. The maximum absolute exposure for any investment is x.u. The default value is 1. The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
random.general.test(n = 2, k = n, segments = NULL, p = 0.5, x.u = 1)
```

Arguments

`n`	A positive integer value for the number of investments in the portfolio
`k`	A positive integer value for the number of non-zero positions
`segments`	A vector or list of vectors that defines the portfolio segments
`p`	A positive numeric value for the probability that an investment weight is positive
`x.u`	A positive numeric value for the maximum absolute exposure to an investment

Details

If $k < n$ the function `random.general.test` is recursively called with n set equal to k to obtain a $k \times 1$ vector of non-zero long and short weights. The R function `sample` is used to generate a simple random sample without replacement of k values from the integers $1, 2, \ldots, n$. These are the subscripts into an $n \times 1$ zero vector to assign the k non-zero weights. This vector is returned.

If $k = n$, the R function `rbinom` is used to generate a vector of plus and minus ones corresponding to the long and short positions. The R function `runif` is used to generate uniformly distributed values between 0 and 1. These are scaled by x.u and then multiplied by the signs. The sum of the investment weights is not restricted.

Value

A list with two named components.

`x`	An $n \times 1$ numerical vector of investment weights
`iter`	An integer value for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

Examples

```
###
### long only portfolio of 30 investments with 30 non-zero positions
###
result.x.long <- random.general.test( 30, p=1.0 )
###
### long only portfolio of 30 investments with 10 non-zero positions
###
result.y.long <- random.general.test( 30, 10, p=1.0 )
###
### short only portfolio of 30 investments with 30 non-zero positions
###
result.x.short <- random.general.test( 30, p=0.0 )
###
### short only portfolio of 30 investments with 10 non-zero positions
###
result.y.short <- random.general.test( 30,10,  p=0.0 )
###
### long short portfolio of 30 investments with 30 non-zero positions
###
result.x.long.short <- random.general.test( 30, p=0.5 )
###
### long short portfolio of 30 investments with 10 non-zero positions
###
result.y.long.short <- random.general.test( 30, 10, p=0.5 )
###
### long bias portfolio of 30 investments with 30 non-zero positions
###
result.x.long.bias <- random.general.test( 30, p=0.7 )
###
### long bias portfolio of 30 investments with 10 non-zero positions
###
result.y.long.bias <- random.general.test( 30, 10, p=0.7 )
###
```

```
### short bias portfolio of 30 investments with 30 non-zero positions
###
result.x.short.bias <- random.general.test( 30, p=0.3 )
###
### short bias portfolio of 30 investments with 10 non-zero positions
###
result.y.short.bias <- random.general.test( 30, 10, p=0.3 )
```

`random.longonly` *Random long only portfolio*

Description

This function generates a vector of investment weights for a portfolio where the weights are non-negative, do not exceed a given upper and and the sum of the weights is a given total. The number of non zero positions is k.

Usage

```
random.longonly(n = 2, k = n, segments = NULL, x.t = 1, x.l=0,
x.u = x.t, max.iter = 1000)
```

Arguments

`n`	An integer value for the number of investments in the portfolio
`k`	An integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t`	Numeric value for the sum of the investment weights
`x.l`	Numeric value for the lower bound on an investment weight
`x.u`	Numeric value for the upper bound on an investment weight
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The simulation method combines the acceptance rejection method used for generating gamma and gaussian random variables with a continuous analog of the method used in Ross (2006) to generate a vector of multinomial random variables. $n-1$ random variables are constructed where the first U_1 is uniformly distributed in the interval $[X_l, X_t]$. Random variable U_2 is a uniform random variable in $[X_l, X_t - U_1]$ given U_1. Random variable U_3 is a uniform random variable in $[0, X_t - U_1 - U_2]$ given U_1 and U_2. This conditional generation of uniform random variables stops with U_{n-1} which is uniform on $\left[X_l, X_t - \sum_{j=1}^{n-2} U_j\right]$ given the first $n-2$ random variables. if $X_t - \sum_{j=1}^{n-1} U_j$ is less than or equal to X_u, then the final random variable is $U_n = X_t - \sum_{j=1}^{n-1} U_j$. Otherwise, the above procedure of generating uniform random variables conditionally is repeated until this condition is satisfied. The vector $\mathbf{W}$ is a random sample of size n of the values in vector $\mathbf{X}$ where the sampling is performed without replacement.

Value

A numeric vector with investment weights.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Cheng, R. C. H., 1977. The Generation of Gamma Variables with Non-integral Sape Parameter, *Journal of the Royal Statistical Society*, Series C (Applied Statistics), 26(1), 71.

Kinderman, A. J. and J. G. Ramage, 1976. Computer Generation of Normal Random Variables, *Journal of the American Statistical Association*, December 1976, 71(356), 893.

Marsaglia, G. and T. A. Bray, 1964. A Convenient method for generating normal variables, *SIAM Review*, 6(3), July 1964, 260-264.

Ross, S. M. (2006). *Simulation*, Fourth Edition, Academic Press, New York NY.

Tadikamalla, P. R., (1978). Computer generation of gamma random variables - II, Communications of the ACM, 21 (11), November 1978, 925-928.

Examples

```
###
### long only portfolio of 30 investments with 30 non-zero positions
###
x <- random.longonly( 30 )
###
### long only portfolio of 30 investments with 10 non-zero positions
###
y <- random.longonly( 30, 10 )
```

`random.longonly.test` *Random long only portfolio*

Description

This function generates a vector of investment weights for a portfolio where the weights are non-negative, do not exceed a given upper and and the sum of the weights is a given total. The number of non zero positions is k. This function is used to evaluation the computational performance of the portfolio generation algorithm.

Usage

```
random.longonly.test(n = 2, k = n, segments = NULL, x.t = 1, x.l=0,
x.u = x.t, max.iter = 1000)
```

Arguments

`n`	An integer value for the number of investments in the portfolio
`k`	An integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t`	Numeric value for the sum of the investment weights
`x.l`	Numeric value for the lower bound on an investment weight
`x.u`	Numeric value for the upper bound on an investment weight
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The simulation methods combines the acceptance rejection method used for generating gamma and gaussian random variables with a continuous analog of the method used in Ross (2006) to generate a vector of multinomial random variables. $n-1$ random variables are constructed where the first U_1 is uniformly distributed in the interval $[X_l, X_t]$. Random variable U_2 is a uniform random variable in $[X_l, X_t - U_1]$ given U_1. Random variable U_3 is a uniform random variable in $[0, X_t - U_1 - U_2]$ given U_1 and U_2. This conditional generation of uniform random variables stops with U_{n-1} which is uniform on $\left[X_l, X_t - \sum_{j=1}^{n-2} U_j\right]$ given the first $n-2$ random variables. if $X_t - \sum_{j=1}^{n-1} U_j$ is less than or equal to X_u, then the final random variable is $U_n = X_t - \sum_{j=1}^{n-1} U_j$. Otherwise, the above procedure of generating uniform random variables conditionally is repeated until this condition is satisfied. The vector $\mathbf{W}$ is a random sample of size n of the values in vector $\mathbf{U}$ where the sampling is performed without replacement.

Value

A list with two named components.

`xmatrix`	An $m \times n$ matrix of investment weights
`iters`	An $m \times 1$ vector with the number of iterations used to obtain the portfolios

A list with two named components.

`x`	An $n \times 1$ numerical vector of investment weights
`iter`	An integer value for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky `<fn334@nyu.edu>`

References

Cheng, R. C. H., 1977. The Generation of Gamma Variables with Non-integral Sape Parameter, *Journal of the Royal Statistical Society*, Series C (Applied Statistics), 26(1), 71.

Kinderman, A. J. and J. G. Ramage, 1976. Computer Generation of Normal Random Variables, *Journal of the American Statistical Association*, December 1976, 71(356), 893.

Marsaglia, G. and T. A. Bray, 1964. A Convenient method for generating normal variables, *SIAM Review*, 6(3), July 1964, 260-264.

Ross, S. M. (2006). *Simulation*, Fourth Edition, Academic Press, New York NY.

Tadikamalla, P. R., (1978). Computer generation of gamma random variables - II, Communications of the ACM, 21 (11), November 1978, 925-928.

Examples

```
###
### long only portfolio of 30 investments with 30 non-zero positions
###
result.x <- random.longonly.test( 30 )
###
### long only portfolio of 30 investments with 10 non-zero positions
###
result.y <- random.longonly.test( 30, 10 )
```

---

random.longshort *Generate random long short porfolio*

Description

This function generates a vector of investment weights for a long short portfolio where the the gross notional exposure is x.t.long + x.t.short and the net notional exposure is x.t.long - x.t.short. There are k non-zero positions in the portfolio.

Usage

```
random.longshort(n = 2, k = n, segments = NULL, x.t.long = 1, x.t.short = x.t.long,
max.iter = 2000, eps = 0.001)
```

Arguments

`n`	A positive integer value for the number of investments in the portfolio
`k`	A positive integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t.long`	A positive real value for the sum of the long exposures
`x.t.short`	A positive real value for the sum of the absolute value of the short exposures
`max.iter`	A positive integer value for the maximum iterations in the acceptance rejection method
`eps`	A small positive real value for the convergence criteria for the gross notional exposure

Details

The function implements an algorithm in which the outer structure is the iterative acceptance rejection method. Within each iteration, the R function `random.longonly` is used to construct a long only investment weight vector `x.long` where the sum of these weights is `x.t.long`. The R function `random.shortonly` is used to construct a short only investment eight vector `random.short` such that the sum of the absolute value of these weights is `x.t.long`. The sum of these two weight vectors, `x.longshort`, satisfies the net notional requirement of the desired portfolio. If the absolute value of computed gross notiona exposure for `x.longshort` minus $x.t.long + x.t.short$ is less than the argument `eps`, then the desired portfolio is generated and result is returned. Otherwise, the process is repeated within the acceptance rejection loop until (1) the required portfolio is generated or 2 the iteration limit is exceeded.

Value

An $n \times 1$ vector of investment weights for the long short portfolio.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Jacobs, B. I. and K. N. Levy, 1997. The Long and Short of Long-Short Investing, *Journal of Investing*, Spring 1997, 73-86.

Jacobs, B. I., K. N. Levy and H. M. Markowitz, 2005. Portfolio Optimization with Factors, Scenarios and Realist SHort Positions, *Operations Research*, July/August 2005, 586-599.

See Also

`random.longonly`, `random.shortonly`

Examples

```
###
### long short portfolio of 30 investments with 30 non-zero positions
###
x <- random.longshort( 30 )
###
### long short portfolio of 30 investments with 10 non-zero positions
###
y <- random.longshort( 30, 10 )
```

random.longshort.test *Random long short portfolio test*

Description

This function generates a vector of investment weights for a portfolio with a given net and gross notional exposure. There are k non-zero positions in the portfolio. The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
random.longshort.test(n = 2, k = n, segments = NULL, x.t.long = 1, x.t.short = x.t.long,
max.iter = 2000, eps = 0.001)
```

Arguments

`n`	A positive integer value for the number of investments in the portfolio
`k`	A positive integer value for the number of non zero positions
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t.long`	A positive real value for the sum of the long exposures
`x.t.short`	A positive real value for the sum of the absolute value of the short exposures
`max.iter`	A positive integer value for the maximum iterations in the acceptance rejection method
`eps`	A small positive real value for the convergence criteria for the gross notional exposure

Details

The function uses the same portfolio generation method described in `random.longshort`. The arguments `x.t`, `x.t.long` and `x.t.short` are proportions of total invested capital.

Value

A list with two named components.

`x`	An $n \times 1$ numerical vector of investment weights
`iter`	An integer value for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.eu>

References

Jacobs, B. I. and K. N. Levy, 1997. The Long and Short of Long-Short Investing, *Journal of Investing*, Spring 1997, 73-86.

Jacobs, B. I., K. N. Levy and H. M. Markowitz, 2005. Portfolio Optimization with Factors, Scenarios and Realist SHort Positions, *Operations Research*, July/August 2005, 586-599.

See Also

random.longonly, random.longshort, random.shortonly

Examples

```
###
### long short portfolio of 30 investments with 30 non-zero positions
###
x.result <- random.longshort.test( 30 )
###
### long short portfolio of 30 investments with 10 non-zero positions
###
x.result <- random.longshort.test( 30, 10 )
```

---

random.shortonly *Random short only portfolio*

Description

This function generates a vector of investment weights for a portfolio where the weights are non-positive, absolute weights do not exceed a given upper and and the sum of the absolute weights weights is a given total. The number of non zero positions in the portfolio is k.

Usage

```
random.shortonly(n = 2, k = n, segments = NULL, x.t = 1, x.l = 0,
x.u = x.t, max.iter = 1000)
```

Arguments

n	A positive integer value for the number of investments in the portfolio
k	A positive integer value for the number of non zero weights
segments	A vector or list of vectors that defines the portfolio segments
x.t	A positive numeric value for the sum of the absolute value of investment weights
x.l	A positive numeric value for the lower bound on the absolute value of investment weights
x.u	A positive numeric value for the upper bound on the absolute value of investment weights
max.iter	A positive integer value for the maximum iterations in the rejection method

Details

The function random.longonly is used to generate a long only portfolio that satisfies the lower bound, upper bound and sum of weight conditions. The value returned is a vector with the opposite signs.

Value

An $n \times 1$ numeric vector of investment weights for the short only portfolio.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

random.longonly

Examples

```
###
### generate short only portfolio of 30 investments with 30 non-zero positions
###
x <- random.shortonly( 30 )
###
### generate short only portfolio of 30 investments with 10 non-zero positions
###
y <- random.shortonly( 30, 10 )
```

---

random.shortonly.test *Random short only portfolio*

Description

This function generates a vector of investment weights for a portfolio where the weights are non-positive, absolute weights do not exceed a given upper and and the sum of the absolute weights is a given total. The number of non zero positions in the portfolio is k. The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
random.shortonly.test(n = 2, k = n, segments = NULL, x.t = 1, x.l = 0,
x.u = x.t, max.iter = 1000)
```

Arguments

`n`	An integer value for the number of investments in the portfolio
`k`	An integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t`	Numeric value for the sum of the absolute value of the investment weights
`x.l`	Numeric value for the lower bound on the absolute value of an investment weight
`x.u`	Numeric value for the upper bound on the absolute value of an investment weight
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The function uses `random.longonly.test` to generate a long only portfolio in test mode. The component x compute is used to define the short portfolio. The short portfolio together with the component iter, the number of iterations used to construct the long only portfolio, are stored in a list of named components.

Value

A list with two named components

`x`	An $n \times 1$ numerical vector of investment weights
`iter`	An integer value for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

`random.shortonly`

Examples

```
###
### generate a short only portfolio of 30 investments with 30 non-zero positions
###
x.result <- random.shortonly.test( 30 )
###
### generate a short only portfolio of 30 investments with 10 non-zero positions
###
y.result <- random.shortonly.test( 30, 10 )
```

rbenchmark — *Generate random naive benchmark portfolios*

Description

This function generates m random long only benchmark portfolios with n investments where the sume of the weights equals a given amount. The weights are naively derived from an i.i.d. sample of truncated random variables.

Usage

```
rbenchmark(m, n = 2, k = n, segments = NULL, x.t = 1,
margins = c("unif", "beta", "exp", "frechet",
"gamma", "gev", "gpd", "gumbel", "lnorm", "logis", "norm",
"weibull"), ...)
```

Arguments

`m`	A positive integer value for the number of portfolios
`n`	A positive integer for the number of investments in the portfolio
`k`	A positive integer for the number of non-zero exposures or cardinality
`segments`	A vector or list of vectors that defines the investment segments
`x.t`	A positive real value for the sum of the investment exposures
`margins`	A character value for the underlying distribution of the truncated random variable. The default is a uniform distribution
`...`	Other arguments passed to the random variate simulation function

Details

The function executes the function `random.benchmark` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step.

Value

A numeric $m \times n$ matrix. The rows are the portfolios and the columns are the investment weights for each portfolio

Author(s)

Frederick Novomestky `<fn334@nyu.edu>`

See Also

`random.benchmark`

Examples

```
###
### 100 long only portfolios of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are uniform
###
p.1.matrix <- rbenchmark( 100, 30 )
###
### 100 long only portfolios of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are uniform
###
p.2.matrix <- rbenchmark( 100, 30, 10 )
###
### 100 long only portfolios of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are log normal
### with zero log mean and unit log standard deviation
###
p.3.matrix <- rbenchmark( 100, 30, margins="lnorm", meanlog=0, sdlog=1 )
###
### 100 long only portfolios of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are log norm
### with zero log mean and unit log standard deviation
###
p.4.matrix <- rbenchmark( 100, 30, 10, margins="lnorm", meanlog=0, sdlog=1 )
```

`rbenchmark.test` *Generate random naive benchmark portfolios*

Description

This function generates m random long only benchmark portfolios with n investments where the sume of the weights equals a given amount. The weights are naively derived from an i.i.d. sample of truncated random variables. This function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
rbenchmark.test(m, n = 2, k = n, segments = NULL, x.t = 1,
margins = c("unif", "beta", "exp", "frechet",
"gamma", "gev", "gpd", "gumbel", "lnorm", "logis", "norm",
"weibull"), ...)
```

Arguments

`m`	A positive integer value for the number of portfolios
`n`	A positive integer for the number of investments in the portfolio
`k`	A positive integer for the number of non-zero exposures or cardinality
`segments`	A vector or list of vectors that defines the investment segments

`x.t`	A positive real value for the sum of the investment exposures
`margins`	A character value for the underlying distribution of the truncated random variable. The default is a uniform distribution
`...`	Other arguments passed to the random variate simulation function

Details

The function executes the function `random.benchmark.test` using the R function `lapply`. The result which is a list contains the investment weight vectors and number of iterations. Thse data are stored in a matrix of investment weights and a vector of iterations. These arrays are returned as a list.

Value

A list with two named components.

`xmatrix`	A numerical $m \times n$ matrix of investment weights
`iters`	An integer $m \times 1$ vector for the number iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

`random.benchmark.test`

Examples

```
###
### 100 long only portfolios of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are uniform
###
p.1.result <- rbenchmark.test( 100, 30 )
###
### 100 long only portfolios of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are uniform
###
p.2.result <- rbenchmark.test( 100, 30, 10 )
###
### 100 long only portfolios of 30 investments with 30 non-zero positions
### the margins of the truncated random variables are log normal
### with zero log mean and unit log standard deviation
###
p.3.result <- rbenchmark.test( 100, 30, margins="lnorm", meanlog=0, sdlog=1 )
###
### 100 long only portfolios of 30 investments with 10 non-zero positions
### the margins of the truncated random variables are log norm
### with zero log mean and unit log standard deviation
###
```

```
p.4.result <- rbenchmark.test( 100, 30, 10, margins="lnorm", meanlog=0, sdlog=1 )
```

---

rbounded *Random bounded portfolios*

Description

This function generates m portfolios of n investments where the weights are constrained to be within investment specificd lower and upper bounds.

Usage

```
rbounded(m, n = 2, x.t = 1, x.l = rep(0, n), x.u = rep(x.t, n), max.iter = 1000)
```

Arguments

`m`	An integer value for the number of portfolios to be generated
`n`	An integer value for the number of investments in the portfolio
`x.t`	Numeric value for the sum of the investment weights
`x.l`	Numeric vector for the lower bounds on the investment weights
`x.u`	Numeric vector for the upper bound on the investment weights
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The function executes the function `random.bounded` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step.

Value

A numeric $m \times n$ matrix. The rows are the portfolios and the columns are the investment weights for each portfolio

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Cheng, R. C. H., 1977. The Generation of Gamma Variables with Non-integral Sape Parameter, *Journal of the Royal Statistical Society*, Series C (Applied Statistics), 26(1), 71.

Kinderman, A. J. and J. G. Ramage, 1976. Computer Generation of Normal Random Variables, *Journal of the American Statistical Association*, December 1976, 71(356), 893.

Marsaglia, G. and T. A. Bray, 1964. A Convenient method for generating normal variables, *SIAM Review*, 6(3), July 1964, 260-264.

Ross, S. M. (2006). *Simulation*, Fourth Edition, Academic Press, New York NY.

Tadikamalla, P. R., (1978). Computer generation of gamma random variables - II, Communications of the ACM, 21 (11), November 1978, 925-928.

See Also

random.bounded

Examples

```
###
### standard long only portfolio
###
p.1.matrix <- rbounded( 400, 30, 1, rep( 0, 30), rep( 1, 30 ) )

###
### 3% lower bound for all investments
### 100% upper bound for all investments
###
x.lb.all.3 <- rep( 0.03, 30 )
x.ub.all.100 <- rep( 1, 30 )
p.2.matrix <- rbounded( 400, 30, 1, x.l = x.lb.all.3, x.u= x.ub.all.100 )
###
### 4% upper bound for all investments
### 3% lower bound for all investments
x.ub.all.4 <- rep( 0.04, 30 )
p.3.matrix <- rbounded( 400, 30, 1, x.l = x.lb.all.3, x.u = x.ub.all.4 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
p.4.matrix <- rbounded( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.100 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.5.matrix <- rbounded( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.100 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
p.6.matrix <- rbounded( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.4 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.7.matrix <- rbounded( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.4 )
###
### 3% lower bound for all investments
### 4% upper bound for 1-10 5% for 11-20 and 4% for 21-30
###
x.ub.4.5.4 <- c( rep( 0.04, 10 ), rep( 0.05, 10 ), rep( 0.04, 10 ) )
p.8.matrix <- rbounded( 400, 30, 1, x.l = x.lb.all.3, x.u= x.ub.4.5.4 )
```

```
###
### 3% lower bound for all investments
### 5% upper bound for 1-10 4% for 11-20 and 5% for 21-30
###
x.ub.5.4.5 <- c( rep( 0.05, 10 ), rep( 0.04, 10 ), rep( 0.05, 10 ) )
p.9.matrix <- rbounded( 400, 30, 1, x.l = x.lb.all.3, x.u= x.ub.5.4.5 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.10.matrix <- rbounded( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.4.5.4 )
###
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.11.matrix <- rbounded( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.4.5.4 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.12.matrix <- rbounded( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.5.4.5 )
###
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.13.matrix <- rbounded( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.5.4.5 )
```

`rbounded.test` *Random bounded portfolios*

Description

This function generates m portfolios of n investments where the weights are constrained to be within investment specificd lower and upper bounds. This function is used to evaluation the computational performance of the portfolio generation algorithm.

Usage

```
rbounded.test(m, n = 2, x.t = 1, x.l = rep(0, n), x.u = rep(x.t, n), max.iter = 1000)
```

Arguments

`m`	An integer value for the number of portfolios to be generated
`n`	An integer value for the number of investments in the portfolio
`x.t`	Numeric value for the sum of the investment weights
`x.l`	Numeric vector for the lower bounds on the investment weights
`x.u`	Numeric vector for the upper bound on the investment weights
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The function executes the function `random.bounded` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step.

Value

A list with two named components.

`xmatrix`	An $m \times n$ matrix of investment weights
`iters`	An $m \times 1$ vector with the number of iterations used to obtain the portfolios

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Cheng, R. C. H., 1977. The Generation of Gamma Variables with Non-integral Sape Parameter, *Journal of the Royal Statistical Society*, Series C (Applied Statistics), 26(1), 71.

Kinderman, A. J. and J. G. Ramage, 1976. Computer Generation of Normal Random Variables, *Journal of the American Statistical Association*, December 1976, 71(356), 893.

Marsaglia, G. and T. A. Bray, 1964. A Convenient method for generating normal variables, *SIAM Review*, 6(3), July 1964, 260-264.

Ross, S. M. (2006). *Simulation*, Fourth Edition, Academic Press, New York NY.

Tadikamalla, P. R., (1978). Computer generation of gamma random variables - II, Communications of the ACM, 21 (11), November 1978, 925-928.

See Also

`random.bounded`

Examples

```
###
### standard long only portfolio
###
p.1.matrix <- rbounded.test( 400, 30, 1 )

###
### 3% lower bound for all investments
### 100% upper bound for all investments
###
x.lb.all.3 <- rep( 0.03, 30 )
x.ub.all.100 <- rep( 1, 30 )
p.2.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.all.3, x.u= x.ub.all.100 )
###
### 4% upper bound for all investments
### 3% lower bound for all investments
x.ub.all.4 <- rep( 0.04, 30 )
```

```
p.3.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.all.3, x.u = x.ub.all.4 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
p.4.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.100 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 100% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.5.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.100 )
###
### 2% lower bound for 1-10, 3% lower bound for 11-20, 2% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.2.3.2 <- c( rep( 0.02, 10 ), rep( 0.03, 10 ), rep( 0.02, 10 ) )
p.6.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.all.4 )
###
### 3% lower bound for 1-10, 2% lower bound for 11-20, 3% lower bound for 21-30
### 4% upper bound for all investments
###
x.lb.3.2.3 <- c( rep( 0.03, 10 ), rep( 0.02, 10 ), rep( 0.03, 10 ) )
p.7.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.all.4 )
###
### 3% lower bound for all investments
### 4% upper bound for 1-10 5% for 11-20 and 4% for 21-30
###
x.ub.4.5.4 <- c( rep( 0.04, 10 ), rep( 0.05, 10 ), rep( 0.04, 10 ) )
p.8.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.all.3, x.u= x.ub.4.5.4 )
###
### 3% lower bound for all investments
### 5% upper bound for 1-10 4% for 11-20 and 5% for 21-30
###
x.ub.5.4.5 <- c( rep( 0.05, 10 ), rep( 0.04, 10 ), rep( 0.05, 10 ) )
p.9.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.all.3, x.u= x.ub.5.4.5 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.10.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.4.5.4 )
###
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 4% upper bound for 1-10  5% for 11-20  4% for 21-30
###
p.11.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.4.5.4 )
###
### 3% lower bound for 1-10, 2% for 11-20, 3% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.12.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.3.2.3, x.u = x.ub.5.4.5 )
###
```

```
### 2% lower bound for 1-10, 3% for 11-20, 2% for 21-30
### 5% upper bound for 1-10  4% for 11-20  5% for 21-30
###
p.13.matrix <- rbounded.test( 400, 30, 1, x.l = x.lb.2.3.2, x.u = x.ub.5.4.5 )
```

`requal` *Generate equal weighted portfolios*

Description

This function generates m random equal portfolios with k non-zero, equal weights and the sum of the weights equals x_t.

Usage

```
requal(m, n = 2, k = n, x.t=1)
```

Arguments

`m`	A positive integer for the number of portfolios in the sample
`n`	A positive integer for the number of non-zero equal weights
`k`	A positive integer for the number of investments in the portfolio
`x.t`	A positive number for the sum of the weights

Details

The function executes the function `random.equal` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step.

Value

A numeric $m \times n$ matrix. The rows are the portfolios and the columns are the investment weights for each portfolio

Author(s)

Frederick Novomestky `<fn334@nyu.edu>`

References

Evans, J. and S. Archer, 1968. Diversification and the Reduction of Risk: An Empirical Analysis, *Journal of Finance*, 23, 761-767.

Upson, R. B., P. F. Jessup and K. Matsumoto, 1975. Portfolio Diversification Strategies, *Financial Analysts Journal*, 31(3), 86-88.

Elton, E. J. and M. J. Gruber, 1977. Risk Reduction and Portfolio Size: An Analytical Solution, *Journal of Business*, 50(4), 415-437.

Bird, R. and M. Tippett, 1986. Naive Diversification and Portfolio Risk - A Note, *Management Science*, 32(2), 244-251.

Statman, M., 1987. How many stocks make a diversified portfolio, *Journal of Financial and Quantitative Analysis*, 22, 353-363.

Newbould, G. D. and P. S. Poon, 1993. The minimum number of stocks needed for diversification, *Financial Practice and Education*, 3, 85-87.

O'Neal, E. S., 1997. How Many Mutual Funds Constitute a Diversified Mutual Fund Portfolio, *Financial Analysts Journal*, 53(2), 37-46.

Statman, M., 2004. The diversification puzzle, *Financial Analysts Journal*, 60, 48-53.

Benjelloun, H. and Siddiqi, 2006. Direct diversification with small stock portfolios. *Advances in Investment Analysis and Portfolio Management*, 2, 227-252.

Benjelloun, H., 2010. Evans and Archer - forty years later, *Investment Management and Financial Innovation*, 7(1), 98-104.

See Also

random.equal

Examples

```
###
### generate 100 equal weighted portfolios of 30 investments with 10 non zero positions
###
x.matrix <- requal( 100, 30, 10 )
```

`requal.test` *Generate equal weighted portfolios*

Description

This function generates m random equal portfolios with k non-zero, equal weights and the sum of the weights equals x_t. This function is used to evaluate the computation performance of the portfolio generation algorithm

Usage

```
requal.test(m, n = 2, k = n, x.t = 1)
```

Arguments

m	A positive integer for the number of portfolios in the sample
n	A positive integer for the number of non-zero equal weights
k	A positive integer for the number of investments in the portfolio
x.t	A positive number for the sum of the weights

Details

The function executes the function `random.equal` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step. This is not an iterative function so that the number of iterations is 1 for all of the portfolios.

Value

A list with two named components.

`xmatrix`	An $m \times n$ matrix of investment weights
`iters`	An $m \times 1$ vector with the number of iterations used to obtain the portfolios

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Evans, J. and S. Archer, 1968. Diversification and the Reduction of Risk: An Empirical Analysis, *Journal of Finance*, 23, 761-767.

Upson, R. B., P. F. Jessup and K. Matsumoto, 1975. Portfolio Diversification Strategies, *Financial Analysts Journal*, 31(3), 86-88.

Elton, E. J. and M. J. Gruber, 1977. Risk Reduction and Portfolio Size: An Analytical Solution, *Journal of Business*, 50(4), 415-437.

Bird, R. and M. Tippett, 1986. Naive Diversification and Portfolio Risk - A Note, *Management Science*, 32(2), 244-251.

Statman, M., 1987. How many stocks make a diversified portfolio, *Journal of Financial and Quantitative Analysis*, 22, 353-363.

Newbould, G. D. and P. S. Poon, 1993. The minimum number of stocks needed for diversification, *Financial Practice and Education*, 3, 85-87.

O'Neal, E. S., 1997. How Many Mutual Funds Constitute a Diversified Mutual Fund Portfolio, *Financial Analysts Journal*, 53(2), 37-46.

Statman, M., 2004. The diversification puzzle, *Financial Analysts Journal*, 60, 48-53.

Benjelloun, H. and Siddiqi, 2006. Direct diversification with small stock portfolios. *Advances in Investment Analysis and Portfolio Management*, 2, 227-252.

Benjelloun, H., 2010. Evans and Archer - forty years later, *Investment Management and Financial Innovation*, 7(1), 98-104.

See Also

`random.equal`

Examples

```
###
### generate 100 equal weighted portfolios of 30 investments with 10 non zero positions
###
result <- requal.test( 100, 30, 10 )
```

---

rgeneral *Generate random general portfolios*

Description

This function generates m random general portfolios with n investments each. There are k positions that can be positive or negative. The probability that a given investment weight is positive is p. The maximum absolute exposure is x.u which has 1 as the default

Usage

```
rgeneral(m, n = 2, k = n, segments = NULL, p = 0.5, x.u = 1)
```

Arguments

m	A positive integer value for the number of portfolios
n	A positive integer value for the number of investments in the portfolio
k	A positive integer value for the number of non-zero investment positions
segments	A vector or list of vectors that defines the portfolio segments
p	A positive numeric value for the probability that a non-zero position is positive
x.u	A positive numeric value for the maximum absolute exposure to an investment

Details

The function executes the function `random.general` using the R function `sapply`. The result returned is the transpose of the matrix generated in the previous step.

Value

An $m \times n$ numeric matrix. The rows are the portfolios and the columns are the investment weights for each portfolio

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

random.general

Examples

```
###
### 100 long only portfolios of 30 investments with 30 non-zero positions
###
x.long <- rgeneral( 100, 30, p=1.0 )
###
### 100 long only portfolios of 30 investments with 10 non-zero positions
###
y.long <- rgeneral( 100, 30, 10, p=1.0 )
###
### 100 short only portfolios of 30 investments with 30 non-zero positions
###
x.short <- rgeneral( 100, 30, p=0.0 )
###
### 100 short only portfolios of 30 investments with 10 non-zero positions
###
y.short <- rgeneral( 100, 30, 10, p=0.0 )
###
### 100 long short portfolios of 30 investments with 30 non-zero positions
###
x.long.short <- rgeneral( 100, 30, p=0.5 )
###
### 100 long short portfolios of 30 investments with 10 non-zero positions
###
y.long.short <- rgeneral( 100, 30, 10, p=0.5 )
###
### 100 long bias portfolios of 30 investments with 30 non-zero positions
###
x.long.bias <- rgeneral( 100, 30, p=0.7 )
###
### 100 long bias portfolios of 30 investments with 10 non-zero positions
###
y.long.bias <- rgeneral( 100, 30, 10, p=0.7 )
###
### 100 short bias portfolios of 30 investments with 30 non-zero positions
###
x.short.bias <- rgeneral( 100, 30, p=0.3 )
###
### 100 short bias portfolios of 30 investments with 10 non-zero positions
###
y.short.bias <- rgeneral( 100, 30, 10, p=0.3 )
```

`rgeneral.test` *Generate random general portfolios*

Description

This function generates m random general portfolios with n investments each that can have k positive or negative. The probability that a given investment weight is positive is p. The maximum absolute exposure is x.u which has 1 as the default The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
rgeneral.test(m, n = 2, k = n, segments = NULL, p = 0.5, x.u = 1)
```

Arguments

`m`	A positive integer value for the number of portfolios
`n`	A positive integer value for the number of investments in the portfolio
`k`	A positive integer value for the number of non-zero long and short positions
`segments`	A vector or list of vectors that defines the portfolio segments
`p`	A positive numeric value for the probability that a position is positive
`x.u`	A positive numeric value for the maximum absolute exposure to an investment

Details

The function executes the function `random.general.test` using the R function `lapply`. The result which is a list contains the investment weight vectors and number of iterations. Thse data are stored in a matrix of investment weights and a vector of iterations. These arrays are returned as a list.

Value

A list with two named components.

`xmatrix`	An $m \times n$ numerical matrix of investment weights
`iters`	An $m \times 1$ integer vector for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

`random.general.test`

Examples

```
###
### 100 long only portfolios of 30 investments
###
x.long <- rgeneral.test( 100, 30, p=1.0 )
y.long <- rgeneral.test( 100, 30, 10, p=1.0 )
###
### 100 short only portfolios of 30 investments
###
x.short <- rgeneral.test( 100, 30, p=0.0 )
```

```
y.short <- rgeneral.test( 100, 30, 10, p=0.0 )
###
### 100 long short portfolios of 30 investments
###
x.long.short <- rgeneral.test( 100, 30, p=0.5 )
y.long.short <- rgeneral.test( 100, 30, 10, p=0.5 )
###
### 100 long bias portfolios of 30 investments
###
x.long.bias <- rgeneral.test( 100, 30, p=0.7 )
y.long.bias <- rgeneral.test( 100, 30, 10, p=0.7 )
###
### 100 short bias portfolios of 30 investments
###
x.short.bias <- rgeneral.test( 100, 30, p=0.3 )
y.short.bias <- rgeneral.test( 100, 30, 10, p=0.3 )
```

`rlongonly` *Generate random long only portfolios*

Description

This function generates m random long only portfolios with n investments with each investment weight bounded in an interval and the sum of the weights equals a given amount. The number of non-zero positions is k.

Usage

```
rlongonly(m, n = 2, k = n, segments = NULL, x.t = 1, x.l = 0,
x.u = x.t, max.iter = 1000)
```

Arguments

`m`	A positive integer value for the number of portfolios
`n`	A positive integer value for the number of investments in each portfolio
`k`	A positive integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t`	A positive numeric value for the sum of investment weights
`x.l`	A positive numeric value for the lower bound of an investment weight
`x.u`	A positive numeric value for the upper bound of an investment weight
`max.iter`	A positive integer value for the number of rejection iterations

Details

The function executes the function `random.longonly` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step.

Value

A numeric $m \times n$ matrix. The rows are the portfolios and the columns are the investment weights for each portfolio

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

`random.longonly`

Examples

```
###
### 100 long only portfolios of 30 investments with 30 non-zero positions
###
x.matrix <- rlongonly( 100, 30 )
###
### 100 long only portfolios of 30 investments with 10 non-zero positions
###
y.matrix <- rlongonly( 100, 30, 10 )
```

`rlongonly.test` *Generate random long only portfolios*

Description

This function generates m random long only portfolios with n investments with each investment weight bounded in an interval and the sum of the weights equals a given amount. The number of non-zero positions is k. This function is used to test the algorithm that generates the random portfolios.

Usage

```
rlongonly.test(m, n = 2, k = n, segments = NULL, x.t = 1, x.l = 0,
x.u = x.t, max.iter = 1000)
```

Arguments

`m`	A positive integer value for the number of portfolios
`n`	A positive integer value for the number of investments in each portfolio
`k`	A positive integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t`	A positive numeric value for the sum of investment weights
`x.l`	A positive numeric value for the lower bound of an investment weight
`x.u`	A positive numeric value for the upper bound of an investment weight
`max.iter`	A positive integer value for the number of rejection iterations

Details

The function executes the function `random.longonly.test` using the R function `lapply`. The result which is a list contains the investment weight vectors and number of iterations. Thse data are stored in a matrix of investment weights and a vector of iterations. These arrays are returned as a list.

Value

A list with two named components.

`xmatrix` A numerical $m \times n$ matrix of investment weights

`iters` An integer $m \times 1$ vector for the number iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Cheng, R. C. H., 1977. The Generation of Gamma Variables with Non-integral Shape Parameter, *Journal of the Royal Statistical Society*, Series C (Applied Statistics), 26(1), 71.

Marsaglia, G. and T. A. Bray, 1964. A convenient method for generating normal variables, *SIAM Review*, 6(3), July 1964, 260-264.

Ross, S. M. (2006). *Simulation*, Fourth Edition, Academic Press, New York NY.

See Also

`random.longonly.test`

Examples

```
###
### generate 100 long only portfolios with 30 investments and 30 non-zero positions
###
x.result <- rlongonly.test( 100, 30 )
###
### generate 100 long only portfolios with 30 investments and 10 non-zero positions
###
y.result <- rlongonly.test( 100, 30, 10 )
```

`rlongshort` *Generate long short portfolios*

Description

This function generates m random long short portfolios with n investments with the given gross and net notional exposure requirements. There are k non-zero positions in the portfolio.

Usage

```
rlongshort(m, n = 2, k = n, segments = NULL, x.t.long = 1, x.t.short = x.t.long,
max.iter = 2000, eps = 0.001)
```

Arguments

`m`	A positive integer value for the number of portfolios generated
`n`	A positive integer value for the number of investments in the portfolio
`k`	A positive integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t.long`	A positive real value for the sum of the long exposures
`x.t.short`	A positive real value for the sum of the absolute value of the short exposures
`max.iter`	A positive integer value for the maximum iterations in the acceptance rejection method
`eps`	A small positive real value for the convergence criteria for the gross notional exposure

Details

The arguments `x.t`, `x.t.long` and `x.t.short` are proportions of total invested capital.

Value

An $m \times n$ numeric matrix of investment weights for the long short portfolios

Author(s)

Frederick Novomestky `<fn334@nyu.edu>`

References

Jacobs, B. I. and K. N. Levy, 1997. The Long and Short of Long-Short Investing, *Journal of Investing*, Spring 1997, 73-86.

Jacobs, B. I., K. N. Levy and H. M. Markowitz, 2005. Portfolio Optimization with Factors, Scenarios and Realist SHort Positions, *Operations Research*, July/August 2005, 586-599.

See Also

random.longshort

Examples

```
###
### 100 portfolios of 30 investments with 30 non-zero positions
###
x.matrix <- rlongshort( 100, 30 )
###
### 100 portfolios of 30 investments with 10 non-zero positions
###
y.matrix <- rlongshort( 100, 30, 20 )
```

---

rlongshort.test *Generate random long short portfolios*

Description

This function generates m random long short portfolios with n investments that satisfy the given gross and net notional exposure requirements. There are k non-zero positions in each portfolio. The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
rlongshort.test(m, n = 2, k = n, segments=NULL, x.t.long = 1, x.t.short = x.t.long,
max.iter = 2000, eps = 0.001)
```

Arguments

m	A positive integer value for the number of portfolios generated
n	A positive integer value for the number of investments in the portfolio
k	A positive integer value for the number of non zero weights
segments	A vector or list of vectors that defines the portfolio segments
x.t.long	A positive real value for the sum of the long exposures
x.t.short	A positive real value for the sum of the absolute value of the short exposures
max.iter	A positive integer value for the maximum iterations in the acceptance rejection method
eps	A small positive real value for the convergence criteria for the gross notional exposure

Details

The function executes the function `random.longshort.test` using the R function `lapply`. The result which is a list contains the investment weight vectors and number of iterations. Thse data are stored in a matrix of investment weights and a vector of iterations. These arrays are returned as a list. Gross notional exposure for each portfolio is `x.t.long + x.t.short` and net notional exposure is `x.t.long - x.t.short`. The argument `eps` is the tolerance applied towards the the gross notional exposure of each portfolio.

Value

A list with two named components.

`xmatrix`	A numerical $m \times n$ matrix of investment weights
`iters`	An $m \times 1$ integer vector for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky `<fn334@nyu.edu>`

References

Jacobs, B. I. and K. N. Levy, 1997. The Long and Short of Long-Short Investing, *Journal of Investing*, Spring 1997, 73-86.

Jacobs, B. I., K. N. Levy and H. M. Markowitz, 2005. Portfolio Optimization with Factors, Scenarios and Realistic Short Positions, *Operations Research*, July/August 2005, 586-599.

See Also

`random.longshort.test`

Examples

```
###
### 100 long short portfolios with 30 investments and 30 non-zero positions
###
x.result <- rlongshort.test( 100, 30 )
###
### 100 long short portfolios with 30 investments and 20 non-zero positions
###
y.result <- rlongshort.test( 100, 30, 20 )
```

`rshortonly` *Generate short only portfolios*

Description

This function generates m random short only portfolios with n investments with each investment absolute weight bounded in an interval and the sum of the absolute value of weights equals a given amount.

Usage

```
rshortonly(m, n = 2, k = n, segments=NULL, x.t = 1, x.l = 0,
x.u = x.t, max.iter = 1000)
```

Arguments

`m`	A positive integer value for the number of portfolios
`n`	A positive integer value for the number of investments in the portfolio
`k`	A positive integer value for the number of non zero weights
`segments`	A vector or list of vectors that defines the portfolio segments
`x.t`	A positive numeric value for the sum of the absolute value of investment weights
`x.l`	A positive numeric value for the lower bound on the absolute value of investment weights
`x.u`	A positive numeric value for the upper bound on the absolute value of investment weights
`max.iter`	A positive integer value for the maximum iterations in the rejection method

Details

The function executes the function `random.shortonly` using the R function sapply. The result returned is the transpose of the matrix generated in the previous step.

Value

A numeric $imesn$ matrix. The rows are the portfolios and the columns are the investment weights for each portfolio

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

`random.shortonly`

Examples

```
x.matrix <- rshortonly( 100, 30 )
y.matrix <- rshortonly( 100, 30, 10 )
```

`rshortonly.test` *Generate random short only portfolios*

Description

This function generates m random short only portfolios with n investments where each investment absolute weight bounded in an interval and the sum of the absolute weights equals a given amount. This function is used to test the algorithm that generates the random portfolios. The number of non zero positions in the portfolio is k. The function is used to evaluate the performance of the portfolio generation algorithm.

Usage

```
rshortonly.test(m, n = 2, k = n, segments = NULL, x.t = 1, x.l = 0,
x.u = x.t, max.iter = 1000)
```

Arguments

`m`	A positive integer value for the number of portfolios
`n`	An integer value for the number of investments in the portfolio
`k`	An integer value for the number of non zero weights
`segments`	A vector or list of vectors that define the portfolio segments
`x.t`	Numeric value for the sum of the absolute value of the investment weights
`x.l`	Numeric value for the lower bound on the absolute value of an investment weight
`x.u`	Numeric value for the upper bound on the absolute value of an investment weight
`max.iter`	An integer value for the maximum iteration in the acceptance rejection loop

Details

The function executes the function `random.shortonly.test` using the R function `lapply`. The result which is a list contains the investment weight vectors and number of iterations. Thse data are stored in a matrix of investment weights and a vector of iterations. These arrays are returned as a list.

Value

A list with two named components.

`xmatrix`	An $m \times n$ numerical matrix of investment weights
`iters`	An $m \times 1$ integer vector for the number of iterations used to obtain the investment weights

Author(s)

Frederick Novomestky <fn334@nyu.edu>

See Also

random.longonly.test

Examples

```
###
### generate 100 short only portfolios of 30 investments with 30 non zero positions
###
x.result <- rshortonly.test( 100, 30 )
###
### generate 100 short only portfolios of 30 investments with 10 non zero positions
###
x.result <- rshortonly.test( 100, 30, 10 )
```

---

segment.complement *Complement of Investment Segments*

Description

This function returns a vector of investments that are in a portfolio with n investments but not in the given investment segments

Usage

```
segment.complement(n, segments)
```

Arguments

n	A positive integer for the number of investments in a portfolio
segments	A vector or list of vectors that defines the investment segments

Details

If the investments in the given segment are for the entire portfolio, a NULL value is returned. If the segments argument is NULL, then the entire portfolio of n investments is returned.

Value

A vector of investments or a NULL value.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

Examples

```
###
### define the segments
###
    I <- list()
    I[[1]] <- c( 1, 2, 3 )
    I[[2]] <- c( 4, 5 )
    I[[3]] <- c( 6, 7 )
    I[[4]] <- c( 8, 9, 10 )
    segment.complement( 10, I )
    segment.complement( 10, NULL )
    segment.complement( 10, I[[1]] )
    segment.complement( 10, I[[2]] )
    segment.complement( 10, I[[3]] )
    segment.complement( 10, I[[4]] )
```

`set.segments` *Set segment weights from portfolios*

Description

This function assigns the given investment weights to larget portfolios using the investment indices in the segments

Usage

```
set.segments(portfolios, n, segments)
```

Arguments

`portfolios`	A vector or matrix of investment weights for the segments
`n`	A positive integer value for the number of investments in the larger portfolio
`segments`	A vector or list of vectors that defines the segment investments

Details

A private function `vector.set.segments` is used to take weights in a given portfolio vector and assign them to a larger vector using the collapsed investment index vector. If the `portfolios` argument is a matrix, then the R function `apply` is used to perform this task for each row vector.

Value

A vector or matrix.

Author(s)

Frederick Novomestky `<fn334@nyu.edu>`

See Also

collapse.segments

Examples

```
###
### simulate 300 long only portfolios with 30 investments
###
portfolios <- rlongonly( 300, 30 )
###
### define six segments with five investments in each
###
segment1 <- 1:5
segment2 <- 11:15
segment3 <- 21:25
segment4 <- 31:35
segment5 <- 41:45
segment6 <- 51:55
segments <- list( segment1, segment2, segment3, segment4, segment5, segment6 )
newPortfolios <- set.segments( portfolios, 60, segments )
```

---

underweight.segments *Underweight Active Investment Segment Exposures*

Description

This function underweight the investment exposures of the given portfolios in the given active investment segments by the proportion x_u of the total exposure in the active segment.

Usage

```
underweight.segments(portfolios, segments, x.u)
```

Arguments

portfolios	A numeric vector or matrix for the portfolio investment exposures
segments	A vector or list of vectors that define the active investment segment
x.u	A positive real value for the proportion of total active exposure allocated to the passive investment exposures

Details

if $x_u = 0$, then the original portfolios are returned. If $x_u = 1$, then the total exposure of the active segment is allocated to the passive investment segment of all the portfolios. The private function `vector.underweight.segments` i performs the actual work and returns a vector. If `portfolios` is a matrix of investment weights, then the apply function is used with the private function to obtain a matrix of weights. The transpose of this matrix is returned.

Value

A vector of adjusted investment exposures for one portfolio or a matrix for more than one portfolio.

Author(s)

Frederick Novomestky <fn334@nyu.edu>

References

Grinold, R. C. and R. H. Kahn, 1999. *Active Portfolio Management: Quantitative Approach for Providing Superior Returns and Controlling Risk*, Second Edition, McGraw-Hill, New York, NY.

See Also

segment.complement

Examples

```
onePortfolio <- random.longonly( 10 )
I <- list()
I[[1]] <- c( 1, 2, 3 )
I[[2]] <- c( 4, 5 )
I[[3]] <- c( 6, 7 )
I[[4]] <- c( 8, 9, 10 )
underweight.segments( onePortfolio, I[[1]], 0 )
underweight.segments( onePortfolio, I[[1]], .1 )
```

Index