ReadMe—Yifei Guo (yg2496)

This file aims to provide further explanations and clarifications that are helpful to understand my codes. All codes are included in a single .Rmd file divided into 46 chunks or sections. All my explanations are organized according to the same order as in the code file.

Section 1

1. First, please specify the file location for your stock-data inside setwd(' ').
2. This chunk combines the "adjclose" prices for all stocks with at least ten years data into one dataframe called df. One stock occupies one column of df and the row names of df are the dates. df has 6580 rows and 8055 columns (stocks).
3. This section may take about one hour to run.

Section 2

1. This chunks extract data from "2009-04-15" to "2019-04-18", last ten years with 2521 trading days.
2. Using "which" functions to find out row numbers for dates "2009-04-15" and "2019-04-18" reveals that the portion of df including data for the last 10 years is already in increasing order. (The row number of "2009-04-15" is smaller than the row number of "2019-04-18") Therefore, we do not need to reorder my data.
3. For some reasons originated from the stock-price-excels, there are two rows with the same date as 2019-04-18. Further examination reveals that these two rows are identical so it is ok just extract data from 2009-04-15 to 2019-04-18 and ignore the identical row.

Section 3

This chunk simply extracts all stocks with less than 5% missing variables using a simple for loop.

Section 4

1. This chunk computes log returns for all stocks extracted in Section 3.
2. My formula for log return computation is $logReturn_t = \log(\frac{P_{t-1}}{P_t})$. Therefore, this explains why my stock-price-data starts from 2009-04-15. I need stock prices on 2009-04-15 to get log returns for 2009-04-16.
3. After running this chunk, we get a dataframe "df.logReturn" with log returns.

Section 5

This chunk re-examines the number of missing values for each stock in the "df.logReturn"

dataframe and extracts stocks with less than 5% missing variables after calculating the log returns.

Section 6

1. Because checking volumes for the entire 8000 stocks is time-consuming, I checked volumes only for the 3903 stocks with less than 5% missing variables.
2. This chunk reports a list of stock names with less than 5% missing variables.

Section 7

1. If you have already specified your file location in section 1 you do not need to do again.
2. After running this section, we get another dataframe "df.vol.1", recording volumes for the 3903 stocks selected over the last ten years.

Section 8

1. Vector "vol.ext" reports a list of stock names with insignificant trading volumes.
2. The final command in this chunk deletes columns, with stock names specified in vector "vol.ext" , in the log-return dataframe. Then, this dataframe is ready for missing values and outliers detections and cleanings.

Section 9

1. I implemented functions "na.interp()" and "tsclean()" to clean the data. The entire data cleaning procedure ends at this section. Finally, I have 3258 stocks in total.
2. If your R software has not installed the "forecast" and "tsoutliers" packages yet, the installation commands are added as comments.

Section 10

1. This section extracts data for s&p 500 over the last 10 years. The file I used is already in increasing order from "2009-04-15" to "2019-04-18". Again, I need s&p 500 on 2009-04-15 to calculate the log return for 2009-04-16.

Section 11

1. This section calculates log returns for s&p 500.
2. This section also changes the first column(dates) of the stock-log-return dataframe into the row names of the dataframe.

Section 12

1. This chunk creates a matrix to record all relevant statistics (mean returns (column 1),

volatilities (column 2), Sharpe Ratio (column 3), beta (column 4)) for all stocks in each rolling window. All values are annualized.
2. This matrix has dimension 3258(# of stocks)*4.

Section 13

1. Since I need stock prices to do trend detections in the following codes and my log returns for all stocks have been cleaned, I need to recalculate stock prices according to the log returns. This section does this calculation to get cleaned prices.
2. I specified the initial prices for all stocks to be 100 for simplicity and to avoid missing values. This specification does not affect any computations because the cleaned price data is only used for trend detections.

Section 14 (Stock Selection for the first rolling window)

1. This section includes two functions.
2. Function "check.include" takes two inputs: a vector (v) with stock names and a string variable (s) of one stock name. This function returns a boolean variable which indicates whether the stock name (s) has already been included in the vector v or not.
3. Function "stock.select.r.w.1" does the stock selection for the first rolling window. Based on results from the midterm project, I chose 20 stocks instead of 30 stocks for each rolling window.

Section 15 (Use Dynamic CAPM and Kalman Filter to get the vector $\widehat{\boldsymbol{\beta}}$ and matrix $\widehat{\boldsymbol{V}}$)

1. In order to get the estimated F matrix I need three components: a vector $\widehat{\boldsymbol{\beta}}$, matrix $\widehat{\boldsymbol{V}}$, and $\hat{\Sigma}$.
2. $\widehat{\boldsymbol{\beta}}$ is a vector with 20 elements, each corresponding to one $\hat{\beta}$ for one selected stock.
3. Matrix $\widehat{\boldsymbol{V}}$ is the residual matrix, calculated using command $diag\left(diag\left(\widehat{cov}(r_{ti} - \widehat{\beta_{ti}} * r_{ti}{}^{M})\right)\right)$ for all 20 stocks over one rolling window. Function "estimate.v.one.asset" calculates $r_{ti} - \widehat{\beta_{ti}} * r_{ti}{}^{M}$, which is a matrix with dimension m*20. m is the rolling window size.
4. $\hat{\Sigma}$ in this case is the variance of S&P 500.
5. Functions "estimate.beta.one.asset", "kalmanf.estx" and "kalmanf.update" are inspired by codes uploaded on coursework. "kalmanf.estx" and "kalmanf.update" are called within "estimate.beta.one.asset", finally reporting a vector with dynamic CAPM betas calculated over one rolling window for one asset. In the following codes, the last beta from this vector for each stock is used to form the $\widehat{\boldsymbol{\beta}}$ vector.

Section 16

1. This section includes only one function "get.F", which uses the three components calculated above to get the F matrix. $F = \hat{\Sigma}\hat{\beta}\hat{\beta}^T + \hat{V}$

Section 17 (Portfolio Optimization)

1. This section includes two functions: "portfolio.opt" and "rollingWindow.opt"
2. "portfolio.opt" function first calculates the optimal shrinkage parameter δ suing formulas from the paper Ledoit & Wolf (2002), then implementing the formula $\hat{S} = \delta F + (1-\delta)S$, where S is the sample covariance matrix, to get the estimated covariance matrix. Finally, using $w_{globalMVP} = \hat{S}^{-1}1/C$, where $C = 1^T\hat{S}^{-1}1$, to get weights for each chosen stock (20 stocks total).
3. "rollingWindow.opt" function is the main function of this project, which does the rolling window exercise.
   - This function takes two inputs: m (rolling window size, I choose m = 504) and h (time-period length without rebalancing)
   - This function returns a list of 5 objects: 1. a vector reporting period returns for each rolling window. 2. a list of weights for each rolling window. 3. a vector reporting shrinkage parameter δ for each rolling window. 4. a list of 20 stock-names for each rolling window. 5. a list of stock-specified returns for each rolling window.
   - Stock replacements for rolling window 2 and beyond (described in the PPT) are done within this function.

Section 18

1. This section includes one function "cumulative.return.function", which converts period returns for each rolling window to cumulative returns.

Section 19

1. This section calls the function rollingWindow.opt(504, 5) to do the entire rolling window exercise for m=504 and h=5. This section may take 20 mins to run.

Section 20

1. This section calls the function rollingWindow.opt(504, 20) to do the entire rolling window exercise for m=504 and h=20.

Section 21

1. This section includes one function "rollingWindow.eql.1", which does the rolling window exercise for the equally weighted portfolio 1/20.

Section 22

1. This section calls the function rollingWindow.eql.1(504, 5, matr.stock.name) to do the entire rolling window exercise for the equally weighted portfolio with m=504 and h=5. Input matr.stock.name passes the list of stocks selected in Section 19 to the equally weighted portfolio.

Section 23

1. This section calls the function rollingWindow.eql.1(504, 20, matr.stock.name) to do the entire rolling window exercise for the equally weighted portfolio with m=504 and h=20. Input matr.stock.name passes the list of stocks selected in Section 20 to the equally weighted portfolio.

Section 24

1. This section includes one function "rollingWindow.sp", which does the rolling window exercise for S&P 500.

Section 25

1. This section calls the function rollingWindow.sp (504, 5) to do the entire rolling window exercise for S&P 500 with m=504 and h=5.

Section 26

1. This section calls the function rollingWindow.sp (504, 20) to do the entire rolling window exercise for S&P 500 with m=504 and h=20.

Section 27

1. This section includes one function x.lab which reports a vector of dates indicating the starting date for each rolling window.

Section 28 and Section 29

1. These two sections plot "Gross Cumulative Return" graphs for h=5 and h=20.

Section 30

1. This section includes a function "turnover.vector", which calculates the total change in weight for each rolling window and returns a vector. This function is used for optimized portfolios only.

2. To calculated turnovers for each rolling window, we need to first calculate the adjusted weight for each stock w_{adj_i}. I calculated $w_{adj_i}{}' = w_{orig_i} * e^{r_i}$ (r_i represents the log return for stock i over h periods) and then rescaled $w_{adj_i}{}'$ so that w_{adj_i} for the 20 selected stocks add up to 1.

Section 31

1. This section includes a function "net.return", which calculates net returns for each rolling window using the gross returns.

Section 32

1. This section includes a function "turnover.vector.eql", which calculates the total change in weights for each rolling window and returns a vector. This function is used for equally weighted 1/20 portfolios only. Adjusted weights are calculated using the same strategy as in Section 30.

Section 33

1. This section implements the "net.return" function defined in Section 31 to generate net returns from gross returns.

Section 34 and Section 35

1. These two sections calculate net cumulative returns for both optimized portfolios and equally weighted portfolios for h=5 and h=20, then plotting net cumulative return graphs for h=5 and h=20.

Section 36

1. This section includes one function "fl.stat.1", which calculates final statistics and organizes all statistics into a matrix. Final statistics include: Mean return, Volatility, Sharpe Ratio; Sortino Ratio; Max Drawdown and Turnover. Final statistics calculated by this matrix are not yet annualized.

Section 37

1. This section implements the "fl.stat.1" function defined above.

Section 38

1. This section converts statistics calculated in Section 37 to annualized statistics.

Section 39

1. Section 39 prints out the matrices.

Following sections are implemented to obtain illustrative graphs and statistics for my presentation.

Section 40 and Section 41

1. These two sections are meant to put gross cumulative return curves and net cumulative returns into one graphs for h = 5 and h = 20.

Section 42

1. This section is used to get an illustrative graph for dynamic betas.

Section 43 and Section 44

1. These two sections try a different rolling window size. m = 252

Section 45 and Section 46

1. I used these two sections to get final statistics for m = 252 and to print out the matrices.