2022.10.24 FYP 多因子策略开发 阶段小记

backtest()里会调用 getOrders()循环整个测试集，即 getOrders()只是一天的策略，backtest()循环 n 次来模拟策略进行 n 天的情况。

getOrders()有一个 parameter 可以得到循环内的天数（backtest()跑到第几次循环了）：store$iter

今天之前的想法：计算出一个固定的α值，再假定一个阈值 T，看 α 和 T 之间的关系来决定某天交易的方向（做多或做空）。
至于阈值定位多少，通过循环计算每个阈值下运行改策略能得到的总收益，取总收益最高者来确定最优 T 值。

逻辑问题：假如股票数据共 1000 天，将策略运行 1000 天并循环得到最优 T 值，再取最优 T 值并运用到策略中，是“预测未来”的逻辑错误。
因为假如 getOrders()循环到第 10 天，而判断第 11 天是买是卖的的阈值 T 缺失是通过总 1000 天的数据得出的，而实际上在第 11 天时根本无法知道未来(11 到 1000)天的股票数据，因此知道最优阈值 T 是个悖论。

10 支股票数据是用来测试策略的，而不是根据 1000 天的数据求最优策略来预测这几支股票 1000 天后的走势。

可用解决方法：将股票数据分割 in-sample 为 0 到 500 天，out-sample 分割为 500-1000 天，根据 0-500 天的数据求出最优阈值 T，再将该 T 用于 500-1000 天的测试。
这样 getOrders()是从第 501 天开始循环的，不会用到“未来”的数据，可以解决逻辑问题。

现核心问题：无法在 getOrders()里计算总收益率(PD ratio)来得到最优阈值 T。
由于 getOrders()是每天的交易策略，要算出最佳阈值 T，得跑出总收益需要用到 backtest()。
而 backtest()又会调用 getOrders()n 次来计算第 n 天的收益。
于是会遇到 getOrders()和 backtest()互相调用的无限循环情况。
尝试了不调用 backtest()将其拆开来自己写，也摆脱不了在 getOrders()内调用自己 getOrders()n 次的无限递归情况。

getOrders()是一天的策略，在没写好一天策略的情况下又如何在 getOrders()内调用自己 n 次模拟 n 天策略？

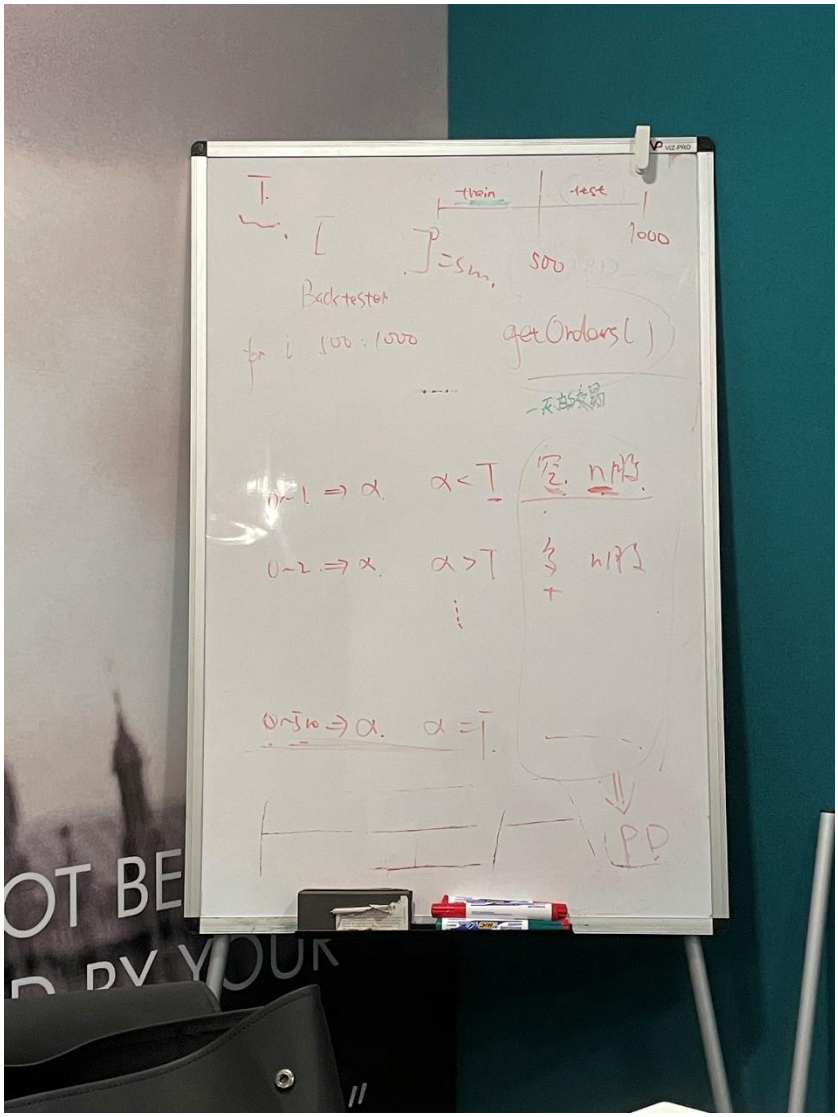
什么狗屁逻辑，头疼。凌晨 1 点多了。反正就是有很多 bug 和逻辑问题。

目前想的三个可能的解决方案：

1.在 getOrders()里增加一个 parameter 就叫 T。然后在 main 里面跑 getOrders 的时候，使用循环将某范围的 T 当 parameter 输入进 getOrders()。
在 main 里面 save.csv 将所有 T 值和其对应的 PD-ratio 存进一个 csv 里。
再另起一个 R 文件，得到新 csv 里最大 PD-ratio 的 T 值，再在新 csv 里重写 getOrders(),

并假如最佳 T 值。

2.在 getOrders()里用公式只得到该天的收益，避免算总收益。backtest()循环得到每天的收益的累加，循环完训练集 500 次得到前 500 天的收益。
T 是个 getOrders()内的变量，backtest()循环后求二次函数。



当时的白板。我现在也看不懂了

3.摆烂，不求最佳阈值 T 了，这样就避免了在 getOrders()里算总收益。T 直接随便取个值，爱赚钱赚钱亏死也不关我事，反正这学期的 report 能讨论目前的缺陷。要优化等下学期再说。

附上目前出来的学术垃圾：

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

```
maxRows <- 3100 # used to initialize a matrix to store closing prices

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

  #Initializing
  allzero    <- rep(0,length(newRowList)) # used for initializing vectors

  if (is.null(store)) store <- initStore(newRowList,params$series)
  store <- updateStore(store, newRowList, params$series)

  marketOrders <- -currentPos; pos <- allzero

  #Get data
  dataList <- getData(directory="PART1")

  #Initialize PD ratio and Threshold
  BestThreshold <- 0
  PD <- -Inf
  Iteration <- 0

  #Iterate through all possible alpha rate
  for (a in -100:100){

    #Iterate through the series in params$series
    for (i in params$series){

      #Get every stock's volume and closed price data
      VOLUME = store$vol[,i]
      CLOSE = store$cl[,i]

      #Identify VWAP index
      VWAP = sum(CLOSE*VOLUME, 10)/sum(VOLUME, 10)

      #Alpha006
      alpha = -1*cor(CLOSE, VOLUME, method="pearson")

      #Change Position
      if (alpha*100 < a){
        pos[params$series[i]] <- -1
      }
      else if (alpha*100 > a ){
        pos[params$series[i]] <- 1
      }
      else if (alpha*100 == a){
```

```
            pos[params$series[i]] <- 0
        }

        #Update market orders
        marketOrders <- -currentPos + pos
    }

    #The return list of the strategy
    store=store
    marketOrders=marketOrders
    limitOrders1=allzero
    limitPrices1=allzero
    limitOrders2=allzero
    limitPrices2=allzero

    #Store a temporary "getOrders" list
    temOrders <- list(store, marketOrders, limitOrders1, limitPrices1, limitOrders2,
limitPrices2)

    #Calculate the result by calling backtester
    sMult <- 0.20 # slippage multiplier
    is_valid_example_strategy <- function(strategy) {
        strategy %in% example_strategies
    }
    stopifnot(is_valid_example_strategy(strategy))
    load_strategy(strategy) # function from example_strategies.R



    results <- backtest(dataList,temOrders,params,sMult) #Error-infinite recursive
    ########################################(Calculate
Result)########################################

    numOfDays <- nrow(dataList[[1]])
    numOfSeries <- length(dataList)

    # initialise as 0-vector of length length(dataList)
    newPosList <- vector(mode="numeric", length = length(dataList))

    # Initialisation of getOrders with first row of data, via is.null(store)
    store <- NULL #(Note: Variable "store" occur potentially bug!)

    # pnlList will store trading results
    # initialize lists of 0 rows; getRowList(dataList,1) used to get date for each via index()
```

```
    # pnlList <- mapply(function(x, y) xts(x, index(y)),0, getRowList(dataList,1), SIMPLIFY =
FALSE)
    pnlList <- lapply(1:numOfSeries,function(x) matrix(0,nrow=numOfDays,ncol=1))
    positionValuesList <- lapply(1:numOfSeries,function(x)
matrix(0,nrow=numOfDays,ncol=1))
    netWorthList <- rep(0, numOfDays)

    # vector that stores a 1 for every day a position was taken in some
    # series and a 0 otherwise
    # initialized as all zero vecotr
    posCounter <-  0

    nonxtsDataList <- lapply(dataList, function(x) as.matrix(x))

    balance <- 1000000
    newNetWorth <- balance
    netWorthList[[1]] <- balance

    bankrupt <- FALSE # Are we bankrupt?

    # MAIN LOOP
    for (i in 2: (numOfDays-1)) { # cannot hold on day 1; day 1 data is given to strategy on
day 2

      oldPosList    <- newPosList

      info = list(balance=balance, netWorth=newNetWorth)

      #Modified - Not using "getOrders" function which might occur infinite recursion
      #Instead, using temporary list
      x <- ?????
      x <- getOrders(store, getRowList(dataList,i-1), oldPosList, info, params) #Error-infinite
recursive
    }


##########################################################################
#########################


    pfolioPnL <- plotResults(dataList,results,plotType='ggplot2')

    #Get the PD-ratio
    NewPD <- pfolioPnL$fitAgg
```

```
    #Compare Pd-ratio and Update the best threshold
    if (NewPD>=PD){
        PD <- NewPD
        BestThreshold <- a/100
    }

    #Print Iteration
    Iteration++
    print("Iteration" +Iteration)
}


#Initializing Again
allzero    <- rep(0,length(newRowList)) # used for initializing vectors

if (is.null(store)) store <- initStore(newRowList,params$series)
store <- updateStore(store, newRowList, params$series)

marketOrders <- -currentPos; pos <- allzero

#Apply the Best Threshold
for (i in params$series){

    VOLUME = store$vol[,i]
    CLOSE = store$cl[,i]

    #Alpha006
    alpha = -1*cor(CLOSE, VOLUME, method="pearson")

    #Change Position
    if (alpha*100 < BestThreshold){
        pos[params$series[i]] <- -1
    }
    else if (alpha*100 > BestThreshold){
        pos[params$series[i]] <- 1
    }
    else if (alpha*100 == BestThreshold){
        pos[params$series[i]] <- 0
    }

    #Update market orders
    marketOrders <- -currentPos + pos
}
```

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



initClStore  <- function(newRowList,series) {
  clStore <- matrix(0,nrow=maxRows,ncol=length(series))
  return(clStore)
}
updateClStore <- function(clStore, newRowList, series, iter) {
  for (i in 1:length(series))
    clStore[iter,i] <- as.numeric(newRowList[[series[i]]]$Close)
  return(clStore)
}



initVolStore  <- function(newRowList,series) {
  volStore <- matrix(0,nrow=maxRows,ncol=length(series))
  return(volStore)
}
updateVolStore <- function(volStore, newRowList, series, iter) {
  for (i in 1:length(series))
    volStore[iter,i] <- as.numeric(newRowList[[series[i]]]$Volume)
  return(volStore)
}



initStore <- function(newRowList,series) {
  return(list(iter=0,cl=initClStore(newRowList,series),vol=initVolStore(newRowList,series)))
}
updateStore <- function(store, newRowList, series) {
  store$iter <- store$iter + 1
  store$cl <- updateClStore(store$cl,newRowList,series,store$iter)
  store$vol <- updateVolStore(store$vol,newRowList,series,store$iter)
  return(store)
}
```

注意——α的表达都是错的。比如 getOrders()循环到第 10 天，应该求前 9 天的 α。α 值每天都在变，不应该像此代码一样只算了 1000 天的总 α。
注意——此代码包含了上述提及的无限递归问题，未解决，小心电脑炸。