Package ‘stockPortfolio’

January 27, 2015

Type Package
Title Build stock models and analyze stock portfolios.
Version 1.2
Date 2012-03-14
Author David Diez and Nicolas Christou
Maintainer David Diez <david.m.diez@gmail.com>
Description Download stock data, build single index, constant
correlation, and multigroup models, and estimate optimal stock
portfolios. Plotting functions for the portfolio possibilities
curve and portfolio cloud are included. A function to test a
portfolio on a data set is also provided.
License GPL (>= 2)
LazyLoad yes
Depends stats, graphics, grDevices, utils
Repository CRAN
Date/Publication 2012-03-14 16:03:05
NeedsCompilation no

R topics documented:

stockPortfolio-package ......................................................... 2
adjustBeta ................................................................. 4
getCorr ................................................................. 5
getReturns ............................................................... 6
optimalPort ............................................................. 8
portCloud ................................................................. 10
portPossCurve .......................................................... 11
portReturn .............................................................. 13
stock04 ................................................................. 14
stock94 ................................................................. 15
stock94Info ............................................................ 16
stock99 ................................................................. 16
The package `stockPortfolio` is a quantitative approach to portfolio allocation among stocks. The package includes functions to download historical data from Yahoo Finance, build models, estimate optimal portfolios, and test portfolios. A large range of graphical features have been included for visual understanding.

Details

Package: `stockPortfolio`
Type: Package
Version: 1.2
Date: 2012-03-14
License: GPL (>= 2)
LazyLoad: yes

A common starting point in the package is with the `getReturns` function, which can be used to obtain stock data using an internet connection. Using an object of class "stockReturns" from the `getReturns` function, one can build a stock model using the `stockModel` function. There are four model options in `stockModel`: no model where a portfolio is selected based on empirical returns, variances, and covariances among the stocks, the single index model, constant correlation model, and the multigroup model. After a stock model has been built, the user can obtain an estimate of the optimal portfolio allocation among those stocks using `optimalPort`. Additionally, one can test out models and portfolios on data sets that are either supplied by the user or are output from `getReturns`.

While most objects can be plotted, there are two specialty plotting functions: `portPossCurve` and `portCloud`. The function `portPossCurve` plots the portfolio possibilities curve based on a model, and `portCloud` plots a cloud of possible portfolios based on a model.

Three data sets and one data "key" have been included as a sample data set: `stock94`, `stock99`, `stock04`, `stock94Info`.

Author(s)

David Diez and Nicolas Christou wrote the package functions.

Maintainer: David Diez <david.m.diez@gmail.com>
References


Examples

```r
### two examples of downloading data ####
## Not run: grEx1 <- getReturns(c('C','BAC'), start='2004-01-01', end='2008-12-31')
## Not run: grEx2 <- getReturns(c('KEY', 'WFC', 'JPM', 'AMR', 'BIIB', 'AMGN'))

### build four models ####
data(stock99)
data(stock94info)
non <- stockmodel(stock99, drop=25, model='none', industry=stock94info$industry)
sim <- stockmodel(stock99, model='SIM', industry=stock94info$industry, index=25)
ccm <- stockmodel(stock99, drop=25, model='CCM', industry=stock94info$industry)
mgm <- stockmodel(stock99, drop=25, model='MGM', industry=stock94info$industry)

### build optimal portfolios ####
opNon <- optimalPort(non)
opSim <- optimalPort(sim)
opCcm <- optimalPort(ccm)
opMgm <- optimalPort(mgm)

### test portfolios on 2004-9 ####
data(stock04)	pNon <- testPort(stock04, opNon)	pSim <- testPort(stock04, opSim)	pCcm <- testPort(stock04, opCcm)	pMgm <- testPort(stock04, opMgm)

### compare performances ####
plot(tpNon)
lines(tpSim, col=2, lty=2)
lines(tpCcm, col=3, lty=3)
lines(tpMgm, col=4, lty=4)
legend('topleft', col=1:4, lty=1:4, legend=c('none', 'SIM', 'CCM', 'MGM'))
```
adjustBeta

Adjust the beta parameters from a single index model

Description

Given a stockModel object that is based on the single index model, the beta parameters may be adjusted using either the Blume or Vasicek technique. This function outputs a new object of class "stockModel" based on the single index model.

Usage

adjustBeta(model, model2 = NULL, method = c("Blume", "Vasicek"))

Arguments

model An object of class "stockModel" that is based on the single index model. If using Blume’s method, then this model should be for the first period.

model2 An object of class "stockModel" that is based on the single index model. If using Blume’s method, then this model should be for the second period. If using Vasicek’s method, then this model is ignored.

method The single index model adjustment method. The options are "Blume" (default) or "Vasicek". Setting it to any character string other than "B", "b", "Blume", "blume", or "1" will result in using the Vasicek method.

Details

The single index model results in a vector of parameter estimates, which is typically labeled using the Greek letter beta. Both the Blume and Vasicek methods adjust the beta parameter vector. Vasicek’s method regresses all of the elements of beta towards the mean of those elements; the amount of this correction is based both on the variability of the elements of beta and also on the estimated standard error of each element of beta. Blume’s method takes beta estimates from two time periods and creates a regression equation: beta2 = b0 + b1*beta1. Blume’s method uses this regression equation to estimate beta for the time following the second period.

Value

adjustBeta returns an object of class "stockModel".

Author(s)

David Diez

References

getCorr

See Also
getReturns, stockModel, optimalPort

Examples

```r
#=== build two single index models ====#
data(stock94)
data(stock99)
data(stock94info)
sim1 <- stockModel(stock94, model='SIM',
                    industry=stock94info$industry, index=25)
sim2 <- stockModel(stock99, model='SIM',
                    industry=stock94info$industry, index=25)

#=== adjust the betas ====
# the output is a new stock model
simBlu <- adjustBeta(sim1, sim2)
simVas <- adjustBeta(sim2, method='Vasicek')

#=== build optimal portfolios ====#
opSim <- optimalPort(sim2)
opBlu <- optimalPort(simBlu)
opVas <- optimalPort(simVas)

#=== test portfolios on 2004-9 ====#
data(stock04)
tpSim <- testPort(stock04, opSim)
tpBlu <- testPort(stock04, opBlu)
 tpVas <- testPort(stock04, opVas)

#=== compare performances ====#
plot(tpSim, ylim=c(1.1, 2.2))
lines(tpBlu, col=2, lty=2)
lines(tpVas, col=3, lty=3)
legend('topleft', col=1:3, lty=1:3, legend=c('none', 'Blume', 'Vasicek'))
```

getCorr Average correlation

Description

Determine the average correlation or average correlation by industry of a variance-covariance matrix.

Usage

getCorr(V, industry = NULL)
getReturns

Arguments

V Variance-covariance matrix.
industry A vector specifying the industry of the stocks in their order given in the columns (and rows) of V. This argument is optional.

Value

If industry is not provided, then the average correlation in V in the matrix is returned but ignoring the diagonal. If industry is provided, then the output is a matrix with dimension k-by-k, where k is the number of unique values in industry.

Author(s)

David Diez

See Also

stockModel

Examples

```r
#=>> the covariance matrix of stock94
getcov(stock94)
getcov(stock94info)
V <- cov(stock94)

#=>> the average correlation
getcorr(V)
getcorr(V, industry=stock94info$industry)
```

Description

Obtain stock data from Yahoo Finance.

Usage

```r
getReturns(ticker, freq = c("month", "week", "day"),
get = c("overlapOnly", "all"), start = "1970-01-01", end = NULL)
```
getReturns

Arguments

ticker
A character vector where each element is a ticker.

freq
The frequency of the stock data to be downloaded. Default is "month" for 12
observations per year and other options are "week" and "day".

get
The default, "overlaponly", will return the stock returns for which all stocks
had data and drop any dates with NA; if it is monthly data, minor corrections are
made when appropriate. The "all" option yields all stock returns regardless
of whether data for all stocks is available; stock data obtained under the "all"
option may not work in the other functions in this package if NA values are
present.

start
Start date in the format "YYYY-MM-DD".

end
End date in the format "YYYY-MM-DD". If using freq="month" (the default) or
freq="week", the DD should be specified as the last day of the month or week,
respectively.

Value

getReturns outputs an object of class "stockReturns", which is a list of the following:

r
Stock returns, where the first row is the most recent and the last row is the oldest.
ticker
The tickers of the stocks.
period
How frequently stock returns are included in the data.
start
The oldest date for which stock returns are included.
end
The most recent date for which stock returns are included.
full
A list, where each item is a data frame containing information from the CSV file
downloaded from Yahoo Finance.

Author(s)

David Diez and Nicolas Christou

See Also

stockModel, optimalPort, testPort, portReturn

Examples

```r
### Citi and Bank of America, 2004-2008 ###
# cBac <- getReturns(c('C','BAC'), start='2004-01-01', end='2008-12-31')
# print(cBac)
# summary(cBac)
# plot(cBac)
# lines(cBac, lwd=2)
# pairs(cBac)
```
optimalPort (model, Rf = NULL, shortSell = NULL, eps = 10^(-4))

Arguments

model An object of class "stockModel".
Rf An optional argument to update the risk free rate.
shortSell An optional argument to update short-selling.
eps An error term to be used in evaluating whether the risk-free rate is acceptable. This argument should not be adjusted except by advanced users.

Details

When the function returns an error regarding the validity of Rf, the risk free rate of return, this is not a bug. That error message means the Rf is too large; it is larger than the expected return of the vertex of the portfolio possibilities curve (the left-most point on this curve). When this occurs, no tangent line can be created along the efficient frontier. The implication is that a lower Rf should be specified to identify a portfolio along the efficient frontier of the portfolio possibilities curve.

This Rf issue has happened relatively frequently with stocks in the last few years. So many stocks are down, which sometimes results in a minimum risk portfolios with an expected return that is negative.

Value

optimalPort outputs an object of class "optimalPortfolio", which is a list of

model An object of class "stockModel".
X The allocation of the optimal portfolio.
R The estimated return associated with allocation X.
risk The estimated risk associated with allocation X.

Author(s)

David Diez and Nicolas Christou
References


See Also

getReturns, stockModel, testPort

Examples

```r
#>>> obtain data ===#
data(stock99)
data(stock94Info)
mgm <- stockModel(stock99, drop=25, model='MGM', industry=stock94Info$industry)

#>>> build optimal portfolios ===#
opMgm1 <- optimalPort(mgm)
opMgm2 <- optimalPort(mgm, Rf=0.004)
print(opMgm1)
summary(opMgm1)

#>>> plot the optimal portfolios ===#
par(mfrow=c(1,2))
# these plots provide a "head coloring" of
# the allocation by optimalPort
plot(opMgm1)
plot(opMgm2)

#>>> additional plotting 1 ===#
par(mfrow=c(1,1))
plot(opMgm1, addNames=TRUE)

#>>> additional plotting 2 ===#
plot(opMgm1, optPortOnly=TRUE, colOP=2, pchOP=2)
points(opMgm2, colOP=2, pchOP=4)

#>>> Watch out -- choosing Rf too large causes errors ===#
data(stock99)
data(stock94Info)
on <- stockModel(stock99, drop=25, model='none',
industry=stock94Info$industry)
portPossCurve(on)
```
portCloud

Description

Given a model with short-selling permitted, portCloud plots a cloud of possible portfolios.

Usage

```
portCloud(model, riskRange = 2, detail = 25, N = 3000, add = TRUE,
          col = c("#55550044"), pch = 20, subSamp = 1000, xlim = "default",
          ylim = "default", xlab = "Risk", ylab = "Return", ...)
```

Arguments

- **model**: An object of class "stockModel".
- **riskRange**: A factor to specify how large you would like the portfolio cloud to be. If X is the portfolio with minimum risk, then the portfolio cloud will look aesthetically best to approximately risk of riskRange*X.
- **detail**: A parameter that adjusts the appearance of the portfolio cloud. The default value is often adequate.
- **N**: A parameter for the number of portfolios to consider.
- **add**: If TRUE, then the points are added to the plot. If FALSE, a new plot is created.
- **col**: Color of the portfolios in the plot.
- **pch**: Plotting character of the portfolios.
- **subSamp**: The maximum number of portfolios to plot.
- **xlim**: Limits for the x axis. Only applied if add=FALSE.
- **ylim**: Limits for the y axis. Only applied if add=FALSE.
- **xlab**: Label for the x axis. Only applied if add=FALSE.
- **ylab**: Label for the y axis. Only applied if add=FALSE.
- **...**: If add=FALSE, additional arguments for plot. If add=TRUE, additional arguments for points.
Details

Which portfolios are actually plotted is dependent on the portfolio possibilities curve. A number of points along the curve, specified by detail and the stocks themselves are used to produce a relatively uniform looking portfolio cloud.

Value

A list of the following items:

- **ports** The portfolios plotted.
- **R** The estimated return associated with each row of ports.
- **risk** The estimated risk associated with each row of ports.

Author(s)

David Diez and Nicolas Christou

See Also

stockModel, portPossCurve

Examples

```r
data(stock94)
sm <- stockModel(stock94, model='SIM', index=25)
portCloud(sm, add=FALSE)
portPossCurve(sm, 2.5, add=TRUE)
```

---

**portPossCurve**  
*Plot the portfolio possibilities curve*

Description

Plot the portfolio possibilities curve or the efficient frontier for models that permit short-selling.

Usage

```r
portPossCurve(model, riskRange = 2, detail = 100, effFrontier = FALSE, 
add = FALSE, type = "l", xlab = "Risk", ylab = "Expected Return", 
doNotPlot = FALSE, ...)
```
Arguments

model  An object of class "stockModel".
riskRange A parameter to specify how much of the portfolio possibilities curve to plot. If X is the portfolio with minimum risk without respect to the risk free rate, then the portfolio possibilities curve will be shown up to approximately the risk riskRange*X.
detail The number of points to include on the portfolio possibilities curve. A small number will result a curve that is evidently made up of lines while a large number will provide more detail but takes more memory. The default value is generally adequate.

effFrontier If TRUE, only the efficient frontier is drawn.
add If TRUE, the curve is added to a plot. Otherwise a new plot is created.
type Plotting method. "p" for points, "l" for lines, "b" for both lines and points, and "n" to produce no points or lines.
xlab Label for the x axis. Only applied if add=FALSE.
ylab Label for the y axis. Only applied if add=FALSE.
doNotPlot If FALSE, nothing is plotted. This option may be useful if the points along the curve are of interest and only the values returned by portPossCurve are of interest.

Value

portPossCurve returns a list of the following items:

R The returns of points along the curve.
risk The risk of points along the curve.
ports The portfolios corresponding to R and risk.

Details

If the curve is not smooth, first try decreasing the riskRange. If this is unsuccessful in producing a plot to the detail desired, increase the detail. Generally it is advisable to attempt to adjust the riskRange before adjusting detail.

Author(s)

David Diez

See Also

stockModel, portCloud
Examples

```r
data(stock94)
sm <- stockModel(stock94, model='SIM', index=25)
portPossCurve(sm, 2)
portCloud(sm, 2.5)
```

---

**portReturn**  
*Estimate return and risk of a portfolio*

Description

Given a portfolio allocation \(X\) and a model, identify the estimated return and risk associated with \(X\).

Usage

```r
portReturn(model, X)
```

Arguments

- `model`  
  An object of class "stockModel".
- `X`  
  The portfolio allocation.

Value

`portReturn` returns a list of the following items:

- `R`  
  The estimated return.
- `V`  
  The estimated risk squared.
- `X`  
  The allocation, which is the second argument.
- `ticker`  
  The tickers from the model.
- `model`  
  An object of class "stockModel", which is the same model provided to the function.

Author(s)

David Diez

See Also

- `stockModel`
Examples

```r
### basics

data(stock04)
sm <- stockModel(stock04, model='SIM', index=25)
op <- optimalPort(sm)
prOp <- portReturn(sm, op$X)
prUn <- portReturn(sm, rep(1, 24)/24)
print(prOp)
summary(prOp)
summary(prUn)

### plotting a "portReturn" object

par(mfrow=c(2,2))
plot(prOp) # provides a heat map of the allocation
plot(prUn) # a boring heat map of allocation
plot(prOp, col=2:5) # many random colors
plot(prUn, col=1) # all black
```

---

**stock04**

*Data for 24 stocks and 1 index, 2004-9*

---

**Description**

Sixty monthly stock observations from 2004-10-01 to 2009-09-01 for 24 stocks in six industries. There is also a 25th column for an index, the S&P500.

**Usage**

```r
data(stock04)
```

**Format**

The format is numerical with 60 rows and 25 columns. The column names provide the tickers, and row names describe the dates.

**Details**

See `stock94Info` for a breakdown of the stocks by industry.

**Source**

Yahoo Finance.

**See Also**

`stock94Info, stock94, stock99, stockModel`
Examples

```r
data(stock04)
data(stock94info)
sm <- stockModel(stock04, model='SIM', index=25,
                 industry=stock94Info$industry)
```

Description

Sixty monthly stock observations from 1994-10-03 to 1999-09-01 for 24 stocks in six industries. There is also a 25th column for an index, the S&P500.

Usage

```r
data(stock94)
```

Format

The format is numerical with 60 rows and 25 columns. The column names provide the tickers, and row names describe the dates.

Details

See `stock94Info` for a breakdown of the stocks by industry.

Source

Yahoo Finance.

See Also

`stock94Info, stock99, stock04, stockModel`

Examples

```r
data(stock94)
data(stock94info)
sm <- stockModel(stock94, model='SIM', index=25,
                 industry=stock94Info$industry)
```
Description

A data frame showing the industries of each of the stocks given in stock94, stock99, and stock04.

Usage

data(stock94Info)

data(stock99)

data(stock04)

Examples

sm <- stockModel(stock04, model='SIM', index=25,
                 industry=stock94Info$industry)

Description

Sixty monthly stock observations from 1999-10-01 to 2004-09-01 for 24 stocks in six industries. There is also a 25th column for an index, the S&P500.

Usage

data(stock99)

Format

The format is numerical with 60 rows and 25 columns. The column names provide the tickers, and row names describe the dates.
Details

See stock94Info for a breakdown of the stocks by industry.

Source

Yahoo Finance.

See Also

stock94Info, stock94, stock04, stockModel

Examples

data(stock99)
data(stock94info)
sm <- stockModel(stock09, model='SIM', index=25,
industry=stock94Info$industry)

stockModel Create a stock model

Description

Input an object of class "stockReturns" and select a model. Available choices are "none", "SIM" (single index model), "CCM" (constant correlation model), and "MGM" (multigroup model).

Usage

stockModel(stockReturns, drop = NULL, Rf = 0, shortSelling = c("y", "n"),
model = c("none", "SIM", "CCM", "MGM"), industry = NULL, index = NULL,
get = c("overlapOnly", "all"), freq = c("month", "week", "day"),
start = "1970-01-01", end = NULL, recentLast = FALSE,
rawStockPrices = FALSE)

Arguments

stockReturns An object of class "stockReturns". Additionally, a character vector of tickers can also be used here, in which case also see argument freq, get, start, and end. Additionally, an object of class "stockModel" can also be input here, which will permit model adjustments, including switching the model altogether. Finally, stock data can also be submitted here as a matrix; the column names should be the ticker names and the row names should be the dates of the returns, YYYY-MM-DD. Additionally, for outside data sets where the oldest stock return is in row 1 (and not the last row), see argument recentLast.

drop Declare any stocks to be dropped. For instance, if the model "none", "CCM", or "MGM" is used, stock indices might be dropped.
The risk free rate of return, which must be standardized for the period (e.g. a 2% yearly rate for monthly data would imply $R_f = 0.02/12$, or 2% with daily data would imply $R_f = 0.02/250$ if there are 250 trading days per year). The default value is 0.

Either "yes" (default) or "no". Some models, "none" and "MGM", will permit short-selling regardless of this selection.

Either no model ("none", the default), the single index model ("SIM"), constant correlation model ("CCM"), or the multigroup model ("MGM").

A character or factor vector containing the industries corresponding to stockReturns. This argument is optional except when model="MGM", however, it may be included in any model for slightly enhanced graphics.

When using model="SIM", the index is the column number indicating the stock index. Warning if using drop and also specifying the index: The value of index should correspond to the column number AFTER dropping columns. See Details below.

"overlaponly" (default) obtains stock returns for which all stocks had data and drops any dates with NA. "all" yields all stock returns regardless of whether data for all stocks is available. This argument is ignored unless stockReturns is a vector of tickers.

The time period between each stock return. Default is "month" and other options are "week" and "day". This argument is ignored unless stockReturns is a vector of tickers.

Start date in the format "YYYY-MM-DD". This argument is ignored unless stockReturns is a vector of tickers.

End date in the format "YYYY-MM-DD". This argument is ignored unless stockReturns is a vector of tickers.

Set this argument to TRUE if (1) you are using your own data that was not obtained by getReturns and (2) your matrix of returns runs from oldest returns (row 1) to most recent returns (last row).

Set to TRUE if (1) you are using your own data that was not obtained by getReturns and (2) your matrix is of stock prices and not of stock returns.

The multigroup model is the least known of the models presented here. It is similar to the constant correlation model, except that instead of assuming a constant correlation across all stocks, correlations are only dependent on the industry of a stock.

If stocks are dropped using the argument drop, then index must correspond to the position of the index AFTER those stocks are dropped. For instance, if there are seven stocks, the index is in position six, and the fourth stock is dropped, then we should use index=5.

stockModel outputs an object of class "stockModel", which is a list of the following items, many of which might be NA:
model The model selected.
ticker A vector of the tickers of the stocks included in the model.
index The index number, if provided by the user.
theIndex Ticker of the index.
industry Industries associated with the stocks.
returns Return data used to build the model.
marketReturns Return data of the index.
n Number of observations per stock.
start The oldest date for which stock returns are included.
end The most recent date for which stock returns are included.
period How frequently stock returns are included in the data.
R Average returns of the stocks.
COV Variance-covariance matrix of the stock returns.
sigma Standard deviation of the returns of the stocks (square root of the diagonal of COV).
shorts Whether short sales are allowed.
Rf Risk free return rate.
alpha Vector of intercepts in the linear model for the single index model.
vAlpha The square of the standard errors of alpha.
beta Vector of coefficients in the linear model for the single index model.
vBeta The square of the standard errors of beta.
betaAdj Whether the model was adjusted via adjustBeta.
MSE Variance of error term associated with single index model for each stock.
RM Mean market return.
VM Variance of the market return.
rho Mean correlation or, if using model="MGM", the matrix of averaged correlations. See getCorr.

Author(s)
David Diez and Nicolas Christou

References
See Also

getReturns, adjustBeta, optimalPort, testPort

Examples

### build four models ###
data(stock99)
data(stock94info)
on <- stockModel(stock99, drop=25, model='none', industry=stock94info$industry)
sim <- stockModel(stock99, model='SIM', industry=stock94info$industry, index=25)
ccm <- stockModel(stock99, drop=25, model='CCM', industry=stock94info$industry)
mgm <- stockModel(stock99, drop=25, model='MGM', industry=stock94info$industry)

### build optimal portfolios ###
opNon <- optimalPort(non)
opSim <- optimalPort(sim)
opCcm <- optimalPort(ccm)
opMgm <- optimalPort(mgm)

### test portfolios on 2004-9 ###
data(stock04)
tpNon <- testPort(stock04, opNon)
tpSim <- testPort(stock04, opSim)
tpCcm <- testPort(stock04, opCcm)
tpMgm <- testPort(stock04, opMgm)

### compare performances ###
plot(tpNon)
lines(tpSim, col=2, lty=2)
lines(tpCcm, col=3, lty=3)
lines(tpMgm, col=4, lty=4)
legend('topleft', col=1:4, lty=1:4, legend=c('none', 'SIM', 'CCM', 'MGM'))

---

testPort

*Test a portfolio on a data set*

Description

Test a portfolio allocation on a new data set. This function is useful for comparing portfolios under different data scenarios.

Usage

testPort(theData, model = NULL, X = NULL, newestFirst = TRUE, isReturns = NULL)
Arguments

theData  The data set to be used. This may be an object of class "stockReturns", a vector of 1 plus the returns on each stock, or a matrix of stock returns where rows are ordered observations and columns represent individual stocks (see also argument newestFirst). The matrix may also be stock prices, in which case see argument isReturns.

model  An object of class "stockModel" or of class "optimalPortfolio". The allocation will be set as the optimal portfolio’s allocation. To set a different allocation, leave this argument as NULL and use argument X.

X  The stock allocation of the portfolio, where element i corresponds to stock i in argument theData. If model is given, this argument is ignored.

newestFirst  If argument theData is a matrix of stock returns or stock prices, and the rows run from oldest (row 1) to most recent (last row), set newestFirst=FALSE.

isReturns  If argument theData is a matrix of stock prices and not stock returns, set this argument as FALSE.

Details

When the argument X is used or if theData is not from getReturns, provide column names to theData that correspond with the names of the elements of X. If theData is a vector of one plus the returns of each stock, then this vector should have its element names corresponding to those elements in X.

If theData is an object of class "stockReturns" or is a matrix of returns or prices, then this will allow the resulting object of class "testPort" to be plotted. See the examples for details.

Value

testPort outputs an object of class "testPort", which consists of the following items:

X  The allocation used.

sumRet  Summary of the returns for each stock.

change  The value of the portfolio if it started at 1.

returns  Return data, if provided.

Author(s)

David Diez and Nicolas Christou

See Also

getReturns, stockModel, optimalPort, portReturn
Examples

```r
#>>>> build two single index models <<<<#
data(stock99)
data(stock94Info)
non <- stockModel(stock99, drop=25, model='none', industry=stock94Info$industry)
sim <- stockModel(stock99, model='SIM', industry=stock94Info$industry, index=25)
ccm <- stockModel(stock99, drop=25, model='CCM', industry=stock94Info$industry)
mgm <- stockModel(stock99, drop=25, model='MGM', industry=stock94Info$industry)

#>>>> build optimal portfolios <<<<#
opNon <- optimalPort(non)
opSim <- optimalPort(sim)
opCcm <- optimalPort(ccm)
opMgm <- optimalPort(mgm)

#>>>> test portfolios on 2004-9 <<<<#
data(stock94)
.tpEqu <- testPort(stock94[,-25], X=rep(1,24)/24)
.tpNon <- testPort(stock94, opNon)
.tpSim <- testPort(stock94, opSim)
.tpCcm <- testPort(stock94, opCcm)
.tpMgm <- testPort(stock94, opMgm)
print(tpEqu)
summary(tpEqu)

#>>>> compare performances <<<<#
plot(tpEqu, ylim=c(1, 3))
lines(tpNon, col=2, lty=2)
lines(tpSim, col=3, lty=3)
lines(tpCcm, col=4, lty=4)
# a sample of how to use points on an object of
# class "testPort", however, its use makes the
# plot somewhat ugly
points(tpMgm, col=5, lty=5, type='b')
legend('topleft', col=1:5, lty=1:5, legend=c('equal all.', 'none', 'SIM', 'CCM', 'MGM'), pch=c(NA, NA, NA, NA, 1))
```


Index

*Topic datasets
stock04, 14
stock94, 15
stock94Info, 16
stock99, 16

*Topic package
stockPortfolio-package, 2

adjustBeta, 4, 19, 20
getCorr, 5, 19
getReturns, 2, 5, 6, 9, 18, 20, 21
lines.stockReturns (getReturns), 6
lines.testPort (testPort), 20

optimalPort, 2, 5, 7, 8, 20, 21

pairs.stockReturns (getReturns), 6
plot, 12
plot.optimalPortfolio (optimalPort), 8
plot.portReturn (portReturn), 13
plot.stockModel (stockModel), 17
plot.stockReturns (getReturns), 6
plot.testPort (testPort), 20
points, 12
points.optimalPortfolio (optimalPort), 8
points.portReturn (portReturn), 13
points.stockModel (stockModel), 17
points.testPort (testPort), 20
portCloud, 2, 10, 12
portPossCurve, 2, 11, 11
portReturn, 7, 13, 21
print.optimalPortfolio (optimalPort), 8
print.portReturn (portReturn), 13
print.stockModel (stockModel), 17
print.stockReturns (getReturns), 6
print.testPort (testPort), 20

stock04, 2, 14, 15–17
stock94, 2, 14, 15, 16, 17
stock99, 2, 14–16, 16
stockModel, 2, 5–7, 9, 11–15, 17, 17, 21
stockPortfolio
(stockPortfolio-package), 2
summary.optimalPortfolio (optimalPort), 8
summary.portReturn (portReturn), 13
summary.stockModel (stockModel), 17
summary.stockReturns (getReturns), 6
summary.testPort (testPort), 20

testPort, 7, 9, 20, 20

stock94Info, 2, 14, 15, 16, 17

23