Package 'portfolioBacktest'

October 14, 2022

Title Automated Backtesting of Portfolios over Multiple Datasets

Version 0.4.1

Date 2022-04-22

Description Automated backtesting of multiple portfolios over multiple
datasets of stock prices in a rolling-window fashion. Intended for
researchers and practitioners to backtest a set of different portfolios,
as well as by a course instructor to assess the students in their portfolio
design in a fully automated and convenient manner, with results conveniently
formatted in tables and plots. Each portfolio design is easily defined as a
function that takes as input a window of the stock prices and outputs the
portfolio weights. Multiple portfolios can be easily specified as a list
of functions or as files in a folder. Multiple datasets can be conveniently
extracted randomly from different markets, different time periods, and
different subsets of the stock universe. The results can be later assessed
and ranked with tables based on a number of performance criteria (e.g.,
expected return, volatility, Sharpe ratio, drawdown, turnover rate, return
on investment, computational time, etc.), as well as plotted in a number of
ways with nice barplots and boxplots.

Maintainer Daniel P. Palomar <daniel.p.palomar@gmail.com>

URL https://CRAN.R-project.org/package=portfolioBacktest,
https://github.com/dppalomar/portfolioBacktest

BugReports https://github.com/dppalomar/portfolioBacktest/issues

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Depends R (>= 2.10)

Imports digest, evaluate, ggplot2, pbapply, PerformanceAnalytics,
parallel, quadprog, quantmod, R.utils, rlang, stats, utils,
xts, zoo

Suggests CVXR, DT, ggfortify, gridExtra, knitr, prettydoc, readtext,
rmarkdown, R.rsp, scales, stringi, testthat

VignetteBuilder knitr, rmarkdown, R.rsp

NeedsCompilation no

Author Daniel P. Palomar [cre, aut],
    Rui Zhou [aut]

Repository CRAN

Date/Publication 2022-04-22 08:30:09 UTC

R **topics documented:**

---

portfolioBacktest-package

                        portfolioBacktest: Automated Backtesting of Portfolios over Multiple
                        Datasets

Description

Automated backtesting of multiple portfolios over multiple datasets of stock prices in a rolling-window fashion. Intended for researchers and practitioners to backtest a set of different portfolios, as well as by a course instructor to assess the students in their portfolio design in a fully automated and convenient manner, with results conveniently formatted in tables and plots. Each portfolio design is easily defined as a function that takes as input a window of the stock prices and outputs

the portfolio weights. Multiple portfolios can be easily specified as a list of functions or as files in a folder. Multiple datasets can be conveniently extracted randomly from different markets, different time periods, and different subsets of the stock universe. The results can be later assessed and ranked with tables based on a number of performance criteria (e.g., expected return, volatility, Sharpe ratio, drawdown, turnover rate, return on investment, computational time, etc.), as well as plotted in a number of ways with nice barplots and boxplots.

Functions

stockDataDownload, financialDataResample, portfolioBacktest, backtestSelector, backtestTable, backtestBoxPlot, backtestLeaderboard, backtestChartCumReturn, backtestChartDrawdown, backtestChartStackedBar backtestSummary, summaryTable, summaryBarPlot

Data

dataset10, SP500_symbols

Help

For a quick help see the README file: GitHub-README.

For more details see the vignette: CRAN-vignette.

Author(s)

Daniel P. Palomar and Rui ZHOU

---

add_performance   *Add a new performance measure to backtests*

Description

Add a new performance measure to backtests

Usage

```
add_performance(bt, name, fun, desired_direction = 1)
```

Arguments

bt	Backtest results as produced by the function portfolioBacktest.
name	String with name of new performance measure.
fun	Function to compute new performance measure from any element returned by portfolioBacktest, e.g., return, wealth, and w_bop.
desired_direction	
	Number indicating whether the new measure is desired to be larger (1), which is the default, or smaller (-1).

Value

List with the portfolio backtest results, see `portfolioBacktest`.

Author(s)

Daniel P. Palomar

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
EWP_portfolio <- function(dataset, ...) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("EWP" = EWP_portfolio), dataset10)

# add a new performance measure
bt <- add_performance(bt, name = "SR arithmetic",
                      fun = function(return, ...)
                                PerformanceAnalytics::SharpeRatio.annualized(return,
                                                                     geometric = FALSE))

bt <- add_performance(bt, name = "avg leverage", desired_direction = -1,
                      fun = function(w_bop, ...)
                                if(anyNA(w_bop)) NA else mean(rowSums(abs(w_bop))))
```

---

backtestBoxPlot   *Create boxplot from backtest results*

Description

Create boxplot from a portfolio backtest obtained with the function `portfolioBacktest`. By default the boxplot is based on the package `ggplot2` (also plots a dot for each single backtest), but the user can also specify a simple base plot.

Usage

```
backtestBoxPlot(
  bt,
  measure = "Sharpe ratio",
  ref_portfolio = NULL,
```

```
    type = c("ggplot2", "simple"),
    ...
  )
```

Arguments

bt	Backtest results as produced by the function [portfolioBacktest]().
measure	String to select a performane measure from "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", and "ROT bps". Default is "Sharpe ratio".
ref_portfolio	Reference portfolio (whose measure will be subtracted). Default is NULL.
type	Type of plot. Valid options: "ggplot2", "simple". Default is "ggplot2".
...	Additional parameters. For example: mar for margins as in par() (for the case of plot type = "simple"); and alpha for the alpha of each backtest dot (for the case of plot type = "ggplot2"), set to 0 to remove the dots.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

[summaryBarPlot](), [backtestChartCumReturn](), [backtestChartDrawdown](), [backtestChartStackedBar]()

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
quintile_portfolio <- function(data, ...) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
                        benchmark = c("1/N", "index"))

# now we can plot
backtestBoxPlot(bt, "Sharpe ratio")
backtestBoxPlot(bt, "Sharpe ratio", type = "simple")
```

---

backtestChartCumReturn

Chart of the cumulative returns or wealth for a single backtest

---

Description

Create chart of the cumulative returns or wealth for a single backtest obtained with the function
[portfolioBacktest](). By default the chart is based on the package ggplot2, but the user can also
specify a plot based on PerformanceAnalytics.

Usage

```
backtestChartCumReturn(
  bt,
  portfolios = names(bt),
  dataset_num = 1,
  type = c("ggplot2", "simple"),
  ...
)
```

Arguments

bt	Backtest results as produced by the function [portfolioBacktest]().
portfolios	String with portfolio names to be charted. Default charts all portfolios in the backtest.
dataset_num	Dataset index to be charted. Default is dataset_num = 1.
type	Type of plot. Valid options: "ggplot2", "simple". Default is "ggplot2".
...	Additional parameters.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

[summaryBarPlot](), [backtestBoxPlot](), [backtestChartDrawdown](), [backtestChartStackedBar](), [backtestChartSharpeRati](

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
quintile_portfolio <- function(data, ...) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
```

```
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
                        benchmark = c("1/N", "index"))

# now we can chart
backtestChartCumReturn(bt)
```

---

backtestChartDrawdown   *Chart of the drawdown for a single backtest*

Description

Create chart of the drawdown for a single backtest obtained with the function portfolioBacktest. By default the chart is based on the package ggplot2, but the user can also specify a plot based on PerformanceAnalytics.

Usage

```
backtestChartDrawdown(
  bt,
  portfolios = names(bt),
  dataset_num = 1,
  type = c("ggplot2", "simple"),
  ...
)
```

Arguments

bt	Backtest results as produced by the function portfolioBacktest.
portfolios	String with portfolio names to be charted. Default charts all portfolios in the backtest.
dataset_num	Dataset index to be charted. Default is dataset_num = 1.
type	Type of plot. Valid options: "ggplot2", "simple". Default is "ggplot2".
...	Additional parameters.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

summaryBarPlot, backtestBoxPlot, backtestChartCumReturn, backtestChartStackedBar, backtestChartSharpeRat

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
quintile_portfolio <- function(data, ...) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
                        benchmark = c("1/N", "index"))

# now we can chart
backtestChartDrawdown(bt)
```

---

backtestChartSharpeRatio
                           Chart of the rolling Sharpe ratio over time for a single backtest

---

Description

Create chart of the rolling Sharpe ratio over time for a single backtest obtained with the function
portfolioBacktest. By default the chart is based on the package ggplot2, but the user can also
specify a plot based on PerformanceAnalytics.

Usage

```
backtestChartSharpeRatio(
  bt,
  portfolios = names(bt),
  dataset_num = 1,
  lookback = 100,
  by = 1,
  gap = lookback,
  bars_per_year = 252,
```

```
    type = c("ggplot2", "simple"),
    ...
)
```

Arguments

bt	Backtest results as produced by the function [portfolioBacktest]().
portfolios	String with portfolio names to be charted. Default charts all portfolios in the backtest.
dataset_num	Dataset index to be charted. Default is dataset_num = 1.
lookback	Length of the lookback rolling window in periods (default is 100).
by	Intervals at which the Sharpe ratio is to be calculated (default is equal to 1).
gap	Initial number of periods to skip (default is equal to lookback).
bars_per_year	Number of bars/periods per year (default is 252).
type	Type of plot. Valid options: "ggplot2", "simple". Default is "ggplot2".
...	Additional parameters.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

[summaryBarPlot](), [backtestBoxPlot](), [backtestChartCumReturn](), [backtestChartStackedBar](), [backtestChartDrawdown]()

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
quintile_portfolio <- function(data, ...) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
                        benchmark = c("1/N", "index"))

# now we can chart
backtestChartSharpeRatio(bt)
```

backtestChartStackedBar

> *Chart of the weight allocation over time for a portfolio over a single*
> *backtest*

Description

Create chart of the weight allocation over time for a portfolio over a single backtest obtained with
the function [portfolioBacktest](). By default the chart is based on the package ggplot2, but the
user can also specify a plot based on PerformanceAnalytics.

Usage

```
backtestChartStackedBar(
  bt,
  portfolio = names(bt[1]),
  dataset_num = 1,
  num_bars = 100,
  type = c("ggplot2", "simple"),
  legend = FALSE
)
```

Arguments

bt	Backtest results as produced by the function [portfolioBacktest]().
portfolio	String with portfolio name to be charted. Default charts the first portfolio in the backtest.
dataset_num	Dataset index to be charted. Default is dataset_num = 1.
num_bars	Number of bars shown over time (basically a downsample of the possibly long sequence).
type	Type of plot. Valid options: "ggplot2", "simple". Default is "ggplot2".
legend	Boolean to choose whether legend is plotted or not. Default is legend = FALSE.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

[summaryBarPlot](), [backtestBoxPlot](), [backtestChartCumReturn](), [backtestChartDrawdown]()

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# for better illustration, let's use only the first 5 stocks
dataset10_5stocks <- lapply(dataset10, function(x) {x$adjusted <- x$adjusted[, 1:5]; return(x)})

# define GMVP (with heuristic not to allow shorting)
GMVP_portfolio_fun <- function(dataset, ...) {
  X <- diff(log(dataset$adjusted))[-1]  # compute log returns
  Sigma <- cov(X)  # compute SCM
  # design GMVP
  w <- solve(Sigma, rep(1, nrow(Sigma)))
  w <- abs(w)/sum(abs(w))
  return(w)
}

# backtest
bt <- portfolioBacktest(list("GMVP" = GMVP_portfolio_fun), dataset10_5stocks, rebalance_every = 20)

# now we can chart
backtestChartStackedBar(bt, "GMVP", type = "simple")
backtestChartStackedBar(bt, "GMVP", type = "simple", legend = TRUE)
backtestChartStackedBar(bt, "GMVP")
backtestChartStackedBar(bt, "GMVP", legend = TRUE)
```

---

backtestLeaderboard *Leaderboard of portfolios from the backtest results*

Description

Leaderboard of portfolios according to the backtesting results and a ranking based on the combination of several performance criteria. Since the different performance measures hava different ranges and distributions, each is first transformed according to its empirical distribution function (along the empirical distribution of the portfolios being ranked) to obtain percentile scores. After that transformation, each of the measures has an empirical uniform distribution in the interval [0, 100] and can be weighted to obtain the final ranking.

Usage

```
backtestLeaderboard(
  bt = NA,
  weights = list(),
  summary_fun = median,
  show_benchmark = TRUE
)
```

Arguments

bt	Backtest results as produced by the function [portfolioBacktest]().
weights	List of weights for the different performance measures as obtained in [backtestSummary]()$performance (i.e., "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", "ROT bps", as well as "cpu time" and "failure rate". For example: weights = list("Sharpe ratio" = 8, "max drawdown" = 4).
summary_fun	Summary function to be employed (e.g., median or mean).
show_benchmark	Logical value indicating whether to include benchmarks in the summary (default is TRUE).

Value

List with the following elements:

leaderboard_scores
> Matrix with the individual scores for the portfolios (as chosen in weights) and the final score.

leaderboard_performance
> Matrix with all the performance measures for the portfolios.

error_summary Error messages generated by each portfolio on each dataset. Useful for debugging and give feedback to the portfolio managers of the different portfolios.

Author(s)

Daniel P. Palomar and Rui Zhou

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
quintile_portfolio <- function(data, ...) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(quintile_portfolio, dataset10,
                        benchmark = c("1/N", "index"))

# see all performance measures available for the ranking
backtestSummary(bt)$performance
```

```
# show leaderboard
leaderboard <- backtestLeaderboard(bt, weights = list("Sharpe ratio"  = 6,
                                                      "max drawdown"  = 1,
                                                      "ROT (bps)"     = 1,
                                                      "cpu time"      = 1,
                                                      "failure rate"  = 1))
leaderboard$leaderboard_scores
```

---

backtestSelector *Selector of portfolio backtest results*

Description

Select the results from a portfolio backtest.

Usage

```
backtestSelector(
  bt,
  portfolio_index = NULL,
  portfolio_name = NULL,
  measures = NULL
)
```

Arguments

bt   Backtest results as produced by the function [portfolioBacktest](#).

portfolio_index
                  Index number of a portfolio, e.g., 1 means to select the performance of the first
                  portfolio recorded in bt.

portfolio_name   String name of a portfolio, e.g., "GMVP" means to select the performance of
                  portfolio with name "GMVP" in bt. Only considered when portfolio_index is
                  not passed.

measures   String vector to select performane measures (default is all) from "Sharpe ratio",
                  "max drawdown", "annual return", "annual volatility", "Sterling ratio",
                  "Omega ratio", and "ROT bps".

Value

List with the following elements:

performance   Performance measures selected by argument measures.

error   Error status (TRUE or FALSE) of portfolio over each dataset (TRUE is when the
                  portfolio function generates an error or the maximum CPU time is exceeded).

error_message	Error messages generated by portfolio function over each dataset. Useful for debugging purposes.
cpu time	CPU usage by portfolio function over each dataset.
portfolio	Portfolio weights generated by portfolio function over each dataset.
return	Portfolio returns over each dataset.
wealth	Portfolio wealth (aka cumulative returns or cumulative P&L) over each dataset.

Author(s)

Rui Zhou and Daniel P. Palomar

Examples

```
library(portfolioBacktest)
data("dataset10")  # load dataset

# define your own portfolio function
EWP_portfolio <- function(dataset, ...) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("EWP" = EWP_portfolio), dataset10)

# extract your interested portfolio result
bt_sel <- backtestSelector(bt, portfolio_name = "EWP")
names(bt_sel)
```

---

backtestSummary   *Summary of portfolio backtest*

Description

Summarize the results from a portfolio backtest.

Usage

```
backtestSummary(
  bt,
  portfolio_indexes = NA,
  portfolio_names = NA,
  summary_fun = median,
  show_benchmark = TRUE
)
```

Arguments

bt	Backtest results as produced by the function [portfolioBacktest]().
portfolio_indexes	
	Numerical vector of portfolio indexes whose performance will be summarized, e.g., c(1, 2) means to summarize the performance of the first and second portfolios recorded in bt.
portfolio_names	
	String vector of portfolio names whose performance will be summarized, e.g., c("EWP", "GMVP") means to summarize the performance of portfolios with names "EWP" and "GMVP" in bt (default is names(bt) except the benchmark names). Only considered when portfolio_indexes is not passed.
summary_fun	Summary function to be employed (e.g., median or mean).
show_benchmark	Logical value indicating whether to include benchmarks in the summary (default is TRUE).

Value

List with the following elements:

performance_summary	
	Performance criteria: "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", "ROT bps", "VaR (0.95)", "CVaR (0.95)", "cpu time", and "failure rate". Default is "Sharpe ratio".
error_message	Error messages generated by each portfolio function over each dataset. Useful for debugging purposes.

Author(s)

Rui Zhou and Daniel P. Palomar

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
EWP_portfolio <- function(dataset, ...) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("EWP" = EWP_portfolio), dataset10)

# show the summary
bt_sum <- backtestSummary(bt)
names(bt_sum)
```

```
bt_sum$performance_summary
```

---

backtestTable   *Table with portfolio backtest results*

Description

Create table with the results from a portfolio backtest.

Usage

```
backtestTable(
  bt,
  portfolio_indexes = NA,
  portfolio_names = NA,
  show_benchmark = TRUE,
  measures = NULL
)
```

Arguments

bt   Backtest results as produced by the function [portfolioBacktest](#).

portfolio_indexes

                    Numerical vector of portfolio indexes whose performance will be summarized, e.g., `c(1, 2)` means to summarize the performance of the first and second portfolios recorded in `bt`.

portfolio_names

                    String vector of portfolio names whose performance will be summarized, e.g., `c("EWP", "GMVP")` means to summarize the performance of portfolios with names `"EWP"` and `"GMVP"` in `bt` (default is `names(bt)` except the benchmark names). Only considered when `portfolio_indexes` is not passed.

show_benchmark   Logical value indicating whether to include benchmarks in the summary (default is `TRUE`).

measures   String vector to select performane measures (default is all) from `"Sharpe ratio"`, `"max drawdown"`, `"annual return"`, `"annual volatility"`, `"Sterling ratio"`, `"Omega ratio"`, `"ROT bps"`, `"error"`, `"cpu time"`, and `"error_message"`.

Value

List with the following elements:

<performance criterion>

                    One item per performance measures as selected by argument `measures`.

error   Error status (`TRUE` or `FALSE`) for each portfolio over each dataset (`TRUE` is when the portfolio function generates an error or the maximum CPU time is exceeded).

| cpu time | CPU usage by each portfolio function over each dataset. |
| error_message | Error messages generated by each portfolio function over each dataset. Useful for debugging purposes. |

Author(s)

Rui Zhou and Daniel P. Palomar

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
EWP_portfolio <- function(dataset, ...) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("EWP" = EWP_portfolio), dataset10)

# show the backtest results in table
bt_tab <- backtestTable(bt)
bt_tab[c("Sharpe ratio", "max drawdown")]
```

---

dataset10   *Ten datasets obtained by resampling the S&P 500*

Description

Ten datasets of stock market data resampled from the S&P 500. Each resample contains a random selection of 50 stocks from the S&P 500 universe and a period of two years with a random initial point.

Usage

```
data(dataset10)
```

Format

List of 10 datasets, each contains two xts objects:

adjusted 505 x 50 xts with the adjusted prices of the 50 stocks

index 505 x 1 xts with the market index prices

Source

Yahoo! Finance

---

financialDataResample *Generate random resamples from financial data*

Description

This function resamples the financial data (e.g., downloaded with `stockDataDownload`) to obtain many datasets for a subsequent backtesting with `portfolioBacktest`. Given the original data, each resample is obtained by randomly choosing a subset of the financial instruments and randomly choosing a time period over the available long period.

Usage

```
financialDataResample(
  X,
  N_sample = 50,
  T_sample = 2 * 252,
  num_datasets = 10,
  rm_stocks_with_na = TRUE
)
```

Arguments

X	List of `xts` objects matching the structure returned by the function `stockDataDownload`.
N_sample	Desired number of financial instruments in each resample.
T_sample	Desired length of each resample (consecutive samples with a random initial time).
num_datasets	Number of resampled datasets (chosen randomly among the financial instrument universe).
rm_stocks_with_na	
	Logical value indicating whether to remove instruments with inner missing values. Default is `TRUE`.

Value

List of datasets resampled from `X`.

Author(s)

Rui Zhou and Daniel P. Palomar

See Also

`stockDataDownload`, `portfolioBacktest`

Examples

```
## Not run:
library(portfolioBacktest)
data(SP500_symbols)

# download data from internet
SP500_data <- stockDataDownload(stock_symbols = SP500_symbols,
                                from = "2009-01-01", to = "2009-12-31")

# generate 20 resamples from data, each with 10 stocks and one quarter continuous data
my_dataset_list <- financialDataResample(SP500_data, N = 10, T = 252/4, num_datasets = 20)

## End(Not run)
```

---

genRandomFuns   *Generate multiple versions of a function with randomly chosen parameters*

Description

Portfolio functions usually contain some parameters that can be tuned. This function creates multiple versions of a function with randomly chosen parameters. After backtesting those portfolios, the plotting function plotPerformanceVsParams can be used to show the performance vs parameters.

Usage

```
genRandomFuns(portfolio_fun, params_grid, name = "portfolio", N_funs = NULL)
```

Arguments

portfolio_fun	Portfolio function with parameters unspecified.
params_grid	Named list containing for each parameter the possible values it can take.
name	String with the name of the portfolio function.
N_funs	Number of functions to be generated.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

plotPerformanceVsParams

Examples

```
library(portfolioBacktest)

# define GMVP with parameters "delay", "lookback", and "regularize"
GMVP_portfolio_fun <- function(dataset, ...) {
  prices <- tail(lag(dataset$adjusted, delay), lookback)
  X <- diff(log(prices))[-1]
  Sigma <- cov(X)
  if (regularize)
    Sigma <- Sigma + 0.1 * mean(diag(Sigma)) * diag(ncol(Sigma))
  # design GMVP
  w <- solve(Sigma, rep(1, ncol(Sigma)))
  return(w/sum(w))
}

# generate the functions with random parameters
portfolio_list <- genRandomFuns(portfolio_fun = GMVP_portfolio_fun,
                                params_grid = list(lookback = c(100, 120, 140, 160),
                                                   delay = c(0, 5, 10, 15, 20),
                                                   regularize = c(FALSE, TRUE)),
                                name = "GMVP",
                                N_funs = 40)
names(portfolio_list)
portfolio_list[[1]]
rlang::env_print(portfolio_list[[1]])
rlang::fn_env(portfolio_list[[1]])$lookback
rlang::fn_env(portfolio_list[[1]])$delay
rlang::fn_env(portfolio_list[[1]])$regularize
```

---

listPortfoliosWithFailures
                    List portfolios with failures

Description

List portfolios with failures

Usage

```
listPortfoliosWithFailures(bt)
```

Arguments

bt   Backtest results as produced by the function `portfolioBacktest`.

Author(s)

Daniel P. Palomar

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
portfolio_with_errors <- function(dataset, ...) {
  return(NA)
}

# do backtest
bt <- portfolioBacktest(list("Portfolio with errors" = portfolio_with_errors), dataset10)

listPortfoliosWithFailures(bt)
```

---

plotPerformanceVsParams

Plot performance of portfolio function vs choice of parameters

---

Description

Portfolio functions usually contain some parameters that can be tuned. After generating multiple versions of a portfolio function with randomly chosen parameters with the function genRandomFuns and doing the backtesting, this function can be used to plot the performance vs choice of parameters.

Usage

```
plotPerformanceVsParams(
  bt_all_portfolios,
  params_subset = NULL,
  name_performance = "Sharpe ratio",
  summary_fun = median
)
```

Arguments

bt_all_portfolios

Backtest results as produced by the function portfolioBacktest.

params_subset   List of named parameters with a subset of the values to be considered. By default all the possible values will be considered.

name_performance

String with the name of the performance measure to be used.

summary_fun   Summary function to be employed (e.g., median or mean). Defult is median.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

genRandomFuns

Examples

```
library(portfolioBacktest)

# define GMVP with parameters "delay", "lookback", and "regularize"
GMVP_portfolio_fun <- function(dataset, ...) {
  prices <- tail(lag(dataset$adjusted, delay), lookback)
  X <- diff(log(prices))[-1]
  Sigma <- cov(X)
  if (regularize)
    Sigma <- Sigma + 0.01*diag(ncol(Sigma))
  # design GMVP
  w <- solve(Sigma, rep(1, ncol(Sigma)))
  return(w/sum(w))
}

# generate the functions with random parameters
portfolio_list <- genRandomFuns(portfolio_fun = GMVP_portfolio_fun,
                                params_grid = list(lookback = c(100, 120, 140, 160),
                                                   delay = c(0, 5, 10, 15, 20),
                                                   regularize = c(FALSE, TRUE)),
                                name = "GMVP",
                                N_funs = 40)

# backtest portfolios
bt <- portfolioBacktest(portfolio_list, dataset10)

# plot
plotPerformanceVsParams(bt)
plotPerformanceVsParams(bt, params_subset = list(regularize = TRUE))
plotPerformanceVsParams(bt, params_subset = list(delay = 5))
plotPerformanceVsParams(bt, params_subset = list(delay = 5, regularize = TRUE))
```

---

portfolioBacktest   *Backtest multiple portfolios over multiple datasets of stock prices in a rolling-window basis*

Description

Automated backtesting of multiple portfolios over multiple datasets of stock prices in a rolling-window fashion. Each portfolio design is easily defined as a function that takes as input a window of the stock prices and outputs the portfolio weights. Multiple portfolios can be easily specified as a list of functions or as files in a folder. Multiple datasets can be conveniently obtained with the function `financialDataResample` that resamples the data downloaded with the function `stockDataDownload`. The results can be later assessed and arranged with tables and plots. The backtesting can be highly time-consuming depending on the number of portfolios and datasets can be performed with parallel computation over multiple cores. Errors in functions are properly catched and handled so that the execution of the overal backtesting is not stopped (error messages are stored for debugging purposes). See vignette for a detailed explanation.

Usage

```
portfolioBacktest(
  portfolio_funs = NULL,
  dataset_list,
  folder_path = NULL,
  source_to_local = TRUE,
  price_name = NULL,
  paral_portfolios = 1,
  paral_datasets = 1,
  show_progress_bar = FALSE,
  benchmarks = NULL,
  shortselling = TRUE,
  leverage = Inf,
  lookback = 252,
  T_rolling_window = NULL,
  optimize_every = 20,
  rebalance_every = 1,
  bars_per_year = 252,
  execution = c("same period", "next period"),
  cost = list(buy = 0, sell = 0, short = 0, long_leverage = 0),
  cpu_time_limit = Inf,
  return_portfolio = TRUE,
  return_returns = TRUE
)
```

Arguments

portfolio_funs   List of functions (can also be a single function), each of them taking as input a dataset containing a list of xts objects (following the format of each element of the argument `dataset_list`) properly windowed (following the rolling-window approach) and returning the portfolio as a vector of normalized weights. See vignette for details.

dataset_list   List of datasets, each containing a list of xts objects, as generated by the function `financialDataResample`.

folder_path   If `portfolio_funs` is not defined, this should contain the path to a folder containing the portfolio functions saved in files. See [vignette](#) for details.

source_to_local

Logical value indicating whether to source files to local environment (default is `TRUE`). It might be dangerous to set it to `FALSE` as in such case the global environment may be changed. We suggest only to allow `FALSE` when the code in the source files does not work when locally sourced, e.g., with some versions of package CVXR. In that case, we further recommend to set `paral_portfolios` > 1 to avoid changing the global environment.

price_name   Name of the `xts` object in each dataset that contains the prices to be used in the portfolio return computation (default is the name of the first `xts` object).

paral_portfolios

Interger indicating number of portfolios to be evaluated in parallel (default is 1), see [vignette-paralle-mode](#) for details.

paral_datasets   Interger indicating number of datasets to be evaluated in parallel (default is 1), see [vignette-paralle-mode](#) for details.

show_progress_bar

Logical value indicating whether to show progress bar (default is `FALSE`).

benchmarks   String vector indicating the benchmark portfolios to be incorporated, currently supports:

- `1/N` - the 1/N portfolio, $w = [1/N, ..., 1/N]$ with N be number of stocks;
- `IVP` - the inverse-volatility portfolio, with weights be inversely proportional the standard deviation of returns;
- `index` - the market index, requires an `xts` named 'index' in the datasets.

shortselling   Logical value indicating whether shortselling is allowed or not (default is `TRUE`, so no control for shorselling in the backtesting).

leverage   Amount of leverage as in $||w||_1 <= leverage$ (default is `Inf`, so no control for leverage in the backtesting).

lookback   Length of the lookback rolling window in periods (default is 252).

T_rolling_window

Deprecated: use `lookback` instead.

optimize_every   How often the portfolio is to be optimized in periods (default is 20).

rebalance_every

How often the portfolio is to be rebalanced in periods (default is 1).

bars_per_year   Number of bars/periods per year. By default it will be calculated automatically (e.g., for daily data there are 252 bars/periods per year).

execution   String that can be either `"same period"` (default) or `"next period"`. At the rebalancing period `t`, the portfolio has used information up to (and including) period `t`. Same period execution means one can get into the position at that period `t`, whereas the next period execution means that one can only get into the position the following period.

cost   List containing four different types of transaction costs (common for all assets) for buying, selling, shorting, and long leveraging. The default is `cost = list(buy = 0e-4, sell = 0e-4, short = 0e-4, long_leverage = 0e-4)`. If some elements are not specified then they will be automatically set to zero.

cpu_time_limit   Time limit for executing each portfolio function over a single data set (default is
                 Inf, so no time limit).

return_portfolio

> Logical value indicating whether to return the portfolios (default is TRUE). Three
> portfolio series are returned: w_optimized is the optimized portfolio at each
> given optimization period (using all the information up to and including that
> period, which can be executed either on the same period or the following period),
> w_rebalanced is the rebalanced portfolio at each given rebalancing period, and
> w_bop is the "beginning-of-period" portfolio (i.e., at each period it contains the
> weights held in the market in the previous period so that the portfolio return at
> that period is just the product of the asset returns and w_bop at that period.)

return_returns   Logical value indicating whether to return the portfolio returns (default is TRUE).
                 Three series are returned: return with the portfolio returns, wealth with the
                 portfolio wealth (aka cumulative P&L), and X_lin with the returns of the as-
                 sets in the universe (note that the portfolio returns can also be obtained as
                 rowSums(X_lin * w_bop) in the absence of transaction costs).

Value

List with the portfolio backtest results, see vignette-result-format for details. It can be accessed di-
rectly, but we highly recommend the use of the package specific functions to extract any required in-
formation, namely, backtestSelector, backtestTable, backtestBoxPlot, backtestLeaderboard,
backtestSummary, summaryTable, summaryBarPlot.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

stockDataDownload, financialDataResample, backtestSelector, backtestTable, backtestBoxPlot,
backtestLeaderboard, backtestSummary, summaryTable, summaryBarPlot.

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
ewp_portfolio <- function(dataset, ...) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("EWP" = ewp_portfolio), dataset10)

# check your result
names(bt)
```

```
backtestSelector(bt, portfolio_name = "EWP", measures = c("Sharpe ratio", "max drawdown"))
backtestTable(bt, measures = c("Sharpe ratio", "max drawdown"))
bt_summary <- backtestSummary(bt)
summaryTable(bt_summary)
```

SP500_symbols   *Stock symbols of the S&P 500 constituents*

Description

Stock symbols of the S&P 500 constituents

Usage

```
data(SP500_symbols)
```

Format

String vector of stock symbols of the S&P 500 constituents. The market index symbol is concluded as the attribute "index_symbol".

Source

[Yahoo! Finance](#)

---

stockDataDownload   *Download stock data from the Internet*

Description

This function is basically a robust wrapper for [quantmod:getSymbols](#) to download stock data from the internet. It will return 6 xts objects of the same dimensions named 'open', 'high', 'low', 'close', 'volume', 'adjusted' and 'index'. Additionally, it can return an xts object with an index. If the download for some stock fails after a few attempts they will be ignored and reported. Also, stocks with missing values can be optionally removed.

Usage

```
stockDataDownload(
  stock_symbols,
  index_symbol = NULL,
  from,
  to,
  rm_stocks_with_na = TRUE,
  local_file_path = getwd(),
  ...
)
```

Arguments

stock_symbols	String vector containing the symbols of the stocks to be downloaded. User can pass the market index symbol as its attribute 'index_symbol' (only considered when argument 'index_symbol' is not passed).
index_symbol	String of the market index symbol.
from	String as the starting date, e.g., "2017-08-17".
to	String as the ending date (not included), e.g., "2017-09-17".
rm_stocks_with_na	
	Logical value indicating whether to remove stocks with missing values (ignoring leading missing values). Default is TRUE.
local_file_path	
	Path where the stock data will be saved after the first time is downloaded, so that in future retrievals it will be locally loaded (if the same arguments are used). Default is getwd(). If local caching is not desired, it can be deactivated by setting local_file_path = NULL.
...	Additional arguments to be passed to [quantmod:getSymbols]().

Value

List of 7 xts objects named 'open', 'high', 'low', 'close', 'volume', 'adjusted' and 'index'. Note that 'index' will only be returned when correct index symbols is passed.

Author(s)

Rui Zhou and Daniel P. Palomar

See Also

[financialDataResample]()

Examples

```
## Not run:
library(portfolioBacktest)
data(SP500_symbols)

# download data from internet
SP500_data <- stockDataDownload(stock_symbols = SP500_symbols,
                                from = "2009-01-01", to = "2009-12-31")

## End(Not run)
```

---

stockDataResample   *Generate random resamples from financial data*

Description

This function is deprecated. Use instead financialDataResample().

Usage

```
stockDataResample(
  X,
  N_sample = 50,
  T_sample = 2 * 252,
  num_datasets = 10,
  rm_stocks_with_na = TRUE
)
```

Arguments

X	List of xts objects matching the structure returned by the function stockDataDownload.
N_sample	Desired number of financial instruments in each resample.
T_sample	Desired length of each resample (consecutive samples with a random initial time).
num_datasets	Number of resampled datasets (chosen randomly among the financial instrument universe).
rm_stocks_with_na	
	Logical value indicating whether to remove instruments with inner missing values. Default is TRUE.

---

summaryBarPlot   *Create barplot from backtest summary*

Description

After performing a backtest with portfolioBacktest and obtaining a summary of the performance measures with backtestSummary, this function creates a barplot from the summary. By default the plot is based on the package ggplot2, but the user can also specify a simple base plot.

Usage

```
summaryBarPlot(bt_summary, measures = NULL, type = c("ggplot2", "simple"), ...)
```

Arguments

bt_summary	Backtest summary as obtained from the function backtestSummary.
measures	String vector to select performane measures (default is all) from 'Sharpe ratio', 'max drawdown', 'annual return', 'annual volatility', 'Sterling ratio', 'Omega ratio', 'ROT bps', etc.
type	Type of plot. Valid options: "ggplot2", "simple". Default is "ggplot2".
...	Additional parameters (only used for plot type = "simple"); for example: mar for margins as in par(), inset for the legend inset as in legend(), legend_loc for the legend location as in legend().

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

summaryTable, backtestBoxPlot, backtestChartCumReturn, backtestChartDrawdown, backtestChartStackedBar

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
quintile_portfolio <- function(data, ...) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
                        benchmark = c("1/N", "index"))

# now we can obtain the table
bt_summary_median <- backtestSummary(bt)
summaryBarPlot(bt_summary_median, measures = c("max drawdown", "annual volatility"))
summaryBarPlot(bt_summary_median, measures = c("max drawdown", "annual volatility"),
               type = "simple")
```

summaryTable   *Create table from backtest summary*

Description

After performing a backtest with [portfolioBacktest](#) and obtaining a summary of the performance measures with [backtestSummary](#), this function creates a table from the summary. By default the table is a simple matrix, but if the user has installed the package DT or grid.table nicer tables can be generated.

Usage

```
summaryTable(
  bt_summary,
  measures = NULL,
  caption = "Performance table",
  type = c("simple", "DT", "kable", "grid.table"),
  digits = 2,
  order_col = NULL,
  order_dir = c("asc", "desc"),
  page_length = 10
)
```

Arguments

bt_summary	Backtest summary as obtained from the function backtestSummary.
measures	String vector to select performane measures (default is all) from 'Sharpe ratio', 'max drawdown', 'annual return', 'annual volatility', 'Sterling ratio', 'Omega ratio', 'ROT bps', etc.
caption	Table caption (only works for type = "DT").
type	Type of table. Valid options: "simple", "DT", "kable", "grid.table". Default is "simple" and generates a simple matrix (with the other choices the corresponding package must be installed).
digits	Integer indicating the number of decimal places when rounding (default is 2).
order_col	Column number or column name of the performance measure to be used to sort the rows (only used for table type = "DT"). By default the last column will be used.
order_dir	Direction to be used to sort the rows (only used for table type = "DT"). Valid options: "asc", "desc". Default is "asc".
page_length	Page length for the table (only used for table type = "DT"). Default is 10.

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

summaryBarPlot

Examples

```
library(portfolioBacktest)
data(dataset10)  # load dataset

# define your own portfolio function
quintile_portfolio <- function(data, ...) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio),
                        dataset10,
                        benchmark = c("1/N", "index"))

# now we can obtain the table
bt_summary_median <- backtestSummary(bt)
summaryTable(bt_summary_median, measures = c("max drawdown", "annual volatility"))
summaryTable(bt_summary_median, measures = c("max drawdown", "annual volatility"), type = "DT")
summaryTable(bt_summary_median, type = "kable")
# this returned kable object can be combined with: " |> kableExtra::kable_styling()"
```

Index