# COMP1100-Assignment2 Report

Stock Market

Name: Yutong WANG

ID: u6293753

# Contents

# 1 Introduction

The aim of the assignment this time is to implement the ?makeOrders? function using our own strategy to buy or sell stocks thus increase our total without bankrupt.

# 2 Descriptions of Functions

## 2.1 Types

To fully understand these function in the file, we need to first get familiar with the basic types. The types are defined in the file like below:

```
type Portfolio = (Cash, Holdings)
type Holding  = (Stock, Quantity)
type Holdings = [Holding]
type Cash = Double
type Price = Double
type Quantity = Integer
type Stock = String
type StockHistory = (Stock, [Price])
```

to make it easier to understand, we can write them In the format below:

```
Portfolio = (Double, [String, Integer],[String, Integer]?)
StockHistory = (String, [Double,Double,?])
```

## 2.2 calculateWealth

```
calculateWealth :: Portfolio -> [StockHistory] -> Cash
calculateWealth (cash, stocks) histories
        = cash + sum (map currentValue stocks)
    where
        currentValue :: (Stock, Quantity) -> Cash
        currentValue (ticker, quantity)
            = fromIntegral quantity
              * getStockPrice ticker histories
```

the total wealth is the combination of cash and the value of all stocks in your holdings:

```
cash + sum (map currentValue stocks)
```

As we all know, cash is Double, then the main problem here is to calculate the value of our holdings.

To calculate that, we need to know the current price of corresponding stock and the quantity of that stock that we hold, then multiply the current price with quantity of each stock, that's why we use map here to calculate the value of each stocks. In addition, since the type of quantity is Integer, and we need to use it with the '*' operation. Thus we need to use 'fromIntegral' to change the type from Integral to Num. Finally, add the value of each price together:

```
cash + sum (map currentValue stocks)

currentValue (ticker, quantity)
            = fromIntegral quantity
              * getStockPrice ticker histories
```

### 2.3 getStockPrice

```
getStockPrice :: Stock -> [StockHistory] -> Price
getStockPrice s histories = head $ snd getStock
    where
        getStock = head $ filter (\x -> fst x == s) histories
```

The next problem is how to get the current price. We need to use the getStockPrice function, this function takes a specific stock(s) and a list of stock history([StockHistory] = [(Stock, [Price])]), then check whether the first place in the tuple which is the Stock is equal to 's'. That why we use filter there. However, the result of filter is a list of a tuple we want. Thus, we use head here to get the tuple out of the list.?

### 2.4 executeOrders

The input of this function is Portfolio, [StockHistory] and [Order]:

```
executeOrders p@(cash, holdings) histories orders
```

@ here is syntactic sugar, we can read @ as ?as?. So @ here actually representing a whole things after it: (cash, holdings). Then the function discuss the possibilities of each orders: [], which means no orders There are two possibilities here, if the cash is negative which means we are borrowing money from the bank, thus, the current cash should be: original cash - 5.5% which is the loan interest. if we are not borrowing money from the bank, then we still get an interest of 3% on our cash savings. B) regular order First, the function defines some possible situations when we need to skip the order. a)s == ?XJO": when stock s is certain stock ?XJO?

b)invalidShort: when the capital we got through short selling is greater than our current wealth.

```
cost - commission ssCommission cost
cost = fromIntegral q * price
```

Cost is the capital we get from short selling and the cost minus commission fees will be actual capital we get from short selling.

c)invalidLong: when the money we use to buy a stock is larger than our current wealth.

```
cost + commission regCommission cost
cost = fromIntegral q * price
```

the cost here is the money we will spend on buying certain quantity of a stock, this cost minus the commission fee will be the total cost we need to spend to execute the order.

d)buyingMarket: when the number of shares we try to buy is larger the number of shares available to buy.

e)cash ¡ (-500000) when cash is below -500000. Then the function checks if it is regular short selling, which means the order is to short sell and the quantity of the stock we held is 0 or negative.

As opposed to regular short selling, when the order is to short sell certain stock, however, we still have some shares of that stock which means quantityHeld ¿ 0 and it is short selling, so the number of share we need to sell is larger than the number of share we already held.

The difference between ?isShortingHeld? and ?isShortingReg? is that ?isShortingHeld? needs to do the regular sell first to sell all of its quantityHeld then do the regular short selling process.

Eliminating some special situation, then we only need to deal with regular selling and buying. In addition, it is important to remember that when selling, the quantity is negative in order.

### 2.5 simulate

We can seen from the function 'convertToStockHistory' in Main.hs that 'StockHistory' is a list of prices from the latest to the oldest.

Next, let's start by the helper functions: unfoldPrices takes a list of prices and transform it into certain format. For example if the list of prices is [1,2,3]:

```
>>> unfoldPrices [1,2,3]
[[3],[2,3],[1,2,3]]
```

the process is as follow:

```
\3 [] -> [3]:[] = [[3]]
\2 [[3]] -> (2 : head[[3]]):[[3]] = [[2,3],[3]]
\1 [[2,3],[3]] -> (1 : head[2,3],[3]):[[2,3],[3]]
                                        = [[1,2,3],[2,3],[3]]
```

then apply reverse: the final answer is [[3],[2,3],[1,2,3]]

The 'distribute' omit the input here:

```
distribute st = map (\pr -> (st, pr))
```

the complete format is:(p is type of [[Price]])

```
distribute st p = map (\pr -> (st, pr)) p
```

So, how do this function works, let me give you an example:

```
>>> distribute "a" [[1],[20],[100]]
[("a",[1.0]),("a",[20.0]),("a",100.0)]
```

Next is ?convert?, this function change a list of Stock history into seperate tuples with only one price in the second place using ?distribute?. For example:

```
>>> convert [("a",[[1],[2],[3]]),("b",[[4],[5],[6]])]
[[("a",[1.0]),("a",[2.0]),("a",[3.0])],[("b",[4.0]),("b",[5.0]),(
"b",[6.0])]]
```

?unfoldHistoriesAux? takes a list of StockHistory into certain format. For example:

```
>>> unfoldHistoriesAux [("a",[1,2,3]),(?b?,[4,5,6])]
[("a",[[3.0],[2.0,3.0],[1.0,2.0,3.0]]),("b",[[6.0],[5.0,6.0],[4.0,
5.0,6.0]])]
```

?unfoldHistories? simulates everyday prices from the very first day, For example:

```
>>> unfoldHistories [("a",[1,2,3]),]
[[("a",[3.0]),("b",[6.0])],[("a",[2.0,3.0]),("b",[5.0,6.0])],[("a"
,[1.0,2.0,3.0]),("b",[4.0,5.0,6.0])]]
```

Finally, the ?simulate? function. This function takes the current portfolio and the current StockHistory to feed price data from the left hand side of the list, which is the first day of the simulation, thus simulating the every day stock market.

# 3 Design of 'makeOrders'

## 3.1 Draft idea

The aim of implementing makeOrders is to earn money, and if we want to make profits in the stock market, we need to use appropriate method to find out the situation of each stock, at least we need to find out whether the trend of this stock is increasing or decreasing. If it is increasing, then we can buy, and if the quantityHeld ¿ 0, then we can sell it at appropriate time; if the line is decreasing, then if the quantityHeld ¿ 0, and we certainly want to avoid losing more money, thus we need to sell it at appropriate time, however if quantityHeld = 0, and the price is continue decreasing is quite a long time so that we assume that the price is highly likely to decreasing in the future, then we can implement short selling, which means we can borrow shares from the company and sell them to someone else at that day and buy shares to give back to that company. While we can not predict the future price accurately, at least we can make sure the profit we make is bigger than the deficit.

## 3.2 Linear regression

The problem is how to describe the situation of the prices of a stock without actually seeing it. That is why I use linear regression, since we can use it to simulate the general trend of the stock prices. The first step of calculating a regression line is to build the points we need to use first, since we have only the list of prices here. In order to use points to calculate the regression line, we need to build a list of points, with x equals to days, y equals to the corresponding price:

```
(days, currentPrice)
```

As we all know, we can use filter to find out the list of prices of the corresponding stock, the first one is the latest price, and the last one is the oldest, which means that is day 1. Thus, we reverse that list of prices so that it starts from the very first day of the simulation. Then we can match the price list with a list of number from 1 to the last day of the list using ?zip? function:

```
Zip [1..] (reverse p)
```

At first, I used ordinary least square to calculate the regression line:

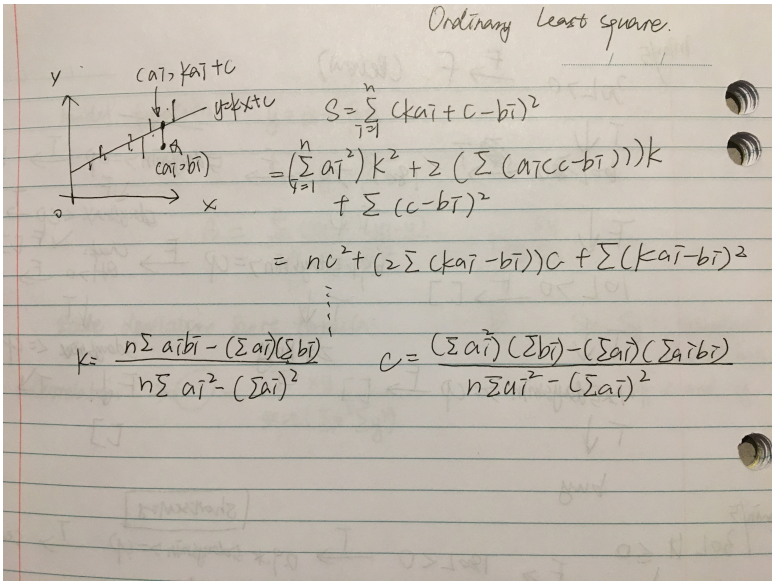


Figure 1: linear regression using ordinary least squares

The implementation of ordinary least squares is as follows:

```
calSlopeOfLine :: (Fractional a, Floating a) => [(a, a)] -> a
calSlopeOfLine pps = (z * timeandSumTwoLists pps -sum (map fst
            pps) * sum (map snd pps)) /
          (z * sum (squareElements pps) - (sum $ map fst pps)**2)
                where z = fromIntegral $ length pps

squareElements :: Floating a => [(a, a)] -> [a]
squareElements pss = [x ** 2| x <- map fst pss]

timeandSumTwoLists :: Num a => [(a, a)] -> a
timeandSumTwoLists pts = case pts of
    [] -> 0
    y : ys -> fst y * snd y + timeandSumTwoLists ys

calculateB :: (Fractional a, Floating a) => [(a, a)] -> a
calculateB spot = (sum (squareElements spot) * (sum (map snd spot
                )) - (sum (map fst spot))*timeandSumTwoLists spot)
                / fromIntegral (length spot) * sum (squareElements
                 spot) - (sum (map fst spot))**2)
```

After reviewing these function, I found that the function 'timeandSumTwoLists' can

write in another way which is more efficient than the previous one. The new timeandSumTwoLists is as follow:

```
newtimeandSum :: Num a => [(a, a)] -> a
newtimeandSum list = sum $ map (\z -> fst z * snd z) list
```

Then try the algorithm in wikipedia of simple linear regression:

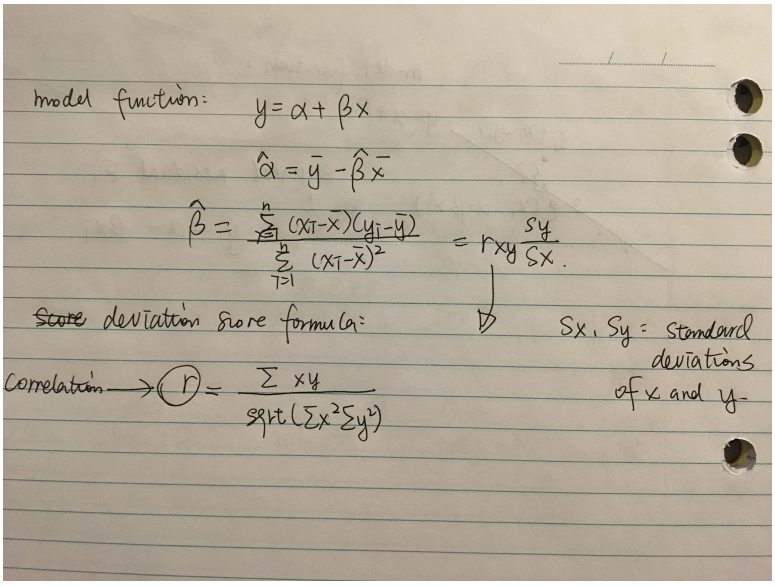


Figure 2: the algorithm used for the final version of linear regression

Finally, to find out which algorithm is more accurate, I choose a set of sample points from wikipedia:

Table 1: Sample points

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.47 | 1.50 | 1.52 | 1.55 | 1.57 | 1.60 | 1.63 | 1.65 | 1.68 | 1.70 | 1.73 | 1.75 | 1.78 | 1.80 |
| 52.21 | 53.12 | 54.48 | 55.84 | 57.20 | 58.57 | 59.93 | 61.29 | 63.11 | 64.47 | 66.28 | 68.10 | 69.92 | 72.19 |

The result calculated by first algorithm is around 65, and the result of second algorithm is around 61, according to the answer which is 61.6746, I choose the second algorithm to perform the regression line.

### 3.3 Main algorithm

There are several possibilities of the trend of the stock price. One thing I need to mention here is that when using the price data in 30 days. The points I use is the points that

has the minimum prices every 5 points. The reason I choose minimum points instead of using all points or the average of every 5 points is that if the line calculated by relatively lower points in 30 days is still increasing, then this line is more likely to increase. K30Min represents the slope of regression line in 30 days (using minimum points every 5 days), K30Max represents the slope of regression line in 30 days (using maximum points every 5 days),K60 represents the slope of regression line in 60 days, K180 represents the slope of regression line in 180 days. First check if the quantityHeld ¡ 0 , if so, then it means that we have used short selling, then check if it's the suitable point to buy these loan shares, if it is not, then skip this stock. If

$$K30Min > 0 and K60 > 0 and K180 > 0$$

Then, find a relatively lower price to buy: current price should equal to or smaller than 0.9 * the minimum price in 5 days. If it?s not a good point to buy, then check if I still hold shares of this stock, if so, then find a relatively higher price to sell all the holdings: Current price should larger than or equal to the maximum price in 5 days. If it?s not a good point to sell, then skip this stock.

If

$$K30Min > 0 and K60 > 0 and K180 <= 0$$

Then, find a relatively lower price to buy: Current price should equal to or smaller than the minimum price in 5 days. If it?s not a good point to buy, then check if it is a relatively higher points for selling: current price should larger than the maximum price in 5 days. If not, then skip this stock.

If

$$K30Min > 0 and K60 <= 0 and K10 > 0$$

Find a relatively lower price to buy Current price should equal to or larger than 1.1 * minimum price in 5 days.

If

$$K30Min > 0 and K60 <= 0 and k10 <= 0$$

Skip this stock

If

$$K30Min <= 0 and K30Max >= 0 and K10 > 0$$

Skip this stock

If

$$K30Min <= 0 and K30Max >= 0 and K10 <= 0$$

check if I have hold shares of this stock, if so, then sell all quantity. If not, skip this stock.

If

$$K30Min <= 0 and K30Max <= 0 and K180 >= 0$$

check if I have hold shares of this stock, if so, then find a relatively higher point to sell all quantity, if the current price is not suitable for selling then skip this stock. If not, skip this stock.

If

$$K30Min <= 0 and K30Max <= 0 and K180 < 0$$

try short selling. find a relatively higher points to sell, current price should equal to or smaller than 0.9 * minimum in 5 days, if not the right point, then check quantity held, if it is negative, then buy it; if it is not, then skip this stock.

### 3.4 Remaining question

Trying to find a way to record the price when I bought the stock, since it will be much easier to determine the right points to sell. Guessing maybe record type is a good way to implement that.

# 4 Testing

Using doctest to test the other helper functions: Since these functions are either deal with list of points or with list of numbers. Thus, we only need to test the function with all kinds of numbers including positive number, negative number and zero, also should containing integer and double.

Using QuickCheck to test three helper functions:

```
prop_groupEveryFive :: Eq a => [a] -> Bool
prop_groupEveryFive glist = glist == (foldl (++) [] $ groupEveryFive glist)
```

Check after applying groupEveryFive function to a list, whether each element in the result list can add to the original list.

```
prop_findMaxPoint :: (Num a, Eq a, Ord a) => [(a, a)] -> Bool
prop_findMaxPoint [] = (findMaxPoint []) == (0,0)
prop_findMaxPoint flist = (snd $ findMaxPoint flist) == (last $ sort $ map snd

prop_findMinPoint :: (Num a, Eq a, Ord a) => [(a, a)] -> Bool
prop_findMinPoint [] = (findMinPoint []) == (0,0)
prop_findMinPoint fmlist = (snd $ findMinPoint fmlist) == (head $ sort $ map snd
```

Choose the second place of each tuple in the list, then sort the list, here I import Data.List to enable the built in function ?sort?. Then, if it is the maximum one then it should be the last of the list, if it is the minimum one the it should be head of the list. Compare this result with the result of the tested function.

# 5 Conclusion

In conclusion, there are many different solutions to make profit without bankrupt in this assignment, however, the hard part to consider every situation and keep improving or even making big changes to the code, since it seems that there is always a better way to perform it. Moreover, it is really important to spend enough time writing and classifying all your thoughts before jump into typing codes. Furthermore, I wrote the report after finishing my code, however, when I try to recall the error and difficult part when testing my code, I find I could not recall most of them which is a pity. In the next assignment, I will spend more time thinking about the question thoroughly and start writing the draft of the report earlier.