# Project: Pairs Trading

Updated to clarify the input arguments and return values for the functions.

## 1 Introduction

This assignment will serve as HW 4 and your project. The assignment is divided into 12 parts, where for each part you are to write a function. The first 4 functions are to be completed for HW 4, and the remaining 8 are for the project. For HW 4, you need only turn in your R code, in a .R file. That is, there is no need to create an Rmd file. It's a good idea to test your functions as you develop them. We will provide a suite of tests for you to check your code against, but it is a good idea to design some tests of your own as the tests provided do not cover all cases.

## 2 The Problem

Finance and financial trading are important elements of all economies. Historically, people used various strategies and hunches to choose when to buy and sell stock and which stocks to trade. Increasingly, people try to use data, algorithms and, importantly, statistical methods to develop more automated strategies for determining what and when to trade in order to increase profits. Over the years, there have been many different approaches and algorithms in this direction, and recent developments illustrate computer trading is increasing rapidly. Some people are even developing statistical/machine learning algorithms that run on network routers to make sub-millisecond trades to exploit time delays other people experience.

In this assignment, you will explore one of the first trading strategies based on computer-intensive analysis of past stock performance. The approach is called pairs trading and was developed at Morgan Stanley in the 1980s. The strategy involves two different stocks, hence the name pairs. The key behind pairs trading is to have two stocks whose prices are positively correlated over time. It is perhaps simplest to understand the idea via an example. The Dow Jones and S&P 500 indices are two "stocks" we can buy and sell. These are not regular stocks such as those offered by companies such as Google, YAHOO, and ATT. Instead, these are actually collections of stocks, but that detail doesn't concern us as the same principle applies for any pair of stock that can be bought and sold. As you can see for the time interval 1990 - 1995 in Figure 1, the prices of these two stocks are highly positively correlated. Figure 2 shows the daily ratio of the prices of the two indices over the same time period. The ratio seems to be fluctuating around a stable value, at least for most of the time period.

The key idea underlying pairs trading is that the movement of the ratio away from its historical average represents an opportunity to make money. For example, if stock 1 is doing better than it typically does, relative to stock 2, then we should sell stock 1 and buy stock 2. This is called "opening a position." Then, when the ratio returns to its historical average, we should buy stock 1 and sell stock 2. This is called "closing the position." The reasoning is quite simple: when stock 1 is priced sufficiently higher than usual, it is likely to go down in value and the price of stock 2

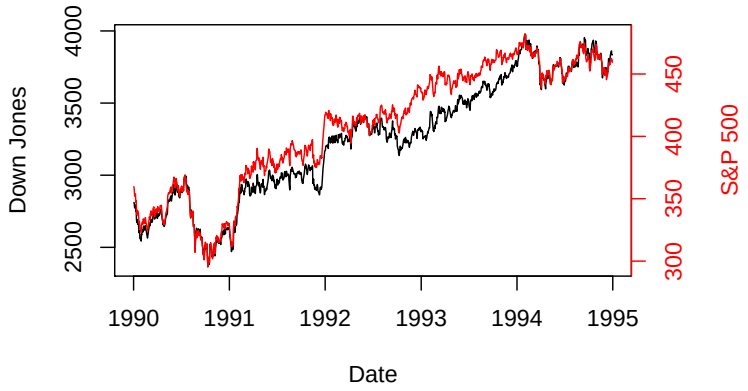


Figure 1: Historical Prices for the Dow Jones and S&P 500 Indices, 1990-1995. *This shows the time series of the two "stock" prices for the 5-year period. There is clearly a high correlation over time between the two financial indices. The values of the two series are quite different and plotted on different scales, with that for the S&P 500 on the right of the plot.*

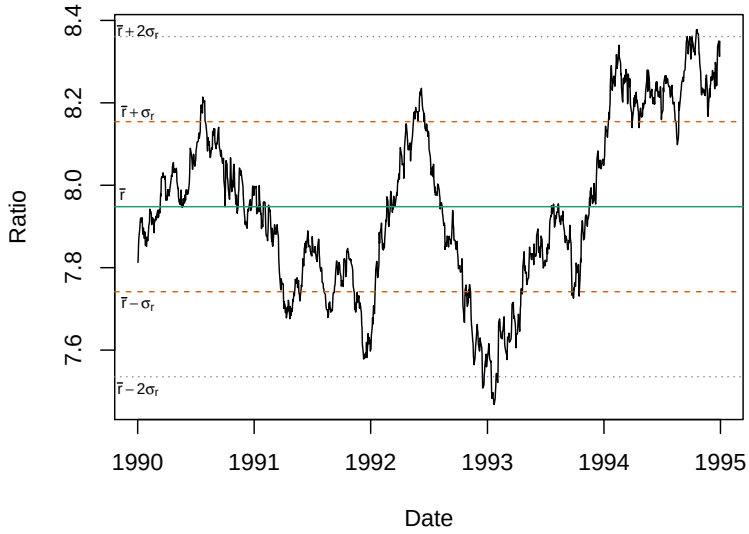


Figure 2: Historical Ratio for the Dow Jones and S&P 500 Indices. *This shows the* ratio *of the two "stock" prices from 1990 to 1995. The ratio appears to move around the mean until 1994 and then to rise above it. The horizontal lines show one and two standard deviations from the mean of the ratio.*

is likely to go up, at least relative to the price of stock 1, since they are positively correlated. Of course, both could increase, but we are interested in relative change as we are looking at the ratio.

To implement this pairs strategy, we need rules for when we should open and close positions. (We'll assume we can always sell a stock, even if we don't actually own it. The mechanism for this is called "short selling," but the details aren't important here.) We can devise rules based on the current value of the price ratio and what it has been in the past. We'll use historical data to determine the "optimal" rule and then use this rule as a trading strategy for the future.

We'll refer to the mean and standard deviation of the ratio for this "training"/historical data as $m$ and $s$, respectively. We'll consider the following class of rules:

- When the ratio of stock 1's price over stock 2's price moves above k standard deviations from the long term mean (i.e., $r > m + ks$), sell \$1 worth of stock 1 and buy \$1 worth of stock 2 (for simplicity, we are assuming we can buy fractions of a share). Then we wait until the ratio is less than or equal to $m$, at which point, we close the position by buying back however many shares of stock 1 we initially sold, and selling the shares of stock 2 we initially bought.

- Similarly, when the ratio is less than $m - ks$, do the same thing but reversing the roles of stock 1 and stock 2. In this case we wait until the ratio rises back up to be greater than or equal to $m$.

There are two things to note. First, $k$ represents how extreme the ratio needs to be before we open a position. Second, \$1 is just an arbitrary amount to invest. Buying a fixed dollar amount, rather than a fixed number of shares, allows us to work with stocks that have very different prices, and also to deal with fractions of stocks.

Now let's see what happens if we implement this strategy with the Dow Jones and S&P 500 prices, going forward from 1995 with $k = 2$. Figure 3 shows the ratios for 1995 to 2010, with a horizontal line displaying the threshold of two standard deviations above and below the mean displayed on the plot. It is important to recognize that we are using the mean and standard deviation from the previous period 1990 to 1995. We are basing our trading strategy on historical values and so use these to determine the thresholds for our trading rules. Also, DJI and GSPC are the shorthand "ticker" names for Dow Jones and S&P 500.

In this time series, we start with the value of the ratio already above our threshold, on the first trading day of 1995 (circle 1 on the plot). DJI is at \$3838.48 and GSPC is at \$459.11, so we sell \$1's worth of the DJI corresponding to $\$1/3838.48 = 0.00026$ units, and we buy $1/459.11 = 0.0022$ units of GSPC. Now we wait until the ratio reverts to the mean, which happens on June 19, 1998 (circle 2 on the plot). At this point both indices have gone up: DJI is at \$8712.87 and GSPC is at \$1100.65. We close the position, meaning we buy back 0.00026 units of DJI for \$2.27 and sell 0.0022 units of GSPC for \$2.40. Our total profit is therefore (\$1 – \$2.27) + (\$2.40 – \$1) = \$0.13. Note that we always subtract the buying price from the selling price, but we bought and sold in different orders for the two indices. We lost money on DJI but earned a bit more than that on GSPC. Pairs trading often works this way, with gains in one stock offsetting losses in the other. Also, note that to get the profit for a different investment amount, you just multiply 0.13 by that amount. Therefore, using \$1 as our baseline amount allows us calculate the *percent* profit.

We have additional opportunities to trade: opening at point 3 and closing at point 4, then opening at point 5 and closing at point 6. Doing this, we end up with a cumulative/total profit at point 6 of \$0.25. However, look at what happens beyond point 7. According to our rule, we open a position at 7, but the ratio does not revert back to the mean. We wait until the end of the entire time period and then close the position, even though the ratio hasn't reverted back to the mean

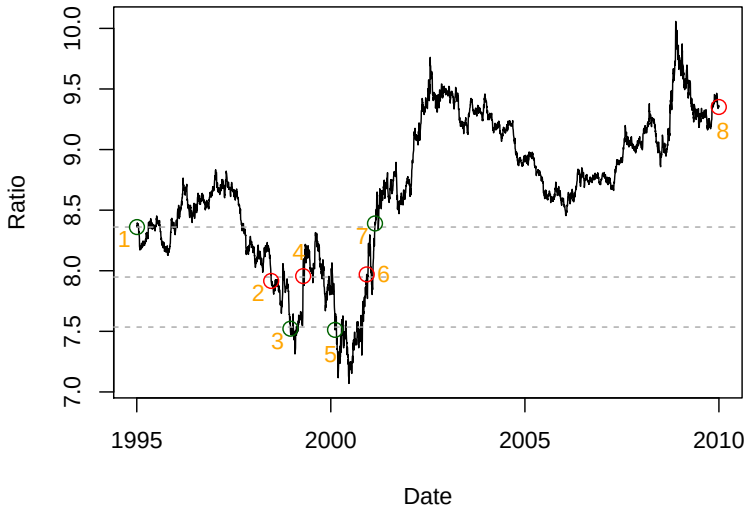


Figure 3: Ratio of Dow Jones to S&P 500 for 19952010. *The circles show the starting and closing positions for trades. We open a position when the ratio is outside of the threshold lines. We close that position when the ratio returns to the mean. The green circles identify the opening of a trading position; the red circles the corresponding close of the position. Circle 7 opens a position but the ratio never returns to the mean. We close the position on the final day of the series. Note that the threshold lines use the mean and standard deviation from the period 1990 to 1995.*

by this point in time. We end up losing about \$0.11 on that final trade, eradicating a significant portion of our profit.

In our example, we arbitrarily used the value $k = 2$. However, the "optimal" (i.e., to maximize earnings) value of $k$ depends on many things. We want to "estimate" or determine the optimal value based on the historical data for our pair of stocks. For this, we use the data from previous years to determine what is the best value of $k$ for the future. That is, we use the old data as "training" data.

One thing we haven't considered yet is that it costs money to make a trade. These costs can vary depending on whether you're an institutional trader or an individual. However, to keep things simple, let's assume that the costs are a fixed proportion $p$ of the total money changing hands (in absolute value). For example, to buy \$30 worth of stock and sell \$100 worth of stock, we would pay $\$130 \times p$. You will not address this in your project.

## 3 For You To Do

In this assignment, you will develop functions to explore pairs trading. We start by reading stock price data and exploring the time series of the prices and their ratios. We then develop functions to determine the start and end of a position and then all positions. The plots will help you validate the code and understand the pairs trading approach. You also write a function to compute the profit for a position and then another for the entire period with multiple positions. Then these functions are used to determine the optimal parameter value ($k$) for pairs trading for two stocks. We divide the data into a training period, and the remainder as a test period. We use the training data to determine the optimal value of $k$ and use that value of $k$ with the test data to assess the

pairs trading approach. Finally, we move from studying actual stock prices to simulating data from a mathematical model in an effort to understand the impact of within- and between-correlation for the two stock price time series on profit when using pairs trading.

# 4 The Data

Pairs trading works best with two stocks that are reasonably correlated. The common example is companies that are in the same business sector such as ATT and Verizon, or Google and YAHOO. When demand for more telecommunications services increases, both companies do well. However, they are also competitors. Companies whose products/services complement each other are also correlated, e.g., hard-drive manufacturers and computer/device vendors. As demand for devices grows, so too does the demand for storage.

We provide historical price data for several companies at `http://www.stat.berkeley.edu/users/nolan/data/stocks`. These are data for IBM (ibm.csv), Intel (intc.csv), Ford (f.csv) and GM (gm.csv). They were downloaded from `http://finance.yahoo.com/q/hp`.

For each record in one of these CSV files, we have the date and details about the price and the number of shares traded that day. There are various different prices reported (opening and closing, minimum and maximum during the day). We are interested in the adjusted closing price (Adj Close). This takes into account other aspects related to the stock that occur before the start of the next day's trading. These records also include events such as stock splits and payment of dividends.

**Function 1: readData()** The first thing to do is write a function to read the data for a stock from a CSV file. Call this function readData(). It takes two arguments: the required *`fileName`*, which is the name of the CSV file; and the optional *`dateFormat`*, which contains a string specifying the format of the `Date` values, e.g., `month/day/year`. Provide a reasonable default value for *`dataFormat`*. When you read the CSV file into R, make sure that the `Date` values remain as strings so that they can be easily converted to `Date` values. The return value from the function is a data frame, with variables `Date` and `Adj.Close`, where the date is formatted as an R class `Date`, Also arrange the observations in increasing order of the date.

**Function 2: combine2stocks()** When you analyze your data, you have to restrict your attention to the common period of time for which we have prices for two stocks. Write a function that computes the days in common between the two data sets of stock prices. This function returns a data frame with rows for each day with the date, adjusted closing prices for the two stocks and the ratio of the two stock prices, i.e.,

$$\text{adjusted price for stock}A/\text{adjusted price for stock}B$$

These variables are to be called `Date`, `Adj.Close.A` , `Adj.Close.B`, and `ratio`. The inputs to this function are the two data frames from the calls to readData(). Call these arguments, *`stockA`* and *`stockB`*. They are required arguments.

# 5 Visualizing the Time Series

**Function 3: plotRatio()** You now have the inputs for the pairs trading scheme, namely the prices and ratio. It will be helpful in debugging your code to create a visualization to see where the ratio

goes outside of some range or limit that would make an opening a position. A visualization will also help you understand the pairs trading strategy. This visualization doesn't define or implement the calculations for our rule. It merely displays the cutoff points. The function plotRatio() makes a line plot of the time series for the ratio and also draws the upper and lower horizontal lines for the pair trading rule, for a given $k$. The parameters for this function are: `ratio`, which is required; `k`, which has a default value of 1; `date`, which has a default value of `seq(along = ratio)` but can also handle a `Date` object; and the special parameter `...`, which is used to pass additional arguments to plot() such as a title, axis labels, etc. The plot should look like the one in Figure 4. Be sure to write this function in such a way that to allow the caller to pass a vector for `k`, not just a single value. In this case, the function draws the bounds/lines for multiple trading rules.

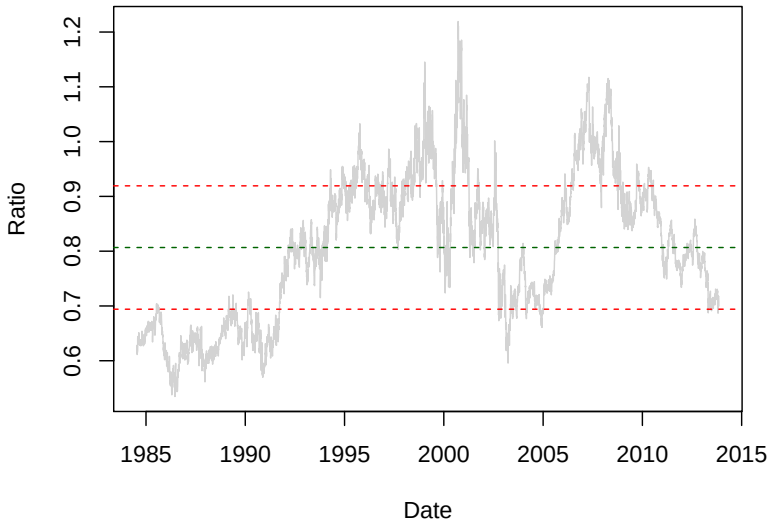


Figure 4: A Simple Plot of the Ratio of Stock Prices for ATT and Verizon. *The mean of the ratio ($\mu_{ratio}$) is shown as a dashed line and thresholds lines indicating $\mu_{ratio} \pm k\sigma_{ratio}$, with $k = 0.85$.*

**Function 4: showPosition()** This function augments the plot created by plotRatio() to display the start and end positions of the ratio time series. We haven't figured out how to compute these positions yet, but we can create this function and test it with example positions. This function has four arguments: `pos`, `ratios`, `col`, and `radius`. The `pos` argument is required and will be a two-element numeric vector of the open and close positions. Likewise, the `ratios` argument is required and will be a two-element numeric vector of the ratio values for the open and close positions. The `col` parameter takes an optional vector of two colors, one for the open and the other for the close; and `radius` is an optional argument to provide the size of the circle to place on the plot (use 100 for the default). Note that if we call plotRatio() with the `Date` values for the horizontal axis, we must pass the actual `Date` values for the open and close dates for our trading position to showPosition(). Alternatively, if our call to plotRatio() uses the indices of the ratio vector (1, 2, ...), then we call showPosition() with the indices for the start and end of our position. The additions should appear as in Figure5.

**The Remaining Functions Are To Be Completed For The Project**

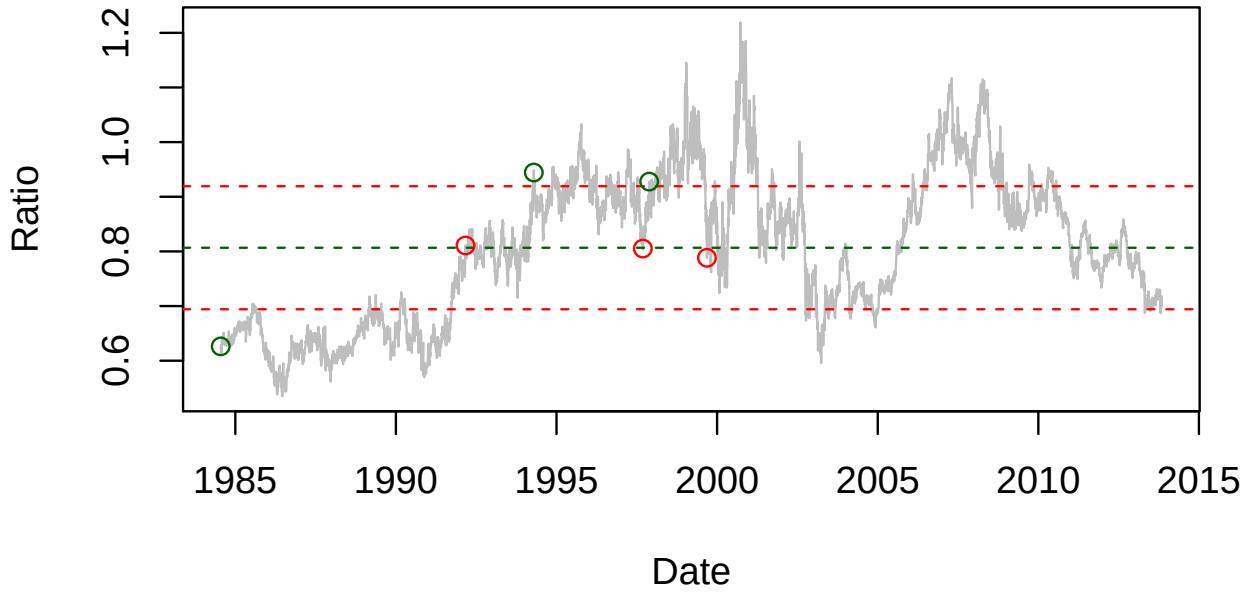


Figure 5: Visualizing the First Three Positions. *This shows the first 3 positions for the ATT/VERIZON stock price ratio. It allows us to verify that our findNextPosition() function is working correctly.*

## 6 Finding Opening and Closing Positions

We are now ready to write code to calculate opening and closing positions. We need to compute all the opening and closing positions for the entire period of interest, but we start by writing a function to identify just one pair of opening and closing positions. This is not the first position, but the "next" position given a starting point, i.e., a day to start from. This will allow us to use the function to find all of the positions.

**Function 5: findNextPosition()** The function takes the entire ratio vector (returned from the call to combine2Stocks()) and an optional starting day that is an index into the vector at which we should start the search. That is, to find the first position, we specify this position/index as 1. This should be the default value. These parameters are to be called, `ratio` and `startDay`, respectively. In addition, include the optional arguments `k` and the mean and standard deviation (m and s) of the ratio. For the mean and SD, the default values are to be computed from `ratio`. That is, we use `m = mean(ratio)` in our function definition. The function returns a vector of length 2, with the first element the open and the second the close. If no open position is found, then the function returns an empty vector, i.e., integer(). After we find the first pair, we call the function again to find the second pair, and we specify `startDay` as the close of the first pair for the starting point from which to search for the next position. We also supply the complete vector or ratios to `ratio`. In this way, we will move across the time series in blocks, starting from where we ended the previous position. There is the possibility that there is no close to the position we opened within the period at which we are looking, as we saw in Figure 3. In this case, we use the index of the last element in the ratio.

**Function 6: getPositions()** This function computes all the positions for a ratio time series. It calls findNextPosition() again and again until the end of the time series is reached. You can use a `while` loop to iterate over the blocks of the time series until you have processed all days and found all positions. We know there are no positions remaining when findNextPosition() returns an empty vector or when the last day of the time series is the close of a position. Collect the individual position vectors into a list and have the getPositions() return it. The arguments and default values to this function are the same as findNextPosition(), except that *`startDay`* is not needed.

**Function 7: positionProfit()** In addition to the start and end day of each position, we also need to compute the profit for that position. Our description of how to do this for the first position for the DJI and GSPC stock in Section 2 is quite explicit, and you are to map that into general *R* code. This function takes the position, which is the vector of the start and end day indices of that position (this input argument is called *`pos`*, which should be a numeric or integer vector of length two). We can use these indices to get the stock prices for the two stocks. We also need to pass the vectors of stock prices to the function as well. These are the parameters `stockPriceA` and `stockPriceB`. We also need the mean of the ratio so that we can determine which stock we sell and which we buy. The function returns the profit for the one position.

**Function 8: getProfit.K()** We defined the function positionProfit() to compute the profit for one position, given the start and end of the position and the vectors of stock prices. The getProfit.K() function should compute all of the trading positions (using getPositions()) and then compute the profit for each position (using positionProfit()). For your function, use the following function definition:

```
getProfit.K = function(x, y, k, m = mean(x/y), s = sd(x/y))
```

Here *`x`* and *`y`* are the stock prices for the two stocks; *`k`* is the number of SDs away from the mean that we use to determine the open position; and *`m`* and *`s`* are the mean and SD to use in determining the positions. This function returns the total profit for all positions.

## 7 Finding the Optimal Value for k

We are now ready to use these functions to find the best value of $k$ from training data and then apply it to test data. The basic machinery is in place with the functions you have written. The first step is to create the training and test data sets. We can look at a particular period for the training data, or we can just split the data in two. We opt for the first approach, and create a 5-year period for the training data. The remainder of the data serves as the test data. We can find the break point for the training and test data with

```
train.period = seq(min(overlap$Date), by = "5 years", length =2)
```

Here `overlap$Date` is the return value from a call to combine2stocks(). In `train.period` we have a vector of length 2 of class `Date` that contains the start and end dates of the training data. We can use this to subset the stock price vectors and the ratio time series vector for this period. Combine the 4 vectors (the date, two stock prices and the ratio) into a data frame for the training. Similarly, we can use `train.period` to subset the stock prices, the ratio, and date for the test period. Then, combine these 4 vectors into a data frame for the test data.

**Function 9: getBest.K()** The getBest.K() function needs to find the trading positions and compute the overall profit for different possible values of $k$ in order to determine the value of $k$ yielding the highest profit for the **training data**. As with getProfit.K(), we have all the building blocks and merely need to connect them. The inputs for this function are: `x` and `y`, the stock prices for the two stocks; `k.min` and `k.max` the smallest and largest values for $k$ and `numK`, the number of $k$ to examine between the min and max; and `m` and `s` are the mean and SD to use in determining the positions. The return value is the $k$ that achieved the maximum profit. If more than one $k$ achieved the maximum, return the median of those $k$s.

Once you have the "optimal" value of $k$ for your training data (`k.star`), you can see how well this $k$ does with the test data. Use getProfit.K() to find the positions for your test data (with the special value of $k$ determined from the training data) and compute the total profit at these positions.

# 8 Simulation Study

For different time periods, the results from pairs trading may be quite different. Similarly, for different pair of stocks, the profits may also be significantly different, either higher or perhaps a lot lower or even negative, meaning a loss. A reasonable question an investor should ask when considering whether to use a pairs trading approach is what are the characteristics of the two stocks that yield a high profit margin?

How do we set about addressing such a general question? It refers to all classes of stock time series. We need to make this more concrete and restrict ourselves to one or two specific classes of stock price series. How do we concretize these? It is hopefully natural to consider a mathematical model for a pair of time series. We introduce a simple *additive* random time series model:

$$\begin{aligned} Y_t^{(1)} &= \beta_0^{(1)} + \beta_1^{(1)} t + X_t^{(1)} \\ Y_t^{(2)} &= \beta_0^{(2)} + \beta_1^{(2)} t + X_t^{(2)}, \end{aligned} \tag{1}$$

where

$$\begin{aligned} X_t^{(1)} &= \rho X_{t-1}^{(1)} + \psi(1-\rho) X_{t-1}^{(2)} + \epsilon_t^{(1)} \\ X_t^{(2)} &= \rho X_{t-1}^{(2)} + \psi(1-\rho) X_{t-1}^{(1)} + \epsilon_t^{(2)} \end{aligned}$$

Here, the subscript $t$ corresponds to time and the superscript $(i)$, $i = 1, 2$, corresponds to the stock. Notice that the two stocks are correlated because $X_t^{(1)}$ depends on the value for the previous day for both stocks, i.e., $X_{t-1}^{(1)}$ and $X_{t-1}^{(2)}$. Here the $\epsilon_t^{(i)}$ are random quantities. Their distributions need to be specified, if we are to analyze how pairs trading behaves under stock prices generated from this model. For simplicity, we can use a Normal distribution for each of the error terms, say, $\epsilon_t^{(i)} \sim N(0, \sigma_i^2)$. Each stock can have a different standard deviation.

Our model has 8 parameters. These include the parameters to generate the $X$s: $\rho$, $\psi$, $\sigma_1$ and $\sigma_2$, and the parameters to generate the $Y$s from the $X$s: $\beta_0^{(i)}$ and $\beta_1^{(i)}$, for $i = 1, 2$.

We'll denote this 8-dimensional vector as $\Theta$. The linear term is a simple function of the day number, i.e., $t$. To this, we add the correlated components $X_t^{(i)}$ for each stock. This makes the $Y_t^{(1)}$ and $Y_t^{(2)}$ values correlated with each other and also with the values from the previous days $t-1$, $t-2$, ....

We now have a concrete mathematical model for two time series. This defines a reasonable class of stock prices and we can use it to address the investor's question about how pairs trading

would work for different time series. Ideally, we would be able to compute the optimal value of $k$ and develop a closed-form solution for the profit as a function of the 8 parameters. Since there is randomness in the model, we will have to use our mathematical statistics skills to compute the distribution of the profit as a function of $k$ and the 8 parameters. We may be able to compute the expected value (mean) and standard deviation of the distribution, but it may require approximations for the distribution. We may not be able to obtain a closed-form solution for these results. Even ostensibly simple mathematical models may prove to be inaccessible for mathematical analysis. In these cases, we can use simulation to explore the models and get answers to our questions.

Rather than working with the equations mathematically, you will simulate actual time series from the two equations. We want to understand how our pairs trading "algorithm" or rule performs as we vary the 8 different parameters. This will then help us understand the characteristics of the algorithm and be able to address the investor's question, i.e., how does profit vary with the values of $\rho$, $\psi$, $\sigma_1$, and $\sigma_2$, and the $\beta$ vector? This will hopefully help us identify good pairs of stock to use together.

How will we use simulation? For a particular vector of the 8 parameters, say $\Theta$, generate $n$ observations for the two series $Y_t^{(i)}$ using the equations in 1 above. Then divide these into a training set and test set. As we did for the actual pair of stocks, use the training set to find the best value of $k$ for pairs trading. Then, use that value with the test data to evaluate how well we would actually do. The value for profit we obtain is for a particular realization of the two time series. We need many independent replications to estimate the distribution of profit for this particular value of $\Theta$. We may be interested in the average profit, some of the quantiles, the standard deviation, or some other statistics. We repeat this for many different values of $\Theta$ to understand the distribution of profit across the parameter space for $\Theta$. This will allow us to understand how these factors change our expected profit and what characteristics of the stock work well for pairs trading, at least for our mathematical model.

**Function 10: stockSim()** This function generates the data for two stock prices for a given value of $\Theta$, i.e., the 8 parameters. This corresponds to implementing the two equations in (1). Since the t$^{th}$ value of each series depends on the (t-1)$^{th}$ value of the time series, a loop is the most obvious way of creating the series. However, be sure to compute as many of the input values to the series using vector calculations as you can. You may also want to consider using matrix multiplication.

We provide the function definition for stockSim():

```
stockSim = function(n = 4000, rho = 0.99, psi = 0, sigma = rep(1, 2),
                    beta0 = rep(100, 2), beta1 = rep(0, 2),
                    epsilon = matrix(rnorm(2*n, sd = sigma),
                                     nrow = n, byrow = TRUE))
```

We have specified default values for the 8 different parameters in $\Theta$. Interestingly, we have also allowed the caller to provide values for the `epsilon` variable. This allows us to specify the random values explicitly, which is useful for testing and reproducing results. The return value is a matrix with two columns. The first column contains the stock prices for stock A and the second column has the prices for stock B.

How do we verify that our function stockSim() is correct? It involves random values, so it is challenging. We can at least control the seed for the random number generator (see set.seed() and `.Random.seed`) to reproduce the exact same random values in the same calls to the random number generation functions. This may not allow us to guarantee the same sequence of random numbers. However, we allowed the caller of stockSim() to specify the matrix of random values via the *epsilon*

parameter. This allows us to provide the same random values across different calls and so remove the randomness.

Now use your simulation function stockSim() to generate time series data and explore the effect of varying $\rho$, $\psi$, $\beta_0^{(i)}$, and $\beta_1^{(i)}$ on the profit rate for pairs trading. For a particular set of values for these 8 parameters, we want many independent and identically distributed values of the profit to provide an estimate for the distribution of the profit. We split this task into two functions.

**Function 11: runSim()** The first function, runSim() carries out one replication for one set of 8 parameters as follows.

- Generate two time series using the 8 parameters.
- Split the time series into training and test datasets.
- Use the training data to determine the optimal value of k for the pairs trading strategy.
- Using that value of k, determine the profit for the test data.

Below is the function definition for runSim():

```
runSim =  function(rho, psi, beta0 = c(100, 100), beta1 = c(0, 0),
                   sigma = c(1, 1), n = 4000)
```

Note we call runSim() with values for each of the 8 parameters in very much the same way as stockSim(). This function returns the profit for the simulated pair of time series.

**Function 12: simProfitDist()** We intend this function to run multiple identically distributed simulations for a given vector of the 8 model parameters and return a collection of profit values. Given runSim(), this is quite easy to implement. We provide it here (i.e., there is no need to write any code - you just need to figure out how to call it):

```
simProfitDist =
function(..., B = 999)
      sapply(1:B,  function(i, ...) runSim(...), ...)
```

The argument `B` is the number of replicates in our simulation for the same parameter settings. This controls the variability of our estimate of the average. The ... allows us to pass any arguments to runSim(). We use it to avoid having to explicitly copy the parameters and their default values from runSim(). Similarly, if we ever add a parameter to runSim(), we will not also have to add it here.

# 9 Project Report

Explore the pairs trading strategy for various pairs of stocks (at least 6 pairs). (More stocks will be posted soon.) You may also download your own historical stock price information from Yahoo. Write a summary of your findings. Consider whether there are there characteristics of stocks that seem to lead to larger gains. Does your analysis provide insight into when pairs trading might work or fail?

Carry out a simulation study of the model presented in Section 8. Be careful with the number of parameter values that you consider. For example, if we have 20 different values for each of `psi`,

`beta1`, and `beta2`, then we have $20 \times 20 \times 20$ different combinations of the three. That amounts to 8000 different settings. If it takes 17 seconds to run each simulation, then all together that takes about 2267 minutes, or about $1\frac{1}{2}$ days, to run. This does not include varying `rho`. If we had 20 separate values for it, our run time would be over 30 days.

Write your findings up in an Rhtml file. Do NOT include your code to carry out your simulation study in the Rhtml file. Instead:

- Save the simulation results and the results from reading your stock into R in an .rda file and load them into your .Rhtml report in a code chunk, e.g., `load("simFindings.rda")`.
- Use the objects in the .rda file to make plots and compute summary statistics. Remember that you can summarize the results of a simulation study with a plot. It doesn't have to be presented as a table of numbers. Note that you may want to use the plot that you designed in HW 4.

Upload to bcourses:

- The .R file that contains your 12 functions.
- The .R file that contains the code to run your simulation and save the results and the code to read the actual stock prices that you analyze in your report.
- The .rda file that contains the findings from your stock comparison and simulation results.
- The .Rhtml file that contains the report. Use the report template posted to bcourses.
- The PDF printout of the knitted .html file (open the knitted html file in a browser and print to PDF).

Keep the report short. Two pages max. You can hide the code for you plots with the "echo=FALSE" parameter in the code chunk.