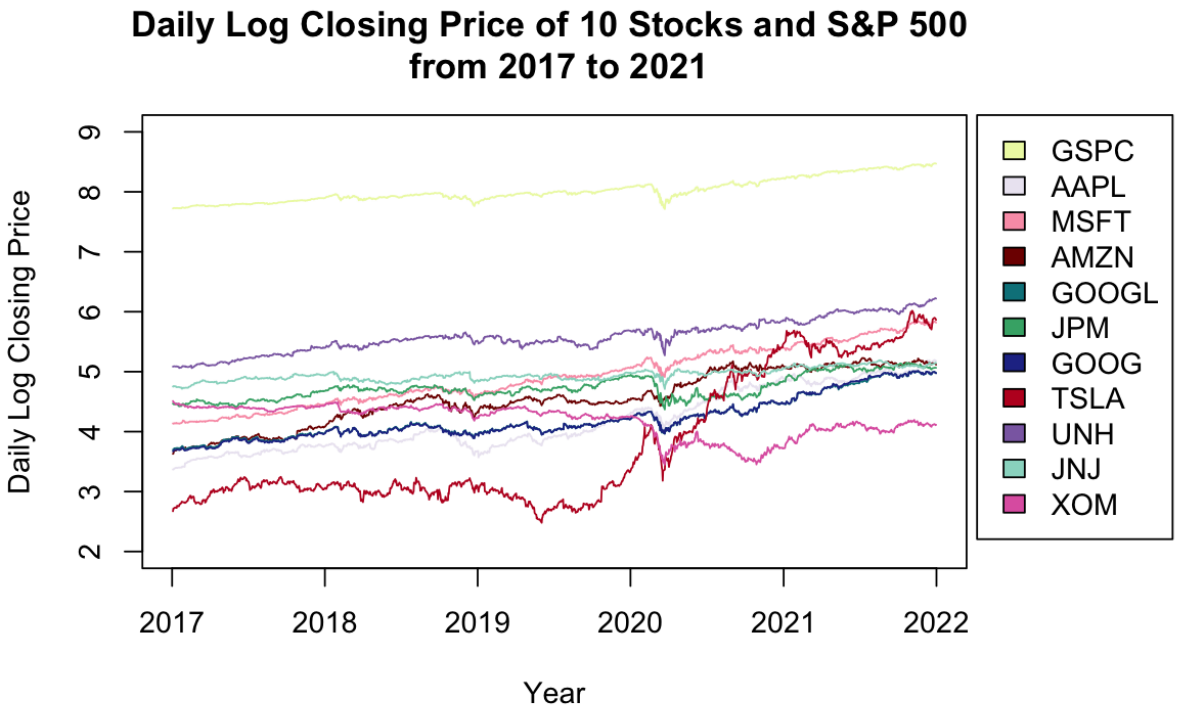
ST326 MT2022
43727

Question 1

The selected 10 stocks from top 100 constituents of S&P500 by index weight as of 10th October 2022 are as follows:

a. Apple Inc. (APPL)
b. Microsoft Corp (MSFT)
c. Amazon.com Inc (AMZN)
d. Alphabet Inc A (GOOGL)
e. Alphabet Inc C (GOOG)
f. Tesla, Inc (TSLA)
g. Unitedhealth Group Inc (UNH)
h. Johnson & Johnson (JNJ)
i. Exxon Mobil Corp (XOM)
j. JPMorgan Chase (JPM)

The daily log-closing prices of 10 stocks and S&P 500 for the past 5 years from 2017 to 2021 are plotted on Figure 1.

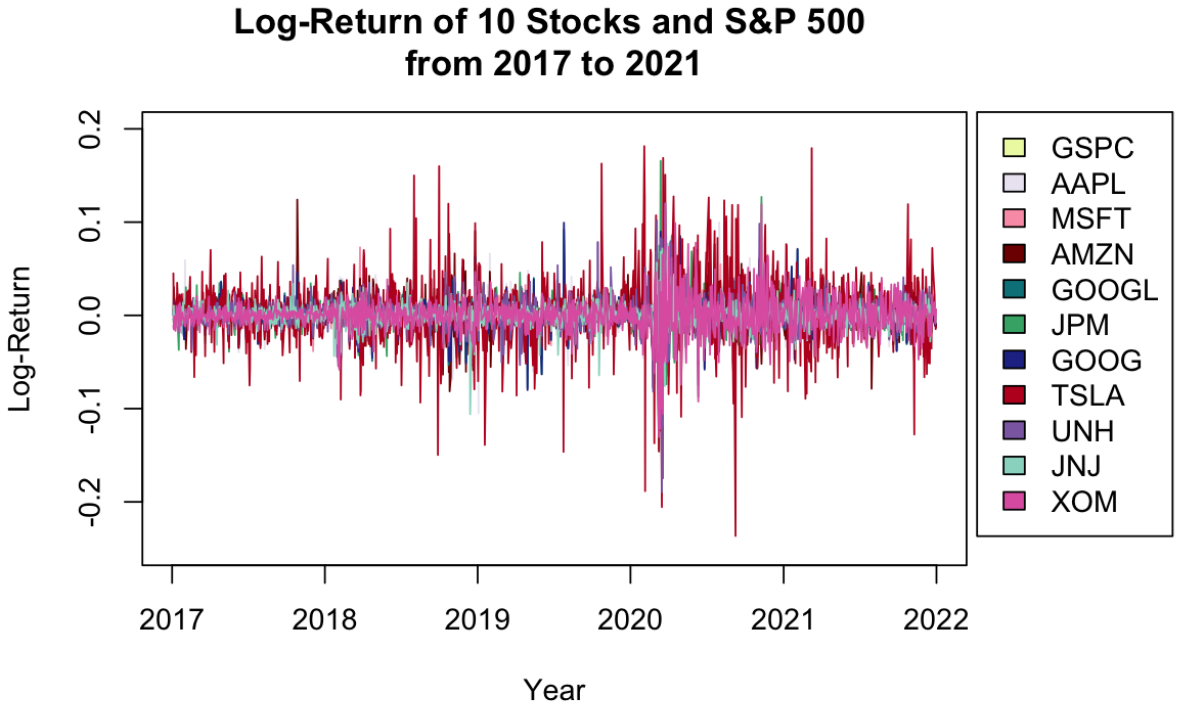


(Figure 1 Daily log-closing price of 10 stocks and S&P 500 from 2017 to 2021)

Based on Figure 1, most of the constituents of S&P 500 and S&P 500 share similar trends most of the time, especially during the recent COVID-19 period, which is the beginning of 2020. Most of them plunged and later growed.

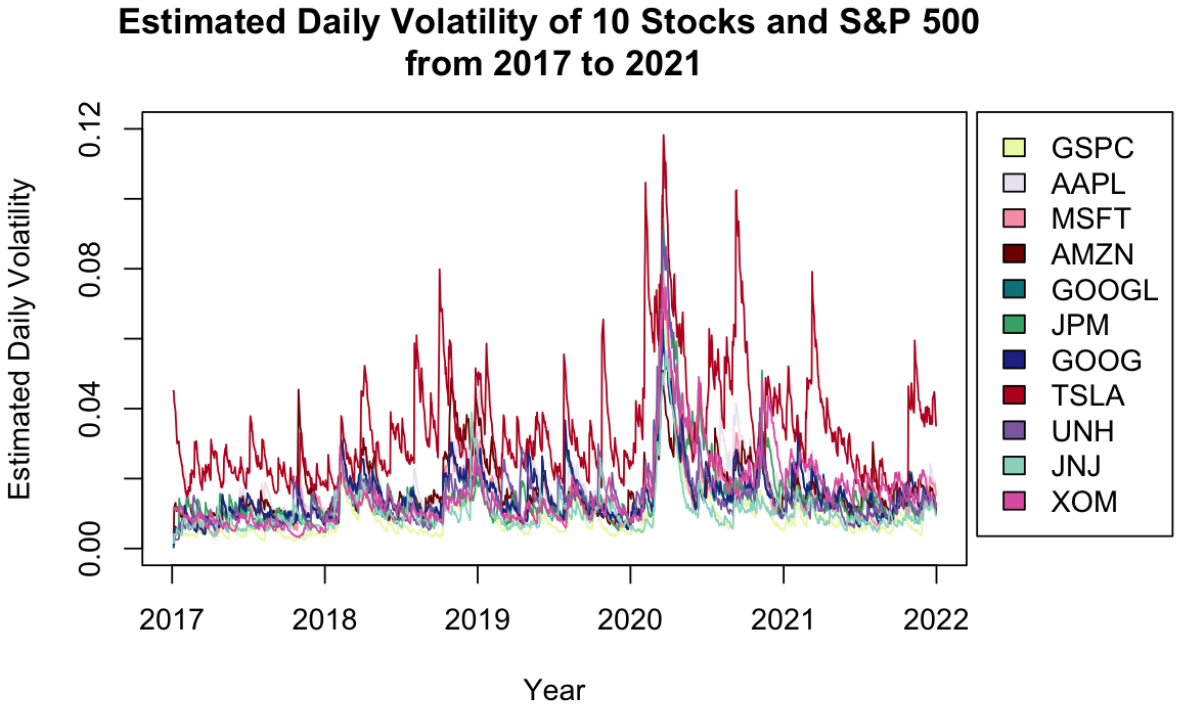
Question 2

Next day S&P 500 return is predicted using q lags of S&P500 and the most up-to-date returns of the 10 stocks by splitting the data set into 50% training, 25% validation and 25% test.A function *pred.snp.prepare* with arguments *max.lag, split and mask* is used. *max.lag* is used to input q lags. *split* takes the percentage term of the training and validation sets respectively. *mask* is used to decide which selected 10 stocks to include, where input of 1 means including the stock and input of 0 for excluding the stock in the analysis. *shift.indices in pred.snp.prepare* function would take values of all 0 as all stocks in S&P 500 trades at the same time.$\boldsymbol{Y}$ is the response variable of S&P 500 at $t+1$ whereas $\boldsymbol{X}$ is the design matrix incorporating S&P 500 returns at $t, t-1, t-2,$ … $t-q+1$ where q is the max number of lag , and returns of 10 stocks at time t .



(Figure 2.1 Daily log-return of 10 stocks and S&P 500 from 2017 to 2021)

Different stocks have different returns as shown in Figure 2.1, which implies that different stocks have different volatilities. For the 11 daily return series, daily volatilities for for each 11 variables (Figure 2.2) can be estimated using exponential filter model in the recursive form, $\sigma_t^2 = (1-\lambda)x_{t-1}^2 + \lambda\sigma_{t-1}^2$ with the training data. Instead of estimating 11 λ for all variables, only a λ for 11 variables is estimated as the λ has similar values for simplicity.



(Figure 2.2 Estimated daily volatility of 10 stocks and S&P 500 from 2017 to 2021)

λ could be estimated with maximum likelihood as the exponential filter model is a reduced GARCH model. Minimizing Residual Sum of Squares is equivalent to maximizing likelihood since
$argmin - f(x) = argmax f(x)$

Likelihood

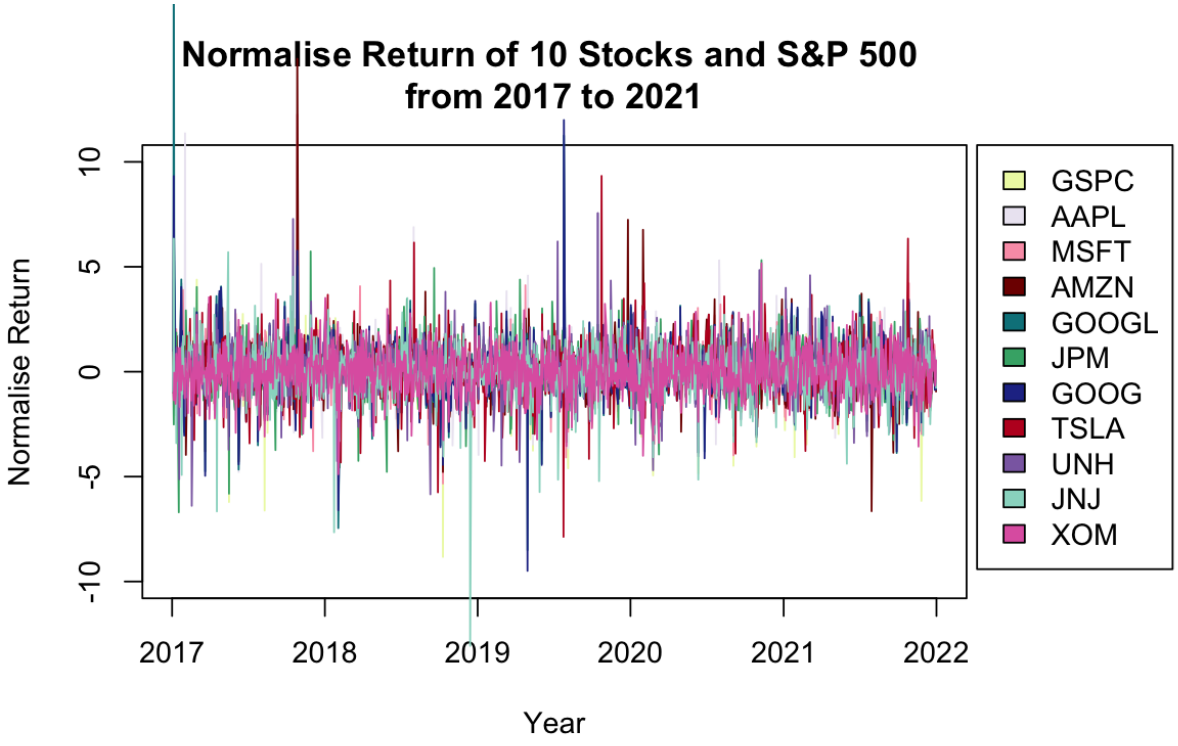
$$p(y|X,w) = \prod_i^N \mathcal{N}(y_i - w^\top x_i, \sigma^2) \approx \prod_i^N \exp(-\frac{1}{2}(y_i - w^\top x_i)^2/\sigma^2)$$

$$\log p(y|X,w) = \log exp(-\frac{1}{2}\sum_i^N (y_i - w^\top x_i)^2/\sigma^2) \approx -\sum_i^N (y_i - w^\top x_i)^2/\sigma^2$$

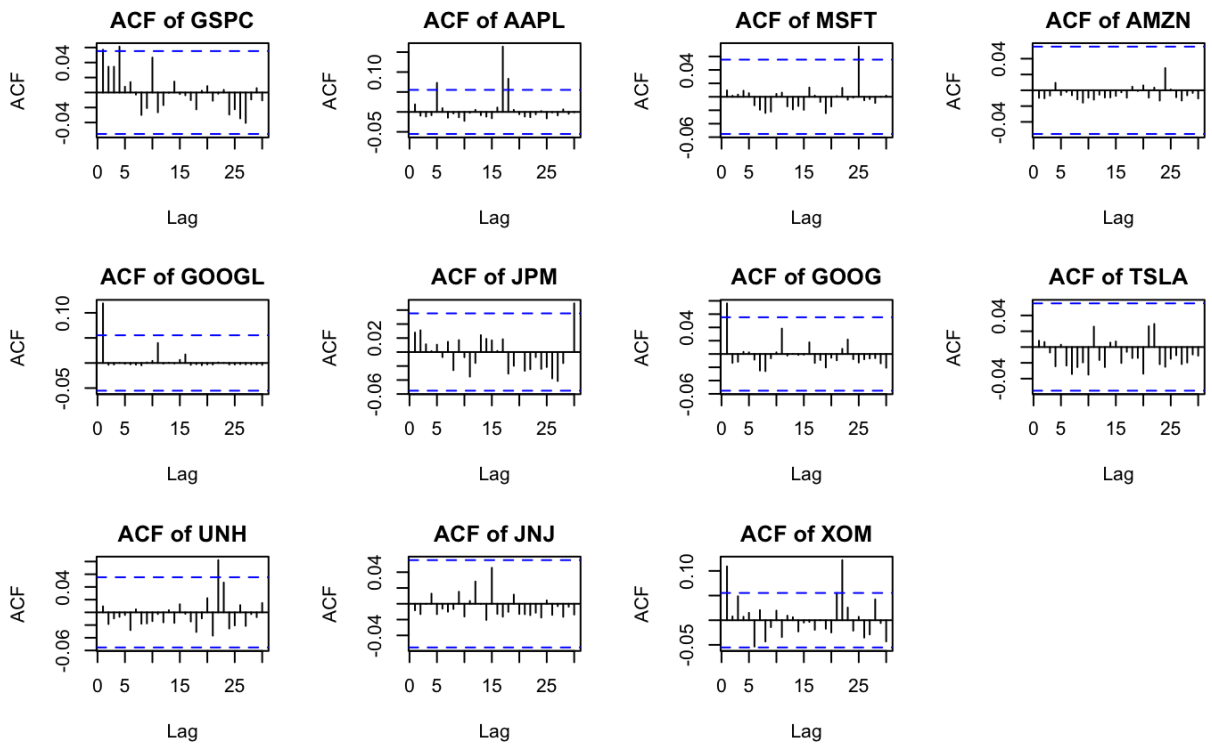
Residual Sum of Squares

$$RSS = \sum_{i}^{N} (y_i - w^\top x_i)^2$$

Estimated $\widehat{\lambda}$ is chosen by minimizing the sum (over 11 variables) of absolute Autocorrelation Function (ACF) at lag 1 of the residual sum of squares over a grid of values (0,1) of λ. This is because the generalized ARCH model of order p,q, GARCH(p,q)is defined by $x_t = \sigma_t \epsilon_t$ where $\epsilon_t \sim IID(0,1)$. The estimated residual or normalized return $\tilde{\epsilon}_t = x_t / \tilde{\sigma}_t$(Figure 2.3), where $\tilde{\sigma}_t^2$, the estimator of σ_t^2, should be approximately independent of each other. Therefore, the estimated residual sum of squares should be independent of each other as well. Hence, ACF of the squared normalized daily returns, $\tilde{\epsilon}_t^2$ should be within the dotted blue interval and treated as insignificant. In Figure 2.4, the squared normalized daily returns of 10 stocks and S&P 500 looks like white noise as the returns are mostly within the dotted blue interval.



(Figure 2.3 Normalised returns of 10 stocks and S&P 500 from 2017 to 2021)



(Figure 2.4 ACF of squared normalized daily returns of 10 stocks and S&P 500 from 2017 to 2021)

The linear regression models with time-varying coefficients of S&P 500's normalized return against selected stocks' normalized return are as follow:

1. Apple Inc. (AAPL)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_1 \boldsymbol{X}_t^1 + \boldsymbol{\epsilon}_{t+1}$$

2. Microsoft Corporation (MSFT)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_2 \boldsymbol{X}_t^2 + \boldsymbol{\epsilon}_{t+1}$$

3. Amazon.com Inc. (AMZN)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_3 \boldsymbol{X}_t^3 + \boldsymbol{\epsilon}_{t+1}$$

4. Alphabet Inc A (GOOGL)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_4 \boldsymbol{X}_t^4 + \boldsymbol{\epsilon}_{t+1}$$

5. Alphabet Inc C (GOOG)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_5 \boldsymbol{X}_t^5 + \boldsymbol{\epsilon}_{t+1}$$

6. Tesla, Inc (TSLA)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_6 \boldsymbol{X}_t^6 + \boldsymbol{\epsilon}_{t+1}$$

7. Unitedhealth Group Inc (UNH)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_7 \boldsymbol{X}_t^7 + \boldsymbol{\epsilon}_{t+1}$$

8. Johnson & Johnson (JNJ)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_8 \boldsymbol{X}_t^8 + \boldsymbol{\epsilon}_{t+1}$$

9. Exxon Mobil Corp (XOM)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_9 \boldsymbol{X}_t^9 + \boldsymbol{\epsilon}_{t+1}$$

10. JPMorgan Chase (JPM)

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \boldsymbol{\gamma}_{10} \boldsymbol{X}_t^{10} + \boldsymbol{\epsilon}_{t+1}$$

11. S&P 500 Against all 10 stocks

$$\boldsymbol{Y}_{t+1} = \sum_{j=1}^{q} \beta \boldsymbol{Y}_{t-j+1} + \boldsymbol{\gamma_0} + \sum_{i=1}^{10} \boldsymbol{\gamma}_i \boldsymbol{X}_t^i + \boldsymbol{\epsilon}_{t+1}$$

Question 3

Step 1

Start from a certain warmup time $t = t_0$, fixed D, where D is the window length and normalized 10+q return series as input.The linear regression model with time-varying coefficients can be written as

$$\boldsymbol{Y}_t = \boldsymbol{X}\boldsymbol{\alpha} + \boldsymbol{e}$$

where $\mathbf{Y} = (Y_{t-D+1}, \cdots, Y_t)^\top$, $\mathbf{X} = (\mathbf{Y}_{-1}, \mathbf{Y}_{-2}, \cdots, \mathbf{Y}_{-q}, \mathbf{1}_D, \mathbf{X}^1, \mathbf{X}^2, \cdots, , \mathbf{X}^{10})$ with $\mathbf{Y}_{-q} = (Y_{t-D+1-q}, Y_{t-D-q}, \cdots, Y_{t-q})^\top$ and $\mathbf{X}^k = (X^k_{t-D}, X^k_{t-D+1}, \cdots, X^k_{t-1})^\top$, $\boldsymbol{\alpha} = (\beta_1, \beta_2, \cdots, \beta_q, \gamma_0, \cdots, \gamma_{10})^\top$ and $\mathbf{e} = (e_{t-D+1}, \cdots, e_t)^\top$. The vector $\mathbf{1}_D$ is a column vector of length D and $\boldsymbol{e} \sim \mathcal{N}(0, \sigma^2)$ is the white noise.

$\mathbf{Y}$ is the normalized return of S&P 500 to be predicted, $\mathbf{Y}_{-q}$ is the normalized return of S&P 500 at lag q, $\mathbf{X}^k$ is the normalized return of selected stocks from S&P 500. k is the index to select the stocks. (not power of k). Lag values of S&P 500 at t,.... $t - q + 1$ and selected 10 stocks at t are used to predict the S&P return at $t + 1$

Estimate α by the least square estimator,
$\hat{\alpha} = \hat{\alpha}_D(t) = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$

Step 2

The next day normalized return for S&P 500, $\hat{Y}_{t+1}$ is estimated using ordinary least square regression over a rolling window length of D.

$$\hat{Y}_{t+1} = \hat{\alpha}^\top \mathbf{x}_t$$

where $\mathbf{x}_t = (Y_t, Y_{t-1}, \cdots, Y_{t+1-q}, 1, X^1_t, \cdots, X^{10}_t)^\top$

The calculation for $\hat{Y}_{t+1}$ ignore all transaction costs.

Step 3

$\hat{Y}_{t+1}$ is now the investment strategy. If $\hat{Y}_{t+1}$ is positive, invest 1 unit of money into S&P 500. If $\hat{Y}_{t+1}$ is negative, short sell 1 unit of money.

The real return of S&P 500 at $t + 1$, R_{t+1} is simplified by not multiplying forecasted volatility effect at $t + 1$.

$$R_{t+1} = \hat{Y}_{t+1} Y_{t+1}$$

R_{t+1} :The real return of S&P 500 at $t + 1$

$\hat{Y}_{t+1}$:Normalized real return of S&P 500 at $t+1$

Step 4

Go back to Step 1 and replace t with $t+1$ till the end of the training set.

Step 5

Repeat Step 1 to Step 4 with a grid of values for window length, D.

Step 6

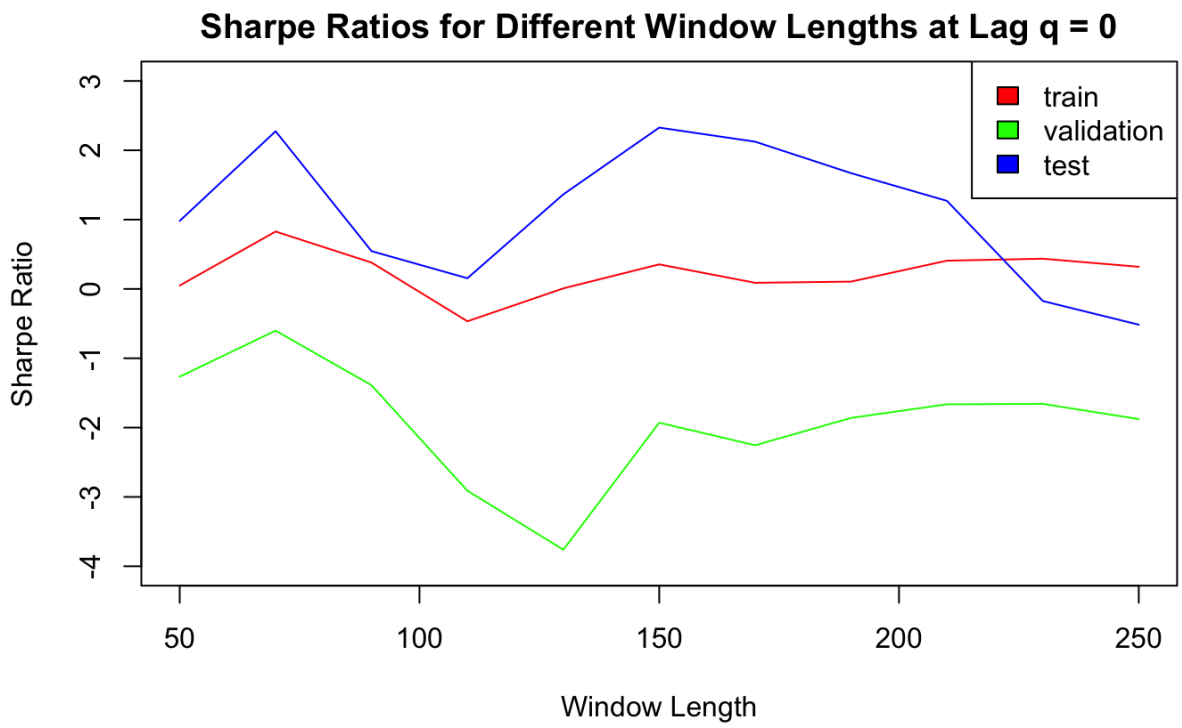
The sharpe ratio is calculated to determine if a investment strategy, $\hat{Y}_{t+1}$ is good.
Compute the annualized sharpe ratio at the end using daily real return, R for different window length, D and time, t with the formula as follows:

$$(\text{Annualised}) \text{ Sharpe ratio} = \sqrt{250} \times \frac{\text{Average Daily return}}{\text{SD of Daily returns}}$$

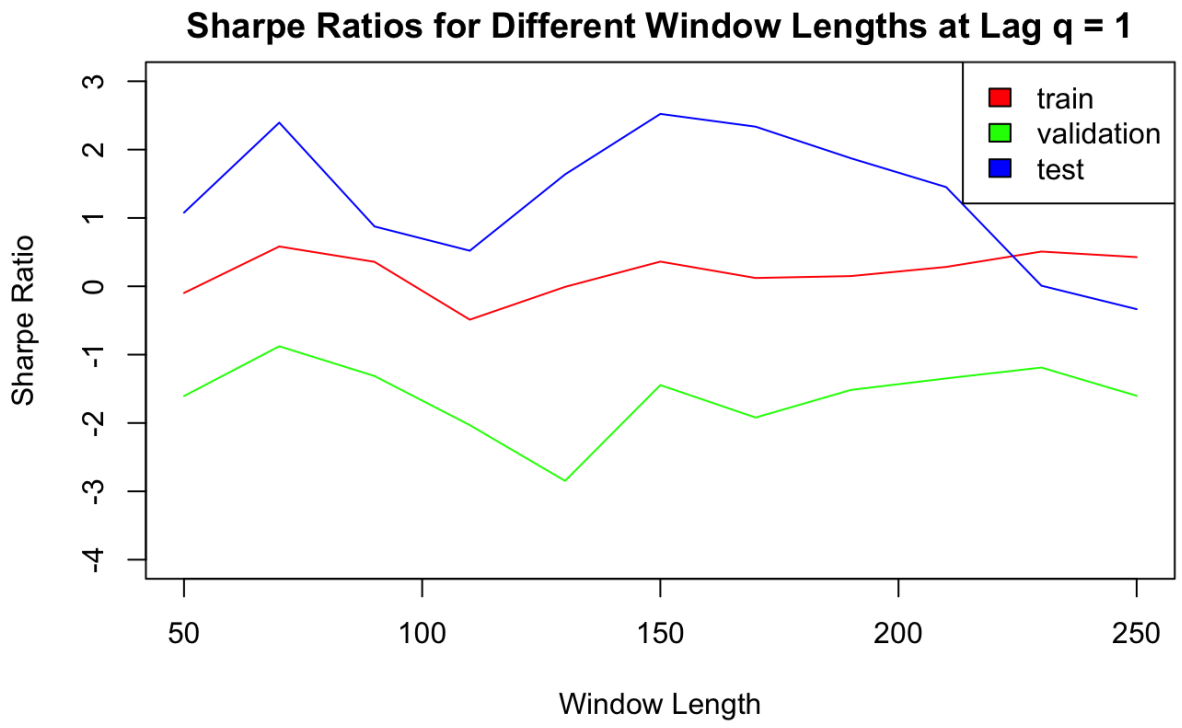
Note: The number of trading days in a year is 250 and transaction costs are ignored.

Question 4

Figure 4.1 and Figure 4.2 demonstrate the Sharpe Ratios for the algorithm (from Step 1 to step 6 in Question 3) running on training, validation and test sets, with D = 50 to 250, adding 20 each time. The estimated lambda, $\widehat{\lambda}$ found in question 3 is used in validation and test sets.

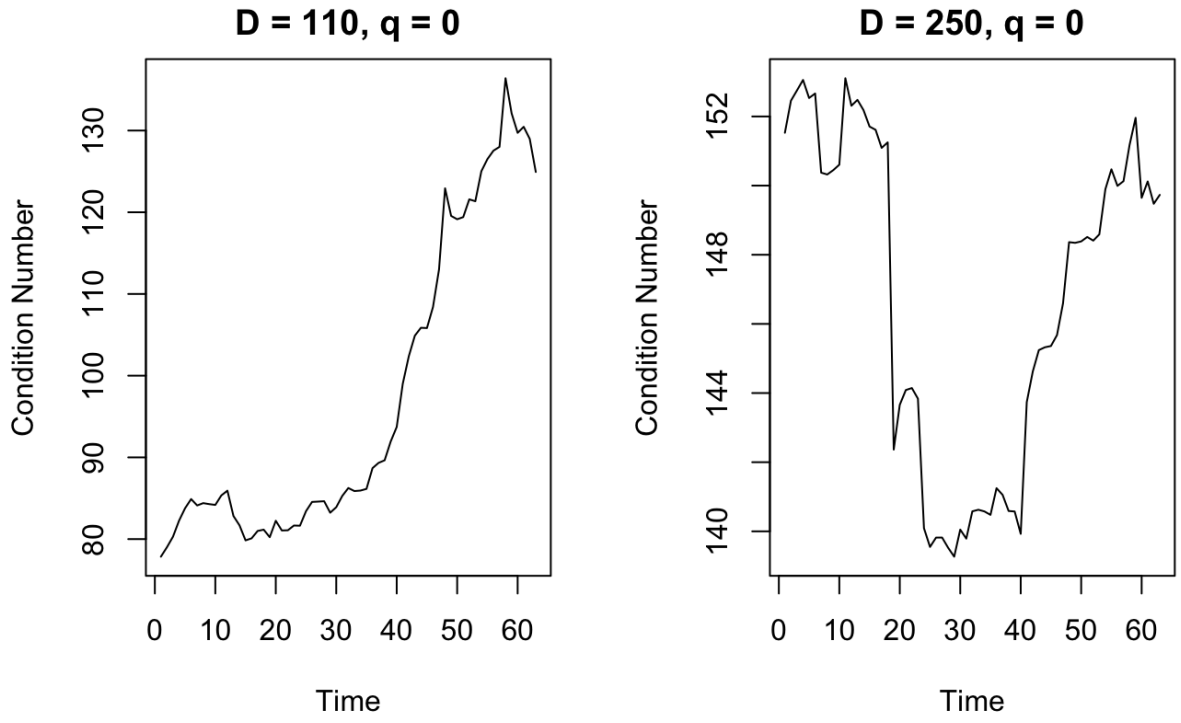


(Figure 4.1 Sharpe ratio for different window lengths at lag q = 0)

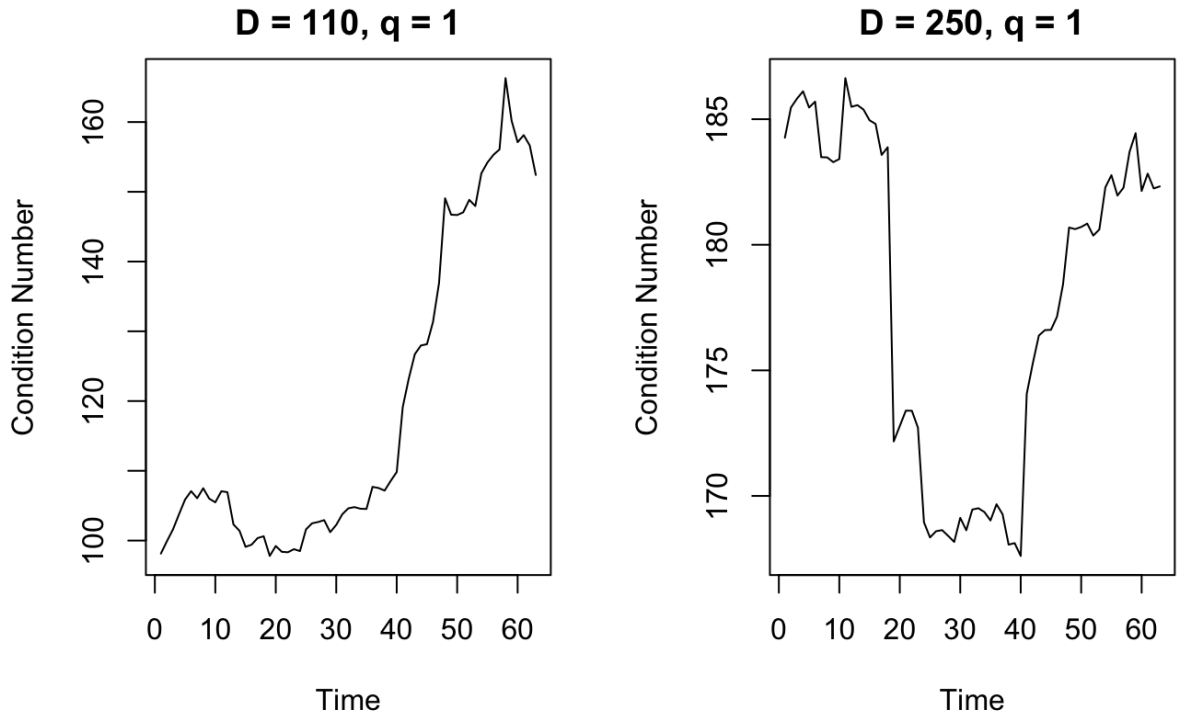


(Figure 4.2 Sharpe ratio for different window lengths at lag q = 1)

The patterns of sharpe ratios for train, validation and test sets for different window lengths at lag q =0 and lag q = 1 seems pretty similar overall. It is expected that sharpe ratios increase with window length. For both lag q =0 and lag q = 1, the sharpe ratio for training sets seems to increase with window length when window length is around 100 to 250 but decreases from window length of 50 to 100. The sharpe ratio of validation and test sets in both cases are at times negative. The trading strategy , $\hat{Y}_{t+1} = \hat{\alpha}^\top \mathbf{x}_t$ deriving from ordinary least squares perhaps fails.



(Figure 4.3 Condition numbers at different time points for $X^T X/10$ for the test set at lag q = 0)



(Figure 4.4 Condition numbers at different time points for $X^T X/10$ for the test set at lag q = 1)

The covariates in the linear regression might be highly correlated with each other, as indicated by high condition numbers in Figure 4.3 and Figure 4.4. To further illustrate this point, the prices of the stocks seem to be quite correlated with each other in Figure 1. Multicollinearity causes the gram matrix, $X^T X$ to have small eigenvalues and $(X^T X)^{-1}$ to have enlarged eigenvalues. $\hat{\alpha}$ estimated by ordinary least squares would be large since $\hat{\alpha} = \hat{\alpha}_D(t) = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{Y}$. The predicted return, $\hat{Y}_{t+1}$, also known as the investment strategy where $\hat{Y}_{t+1} = \hat{\alpha}^\top \mathbf{x}_t$ would be large. When $\hat{Y}_{t+1}$ is strongly negative and strongly positive, we might short and long at ridiculous positions. Using ordinary least squares over training, validation and test sets seem to be inappropriate or could be improved in this case.

Question 5

To improve upon ordinary least squares, reduce multicollinearity by subsetting selected covariates in regression via marginal correlation screening. If absolute value of

correlation of normalized return for S&P 500 at time $t+1, \boldsymbol{Y}_{t+1}$ and $\boldsymbol{X}_t^k$ is greater than θ, include $\boldsymbol{X}_t^k$ as a covariate.

$|corr(\boldsymbol{Y}_{t+1}, \boldsymbol{X}_t^k)| > \theta$, where θ is a threshold to be selected

sim.grid function in R (refer to appendix) considers which θ is the best via marginal correlation screening to obtain the best sharpe ratio.

Lag = 0 for training set. Maximum sharpe ratio is 0.70307753.Hence, θ = 0.03

```
sim.grid(data0, lambda_all$best, 250)
```

```
##   [1]  0.66660324  0.69582817  0.57682497  0.70307753  0.50102145  0.56287899
##   [7]  0.57399063  0.05324117 -0.14765919 -1.02692629 -1.04008981 -0.72275993
##  [13] -0.86998499 -0.71807014 -0.43907072 -0.27299139 -0.02628937 -0.35978742
##  [19] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [25] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [31] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [37] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [43] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [49] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [55] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [61] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [67] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [73] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [79] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [85] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [91] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [97] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
```

(Table 5.1)

Lag = 0 for validation set. Maximum sharpe ratio is 1.42381875.Hence, θ = 0.06

```
sim.grid.valid(data0,lambda_all$best, 250)
```

```
##   [1] -1.97222595 -1.56634634 -1.82888718 -1.59719062 -1.18794458  0.16939087
##   [7]  1.42381875  0.88381310  1.01955958  0.68710569  0.56933834 -0.13015657
##  [13] -0.19474129 -0.21016345 -0.01681136  1.49713662  1.49713662  1.49713662
##  [19]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [25]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [31]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [37]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [43]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [49]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [55]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [61]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [67]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [73]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [79]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [85]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [91]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [97]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
```

(Table 5.2)

Lag = 1 for training set.Maximum sharpe ratio is 0.906319535. Hence, θ = 0.03

```
sim.grid(data1, lambda_all$best, 250)
```

```
##   [1]  0.575746964  0.723257217  0.733794719  0.906319535  0.725371117
##   [6]  0.664724971  0.653803098  0.002847738 -0.106109544 -1.160033747
##  [11] -1.099428313 -0.839110082 -1.132239883 -0.852494887 -0.618992384
##  [16] -0.414557170 -0.028460606 -0.361768794 -0.212428338 -0.212428338
##  [21] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [26] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [31] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [36] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [41] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [46] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [51] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [56] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [61] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [66] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [71] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [76] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [81] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [86] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [91] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [96] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
## [101] -0.212428338
```

(Table 5.3)

Lag = 1 for validation set.Maximum sharpe ratio is 0.948576244 . Hence, θ = 0.07

```
sim.grid.valid(data1,lambda_all$best, 250)
```

```
##   [1] -1.876106210 -2.093384374 -1.330324104 -1.626780481 -1.075438935
##   [6]  0.463818771  1.258647368  0.948576244  1.134035164  0.510018653
##  [11]  0.253977082  0.906454290 -0.185658760 -0.199984830 -0.006066917
##  [16]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [21]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [26]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [31]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [36]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [41]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [46]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [51]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [56]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [61]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [66]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [71]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [76]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [81]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [86]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [91]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [96]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
## [101]  1.497136619
```

(Table 5.4)

For both training and validation set at lag q= 0 and q = 1, marginal correlation screening seems to be a good way as the sharpe ratio is quite high, ranging from 0.70307753 to 1.42381875. It seems like this marginal correlation screening is better than just using ordinary least squares.

Appendix

ST326 Coursework 2022

2022-12-06

```
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: TTR
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tseries)
library(RColorBrewer)
```

Question 1

The selected 10 stocks from top 100 constituents of S&P500 by index weight as of 10th October 2022 are as follows:

1. Apple Inc. (APPL)
2. Microsoft Corp (MSFT)
3. Amazon.com Inc (AMZN)
4. Alphabet Inc A (GOOGL)

5. Alphabet Inc C (GOOG)
6. Tesla, Inc (TSLA)
7. Unitedhealth Group Inc (UNH)
8. Johnson & Johnson (JNJ)
9. Exxon Mobil Corp (XOM)
10. JPMorgan Chase (JPM)

Download data for 10 stocks using `quantmod` package from 2017 to 2021.

```
selected_stocks = c('AAPL','MSFT','AMZN','GOOGL',
                    'JPM','GOOG','TSLA','UNH','JNJ','XOM')

for (i in 1:length(selected_stocks)){
    tmp_ = selected_stocks[i]
    print(sprintf('Downloading %s stocks... ', tmp_))
    getSymbols(tmp_)
    exec0 = sprintf('%s = %s["2017::2021"]', tmp_, tmp_)
    eval(parse(text=exec0))
    exec1 = sprintf('%s = as.data.frame(%s)', tmp_, tmp_)
    eval(parse(text=exec1))
    exec2 = sprintf('%s.Timestamp = as.numeric(as.Date(rownames(%s)))', tmp_, tmp_)
    eval(parse(text=exec2))
    exec3 = sprintf('%s = cbind(%s.Timestamp, %s)',
                    tmp_, tmp_, tmp_)
    eval(parse(text=exec3))
    exec4 = sprintf('write.table(%s, "%s.txt", row.names=FALSE)', tmp_, tmp_)
    eval(parse(text=exec4))
}
```

```
## [1] "Downloading AAPL stocks... "
## [1] "Downloading MSFT stocks... "
## [1] "Downloading AMZN stocks... "
## [1] "Downloading GOOGL stocks... "
## [1] "Downloading JPM stocks... "
## [1] "Downloading GOOG stocks... "
## [1] "Downloading TSLA stocks... "
## [1] "Downloading UNH stocks... "
## [1] "Downloading JNJ stocks... "
## [1] "Downloading XOM stocks... "
```

Download data for S&P 500 using `quantmod` package from 2017 to 2021.

```
getSymbols('^GSPC')
```

```
## [1] "^GSPC"
```

```
GSPC = GSPC["2017::2021"]
GSPC <- as.data.frame(GSPC)
GSPC.Timestamp <- as.numeric(as.Date(rownames(GSPC)))
GSPC <- cbind(GSPC.Timestamp, GSPC)
write.table(GSPC, 'GSPC.txt', row.names=FALSE)
```

Export the downloaded data to text files.

```
read.bossa.data <- function(vec.names) {

    p <- length(vec.names)

    n1 <- 20000
    dates <- matrix(99999999, p, n1)
    closes <- matrix(0, p, n1)
    max.n2 <- 0

    for (i in 1:p) {
      filename <- paste("",vec.names[i], ".txt", sep="")
      tmp <- scan(filename, list(date=numeric(), NULL, NULL, NULL, close=numeric(), NULL, NULL), skip=1
      n2 <- length(tmp$date)
      max.n2 <- max(n2, max.n2)
      dates[i,1:n2] <- tmp$date
      closes[i,1:n2] <- tmp$close
    }

    dates <- dates[,1:max.n2]
    closes <- closes[,1:max.n2]

    days <- rep(0, n1)
    arranged.closes <- matrix(0, p, n1)
    date.indices <- starting.indices <- rep(1, p)
    already.started <- rep(0, p)
    day <- 1

    while(max(date.indices) <= max.n2) {
      current.dates <- current.closes <- rep(0, p)
      for (i in 1:p) {
        current.dates[i] <- dates[i,date.indices[i]]
        current.closes[i] <- closes[i,date.indices[i]]
      }

      min.indices <- which(current.dates == min(current.dates))
      days[day] <- current.dates[min.indices[1]]

      arranged.closes[min.indices,day] <- log(current.closes[min.indices])
      arranged.closes[-min.indices,day] <- arranged.closes[-min.indices, max(day-1, 1)]
      already.started[min.indices] <- 1
      starting.indices[-which(already.started == 1)] <- starting.indices[-which(already.started == 1)] +
      day <- day + 1
      date.indices[min.indices] <- date.indices[min.indices] + 1
    }

    days <- days[1:(day-1)]
    arranged.closes <- arranged.closes[,1:(day-1)]
    max.st.ind <- max(starting.indices)
    r <- matrix(0, p, (day-max.st.ind-1))

    for (i in 1:p) {
      r[i,] <- diff(arranged.closes[i,max.st.ind:(day-1)])
```

```
      r[i,] <- r[i,] / sqrt(var(r[i,]))
      r[i,r[i,]==0] <- rnorm(sum(r[i,]==0))
    }

    return(list(dates=dates, closes=closes, days=days, arranged.closes=arranged.closes, starting.indices
  }
```

Names of the selected stocks and S&P 500

```
index_and_stocks <- c('GSPC',selected_stocks)
print(index_and_stocks)
```

```
##  [1] "GSPC"  "AAPL"  "MSFT"  "AMZN"  "GOOGL" "JPM"   "GOOG"  "TSLA"  "UNH"
## [10] "JNJ"   "XOM"
```

Store all downloaded data to `ind` . Extract `date` and `logprice` from `ind`

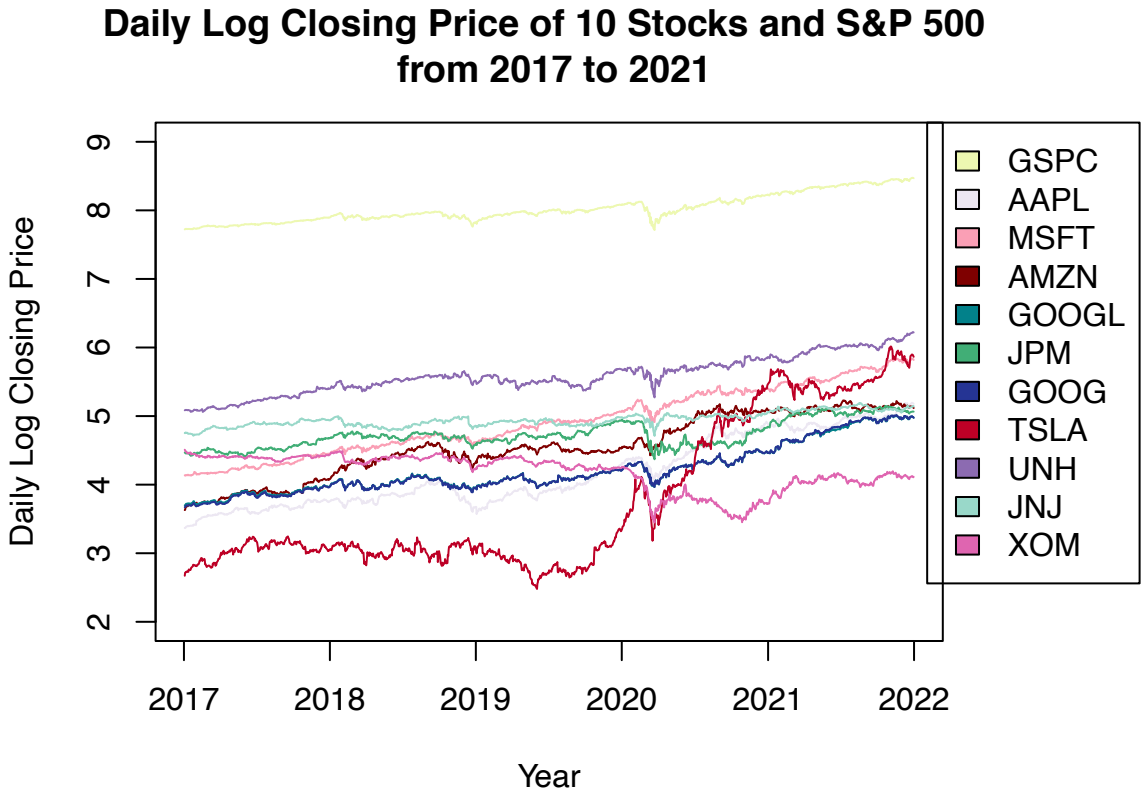
```
ind <- read.bossa.data(index_and_stocks)
date <- ind$date
logprice <- log(ind$close)
```

Generate random colours for 1 index and 10 stocks

```
set.seed(10)
n <- length(index_and_stocks)
qual_col_pals = brewer.pal.info[brewer.pal.info$category == 'seq',]
col_vector = unlist(mapply(brewer.pal, qual_col_pals$maxcolors, rownames(qual_col_pals)))
colors = sample(col_vector, n)
```

Plot the daily log closing price of top 10 stocks and S&P 500 from 2017 to 2021

```
par(mar=c(5, 4, 4, 8), xpd=TRUE)

plot(as.Date(date[1,]), logprice[1,],
    main="Daily Log Closing Price of 10 Stocks and S&P 500 \n from 2017 to 2021",
    ylab="Daily Log Closing Price",
    xlab="Year",
    ylim=c(2,9),
    xlim=c(date[1,1],date[1,dim(date)[2]]),
    type="l",
    col=colors[1])
for (i in 2:length(index_and_stocks)){
    lines(date[i,], logprice[i,], col=colors[i])
}
legend("topright", inset = c(-0.25, 0), index_and_stocks, fill=colors)
```



Question 2

Prepare the data for prediction of S&P 500 at time $t+1$. Include the maximum S&P 500 lag. Split them into a train, validation and test sets. To include the selected stocks, input 1 in mask.

```
pred.snp.prepare <- function(max.lag = 5, split = c(50, 25), mask = rep(1, 10)) {

    ind <- read.bossa.data(index_and_stocks)
    d <- dim(ind$r)

    start.index <- max(3, max.lag + 1)

    y <- matrix(0, d[2] - start.index + 1, 1)

    x <- matrix(0, d[2] - start.index + 1, d[1] - 1 + max.lag)

    y[,1] <- ind$r[1,start.index:d[2]]

    for (i in 1:max.lag) {
        x[,i] <- ind$r[1,(start.index-i):(d[2]-i)]
    }

    shift.indices <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

    for (i in 2:(d[1])) {
```

```
        x[,i+max.lag-1] <- ind$r[i,(start.index-1-shift.indices[i-1]):(d[2]-1-shift.indices[i-1])]
    }

    end.training <- round(split[1] / 100 * d[2])
    end.validation <- round(sum(split[1:2]) / 100 * d[2])
    x <- x[,as.logical(c(rep(1, max.lag), mask))]

    y.train <- as.matrix(y[1:end.training], end.training, 1)
    x.train <- x[1:end.training,]

    y.valid <- as.matrix(y[(end.training+1):(end.validation)], end.validation-end.training, 1)
    x.valid <- x[(end.training+1):(end.validation),]

    y.test <- as.matrix(y[(end.validation+1):(d[2] - start.index + 1)], d[2]-start.index-end.validation
    x.test <- x[(end.validation+1):(d[2] - start.index + 1),]

    list(x=x, y=y, x.train=x.train, y.train=y.train, x.valid=x.valid, y.valid=y.valid, x.test=x.test, y

}
```

Compute the volatility for each covariates and the response in the training set.

```
first.acf.squares.train <- function(x, lambda) {

# x is an object returned by "pred.snp.prepare"
# this function computes the volatility for each covariate and the response in the training part
# it then computes the acfs of the squared residuals after removing the volatility, and adds up
# the first acfs for each covariate and response
# the point is to choose lambda so that as much as possible of the acf has been removed

  d <- dim(x$x.train)

  ss <- 0

  x.train.dev <- x$x.train
  y.train.dev <- x$y.train

  x.valid.dev <- x$x.valid
  y.valid.dev <- x$y.valid

  x.test.dev <- x$x.test
  y.test.dev <- x$y.test


  for (i in 1:(d[2])) {
    v <- vol.exp.sm(x$x.train[,i], lambda)
    ss <- ss + abs(acf(v$sq.resid, plot=FALSE)$acf[2])
    x.train.dev[,i] <- v$resid

    v <- vol.exp.sm(x$x.valid[,i], lambda)
    x.valid.dev[,i] <- v$resid

    v <- vol.exp.sm(x$x.test[,i], lambda)
```

```
    x.test.dev[,i] <- v$resid

  }

  v <- vol.exp.sm(x$y.train, lambda)
  ss <- ss + abs(acf(v$sq.resid, plot=FALSE)$acf[2])
  y.train.dev <- v$resid

  v <- vol.exp.sm(x$y.valid, lambda)
  y.valid.dev <- v$resid

  v <- vol.exp.sm(x$y.test, lambda)
  y.test.dev <- v$resid

  list(ss=ss, y.train.dev=y.train.dev, x.train.dev=x.train.dev, y.valid.dev=y.valid.dev, x.valid.dev=x.v

}
```

Exponential smoothing of x^2 with parameter lambda

```
vol.exp.sm <- function(x, lambda) {

    sigma2 <- x^2
    n <- length(x)

    for (i in 2:n)
        sigma2[i] <- sigma2[i-1] * lambda + x[i-1]^2 * (1-lambda)

    sigma <- sqrt(sigma2)

    resid <- x/sigma
    resid[is.na(resid)] <- 0
    sq.resid <- resid^2

    list(sigma2=sigma2, sigma=sigma, resid = resid, sq.resid = sq.resid)

}
```

Find the optimal lambda that minimises residual sum of squares

```
try_lambda <- function(data) {
    lambda_x = seq(0.01,0.99,0.01)
    lambda_ss = c()
    for (j in 1:length(lambda_x)){
        tmp_ss = first.acf.squares.train(data, lambda_x[j])$ss
        lambda_ss = c(lambda_ss,tmp_ss)
    }
    print(sprintf('lambda: %s, min ss: %s',
        lambda_x[which.min(lambda_ss)],min(lambda_ss)))
    list(xs=lambda_x, sss=lambda_ss, best=lambda_x[which.min(lambda_ss)])
}
```

```
set.seed(4)
data <- pred.snp.prepare(max.lag=5)
lambda_all = try_lambda(data)
```

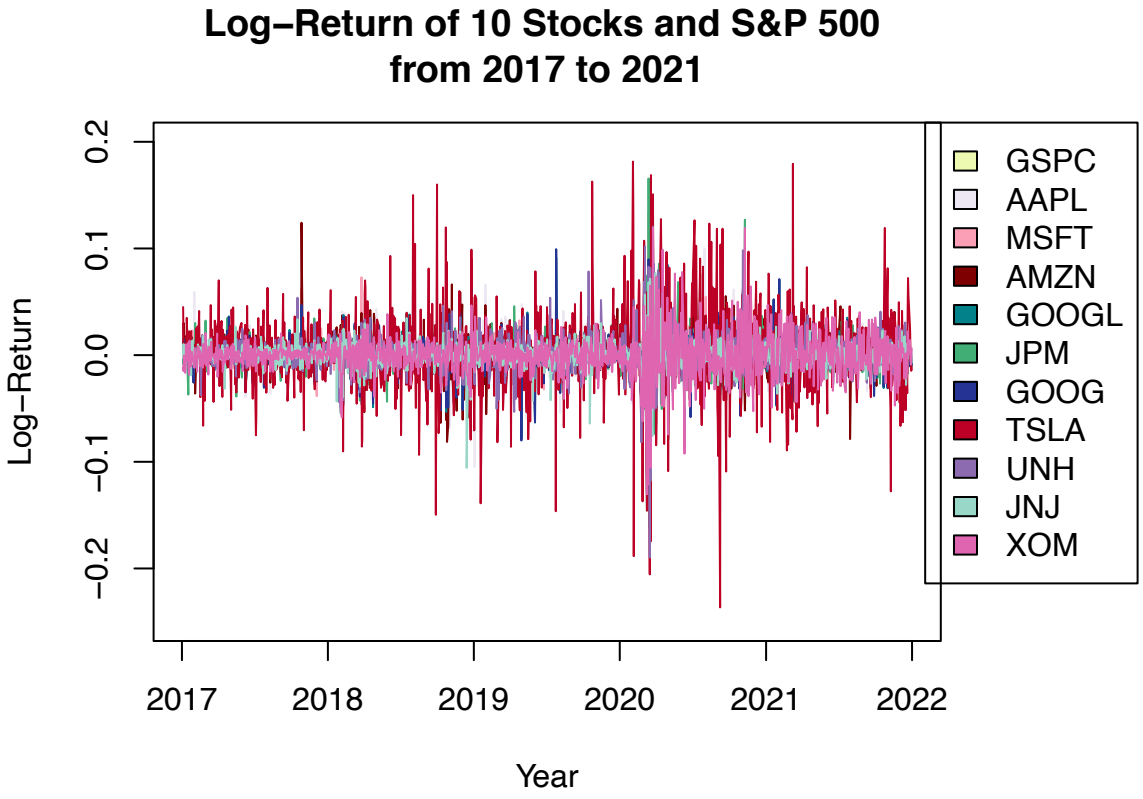
```
## [1] "lambda: 0.87, min ss: 0.237343177028872"
```

Find log-return by using `diff()` function on log-price.

```
logprice <- as.data.frame(ind$arranged.closes) # logprice # 11*1259
logreturn <- as.data.frame(t(diff(t(logprice)))) # return  # 11*1258
```

Plot the log-return of top 10 stocks and S&P 500 from 2017 to 2021

```
par(mar=c(5, 4, 4, 8), xpd=TRUE)

plot(as.Date(date[1,2:dim(date)[2]]),logreturn[1,],
    main="Log-Return of 10 Stocks and S&P 500 \nfrom 2017 to 2021",
    ylab="Log-Return",
    xlab="Year",
    ylim=c(-0.25,0.2),
    xlim=c(date[1,1],date[1,dim(date)[2]]),
    type="l",
    col=colors[1])
for (i in 2:length(index_and_stocks)){
    lines(date[1,2:dim(date)[2]], logreturn[i,], col=colors[i])
}

legend("topright", inset = c(-0.25, 0), index_and_stocks, fill=colors)
```



Get the estimated daily volatility of S&P 500 and 10 stocks

```
vol_GSPC = vol.exp.sm(logreturn[1,],lambda_all$best)
vol_AAPL = vol.exp.sm(logreturn[2,],lambda_all$best)
vol_MSFT = vol.exp.sm(logreturn[3,],lambda_all$best)
vol_AMZN = vol.exp.sm(logreturn[4,],lambda_all$best)
vol_GOOGL = vol.exp.sm(logreturn[5,],lambda_all$best)
vol_JPM = vol.exp.sm(logreturn[6,],lambda_all$best)
vol_GOOG = vol.exp.sm(logreturn[7,],lambda_all$best)
vol_TSLA = vol.exp.sm(logreturn[8,],lambda_all$best)
vol_UNH = vol.exp.sm(logreturn[9,],lambda_all$best)
vol_JNJ = vol.exp.sm(logreturn[10,],lambda_all$best)
vol_XOM = vol.exp.sm(logreturn[11,],lambda_all$best)
```

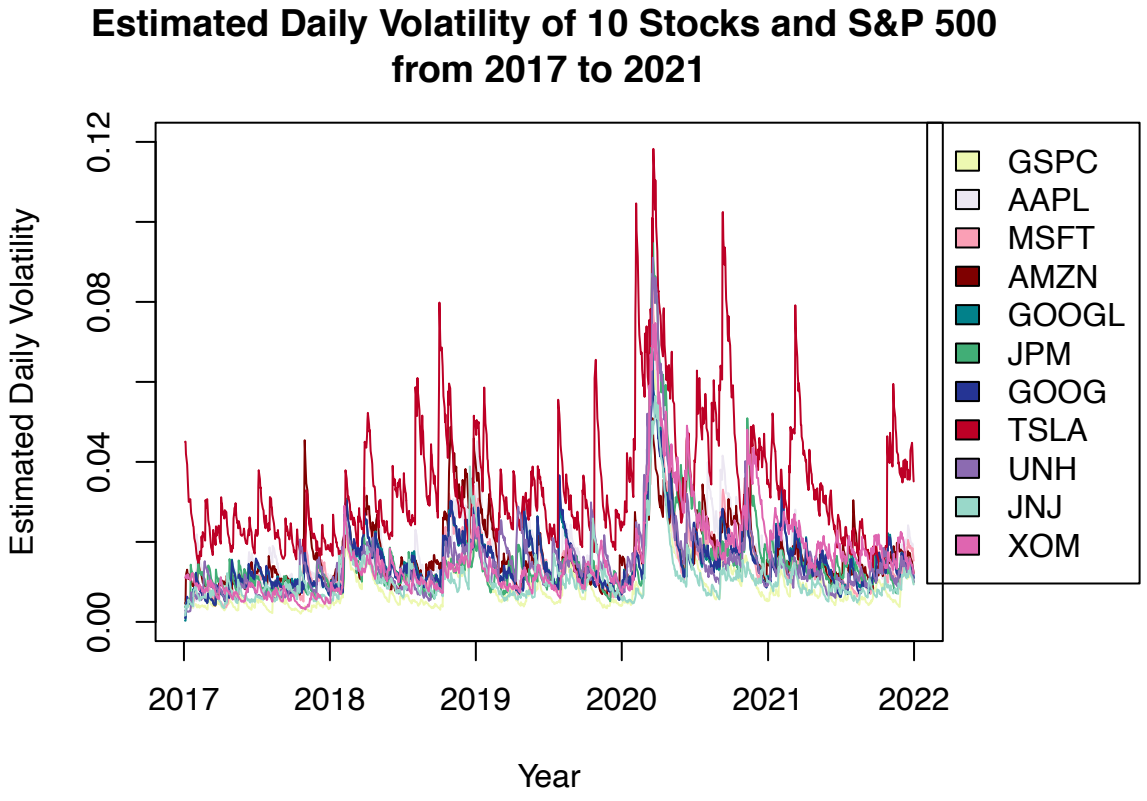
Estimated daily volatility of S&P 500 and 10 stocks

```
volatiliy <- rbind(vol_GSPC$sigma,vol_AAPL$sigma,vol_MSFT$sigma,vol_AMZN$sigma,vol_GOOGL$sigma,
                   vol_JPM$sigma,vol_GOOG$sigma,vol_TSLA$sigma,vol_UNH$sigma,vol_JNJ$sigma,vol_XOM$sigma
dim(volatiliy)
```

```
## [1]   11 1258
```

Plot the estimated daily volatility of top 10 Stocks and S&P 500 from 2017 to 2021

```
par(mar=c(5, 4, 4, 8), xpd=TRUE)

plot(as.Date(date[1,2:dim(date)[2]]),volatiliy[1,],
    main="Estimated Daily Volatility of 10 Stocks and S&P 500 \nfrom 2017 to 2021",
    ylab="Estimated Daily Volatility",
    xlab="Year",
    ylim=c(0,0.12),
    xlim=c(date[1,1],date[1,dim(date)[2]]),
    type="l",
    col=colors[1])
for (i in 2:length(index_and_stocks)){
    lines(date[1,2:dim(date)[2]], volatiliy[i,], col=colors[i])
}

legend("topright", inset = c(-0.25, 0), index_and_stocks, fill=colors)
```



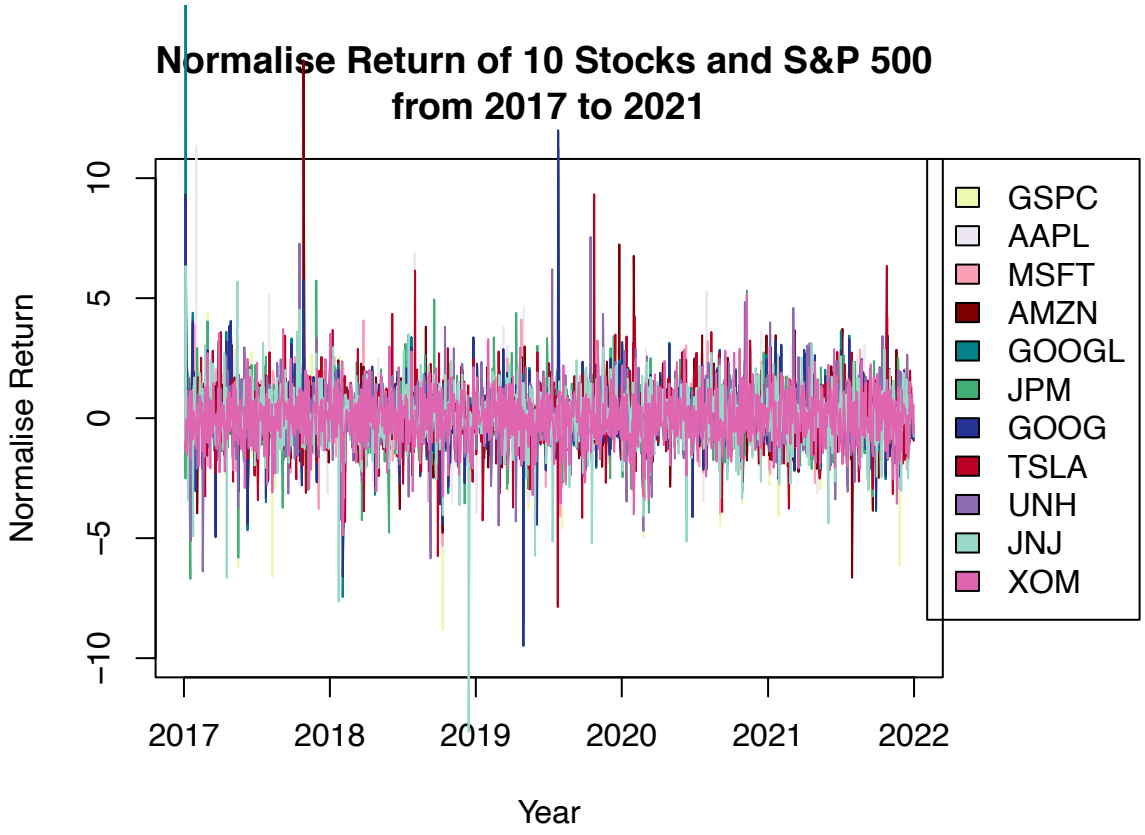
Normalized the returns

```
norm_return <- logreturn/volatiliy
dim(norm_return)
```

```
## [1]   11 1258
```

Plot the normalise return of top 10 stocks and S&P 500 from 2017 to 2021

```r
par(mar=c(5, 4, 4, 8), xpd=TRUE)

plot(as.Date(date[1,2:dim(date)[2]]),norm_return[1,],
    main="Normalise Return of 10 Stocks and S&P 500 \nfrom 2017 to 2021",
    ylab="Normalise Return",
    xlab="Year",
    ylim=c(-10,10),
    xlim=c(date[1,1],date[1,dim(date)[2]]),
    type="l",
    col=colors[1])
for (i in 2:length(index_and_stocks)){
    lines(date[1,2:dim(date)[2]], norm_return[i,], col=colors[i])
}

legend("topright", inset = c(-0.25, 0), index_and_stocks, fill=colors)
```



Create a data frame, `norm_return` with date as column name and index and stocks as row names.

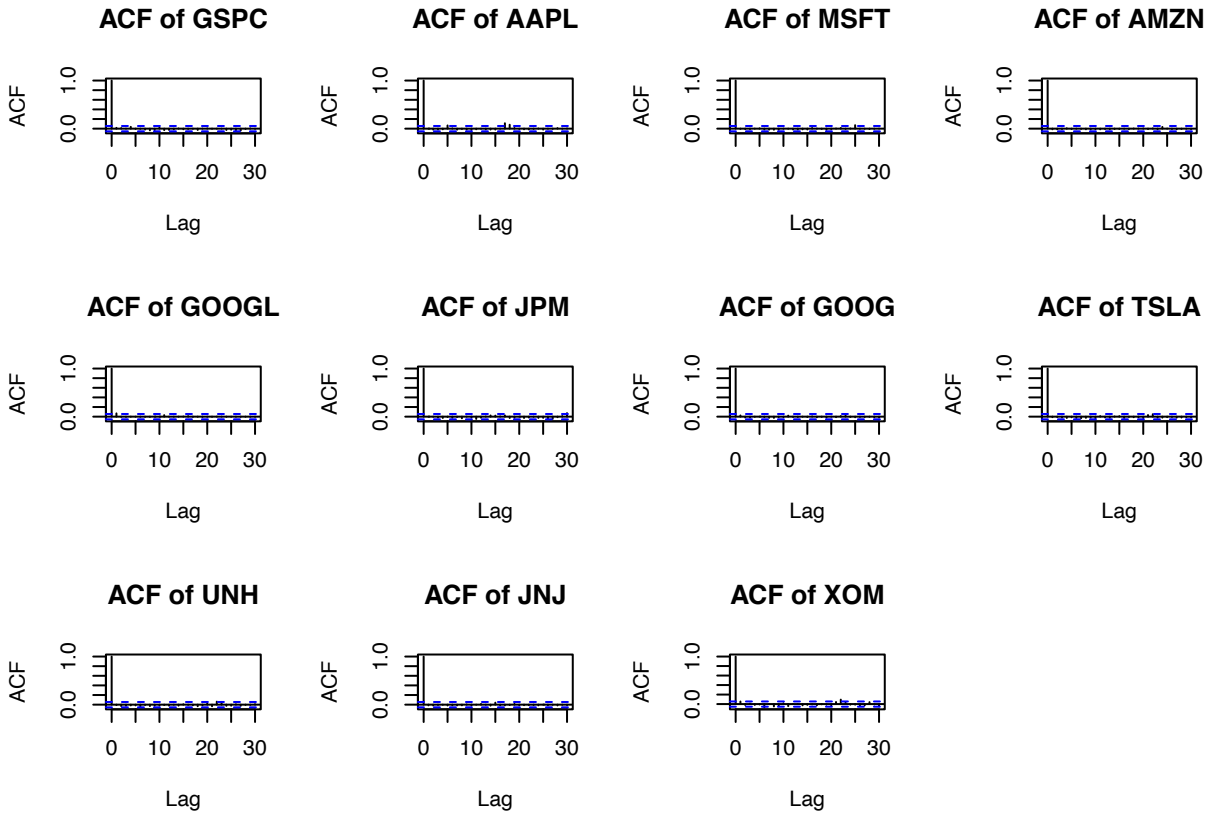
```r
names(norm_return) <- as.Date(date[1,2:dim(date)[2]])
rownames(norm_return) <- index_and_stocks
```

Square the normalized returns

```r
norm_return_sq <- norm_return^2
```

Plot the ACF of the squared normalized returns of S&P 500 and selected 10 stocks.

```
par(mfrow=c(3,4))
for (i in 1:dim(norm_return_sq)[1]){
  acf(ts(data = t(norm_return_sq[i,]), start = c(2017, 1, 3)),
    main= paste('ACF of',colnames(ts(data = t(norm_return_sq[i,]), start = c(2017, 1, 3))), sep = ' '))
}
```



Question 4

```
thresh.reg <- function(x, y, th, x.pred = NULL) {

  # estimation of alpha in y = w + x alpha + epsilon (linear regression)
  # but only using those covariates in x whose marginal correlation
  # with y exceeds th
  # use th = 0 for full regression
  # note the intercept is added
  # x.pred is a new x for which we wish to make prediction

  d <- dim(x)

  ind <- (abs(cor(x, y)) > th)
  n <- sum(ind)

  new.x <- matrix(c(rep(1, d[1]), x[,ind]), d[1], n+1)    ## Adding intercept term
```

```
  gram = t(new.x) %*% new.x

  alpha <- solve(gram) %*% t(new.x) %*% matrix(y, d[1], 1)

  ind.ex <- c(1, as.numeric(ind))

  ind.ex[ind.ex == 1] <- alpha

  condnum = max(svd(gram)$d)/min(svd(gram)$d)

  pr <- 0

  if (!is.null(x.pred)) pr <- sum(ind.ex * c(1, x.pred))

  list(alpha = ind.ex, pr=pr, condnum = condnum)

}


Lasso.reg <- function(x, y, th = -1, x.pred = NULL) {

  # estimation of alpha in y = a + x alpha + epsilon (linear regression) with L1 penalization
  # Penalty is not specified by delta in lecture notes,
  # but the ratio sum of absolute coefficients/max sum of absolute coefficients.
  # th = ratio specified above, has a one-one correspondence with delta.
  # Use th outside [0,1] if want to use 10-fold CV chosen one.
  # Note the intercept is not here since lars will add it by default
  # x.pred is a new x for which we wish to make prediction

  d <- dim(x)
  ratio <- rep(0,d[2]+1)

  fit.lasso <- lars(x, y, normalize = FALSE)  ## Not normalizing individual covariates

  if (th<0 || th>1) { th <- (which.min(cv.lars(x,y, index=seq(from=0, to=1, length=101), plot.it=FALSE)$

  ## Calculate the different ratios at which certain coefficient is exactly shrunk to 0 ##

  maxnorm = sum(abs(fit.lasso$alpha[d[2]+1,]))
  done = 0; i=2
  while (done==0){
    ratio[i] <- sum(abs(fit.lasso$alpha[i,]))/maxnorm
    if (th <= ratio[i] && done == 0) {
      alpha = fit.lasso$alpha[i-1,] + (fit.lasso$alpha[i,] - fit.lasso$alpha[i-1,])*(th-ratio[i-1])/(ra
      done = 1
    }
    i <- i+1
  }

  ## Still need to calculate the intercept ##
  a0 = mean( y - x%*%alpha )

  alpha = c(a0, alpha)
```

```
  pr <- 0

  if (!is.null(x.pred)) pr <- sum(alpha * c(1, x.pred))

  condnum = -1  ## Not really matter here, just conform with thresh.reg outputs

  list(alpha = alpha, pr=pr, th=th, condnum=condnum)

}




sharpe.curves <- function(x, lambda, th, warmup, reg.function = thresh.reg, win = seq(from = 10, to = wa

  # computes Sharpe ratios for a sequence of rolling windows (D in the lecture notes)
  # for the training, validation and test sets

  w <- length(win)

  train.curve <- valid.curve <- test.curve <- rep(0, w)

  n <- length(x$y.train)

  i=1
  rreg <- rolling.thresh.reg(x, lambda, th, win[i], warmup, reg.function)
  rreg.valid <- rolling.thresh.reg.valid(x, lambda, th, win[i], warmup, reg.function)
  rreg.test <- rolling.thresh.reg.test(x, lambda, th, win[i], warmup, reg.function)

  condnum = matrix(0,w,length(rreg$condnum))
  condnum.valid = matrix(0,w,length(rreg.valid$condnum))
  condnum.test = matrix(0,w,length(rreg.test$condnum))

  train.curve[i] <- rreg$err
  valid.curve[i] <- rreg.valid$err
  test.curve[i] <- rreg.test$err

  condnum[i,] <- rreg$condnum
  condnum.valid[i,] <- rreg.valid$condnum
  condnum.test[i,] <- rreg.test$condnum

  for (i in 2:w) {
    rreg <- rolling.thresh.reg(x, lambda, th, win[i], warmup, reg.function)
    rreg.valid <- rolling.thresh.reg.valid(x, lambda, th, win[i], warmup, reg.function)
    rreg.test <- rolling.thresh.reg.test(x, lambda, th, win[i], warmup, reg.function)

    train.curve[i] <- rreg$err
    valid.curve[i] <- rreg.valid$err
    test.curve[i] <- rreg.test$err

    condnum[i,] <- rreg$condnum
    condnum.valid[i,] <- rreg.valid$condnum
```

```
    condnum.test[i,] <- rreg.test$condnum
  }

  list(train.curve = train.curve, valid.curve = valid.curve, test.curve = test.curve, condnum=condnum,
}




rolling.thresh.reg <- function(x, lambda, th, win, warmup, reg.function = thresh.reg) {

  # performs prediction over a rolling window of size win
  # over the training set
  # x - returned by pred.footsie.prepare
  # lambda - parameter for exponential smoothing
  # th - threshold for thresh.reg
  # warmup - t_0 from the lecture notes


  xx <- first.acf.squares.train(x, lambda)

  n <- length(xx$y.train.dev)

  err <- 0

  condnum <- predi <- truth <- rep(0, n-warmup+1)

  for (i in warmup:n) {

    y <- xx$y.train.dev[(i-win):(i-1)]
    xxx <- xx$x.train.dev[(i-win):(i-1),]

    zz <- reg.function(xxx, y, th, xx$x.train.dev[i,])

    predi[i-warmup+1] <- zz$pr
    condnum[i-warmup+1] <- zz$condnum
    truth[i-warmup+1] <- xx$y.train.dev[i]

  }

  ret <- predi * truth

  err <- sqrt(250) * mean(ret) / sqrt(var(ret))

  list(err=err, predi=predi, truth=truth, condnum=condnum)

}


rolling.thresh.reg.valid <- function(x, lambda, th, win, warmup, reg.function = thresh.reg) {

  # The same as the previous function but for the validation set
```

```
  xx <- first.acf.squares.train(x, lambda)

  n <- length(xx$y.valid.dev)

  err <- 0

  condnum <- predi <- truth <- rep(0, n-warmup+1)

  for (i in warmup:n) {

    y <- xx$y.valid.dev[(i-win):(i-1)]
    xxx <- xx$x.valid.dev[(i-win):(i-1),]

    zz <- reg.function(xxx, y, th, xx$x.valid.dev[i,])

    predi[i-warmup+1] <- zz$pr
    condnum[i-warmup+1] <- zz$condnum
    truth[i-warmup+1] <- xx$y.valid.dev[i]

  }


  ret <- predi * truth

  err <- sqrt(250) * mean(ret) / sqrt(var(ret))

  list(err=err, predi=predi, truth=truth, condnum=condnum)

}


rolling.thresh.reg.test <- function(x, lambda, th, win, warmup, reg.function = thresh.reg) {

  # The same as the previous function but for the test set

  xx <- first.acf.squares.train(x, lambda)

  n <- length(xx$y.test.dev)

  err <- 0

  condnum <- predi <- truth <- rep(0, n-warmup+1)

  for (i in warmup:n) {

    y <- xx$y.test.dev[(i-win):(i-1)]
    xxx <- xx$x.test.dev[(i-win):(i-1),]
    zz <- reg.function(xxx, y, th, xx$x.test.dev[i,])

    predi[i-warmup+1] <- zz$pr
    condnum[i-warmup+1] <- zz$condnum
    truth[i-warmup+1] <- xx$y.test.dev[i]
```

```
  }

  ret <- predi * truth

  err <- sqrt(250) * mean(ret) / sqrt(var(ret))

  list(err=err, predi=predi, truth=truth, condnum=condnum)
}

sim.grid <- function(x, lambda, win, warmup = 250, reg.function = thresh.reg, th.grid = seq(from = 0, t
  # Which threshold th best over training set?

  tt <- length(th.grid)

  res <- rep(0, tt)

  for (i in 1:tt) res[i] <- rolling.thresh.reg(x, lambda, th.grid[i], win, warmup, reg.function)$err

  res
}

sim.grid.valid <- function(x, lambda, win, warmup = 250, reg.function = thresh.reg, th.grid = seq(from =
  # The same over the validation set.

  tt <- length(th.grid)

  res <- rep(0, tt)

  for (i in 1:tt) res[i] <- rolling.thresh.reg.valid(x, lambda, th.grid[i], win, warmup, reg.function)$

  res
}
```

Sharpe ratios for different window lengths at lag q = 0

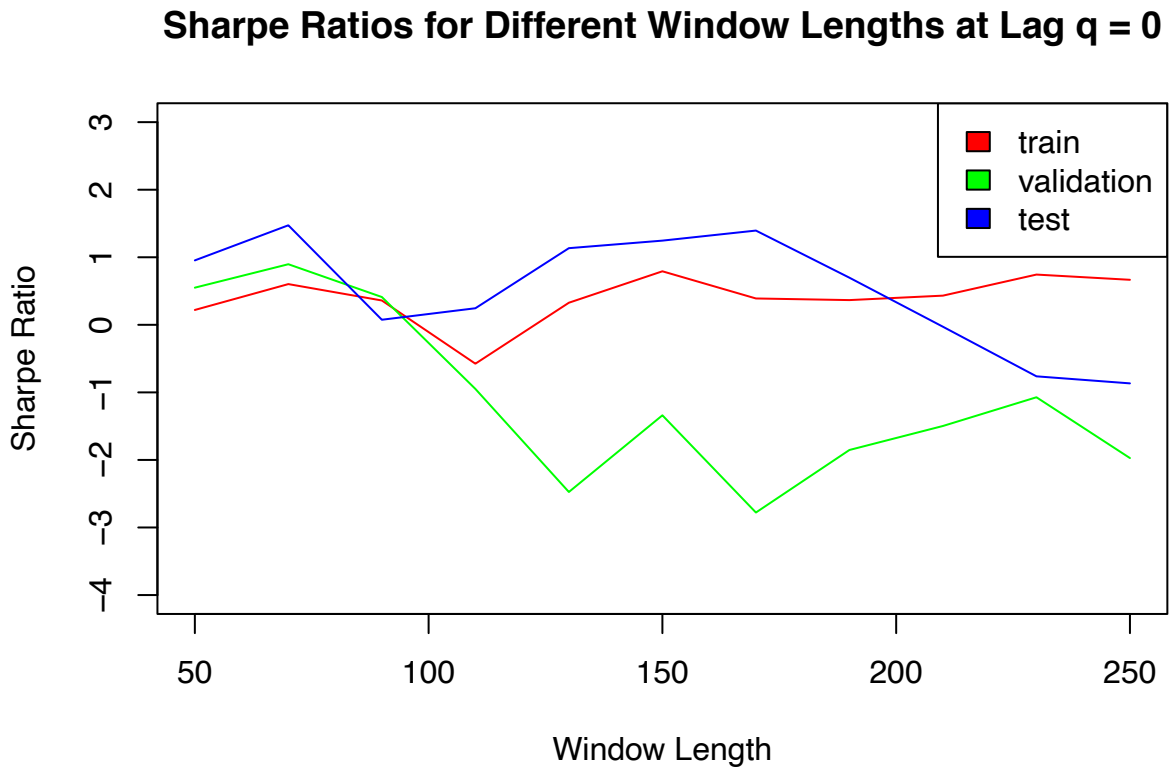
```
q0 <- 0
win <- seq(from = 50, to= 250, by = 20)
data0 <- pred.snp.prepare(q0)
sc0 <- sharpe.curves(data0, lambda_all$best, 0, 250, win =win)

length(sc0$train.curve)
```

```
## [1] 11
```

```
win
```

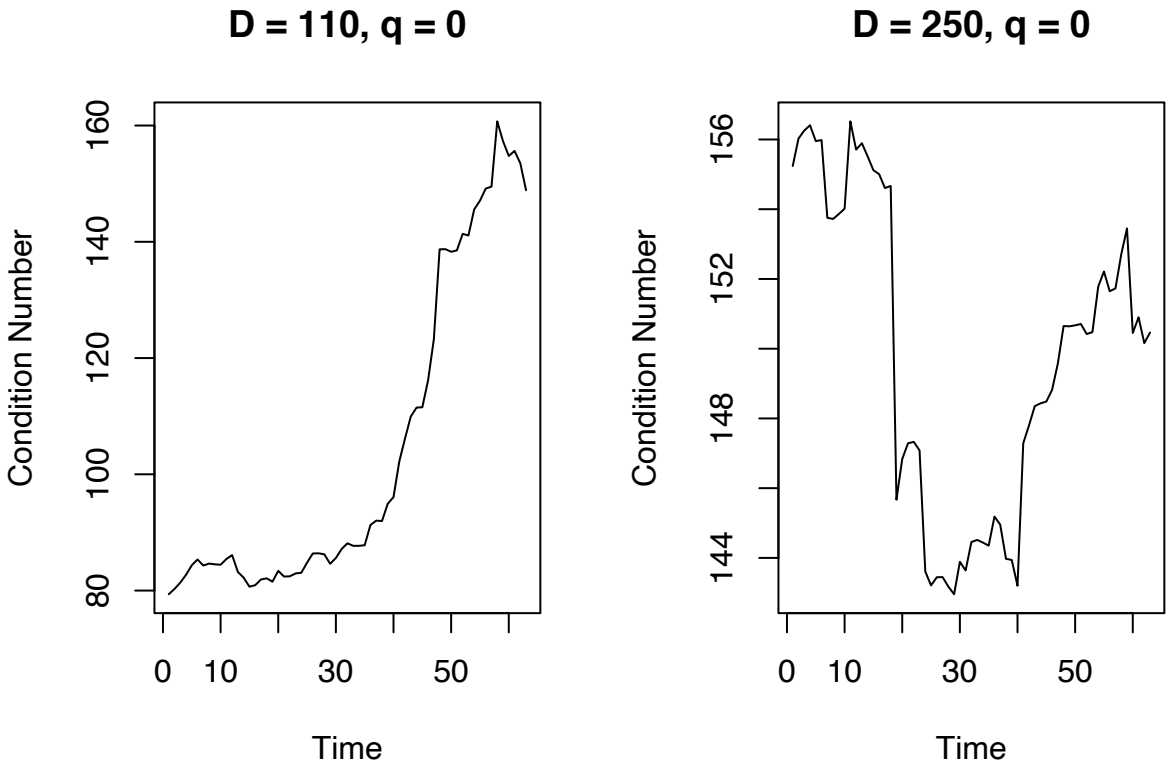
```
##  [1]  50  70  90 110 130 150 170 190 210 230 250
```

```
plot(win, sc0$train.curve, type='l',
     ylim=c(-4,3), col='red',
     ylab='Sharpe Ratio', xlab='Window Length',
     main='Sharpe Ratios for Different Window Lengths at Lag q = 0')
lines(win, sc0$valid.curve, col='green')
lines(win, sc0$test.curve, col='blue')
legend("topright", c('train','validation','test'), fill=c('red','green','blue'))
```



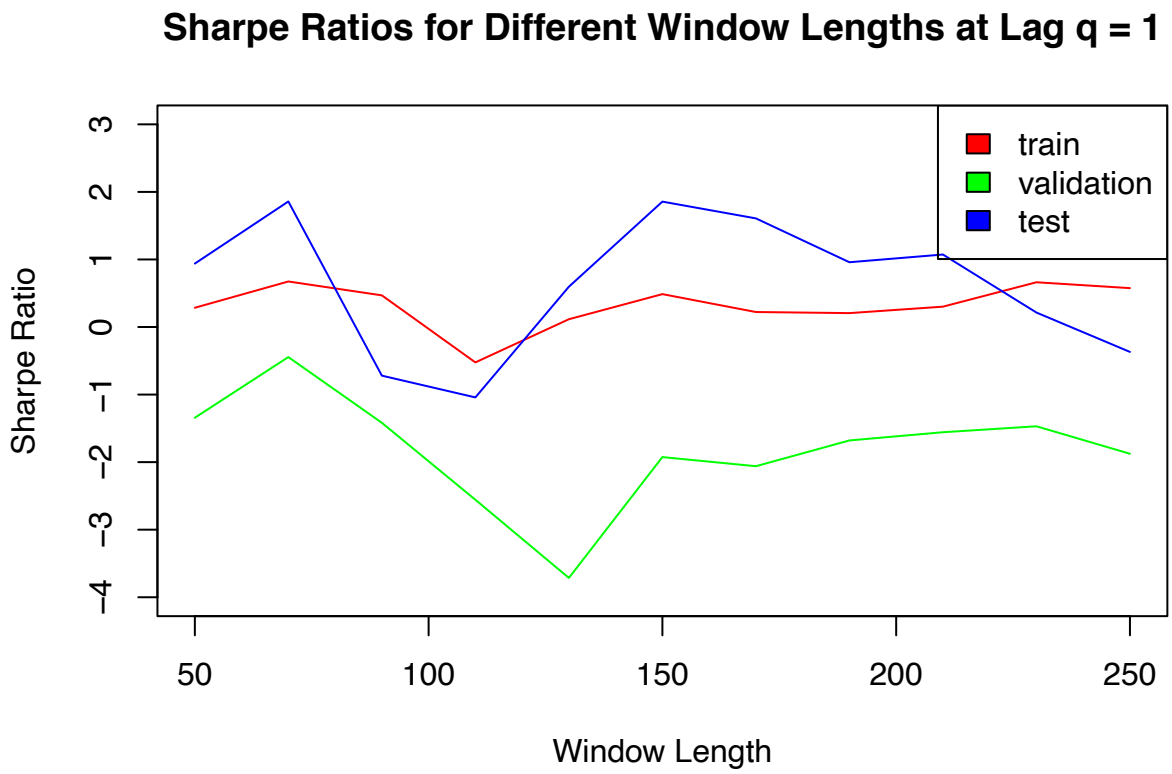
Condition numbers at different time points for $X^T X/10$ of the test set at lag q = 0

```
par(mfrow=c(1,2))
plot.ts(sc0$condnum.test[4,],
        main='D = 110, q = 0',
        ylab='Condition Number', xlab='Time')
plot.ts(sc0$condnum.test[11,],main='D = 250, q = 0',
        ylab='Condition Number', xlab='Time')
```



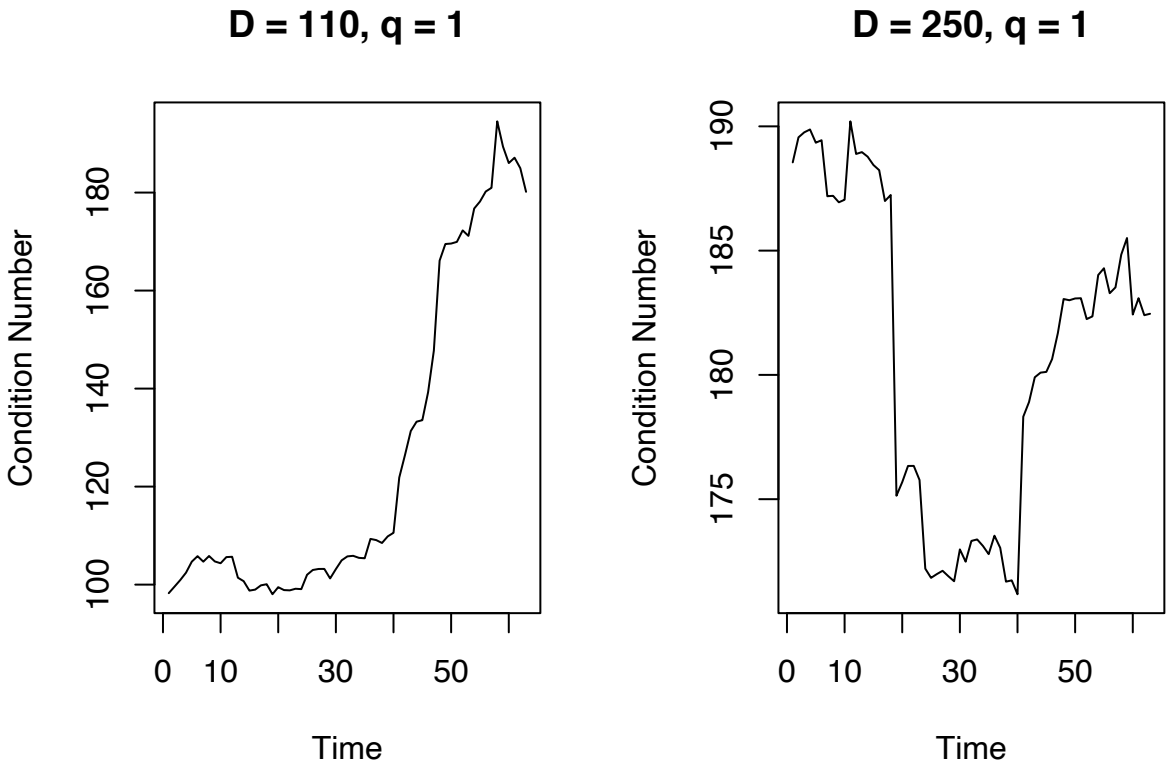
Sharpe ratios for different window lengths at lag q = 1

```
q1 <- 1
data1 <- pred.snp.prepare(q1)
sc1 <- sharpe.curves(data1, lambda_all$best, 0, 250, win =win)
plot(win, sc1$train.curve, type='l',
     ylim=c(-4,3), col='red',
     ylab='Sharpe Ratio', xlab='Window Length',
     main='Sharpe Ratios for Different Window Lengths at Lag q = 1')
lines(win, sc1$valid.curve, col='green')
lines(win, sc1$test.curve, col='blue')
legend("topright", c('train','validation','test'), fill=c('red','green','blue'))
```



Condition numbers at different time points for $X^T X/10$ of the test set at lag q = 1

```
par(mfrow=c(1,2))
plot.ts(sc1$condnum.test[4,],main='D = 110, q = 1',
        ylab='Condition Number', xlab='Time')
plot.ts(sc1$condnum.test[11,],main='D = 250, q = 1',
        ylab='Condition Number', xlab='Time')
```



Question 5

Consider which θ is the best by looking at the output of sharpe ratio obtained by marginal correlation screening.

Lag = 0 for training set. Maximum sharpe ratio is 0.70307753.Hence, $\theta = 0.03$

```
sim.grid(data0, lambda_all$best, 250)
```

```
##   [1]  0.66660324  0.69582817  0.57682497  0.70307753  0.50102145  0.56287899
##   [7]  0.57399063  0.05324117 -0.14765919 -1.02692629 -1.04008981 -0.72275993
##  [13] -0.86998499 -0.71807014 -0.43907072 -0.27299139 -0.02628937 -0.35978742
##  [19] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [25] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [31] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [37] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [43] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [49] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [55] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [61] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [67] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [73] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [79] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [85] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [91] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
##  [97] -0.21023828 -0.21023828 -0.21023828 -0.21023828 -0.21023828
```

Lag = 0 for validation set. Maximum sharpe ratio is 1.42381875.Hence, $\theta = 0.06$

```
sim.grid.valid(data0,lambda_all$best, 250)
```

```
##   [1] -1.97222595 -1.56634634 -1.82888718 -1.59719062 -1.18794458  0.16939087
##   [7]  1.42381875  0.88381310  1.01955958  0.68710569  0.56933834 -0.13015657
##  [13] -0.19474129 -0.21016345 -0.01681136  1.49713662  1.49713662  1.49713662
##  [19]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [25]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [31]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [37]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [43]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [49]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [55]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [61]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [67]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [73]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [79]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [85]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [91]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
##  [97]  1.49713662  1.49713662  1.49713662  1.49713662  1.49713662
```

Lag = 1 for training set.Maximum sharpe ratio is 0.906319535. Hence, $\theta = 0.03$

```
sim.grid(data1, lambda_all$best, 250)
```

```
##   [1]  0.575746964  0.723257217  0.733794719  0.906319535  0.725371117
##   [6]  0.664724971  0.653803098  0.002847738 -0.106109544 -1.160033747
##  [11] -1.099428313 -0.839110082 -1.132239883 -0.852494887 -0.618992384
##  [16] -0.414557170 -0.028460606 -0.361768794 -0.212428338 -0.212428338
##  [21] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [26] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [31] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [36] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [41] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [46] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [51] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [56] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [61] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [66] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [71] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [76] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [81] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [86] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [91] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
##  [96] -0.212428338 -0.212428338 -0.212428338 -0.212428338 -0.212428338
## [101] -0.212428338
```

Lag = 1 for validation set.Maximum sharpe ratio is 0.948576244 . Hence, $\theta = 0.07$

```
sim.grid.valid(data1,lambda_all$best, 250)
```

```
##   [1] -1.876106210 -2.093384374 -1.330324104 -1.626780481 -1.075438935
##   [6]  0.463818771  1.258647368  0.948576244  1.134035164  0.510018653
##  [11]  0.253977082  0.906454290 -0.185658760 -0.199984830 -0.006066917
##  [16]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [21]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [26]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [31]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [36]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [41]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [46]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [51]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [56]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [61]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [66]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [71]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [76]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [81]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [86]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [91]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
##  [96]  1.497136619  1.497136619  1.497136619  1.497136619  1.497136619
## [101]  1.497136619
```