Introduction to the matiming package

Valeriy Zakamulin*

This revision: September 7, 2017

Contents

*School of Business and Law, University of Agder, Service Box 422, 4604 Kristiansand, Norway, E-mail: Valeri.Zakamouline@uia.no, Website: http://vzakamulin.weebly.com/

1 Introduction

The `matiming` package contains functions to compute moving averages, simulate moving average trading strategies, and to perform back-tests and forward tests of these strategies. For the sake of convenience, the package includes the monthly and daily data on the returns (both capital gain and total) of the Standard and Poor's Composite index and the Dow Jones Industrial Average index, as well as the risk-free rate of returns.

The package is provided AS IS, without any implied warranty as to its accuracy or suitability. The main reference book that describes the computation of moving averages, moving average trading rules, performance measurement, and methodology of back- and forward-testing is (2017) "**Market Timing with Moving Averages: The Anatomy and Performance of Trading Rules**" by Valeriy Zakamulin.

2 Data

The following datasets are provided by the package:

Object name	**Index**	**Data frequency**	Start date	**End date**
`sp500.monthly`	S&P Composite	monthly	January 1857	December 2015
`sp500.daily`	S&P Composite	daily	July 1926	December 2015
`djia.monthly`	Dow Jones	monthly	July 1926	December 2015
`djia.daily`	Dow Jones	daily	July 1926	December 2015

Figure 1: The description of the datasets

All datasets represent `zoo` time-series objects with the following data columns:

CAP: Capital gain return

TOT: Total return

RF: Risk-free rate of return

In all datasets, the dates are stored in `Date` format `"%Y-%m-%d"`.

For example, to access the dates and the data in dataset `djia.monthly`, one can use the following command sequence

```
library(zoo)
data <- djia.monthly
dates <- index(data)
data <- coredata(data)
capret <- data[,"CAP"]
totret <- data[,"TOT"]
rfret <- data[,"RF"]
```

As a result, object `dates` is the vector of dates, objects `capret`, `totret`, and `rfret` are the vectors of capital gain returns, total returns, and the risk-free rates of return respectively.

3 The organization of the functions supplied by the package

All functions supplied by the package can be divided into the following families of functions:

- Functions that compute different moving averages;
- Functions that simulate the returns to different trading strategies and merge the results of simulations;
- Functions that compute the performance of trading strategies;
- Functions that perform back- and forward tests of trading strategies;
- Function that report the descriptive statistics of trading rules and plot the results

4 Functions that compute moving averages

The following functions are supplied:

SMA : Computes the Simple Moving Average

LMA : Computes the Linear Moving Average

EMA : Computes the (infinite) Exponential Moving Average

DMA : Computes the Double Exponential Moving Average (see Mulloy (1994))

HMA : Computes the Hull Moving Average (see Hull (2005))

ZMA : Computes the Zero Lag Exponential Moving Average (see Ehlers and Way (2010))

All functions take two arguments: a vector of prices and a scalar that defines the size of the averaging window.[1] Functions `SMA`, `LMA`, and `EMA` are implemented in `C` and use recursive algorithms; hence they compute moving averages very fast. The other moving averages are computed using `SMA`, `LMA`, and `EMA` functions.

An example of using SMA function

```
sma <- SMA(prices, 10)
```

However, there is no necessity to call these functions directly; only pointers to this functions are provided as arguments in other functions. This family of functions can be extended by adding your own functions that compute other types of moving averages.

5 Functions that simulate the returns to different trading rules and merge the results of simulations

To simulate the returns to different trading rules, the following functions are supplied:

`sim.mom.strategy :` Simulate the returns to the Momentum (MOM) strategy

`sim.mac.strategy :` Simulate the returns to the Moving Average Crossover (MAC) strategy

`sim.cdir.strategy :` Simulate the returns to the Moving Average Change of Direction (CDIR) strategy

`sim.macd.strategy :` Simulate the returns to the Moving Average Convergence/Divergence (MACD) strategy

`sim.mae.strategy :` Simulate the returns to the Moving Average Envelope (MAE) strategy

The usage of these functions:

```
sim.mom.strategy(totret, rfret, dates, capret=NULL, tc=0, shorts = FALSE,
                 winsize=200)

sim.mac.strategy(totret, rfret, dates, capret=NULL, tc=0, shorts = FALSE,
                 fast=50, slow=200, FUN=SMA)
```

[1] For EMA, the second argument is the length of the averaging window in SMA that has the same average lag time as EMA.

```
sim.cdir.strategy(totret, rfret, dates, capret=NULL, tc=0, shorts = FALSE,
                  winsize=200, FUN=SMA)

sim.macd.strategy(totret, rfret, dates, capret=NULL, tc=0, shorts = FALSE,
                  fast=12, slow=26, final=9, FUN=EMA)

sim.mae.strategy(totret, rfret, dates, capret=NULL, tc=0, shorts = FALSE,
                 winsize=200, percentage=1, FUN=SMA)
```

Most arguments in these functions are basically the same:

`totret` : the vector of total returns to the passive strategy

`rfret` : the vector of risk-free rates of return

`dates` : the vector of dates

`capret` : the vector of capital gain returns to the passive strategy

`tc` : a scalar that defines the amount of one-way proportional transaction costs. The transaction costs are in decimals, not in percentages. For example, `tc=0.001` defines 0.1% one-way transaction costs;

`shorts` : a logical variable. If `shorts=FALSE`, a Sell trading signal mandates selling the financial asset and moving to cash. If `shorts=TRUE`, a Sell trading signal mandates selling short the financial asset.

`FUN` : a function that defines the type of a moving average used to compute the trading indicator

The vector of capital gain returns is used to construct the time-series of prices. These prices are used in the computation of technical trading indicators. This methodology agrees with the common practice that trading signals are computed using prices **not** adjusted for dividends. In cases where the capital gain returns are not available (if `capret=NULL`) the time-series of prices is computed using the total returns. Hence, in this case (when `capret=NULL`) trading signals are computed using prices adjusted for dividends.

The argument `winsize` (a vector or a scalar) defines the size of the averaging window in MOM, MAE, and CDIR trading rules. `fast` and `slow` (each of them can be a vector or a scalar) define the sizes of the shorter (fast) and longer (slow) averaging windows in MAC and

MACD rules. The argument `final` (a vector or a scalar) defines the size of the averaging window for final smoothing in MACD rule. Finally, `percentage` (a vector or a scalar) defines the envelope percentage in MAE rule.

Each function returns an object (say, results) that contains the description of each simulated strategy and the simulated returns. For example, if `winsize=6` in MOM strategy, then the function simulates only one strategy, MOM(6). However, if `winsize=2:10`, then the function simulates 9 trading strategies (MOM(2), MOM(3), ..., MOM(10)).

All functions perform a check of validity (of the set) of arguments that define the sizes of the averaging windows. In MOM, MAE, and CDIR rules, functions perform a check that `winsize>1`. Similarly, in MACD rule the function perform a check that `final>1`. In MAC and MACD rules, functions perform a check that `slow>fast`. For example, one can define the following vectors and simulate the returns to the MAC strategy

```
fast <- 1:3
slow <- 2:4
res <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
           capret=capret, tc=tc, shorts=shorts, fast=fast, slow=slow, FUN=SMA)
```

Note that both vectors (`fast` and `slow`) have 3 elements, hence there are 9 possible combinations of `fast[i]` and `slow[j]`. However, not all of these combinations are valid. The function simulates only 6 strategies (MAC(1,2), MAC(1,3), MAC(1,4), MAC(2,3), MAC(2,4), MAC(3,4)) for which `fast[i] < slow[j]`.

The result returned by each function serves as an argument in functions that perform either the back- or forward test of different trading rules. For example, one can simulate a set of `MOM(n)` rules (for different values of `n`) and then test whether the MOM rule outperforms the buy-and-hold rule in an out-of-sample test. In this case a trader has only one set of available rules. However, in reality a trader may consider several sets of trading rules at the same time. For instance, in addition to the set of MOM rules a trader can consider a set of MAC rules. In this case, at each time t a trader finds the best trading rule in a back test among all available MOM and MAC rules (this combination of two or several rules is called **COMBI** strategy in the book "Market Timing with Moving Averages"). In order to test whether MOM and MAC rules outperform the buy-and-hold rule in an out-of-sample test, a trader needs to simulate the returns to both the rules and then to merge the results of these simulations. Merging the

results of simulation is performed by `sim.resuls.merge` function.

An example of a code that simulates the returns to different MAC and MOM strategies and merges the results of simulations is given below:

```
fast <- 1:5
slow <- 2:15
winsize <- 2:10
res1 <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
    capret=capret, tc=tc, shorts=shorts, fast=fast, slow=slow, FUN=SMA)
res2 <- sim.mom.strategy(totret=totret, rfret=rfret, dates=dates,
    capret=capret, tc=tc, shorts=shorts, winsize=winsize)
results <- sim.results.merge(res1, res2)
```

6 Functions that compute the performance of trading rules

Three following functions are supplied by the package:

`Excret :` Mean excess returns

`Sharpe :` Sharpe ratio

`Sortino :` Sortino ratio

The sole argument in all these functions is the vector of excess returns (strategy's returns in excess of the risk-free rates of return). An example of using `Sharpe` function

```
SR <- Sharpe(exret)
```

However, there is no necessity to call these functions directly; only pointers to this functions are provided as arguments in other functions. This family of functions can be extended by adding your own functions that compute other performance measures using excess returns.

7 Functions that perform back- and forward tests of trading rules

7.1 Finding the best trading rules in a back test

Function `back.test` finds the optimal trading strategies in a back-test. The usage of this function

```
back.test(results, start.date, end.date, n.first=10, include.bh=FALSE, FUN=Sharpe)
```

where

`results :` the results of simulation of a trading strategy or multiple trading strategies

`start.date :` the start of the in-sample period

`end.date :` the end of the in-sample period

`n.first :` the (maximum) number of the best performing strategies in the returned object

`include.bh :` a logical value indicating whether to consider the Buy-and-Hold (bh) strategy in addition to the moving average strategies

`FUN :` the function that computes the performance of a trading strategy

This function performs the following. Given a set of simulated trading rules (in `results` argument) and a specific historical period (from `start.date` to `end.date`), the function finds and returns the trading rules with the best performance. The number of best trading rules is limited from above by argument `n.first` (a scalar). The function that computes the performance measure is defined by argument `FUN`. In principle, no one moving average strategy might outperform the buy-and-hold strategy over some specific historical period. Argument `include.bh=TRUE` allows one to consider the buy-and-hold strategy as a strategy with potentially one of the best performing strategies in a given historical period.

More formally, object `results` contains the returns to each simulated market timing rule over the full historical sample from time 0 till time T. Assume that there are totally n simulated trading rules. Denote by $(x_0^i, x_1^i, x_2^i, \ldots, x_T^i)$ the excess returns to trading rule number i. For each trading rule $i \in [1, n]$, this function measures the performance over the period from time $t_{\text{start}} \geq 0$ to time $t_{\text{end}} \leq T$:

$$\text{Performance strategy } i = PM(x^i_{\text{start}}, x^i_{\text{start}+1}, x^i_{\text{start}+2}, \ldots, x^i_{\text{end}}),$$

where $PM(\cdot)$ is a function that computes the performance (either `Excret`, `Sharpe`, or `Sortino`). Then the function sorts all trading rules in descending order from the best performing rule

to the worst performing rule, and finally returns the description of the first `n.first` best performing rules.

The following program simulates a set of MAC trading strategies and finds the first 10 best strategies (using the Sortino ratio) over January 1990 to December 2015. Function `best.is.strategies` reports the best strategies and plots their cumulate returns.

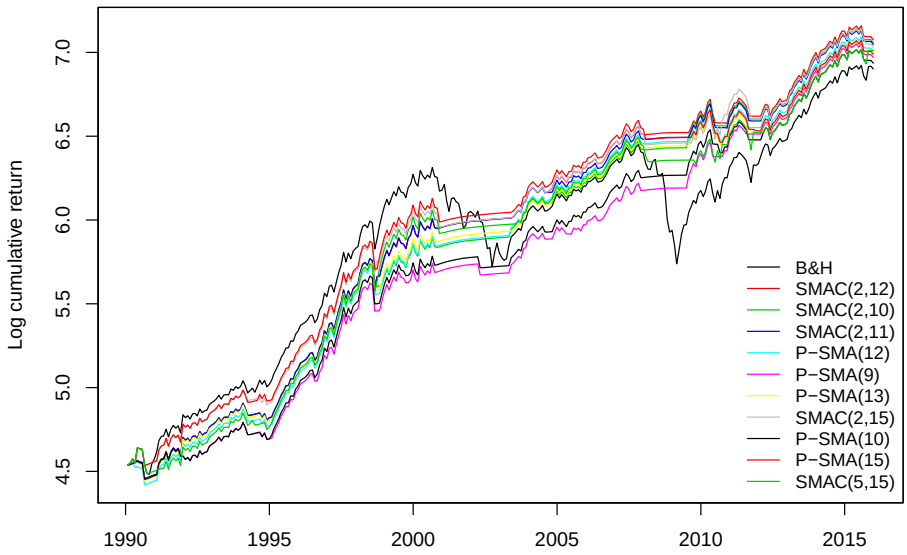
```
> rm(list=ls(all=TRUE))
> library(matiming)
> library(xtable)
> library(zoo)
> # access the data
> data <- sp500.monthly
> dates <- index(data)
> data <- coredata(data)
> capret <- data[,"CAP"]
> totret <- data[,"TOT"]
> rfret <- data[,"RF"]
> # simulate the trading strategies
> tc <- 0.0025
> fast <- 1:5
> slow <- 2:15
> shorts <- FALSE
> results <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
+                             capret=capret, tc=tc, shorts=shorts,
+                             fast=fast, slow=slow, FUN=SMA)
> # perform the back-test
> start.date <- as.Date("1990-01-01", format="%Y-%m-%d")
> end.date <- as.Date("2015-12-31", format="%Y-%m-%d")
> include.bh <- TRUE
> n.first <- 10
> res <- back.test(results, start.date=start.date, end.date=end.date,
```

```
+                     n.first=n.first, include.bh=include.bh, FUN=Sortino)

> # report and plot the results

> df <- best.is.strategies(res, to.annual=sqrt(12), plot.results=TRUE)

> xtab <- xtable(df, digits=3)
```




```
> print(xtab, include.rownames=FALSE)
```

Rank	Rule	Performance
1	SMAC(2,12)	1.089
2	SMAC(2,10)	1.079
3	SMAC(2,11)	1.072
4	P-SMA(12)	1.036
5	P-SMA(9)	1.032
6	P-SMA(13)	1.022
7	SMAC(2,15)	0.992
8	P-SMA(10)	0.990
9	P-SMA(15)	0.987
10	SMAC(5,15)	0.972
	B&H()	0.699

7.2 Simulating the returns to the out-of-sample trading strategy

Function `forward.test` simulates the returns to the out-of-sample trading strategy. The usage of this function:

```
forward.test(results, start.date, end.is.date, end.oos.date,
             refit.every=1, rolling=FALSE, include.bh=FALSE, FUN=Sharpe)
```

where

`results :` the results of simulation of a trading strategy or multiple trading strategies

`start.date :` the start of the in-sample period

`end.is.date :` the end of the initial in-sample period

`end.oos.date :` the end of the out-of-sample period

`refit.every :` this parameter determines how often the best trading strategy in a back test is determined

`rolling :` a logical value indicating whether to use the expanding (if FALSE) or rolling (if TRUE) in-sample window

`include.bh :` a logical value indicating whether to consider the Buy-and-Hold (bh) strategy in addition to the moving average strategies

`FUN :` the function that computes the performance of a trading strategy

In a forward test, the time $t+1$ return to the moving average strategy is defined using the trading signal of the best performing strategy till time t. That is, the trading signal for time $t+1$ is determined using only the information available at time t.

Before simulating out-of-sample returns to a trading strategy, the returns to a set of trading rules are simulated from time 0 till time T. To perform a forward test, one selects a period from time t_{start} (defined by `start.date`) till time t_{end} (defined by `end.oos.date`). This period is split into two segments: the initial in-sample period from time t_{start} till time t_{split} (defined by `end.is.date`) and the out-of-sample period from time $t_{\text{split}}+1$ till time t_{end}.

To generate the trading signal at time t_{split},[2] for each trading rule $i \in [1, n]$, the function measures the performance over the period from time t_{start} to time t_{split}:

$$\text{Performance strategy } i = PM(x^i_{\text{start}}, x^i_{\text{start}+1}, x^i_{\text{start}+2}, \ldots, x^i_{\text{split}}),$$

where $PM(\cdot)$ is a function that computes the performance and $(x^i_{\text{start}}, x^i_{\text{start}+1}, x^i_{\text{start}+2}, \ldots, x^i_{\text{split}})$ are the excess returns to trading rule i in the initial in-sample period. Then the trading signal is generated by the best trading rule (the rule with the highest performance) in the initial in-sample period.

If `refit.every>1`, the function uses the same trading rule to generate the trading signal at the subsequent times from time t_{split} till time t_{split}`+refit.every-1`. At time t_{split}`+refit.every` the function again finds the best trading strategy in the in-sample period. If `rolling=FALSE`, the new in-sample period starts from time t_{start} to time t_{split}`+refit.every`. Otherwise, if `rolling=TRUE`, the new in-sample period starts from time t_{start}`+refit.every` to time t_{split} `+refit.every`. In the former case, the length of the in-sample period increases. In the latter case, the length of the in-sample period remains the same; this type of test is called "walk-forward test".

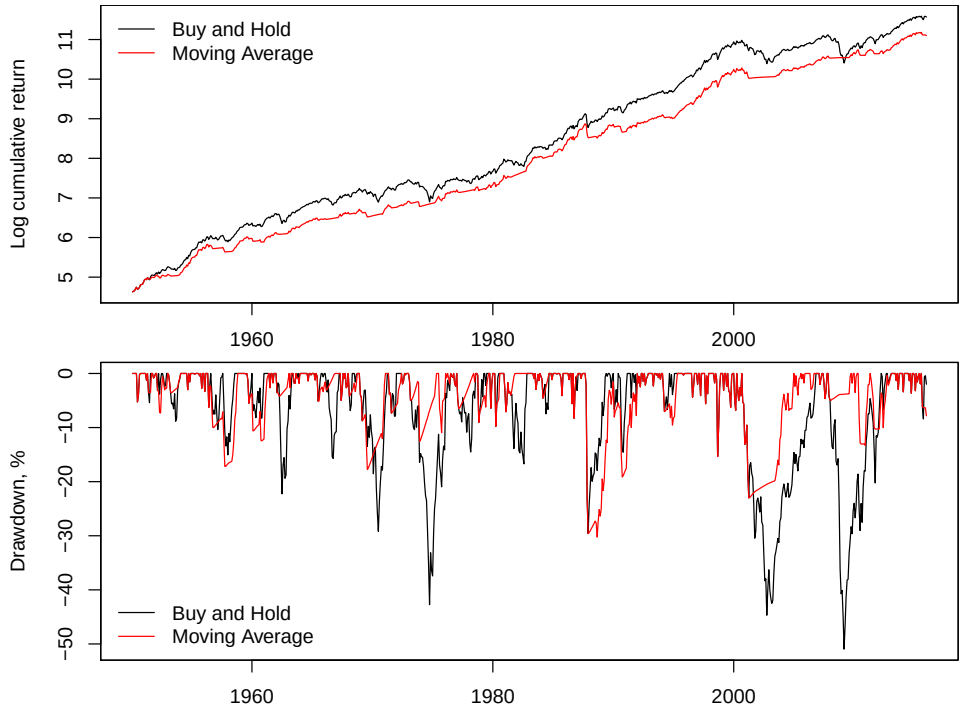
The following program first simulates a set of MAC trading strategies. Then the program simulates the returns to the out-of-sample trading strategy over the period from January 1950 till December 2015. The initial in-sample period is from January 1928 till December 1949; the function uses a rolling window scheme (because `rolling=TRUE`); the best trading rule in a back test is determined every each 5 periods (because `refit.every=5`); the set of the trading rules is augmented by the buy-and-hold strategy (because `include.bh=TRUE`); the performance measure is the Sortino ratio (because `FUN=Sortino`).

```
> rm(list=ls(all=TRUE))
> library(matiming)
> library(zoo)
> library(xtable)
> # access the data
```

[2] Note that the trading signal is generated at the end of the period and, therefore, it defines the return over the next period $t_{\text{split}} + 1$.

```
> data <- sp500.monthly
> dates <- index(data)
> data <- coredata(data)
> capret <- data[,"CAP"]
> totret <- data[,"TOT"]
> rfret <- data[,"RF"]
> # simulate the trading strategies
> tc <- 0.0025
> fast <- 1:5
> slow <- 2:15
> shorts <- FALSE
> results <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
+     capret=capret, tc=tc, shorts=shorts, fast=fast, slow=slow, FUN=SMA)
> # simulate out-of-sample returns
> start.date <- as.Date("1928-01-01", format="%Y-%m-%d")
> end.is.date <- as.Date("1949-12-31", format="%Y-%m-%d")
> end.oos.date <- as.Date("2015-12-31", format="%Y-%m-%d")
> refit.every <- 5
> include.bh <- TRUE
> rolling <- TRUE
> res <- forward.test(results, start.date=start.date, end.is.date=end.is.date,
+       end.oos.date=end.oos.date, refit.every=refit.every, rolling=rolling,
+       include.bh=include.bh, FUN=Sortino)

> # report the descriptive statistics, the p-value of the test, and plot the results
> df <- descriptive.statistics(res, plot.results = TRUE)
> xtab <- xtable(df, digits=2)
```




```
> print(xtab)
```

	BH	MA
Mean returns %	11.64	10.49
Std. deviation %	14.43	11.03
Minimum return %	-21.54	-21.54
Maximum return %	16.78	13.21
Skewness	-0.41	-0.53
Kurtosis	1.71	4.67
Average drawdown %	5.98	4.50
Average max drawdown %	28.23	16.31
Maximum drawdown %	50.96	30.25
Sortino	0.75	0.85
P-value		0.27
Rolling 5-year Win %		51.57
Rolling 10-year Win %		60.33

Note that the result of the simulation of the returns to the out-of-sample trading strategy is supposed to be processed by other functions supplied by the `matiming` package. For example, the program above uses function `descriptive.statistics` to report the descriptive statistics, the p-value of the test, and plot the results. However, if necessary, the user can access the results of simulation and process them using own functions.

The object returned by function `forward.test` represents a list that contains the following vectors (all of them are of the same length):

`totret :` a vector of total returns to the buy-and-hold strategy

`actret :` a vector of total returns to the out-of-sample trading strategy

`rfret :` a vector of risk-free rates of returns

`dates :` a vector of dates (from $t_{\text{split}} + 1$ till t_{end})

Below is an example of a program that simulates the out-of-sample returns to a set of MAC rules (this part is identical to that in the program above) and then processes the results of simulations without using functions in the `matiming` package:

```
rm(list=ls(all=TRUE))
library(matiming)
library(zoo)
library(xtable)
# access the data
data <- sp500.monthly
dates <- index(data)
data <- coredata(data)
capret <- data[,"CAP"]
totret <- data[,"TOT"]
rfret <- data[,"RF"]
# simulate the trading strategies
tc <- 0.0025
fast <- 1:5
slow <- 2:15
shorts <- FALSE
results <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
    capret=capret, tc=tc, shorts=shorts, fast=fast, slow=slow, FUN=SMA)
# simulate out-of-sample returns
start.date <- as.Date("1928-01-01", format="%Y-%m-%d")
end.is.date <- as.Date("1949-12-31", format="%Y-%m-%d")
end.oos.date <- as.Date("2015-12-31", format="%Y-%m-%d")
refit.every <- 5
include.bh <- TRUE
rolling <- TRUE
res <- forward.test(results, start.date=start.date, end.is.date=end.is.date,
      end.oos.date=end.oos.date, refit.every=refit.every, rolling=rolling,
      include.bh=include.bh, FUN=Sortino)
# access the results
totret <- res$totret
actret <- res$actret
```

```
rfret <- res$rfret
dates <- res$dates
# compute the Sharpe ratios
SR.bh <- Sharpe(totret-rfret)*sqrt(12)
SR.ma <- Sharpe(actret-rfret)*sqrt(12)
# plot the cumulative returns
cumret.bh <- cumprod(1+totret)
cumret.ma <- cumprod(1+actret)
data.zoo <- zoo(log(cbind(cumret.bh, cumret.ma)), order.by=dates)
names(data.zoo) <- c("BH","MA")
plot(data.zoo, plot.type="single", col=c("black","red"),
     ylab="Log cumulative returns", xlab="")
legend("topleft", legend=c("BH","MA"), col=c("black","red"), lty=1, bty = "n")
```

7.3 Outperformance test

Note that function `back.test` finds (and ranks) the best trading strategies in a back test without testing whether the best strategy in a back test outperforms its passive counterpart. We remind the reader that, due to data-mining bias, the performance of the best trading strategy in a back test is biased upward, whereas the p-value of the standard test of outperformance is biased downward. Currently, package `matiming` has no functions to correct the data-mining bias in the performance of the best rule in a back test.

Function `forward.test` simulates the returns to the out-of-sample trading strategy without testing whether the out-of-sample strategy outperforms its passive counterpart. To conduct the outperformance test, function `outperformance.test` is used. Specifically, function `outperformance.test` tests whether the moving average strategy outperforms its passive counterpart. All tests are based on using a bootstrap method. The usage of this function:

```
outperformance.test(results, digits=3, to.annual=sqrt(12),
                    type=c("ordinary", "block", "stationary"),
                    bLen=5, R=1000, automatic=FALSE)
```

where

`results :` the results returned by function `forward.test` or `back.test`

`digits :` defines the number of digits after the decimal delimiter to use in reporting the performance of the moving average strategy

`to.annual :` a coefficient used to annualize the performance measure. For example, when the

performance is measured using either `Sharpe` or `Sortino`, use `sqrt(252)` and `sqrt(12)` when the returns are at the daily and monthly frequency respectively

`type :` determines the type of the bootstrap method. The default type is "ordinary" (the method of Efron (1979)). The two other types are "block" (the method of Künsch (1989)) and "stationary" (the method of Politis and Romano (1994))

`bLen :` a scalar that specifies the block length in the block-bootstrap method

`R :` a scalar that specifies the number of bootstrap replicates

`automatic :` a logical variable that specifies whether to compute the optimal block length for the block-bootstrap and stationary bootstrap. If `automatic=TRUE`, the function computes the optimal block length using the method by Patton, Politis, and White (2009).

The goal of this function is to test the following null hypothesis:

$$H_0 : \Delta = PM_{MA} - PM_{BH} \leq 0,$$

where PM_{MA} and PM_{BH} are the performances of the moving average and buy-and-hold strategies respectively. In words, the null hypothesis is that the performance of the moving average strategy is not better than the performance of the buy-and-hold strategy. If one can reject this null hypothesis, then the performance of the moving average strategy is statistically significantly better than the performance of its passive counterpart. Note that even though one can conduct the outperformance test for the best trading strategy in a back test, in this case the p-value of the test in unreliable; function `outperformance.test` should be used mainly to test whether the out-of-sample trading strategy outperforms its passive counterpart.

This function returns the p-value of the test.

Below is an example of a program that simulates the out-of-sample returns to a set of MAC rules (this part is identical to that in the programs above) and then tests the outperformance hypothesis.

```
> rm(list=ls(all=TRUE))
> library(matiming)
> library(zoo)
```

```
> library(xtable)
> # access the data
> data <- sp500.monthly
> dates <- index(data)
> data <- coredata(data)
> capret <- data[,"CAP"]
> totret <- data[,"TOT"]
> rfret <- data[,"RF"]
> # simulate the trading strategies
> tc <- 0.0025
> fast <- 1:8
> slow <- 2:18
> shorts <- F
> res1 <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
+                          capret=capret, tc=tc, shorts=shorts, fast=fast,
+                          slow=slow, FUN=SMA)
> res2 <- sim.mom.strategy(totret=totret, rfret=rfret, dates=dates,
+                          capret=capret, tc=tc, shorts=shorts, winsize=slow)
> results <- sim.results.merge(res1, res2)
> # simulate the out-of-sample trading strategy using the rolling in-sample window
> start.date <- as.Date("1920-01-01", format="%Y-%m-%d")
> end.is.date <- as.Date("1949-12-31", format="%Y-%m-%d")
> end.oos.date <- as.Date("2015-12-31", format="%Y-%m-%d")
> refit.every <- 1
> rolling <- TRUE
> res <- forward.test(results, start.date, end.is.date, end.oos.date,
+                         refit.every=refit.every, rolling=rolling, FUN=Sharpe)
> pval <- outperformance.test(res, type="block")

============================================================
```

```
Performance of the BH strategy = 0.504
Performance of the MA strategy = 0.574

============================================================

The following null hypothesis is tested:
The performance of MA strategy <= the performance of BH strategy
Bootstrap method: block
Block length: 5
Number of simulations: 1000
P-value of the null hypotheis = 0.240
```

8 Function that report the descriptive statistics of trading rules and plot the results

8.1 Descriptive statistics, cumulative returns, and drawdowns

Function `best.is.strategies`

Function `best.is.strategies` reports the best trading strategies in a back-test. The usage of this function:

```
best.is.strategies(results, to.annual=sqrt(12), plot.results=FALSE)
```

where

`results :` the results returned by function `back.test`

`to.annual :` a coefficient used to annualize the performance measure. For example, when the performance is measured using either `Sharpe` or `Sortino`, use `sqrt(252)` and `sqrt(12)` when the returns are at the daily and monthly frequency respectively

`plot.results :` a logical variable that specifies whether to produce the plot of cumulative returns to the best trading strategies in a back test.

The function returns an object of class `data.frame` containing three columns: `Rank`, `Rule`, and `Performance`.

An example of using this function is provided in the description of function `back.test`. Below is a fragment of the program that performs the back tests of a set of trading rules and uses function `best.is.strategies` to plot the results and print the table with the best trading rules:

```
res <- back.test(results, start.date=start.date, end.date=end.date,
                 n.first=n.first, include.bh=include.bh, FUN=Sortino)
# report and plot the results
df <- best.is.strategies(res, to.annual=sqrt(12), plot.results=TRUE)
xtab <- xtable(df, digits=3)
print(xtab, include.rownames=FALSE)
```

Function `descriptive.statistics`

Function `descriptive.statistics` computes the summary statistics of the moving average trading strategy and the corresponding buy-and-hold strategy. This function also conducts the test of the outperformance hypothesis. If required, this function plots the cumulative returns and running drawdowns to the moving average trading strategy and the corresponding buy-and-hold strategy. The usage of this function:

```
descriptive.statistics(results, nobs.a.year = 12, to.annual=sqrt(12),
                       type=c("ordinary", "block", "stationary"),
                       bLen=5, R=1000, automatic=FALSE, plot.results=FALSE)
```

where

`results :` the results returned by function `forward.test` or `back.test`

`nobs.a.year :` a scalar that defines the number of observations a year. For example, for monthly (daily) data the number of observations a year equals 12 (252)

`to.annual :` a coefficient used to annualize the performance measure. For example, when the performance is measured using either `Sharpe` or `Sortino`, use `sqrt(252)` and `sqrt(12)` when the returns are at the daily and monthly frequency respectively. If the performance measure is `Excret`, use `12`

`type :` determines the type of the bootstrap method. The default type is "ordinary" (the method of Efron (1979)). The two other types are "block" (the method of Künsch (1989)) and "stationary" (the method of Politis and Romano (1994))

`bLen :` a scalar that specifies the block length in the block-bootstrap method

`R :` a scalar that specifies the number of bootstrap replicates

`automatic :` a logical variable that specifies whether to compute the optimal block length for the block-bootstrap and stationary bootstrap. If `automatic=TRUE`, the function computes the optimal block length using the method by Patton et al. (2009).

`plot.results :` a logical variable that specifies whether to produce the plot of cumulative returns and running drawdowns to the moving average strategy and the buy-and-hold strategy

If the `results` are returned by function `back.test`, the function computes the descriptive statistics of the best trading strategy in a back test. Otherwise, the function computes the descriptive statistics of the out-of-sample trading strategy.

This function returns an object of class `data.frame` containing two columns: `BH` and `MA`. The first column contains the summary statistics for the Buy-and-Hold strategy, the second column contains the summary statistics for the Moving Average strategy. Each column contains the following rows:

`Mean returns % :` annualized mean returns

`Std. deviation % :` annualized standard deviation

`Minimum return % :` minimum return

`Maximum return % :` maximum return

`Skewness :` skewness of return distribution

`Kurtosis :` kurtosis of return distribution

`Average drawdown % :` average drawdown

`Average max drawdown % :` average of the 10 largest drawdowns

`Maximum drawdown %` : maximum drawdown

`Performance` : Performance measure

`P-value` : P-value of the test of outperformance

`Rolling 5-year Win %` : Probability that the moving average strategy outperforms its passive counterpart over a 5-year horizon

`Rolling 10-year Win %` : Probability that the moving average strategy outperforms its passive counterpart over a 10-year horizon

The outperformance test is performed in a similar manner as in function `outperformance.test`.

An example of using this function is provided in the description of function `forward.test`. Below is a fragment of the program that performs the forward tests of a set of trading rules and uses function `descriptive.statistics` to plot the results and print the table with the descriptive statistics:

```
res <- forward.test(results, start.date=start.date, end.is.date=end.is.date,
      end.oos.date=end.oos.date, refit.every=refit.every, rolling=rolling,
      include.bh=include.bh, FUN=Sortino)
# report the descriptive statistics, the p-value of the test, and plot the results
df <- descriptive.statistics(res, plot.results = TRUE)
xtab <- xtable(df, digits=2)
print(xtab)
```

8.2 Rolling outperformance plots

The outperformance generated by a moving average trading strategy is very non-uniform over time. Therefore it is very useful to analyze the graph of rolling N-year outperformance. For this purpose the package provides two functions described below.

Function `is.outperformance.plot`

Function `is.outperformance.plot` plots the trading strategies rolling outperformance.

The usage of this function:

```
is.outperformance.plot(results, start.date, split.date, end.date, FUN=Sharpe)
```

where

`results :` the results of the back-test or simulation of a trading strategy or multiple trading strategies. The former is the result returned by function `back.test`. The latter is the result returned by function `sim.mom.strategy`, `sim.mac.strategy`, `sim.cdir.strategy`, `sim.macd.strategy`, `sim.results.merge`

`start.date :` the start of the first rolling period

`split.date :` the end of the first rolling period

`end.date :` the end of the total period

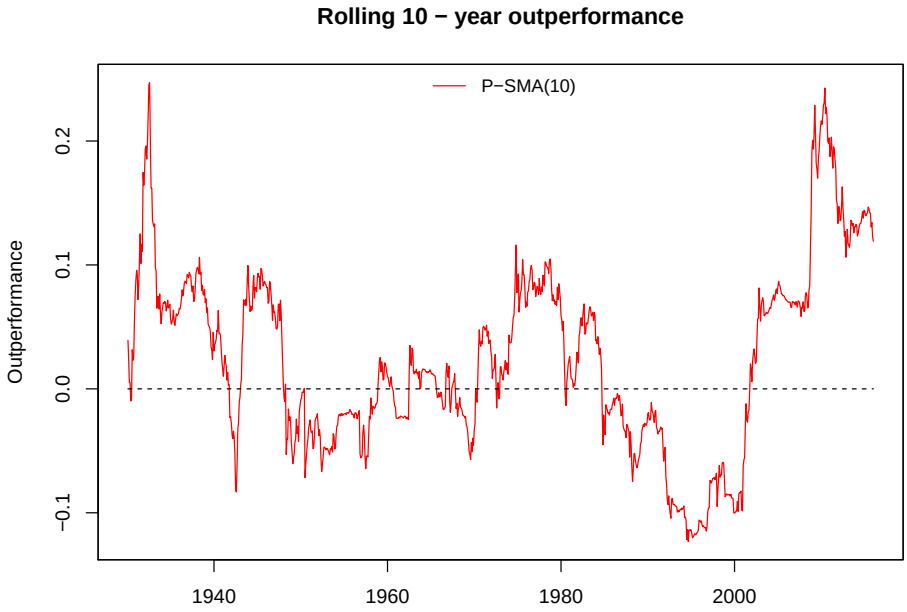
`FUN :` the function that computes the performance of a trading strategy. The function `FUN` is either `Excret`, `Sharpe`, or `Sortino`. The function `FUN` is not used when `results` are the results of the back-test.

The program below simulates the returns to the P-SMA(10) strategy and plots its rolling 10-year outperformance:

```
> rm(list=ls(all=TRUE))
> library(matiming)
> library(zoo)
> # access the data
> data <- sp500.monthly
> dates <- index(data)
> data <- coredata(data)
> capret <- data[,"CAP"]
> totret <- data[,"TOT"]
> rfret <- data[,"RF"]
> # simulate the SMA(10) strategy
> tc <- 0.0025
> fast <- 1
> slow <- 10
> shorts <- F
> results <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
```

```
+                           capret=capret, tc=tc, shorts=shorts, fast=fast,
+                           slow=slow, FUN=SMA)
> # plot the rolling outperformance
> start.date <- as.Date("1920-01-01", format="%Y-%m-%d")
> split.date <- as.Date("1929-12-31", format="%Y-%m-%d")
> end.date <- as.Date("2015-12-31", format="%Y-%m-%d")

> is.outperformance.plot(results, start.date=start.date, split.date=split.date,
+     end.date=end.date, FUN=Sharpe)
```



Function `oos.outperformance.plot`

Function `oos.outperformance.plot` plots the rolling outperformance of the out-of-sample trading strategy.

The usage of this function:

```
oos.outperformance.plot(results, start.date, split.date, end.date)
```

where

`results :` the results returned by function `forward.test`.

`start.date :` the start of the first rolling period

`split.date :` the end of the first rolling period

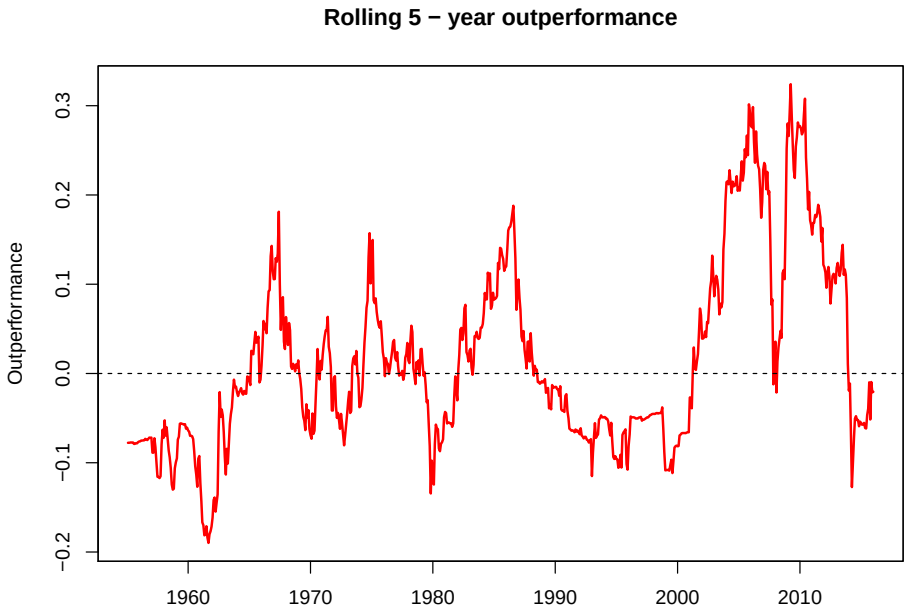
`end.date :` the end of the total period

The program below simulates the returns to a set of MAC and MOM rules, merges the results of simulations, simulates returns to the out-of-sample trading strategy and plots the rolling 5-year outperformance.

```
> rm(list=ls(all=TRUE))
> library(matiming)
> library(zoo)
> data <- sp500.monthly
> dates <- index(data)
> data <- coredata(data)
> capret <- data[,"CAP"]
> totret <- data[,"TOT"]
> rfret <- data[,"RF"]
> tc <- 0.0025
> fast <- 1:9
> slow <- 2:18
> res.mac <- sim.mac.strategy(totret=totret, rfret=rfret, dates=dates,
+                             capret=capret, tc=tc, fast=fast, slow=slow, FUN=SMA)
> res.mom <- sim.mom.strategy(totret=totret, rfret=rfret, dates=dates,
+                             capret=capret, tc=tc, winsize=slow)
> results <- sim.results.merge(res.mac, res.mom)
> start.date <- as.Date("1930-01-01", format="%Y-%m-%d")
> end.is.date <- as.Date("1949-12-31", format="%Y-%m-%d")
> end.oos.date <- as.Date("2015-12-31", format="%Y-%m-%d")
> res <- forward.test(results, start.date=start.date, end.is.date=end.is.date,
+                     end.oos.date=end.oos.date)
> # plot the rolling outperformance
```

```
> start.date <- as.Date("1950-01-01", format="%Y-%m-%d")
> split.date <- as.Date("1954-12-31", format="%Y-%m-%d")
> end.date <- as.Date("2015-12-31", format="%Y-%m-%d")

> oos.outperformance.plot(res, start.date=start.date, split.date=split.date,
+                         end.date=end.date)
```



References

Efron, B. (1979). "Bootstrap Methods: Another Look at the Jackknife", *Annals of Statistics*, *7*(1), 1–26.

Ehlers, J. F. and Way, R. (2010). "Zero Lag (Well, Almost)", *Technical Analysis of Stocks and Commodities*, *28*(12), 30–35.

Hull, A. (2005). "How to Reduce Lag in a Moving Average", `http://www.alanhull.com/hull-moving-average`. [Online; accessed 7-October-2016].

Künsch, H. R. (1989). "The Jacknife and the Bootstrap for General Stationary Observations", *Annals of Statistics*, *17*(3), 1217–1241.

Mulloy, P. G. (1994). "Smoothing Data With Faster Moving Averages", *Technical Analysis of Stocks and Commodities*, *12*(1), 11–19.

Patton, A., Politis, D. N., and White, H. (2009). "Correction to "Automatic Block-Length Selection for the Dependent Bootstrap" by D. Politis and H. White", *Econometric Reviews*, *28*(4), 372–375.

Politis, D. N. and Romano, J. P. (1994). "The Stationary Bootstrap", *Journal of the American Statistical Association*, *89*(428), 1303–1313.