

COMP226 Assignment 2: Strategy Development

The latest version of this document can be found here:

https://www2.csc.liv.ac.uk/~rahul/teaching/comp226/_downloads/a2.pdf

Continuous Assessment Number	2 (of 2)
Weighting	10%
Assignment Circulated	11:00 Monday 25 April 2016
Deadline	17:00 Tuesday 10 May 2016
Submission Mode	Electronic only
Learning Outcomes Assessed	<p>This assignment will address the following learning outcomes:</p> <ul style="list-style-type: none">• Understand the spectrum of computer-based trading applications and techniques, from profit-seeking trading strategies to execution algorithms.• Be able to design trading strategies and evaluate critically their historical performance and robustness.• Understand the common pitfalls in developing trading strategies with historical data.• Understand methods for measuring risk and diversification at the portfolio level.
Summary of Assessment	<p>The goal of this assignment is to implement and optimize a well-defined trading strategy within the backtester_v4.2 framework. The assignment will be assessed via the testing of 6 functions that you need to implement. The input and output behaviour of each function is fully specified and a code template is provided as a starting point.</p> <p>In addition to addressing the learning outcomes above, another purpose of this assignment is for students to familiarize themselves with the backtester framework that will also be used in COMP396.</p>
Marking Criteria	<p>Individual marks are attributed for each of 6 functions that should be implemented. If all 6 function implementations pass all the automated tests then a mark of 100% will be achieved. Partial credit for a function may be awarded if some but not all automated tests for that function are passed. The marks available for each function are given below.</p>
Submission necessary in order to satisfy module requirements	No
Late Submission Penalty	Standard UoL policy; note that no resubmissions after the deadline will be considered.
Expected time taken	Roughly 8 hours

Introduction: the backtester framework

You will write a strategy that should run in the backtester framework, which is available from

<http://www2.csc.liv.ac.uk/~rahul/teaching/comp226/bt.html#backtester>

The first thing you should do is download and unzip `backtester_v4.2.zip`, which will create a directory `backtester_v4.2` on your hard drive. Here is a listing of the zip file contents:

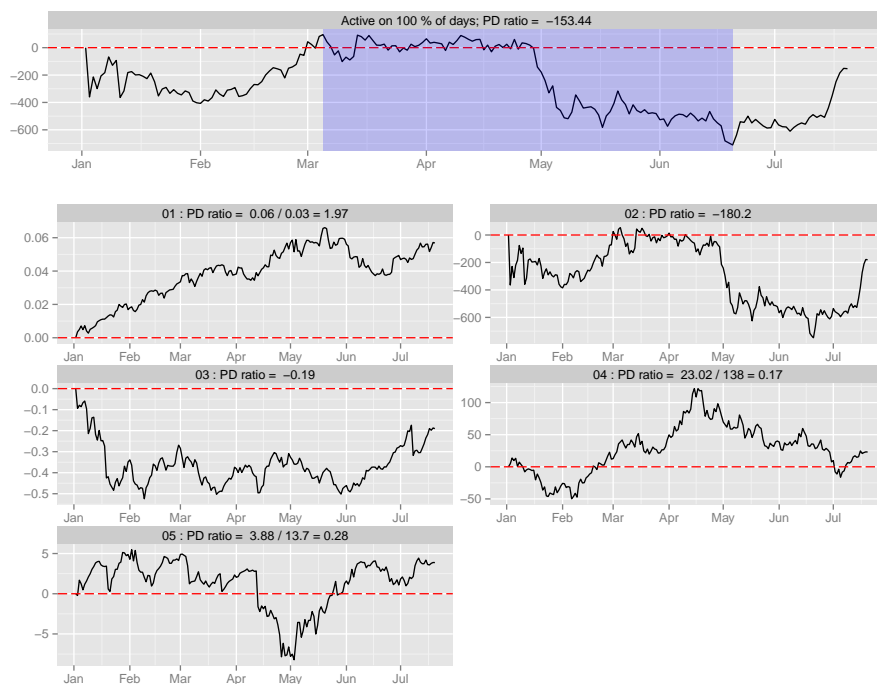
```
backtester_v4.2
├── DATA
│   ├── A2
│   │   ├── 01.csv
│   │   └── 02.csv
│   └── EXAMPLE
│       ├── 01.csv
│       ├── 02.csv
│       ├── 03.csv
│       ├── 04.csv
│       └── 05.csv
├── framework
│   ├── backtester.R
│   ├── data.R
│   ├── pdratio.R
│   ├── processResults.R
│   └── utilities.R
├── in-sample_period.R
├── main.R
├── main_optimize.R
├── main_template.R
└── strategies
    ├── a2_template.R
    ├── bbands.R
    ├── copycat.R
    └── fixed.R
```

5 directories, 20 files

Next you should open R and set the working directory to the `backtester_v4.2` directory on your hard drive. You can now try the example code as follows:

```
source('main.R')
```

If this doesn't work, first make sure you have set the working directory correctly, and then make sure you have installed all the required packages (see the error messages you get to figure out what the problem is). When it works it will produce a plot like the following:



There is one equity curve for each series in the data (5 of them in this case), and one aggregate equity curve. Let's go through `main.R` and see what the individual parts do.

First we source the framework itself.

```
source('framework/data.R')
source('framework/backtester.R')
source('framework/processResults.R')
```

Next, we load in the data using the function `getData` that is defined in `framework/data.R`. It returns a list of xts objects. These will be passed to the function `backtester`, though we may first change the start and end dates of the xts objects (which we will do in assignment 2).

```
# Read in data
dataList <- getData(directory="EXAMPLE")
```

There are 5 series in the directory `backtester_4.2/DATA/EXAMPLE/`, and therefore the list `dataList` has 5 elements too.

```
> length(dataList)
[1] 5
```

Each element is an xts:

```
> for (x in dataList) print(class(x))
[1] "xts" "zoo"
[1] "xts" "zoo"
[1] "xts" "zoo"
[1] "xts" "zoo"
[1] "xts" "zoo"
```

All the series have the same start and end dates:

```
> for (x in dataList) print(paste(start(x),end(x)))
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
[1] "1970-01-02 1973-01-05"
```

The individual series contain Open, High, Low, Close, and Volume columns:

```
> head(dataList[[1]])
      Open   High   Low  Close Volume
1970-01-02 0.7676 0.7698 0.7667 0.7691   3171
1970-01-03 0.7689 0.7737 0.7683 0.7729   6311
1970-01-04 0.7725 0.7748 0.7718 0.7732   4317
1970-01-05 0.7739 0.7756 0.7739 0.7751   3409
1970-01-06 0.7760 0.7770 0.7754 0.7757   2904
1970-01-07 0.7738 0.7744 0.7728 0.7743   3514
```

The next thing we do in main.R is source a strategy file.

```
# Choose strategy
strategyFile <- 'strategies/fixed.R'
# strategyFile <- 'strategies/copycat.R'
# strategyFile <- 'strategies/bbands.R'
# strategyFile <- 'strategies/a2_template.R'

cat("Sourcing",strategyFile,"\n")
source(strategyFile) # load in getOrders
```

Here is the contents of the strategy file backtester_v4.2/strategies/fixed.R:

```
# This strategy uses only market orders (and only in the first period).
# It takes a position in the first period and holds it for the duration.
# The backtester automatically exits all positions as market orders at the end.

getOrders <- function(store, newRowList, currentPos, params) {

  allzero      <- rep(0,length(newRowList))
  marketOrders <- allzero

  if (is.null(store)) {
    # take position during first iteration and hold
    marketOrders <- params$sizes
    store <- 1 # not null
  }
  return(list(store=store,marketOrders=marketOrders,
              limitOrders1=allzero,
              limitPrices1=allzero,
              limitOrders2=allzero,
              limitPrices2=allzero))
}
```

The backtester framework runs a strategy by calling `getOrders`. The arguments to `getOrders` are fixed, i.e., they are the same for all strategies. In the example strategy `fixed.R`, `getOrders` is the only function. The arguments to `getOrders` are as follows:

```
getOrders <- function(store,newRowList,currentPos,params) {
```

- `store`: contains all data you choose to save from one period to the next
- `newRowList`: new day's data (a list of single rows from the series)
- `currentPos`: the vector of current positions in each series
- `params`: a list of parameters that are sent to the function

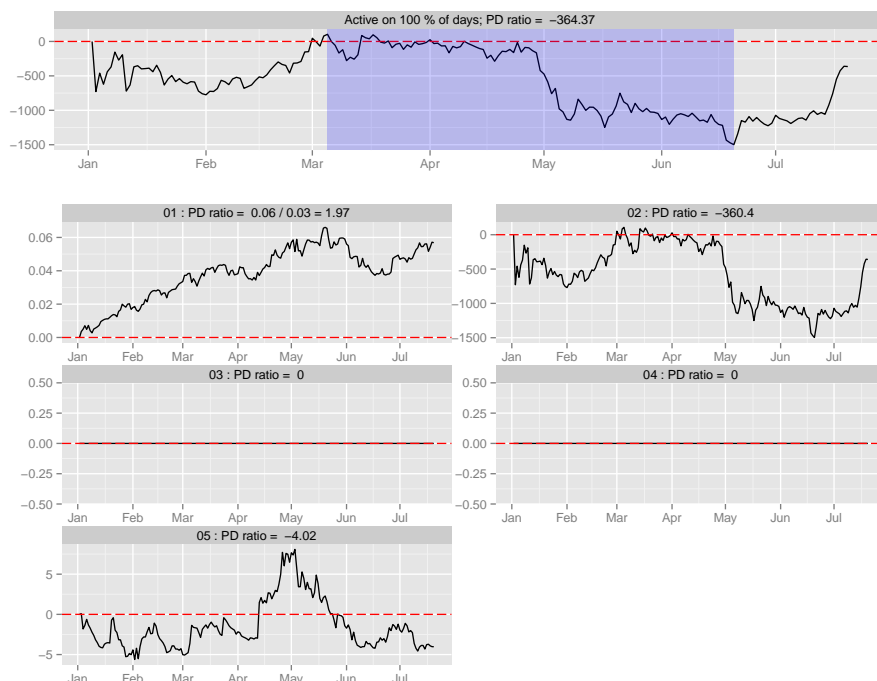
Here's how the strategy `fixed.R` works. In the very first period the backtester always (for every strategy) passes `store` to `getOrders` with `NULL` as its value. Thus in the first period, and the first period only, the vector `marketOrders` will be set to the parameter `params$size`, which should be a vector of positions with length equal to the number of series, which is 5 in this case. In `main.R` we see this parameter `params$series`, which is the only parameter for `fixed.R`, set as follows:

```
params <- list(size=rep(1,5))
```

With these sizes, we buy and hold one unit in every series. We can change the parameters and take positions in only some series and go short in some series, e.g., with:

```
params <- list(size=c(1,2,0,0,-1))
```

Compare with the equity curves above and note that for series 1 they are the same, for series 2 the new one is scaled by 2, for series 3 and 4 we no longer trade, and for series 5 we now take a short position, so the new series 5 is a reflection of the old series 5 equity curve.



The next thing in `main.R` is a subsetting of the time period for the backtest as follows:

```
inSampDays <- 200
# in-sample period
dataList <- lapply(dataList, function(x) x[1:inSampDays])
```

So we are only using the first 200 days.

Hint

You should adapt this use of `lapply` on `dataList` in order to define the in-sample period in assignment 2.

Finally we actually do the backtest and plot the results as follows:

```
# Do backtest
results <- backtest(dataList, getOrders, params, sMult=0.2)
pfolioPnL <- plotResults(dataList, results)
```

The arguments to the function `backtest` are the following:

- `dataList` - list of (daily) `xts` objects (with identical indexes)
- `getOrders` - the strategy
- `params` - the parameters for the strategy
- `sMult` - slippage multiplier (proportion of overnight gap)

Results for individual series are available in `results$pnLList`. The portfolio results are available in `pfolioPnL`, which is produced by `plotResults(dataList, results)`. This function also automatically plots individual and aggregate equity curves, and computes variant of the Calmar Ratio that we call the Profit Drawdown Ratio (PD ratio for short) - it is the final profit divided by the maximum drawdown in terms of profit and loss, or if the strategy makes a loss overall the PD ratio is just that loss (which is negative). You do not need to write code to compute this, since it has already been done for you. In assignment 2 we will optimize the aggregate PD ratio - this value is stored in `pfolioPnL$fitAgg`, e.g., for our first example we have:

```
> print(pfolioPnL$fitAgg)
[1] -153.44
```

This matches up with the PD ratio that appears at the top of the aggregate equity curve produced by `plotResults`.

Recall that market orders specify volume and direction (but not price), and limit orders specify price, volume, and direction. In the backtester framework, trading decisions are made after the close of day k , and trades are executed on day $k+1$. For each day, the framework supports one market order for each series, and two limit orders for each series. These orders are returned from `getOrders` as follows:

```
return(list(store=store, marketOrders=marketOrders,
            limitOrders1=limitOrders1,
            limitPrices1=limitPrices1,
            limitOrders2=limitOrders2,
            limitPrices2=limitPrices2))
```

Market orders will be executed at the open on day $k+1$. They incur **slippage** (20% of the overnight gap for assignment 2). Market orders are specified by

- size (number of units to trade)
- direction (buy/sell)

The sizes and directions of market orders are encoded in the vector `marketOrders` of the return list of `getOrders`. For example, the vector

```
c(0,-5,0,1,0)
```

means place a market order for 5 units short in series 2, and 1 unit long in series 4.

We will not use limit orders for assignment 2, so you can leave `limitOrders1`, `limitPrices1`, `limitOrders2`, `limitPrices2` as zero vectors when you do assignment 2 (i.e., you do not need to edit that part of the template). We will introduce the limit order functionality in detail in COMP396.

As well as `fixed.R`, two other example strategies are available now: `copycat.R`, `bbands.R`. It is left to you to explore how these work. These strategies implement the copycat strategy and mean-reversion Bollinger bands strategies from slides 06, respectively. Note that `bbands.R` and `a2_template.R` make use of the `store` to remember past data. You will need to adapt the `store` for COMP396, but for now, for COMP226 assignment 2 you just need to understand how close prices can be retrieved from the `store` as described below.

Before we move on to assignment 2, we will briefly look at an example of parameter optimization that will be useful for assignment 2. To make it easier to carry out parameter optimizations, `getOrders` takes an argument `params`. This can be used to pass a parameter combination to a strategy. This in turn can be used to do a parameter optimization as `main_optimize.R` demonstrates. Here is the source code for `main_optimize.R`:

```
source('framework/data.R');
source('framework/backtester.R')
source('framework/processResults.R');
source('strategies/bbands.R')

numOfDays <- 200
dataList <- getData(directory="EXAMPLE")
dataList <- lapply(dataList, function(x) x[1:numOfDays])
sMult <- 0.2 # slippage multiplier

lookbackSeq <- seq(from=20,to=40,by=10)
sdParamSeq <- seq(from=1.5,to=2,by=0.5)
paramsList <- list(lookbackSeq,sdParamSeq)
numberComb <- prod(sapply(paramsList,length))

resultsMatrix <- matrix(nrow=numberComb,ncol=3)
colnames(resultsMatrix) <- c("lookback","sdParam","PDratio")
pfolioPnLList <- vector(mode="list",length=numberComb)

count <- 1
for (lb in lookbackSeq) {
  for (sdp in sdParamSeq) {
    params <- list(lookback=lb,sdParam=sdp,series=1:4,posSizes=rep(1,10))
    results <- backtest(dataList, getOrders, params, sMult)
    pfolioPnL <- plotResults(dataList,results)
```

```

        resultsMatrix[count,] <- c(lb,sdp,pfolioPnL$fitAgg)
        pfolioPnLList[[count]]<- pfolioPnL
        cat("Just completed",count,"out of",numberComb,"\n")
        print(resultsMatrix[count,])
        count <- count + 1
    }
}
print(resultsMatrix[order(resultsMatrix[, "PDratio"]),])

```

The code template and data for assignment 2

You are now ready to start working on assignment 2. To do so you should read and work through the rest of this document very carefully.

As a first step, try to run `main_template.R`. It uses `strategies/a2_template.R` which is a code template that should serve as your starting point. If you try to source `main_template.R` you will get an error as follows:

```

Error in if (store$iter > params$lookbacks$long) { :
  argument is of length zero

```

If you read on you will see that the final strategy requires a parameter called `lookbacks`. Read on to see what form this parameter should take.

The code template contains templates for the 6 functions that you need to complete. These functions are:

1. `getTMA`
2. `getPosSignFromTMA`
3. `getPosSize`
4. `getOrders`
5. `getInSampleResult`
6. `getInSampleOptResult`

The rest of the document is split into two parts. The first part describes the function requirements and marking criteria for the first 4 functions, which relate to the strategy implementation. The second part describes the function requirements and marking criteria for the last 2 functions. Hints are given on how best to implement things, so read carefully. The examples below should give you an idea how you can test these functions to check whether you have implemented them correctly.

Note

You can develop the first three functions without running the backtester, which may be easier.

Part 1: strategy implementation

The overall goal of the assignment is the implementation and optimization of a triple moving average crossover (TMA) trading strategy. The specification of the strategy and the functions that it should comprise are given **in full detail**, so the correctness of your code can and will be checked automatically.

The TMA strategy you will implement is related to Example 1 in COMP226 slides 06. However, long and short positions are swapped as compared to that example (so you will here implement a mean-reversion as opposed to a trend following type strategy).

The strategy uses three moving averages with three different lookbacks (window lengths). The short lookback should be smaller than the medium window, which in turn should be smaller than the long lookback. In every trading period, the strategy will compute the value of these three moving averages. You will achieve this by completing the implementation of the function `getTMA`.

The following table indicates the position that the strategy will take depending on the relative values of the three moving averages (MAs). You will compute this position (sign, but not size) by completing the function `getPosSignFromTMA`. The system is out of the market (i.e., flat) when the relationship between the short moving average and the medium moving average does not match the relationship between the medium moving average and long moving average.

MA		MA		MA	Position
short	>	medium	>	long	short
short	<	medium	<	long	long

The function `getPosSignFromTMA` should use a function `getTMA`. The position size, i.e., the number of units to be long or short, will be determined by the function `getPosSize`. Finally, as usual in the backtester framework for COMP226 and COMP396, the position sizes are given to the backtester in the function `getOrders`. Here are the detailed specification and marks available for these first 4 functions.

Function name	Input parameters	Expected behaviour	Marks available for a correct implementation
getTMA	close_prices; lookbacks. The specific form that these arguments should take is specified in the template code via the 6 checks that you need to implement.	You should first implement the checks as described in the template. Hints of how to implement them are given below. The function should return a list with three named elements (named short, medium, and long). Each element should be equal to the value of a simple moving average with the respective window size as defined by lookbacks. The windows should all end in the same period, which should be the final row of close_prices.	18% (3% per check) 12%
getPosSignFromTMA	tma_list is a list with three named elements, short, medium, and long. These correspond to the simple moving averages as returned by getTMA. Note: You do not need to check the validity of the function argument in this case, or for the remaining functions either.	This function should return a single number that should be 0, 1, or -1. If the short value of tma_list is less than the medium value, and the medium value is less than the long value, then the return value should be 1 (indicating a long position). If the short value of tma_list is greater than the medium value, and the medium value is greater than the long value, then the return value should be -1 (indicating a short position). Otherwise, the return value should be 0 (indicating a flat position).	15%
getPosSize	current_close: this is the current close for one of the series; it does not have a default value. constant: this argument should have a default value of 1000.	The function should return (constant divided by current_close) rounded down to the nearest integer.	5%

getOrders	The arguments to this function are always the same for all strategies used in the backtester framework.	This function should implement the strategy outlined above and again below in "Strategy specification".	20%
-----------	---	---	-----

Strategy specification

The strategy should apply the following logic independently for both series.

The strategy does nothing until there have been `params$lookbacks$long`-many periods.

In the `params$lookbacks$long`-th period, and in every period after, the strategy computes three simple moving averages with window lengths equal to:

- `params$lookbacks$short`
- `params$lookbacks$medium`
- `params$lookbacks$long`

The corresponding windows always end in the current period. The strategy should in this period send **market orders** to assume a position (make sure you take into account positions from earlier) according to `getPosSignFromTMA` and `getPosSize`. (Limit orders are not required at all, and can be left as all zero.)

Hints

For the checks for `getTMA` you may find the following functions useful:

- The operator `!` means not, and can be used to negate a boolean.
- `sapply` allows one to apply a function element-wise to a vector or list (e.g., to a vector list `c("short", "medium", "long")`).
- `all` is a function that checks if all elements of a vector are true (for example, it can be used on the result of `sapply`).
- `%in%` can be used to check if a element exists inside a vector.

To compute the moving average in `getTMA` you can use the function `SMA` from the `TTR` package.

Note: The list returned by `getTMA` should work as input to the function `getPosSignFromTMA`.

For `getPosSize`, to round **down** to the nearest integer you can use the function `floor`.

As in the template, use the negative of `currentPos` summed with the new positions you want to take to make sure that you assume the correct position.

In order to help you check whether you have implemented the functions correctly, we next give some examples of how correct implementations of the functions will behave. These examples assume that you have correctly implemented the first 4 functions in `a2_template.R` and sourced the resulting code to make the functions available in the R environment.

Example output for `getTMA`

First you should make sure that you have correctly implemented all **6 checks** on the function arguments. Here are 3 examples of expected behaviour:

```
> close_prices <- c(1,2,3)
> lookbacks <- list(short=as.integer(5),medium=as.integer(10),long=as.integer(20))
> getTMA(close_prices,lookbacks) # bad close_prices
Error in getTMA(close_prices, lookbacks) :
  E04: close_prices is not an xts according to is.xts()

> dataList <- getData(directory="A2")
Read 2 series from DATA/A2
> close_prices <- dataList[[1]]$Close[1:19]
> getTMA(close_prices,lookbacks) # bad close_prices; too short
Error in getTMA(close_prices, lookbacks) :
  E05: close_prices does not enough rows

> lookbacks <- list(5,10,25)
> getTMA(close_prices,lookbacks) # bad lookbacks; list elements not named
Error in getTMA(close_prices, lookbacks) :
  E01: At least one of "short", "medium", "long" is missing from names(lookbacks)
```

Here is an example where we give the function valid arguments.

```
> lookbacks <- list(short=as.integer(5),medium=as.integer(10),long=as.integer(20))
> close_prices <- dataList[[1]]$Close[1:20]
> getTMA(close_prices,lookbacks)
$short
[1] 16.948

$medium
[1] 17.086

$long
[1] 17.1525
```

Example output for `getPosSignFromTMA`

Here are three examples of correct output:

```
> getPosSignFromTMA(list(short=10,medium=20,long=30))
[1] 1
> getPosSignFromTMA(list(short=10,medium=30,long=20))
[1] 0
> getPosSignFromTMA(list(short=30,medium=20,long=10))
[1] -1
```

Example output for getPosSize

Here are two examples of correct output:

```
> current_close <- 100.5
> getPosSize(current_close)
[1] 9
> getPosSize(current_close, constant=100.4)
[1] 0
```

Example output for getOrders

To check your implementation of `getOrders`, see part 2 for examples of correct output for the function `getInSampleResult` below.

Part 2: in-sample tests

There are two more functions that you need to implement: `getInSampleResult` and `getInSampleOptResult`. For both functions you will need to compute **your own** in-sample period, which is based on your computer science (CS) username. This ensures that for part 2 there are different answers for different students. To get your in-sample period you should use `in-sample_period.R` as follows. Source it and run the function `getInSamplePeriod` with your CS username as per the following example. Then use the first number in the returned vector as the start of the in-sample period and the second number as the end.

```
> source('in-sample_period.R')
> getInSamplePeriod('x4xz1')
[1] 230 644
```

So for this example username the start of the in-sample period is day 230 and the end is 644. **Note:** you may need to install the package `digest` to use this code.

Once you have your own in-sample period (and a correct implementation of `getOrders`), you are ready to complete the implementation of `getInSampleResult`.

Function name	Input parameters	Expected behaviour	Marks available for a correct implementation
<code>getInSampleResult</code>	None	This function should return the PD ratio that is achieved when the strategy is run with short lookback 10, medium lookback 20, and long lookback 30, on your username-specific in-sample period.	10%

To complete the final function `getInSampleOptResult` you need to do an in-sample parameter optimization using the following parameter combinations for the:

- short lookback
- medium lookback
- long lookback

You should **not** optimize the constant used with `getPosSize`, and leave it as 1000 as defined in the template code.

The parameter combinations are defined by two things: parameter ranges and a further restriction. Make sure you correctly use both to produce the correct set of parameter combinations. The ranges are:

Parameter	Minimum value	Increment	Maximum Value
short lookback	100	5	110
medium lookback	105	5	120
long lookback	110	5	130

The further restriction is the following:

Further restriction on parameter values

You should further restrict the parameter combinations as follows:

- The medium lookback should always be strictly greater than the short lookback.
- The long lookback should always be strictly greater than the medium lookback.

You need to find the best PD ratio that can be achieved one this set of parameter combinations for the in-sample period that corresponds to your username, and set it as the return value of `getInSampleOptResult`.

Hint

The correct resulting number of parameter combinations is 28.

You can adapt `backtester_v4.2/main_optimize.R`. It is probably easiest to use three nested for loops in order to ensure that you only check valid parameter combinations (where the short < medium < long for the respective window lengths).

Function name	Input parameters	Expected behaviour	Marks available for a correct implementation
<code>getInSampleOptResult</code>	None	This function should return the best PD ratio than can be achieved with the stated allowable parameter combinations on your username-specific in-sample period.	20%

Next we give some example output for these two functions.

Example output for `getInSampleResult`

To help you check the correctness of your code, here are three example return values for made up usernames:

Username	Correct return value
x1xxx	-1664.35

x1yyy	-541.94
x1zzx	-1776.39

Example output for `getInSampleOptResult`

To help you check the correctness of your code, here are three example return values for made up usernames:

Username	Correct return value
x1xxx	3.28
x1yyy	1.42
x1zzx	3.01

Marks summary

Function	Marks
<code>getTMA</code>	30
<code>getPosSignFromTMA</code>	15
<code>getPosSize</code>	5
<code>getOrders</code>	20
<code>getInSampleResult</code>	10
<code>getInSampleOptResult</code>	20

Submission

You need to submit **a single R file** that contains your implementation of 6 functions.

Submission is via the department electronic submission system:

<http://www.csc.liv.ac.uk/cgi-bin/submit.pl>

In what follows replace x1xx by your CS username (which you use to log on to the submission system).

Submit **one** ascii R file

- x1xx.R - an R file containing your code (implementations of all the functions)

Warning

Your code will be put through the department's automatic plagiarism and collusion detection system. Student's found to have plagiarized or colluded will likely receive a mark of zero. Do not discuss or show your work to other students.