Problem 1

Use the stock returns in DailyReturn.csv for this problem. DailyReturn.csv contains returns for 100 large US stocks and as well as the ETF, SPY which tracks the S&P500.

Create a routine for calculating an exponentially weighted covariance matrix. If you have a package that calculates it for you, verify that it calculates the values you expect. This means you still have to implement it.

Vary $\lambda \in (0, 1)$. Use PCA and plot the cumulative variance explained by each eigenvalue for each λchosen.

What does this tell us about values of λ and the effect it has on the covariance matrix?

Problem 1.1

Create a routine for calculating an exponentially weighted covariance matrix. If you have a package that calculates it for you, verify that it calculates the values you expect. This means you still have to implement it.

There were two ways mentioned in the note to generate exponentially weighted covariance.

The first one is a simple exponential smoothing model. The variance for tomorrow is updated based on today's forecast variance and today's deviation from the mean.

$$\sigma_t^2 = \lambda\sigma_{t-1}^2 + (1 - \lambda)\left(x_{t-1} - \bar{x}\right)^2$$

After substitution and normalization we have the second method. The weight of a prior observation is:

$$w_{t-i} = (1 - \lambda)\,\lambda^{i-1}$$

$$\widehat{w_{t-i}} = \frac{w_{t-i}}{\sum_{j=1}^{n} w_{t-j}}$$

Variance and Covariance Estimators are then

$$\widehat{\sigma_t^2} = \sum_{=1}^{n} w_{t-i}\left(x_{t-i} - \overline{x}\right)^2$$

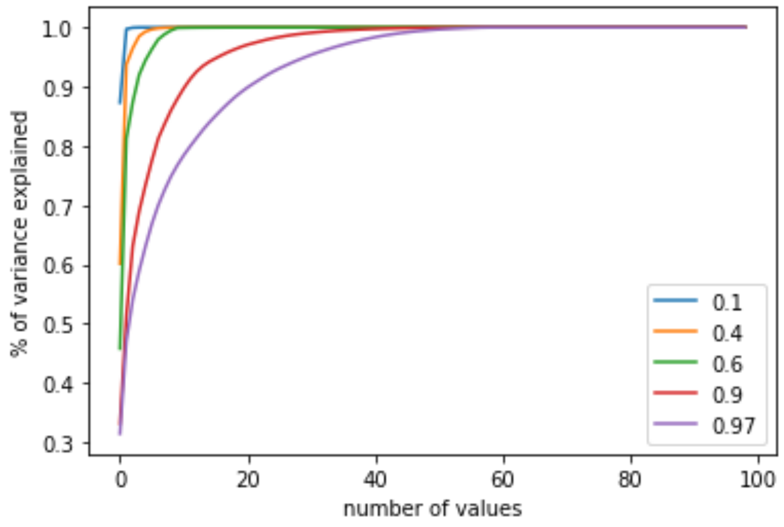
$$\widehat{cov(x, y)} = \sum_{i=1}^{n} w_{t-i}\left(x_{t-i} - \overline{x}\right)\left(y_{t-i} - \overline{y}\right)$$

routine for calculating an exponentially weighted covariance matrix is implemented in this way.

Problem 1.2

Vary $\lambda \in (0, 1)$. Use PCA and plot the cumulative variance explained by each eigenvalue for each λ chosen.

What does this tell us about values of λ and the effect it has on the covariance matrix?



PCA is implemented and the cumulative variance explained graph is shown above. From the graph we can see that as λ becomes larger, the curve becomes smoother. When λ is small, the cumulative variance explained approach to 1 after few values, which means the covariance is explained by a small number of eigenvalues. It can be seen from the formula that when λ is small, the first several weights is quite large. And as λ becomes larger, more eigenvalues are used.

Problem 2

Copy the chol_psd(), and near_psd() functions from the course repository – implement in your programming language of choice. These are core functions you will need throughout the remainder of the class.

Implement Higham's 2002 nearest psd correlation function.
Generate a non-psd correlation matrix that is 500x500. You can use the code I used in class:

```
n=500
sigma = fill(0.9,(n,n))
for i in 1:n
    sigma[i,i]=1.0
end
sigma[1,2] = 0.7357
sigma[2,1] = 0.7357
```

Use near_psd() and Higham's method to fix the matrix. Confirm the matrix is now PSD.

Compare the results of both using the Frobenius Norm. Compare the run time between the two. How does the run time of each function compare as N increases?

Based on the above, discuss the pros and cons of each method and when you would use each. There is no wrong answer here, I want you to think through this and tell me what you think.

Problem 2.1

Copy the chol_psd(), and near_psd() functions from the course repository – implement in your programming language of choice. These are core functions you will need throughout the remainder of the class.

chol_psd(), and near_psd() functions are implemented by python.

Problem 2.2

Implement Higham's 2002 nearest psd correlation function.
Generate a non-psd correlation matrix that is 500x500. You can use the code I used in class:

Function Higham_psd is implemented as the steps shown in the lecture note. There exists function np.linalg.norm(df, ord = 'fro') in python. But I also implement it by my own function Frobenius().

Problem 2.3

Use near_psd() and Higham's method to fix the matrix. Confirm the matrix is now PSD.

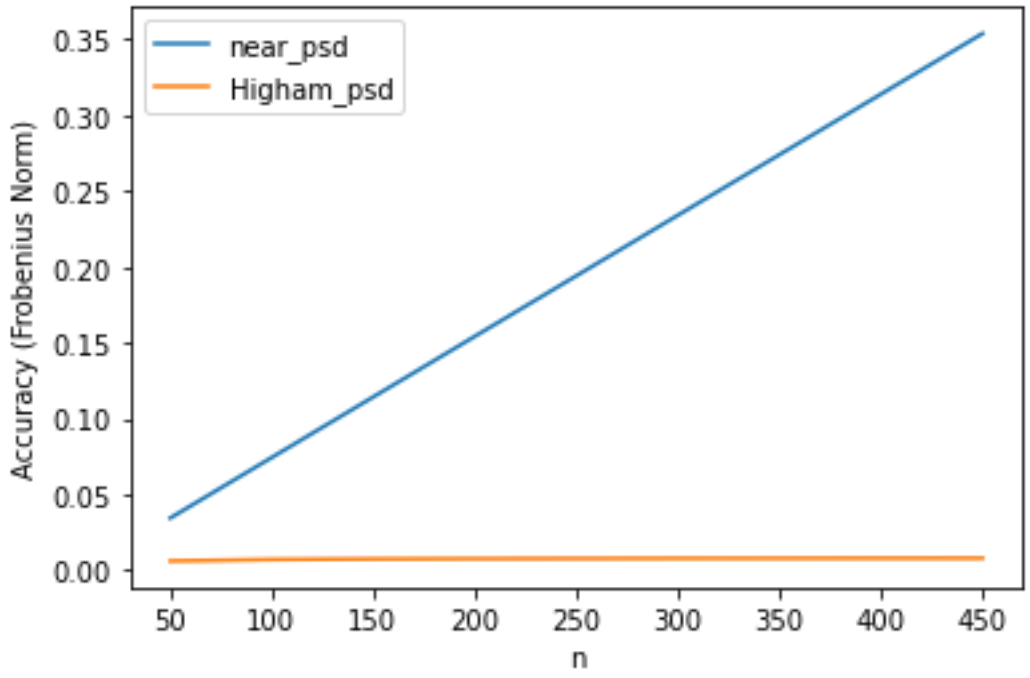
A function is_psd() is implemented to check whether a function is psd or not. And the matrix sigma is transferred by both near_psd() and Higham's method. And the result is proved to be PSD.

Problem 2.4

Compare the results of both using the Frobenius Norm. Compare the run time between the two. How does the run time of each function compare as N increases?

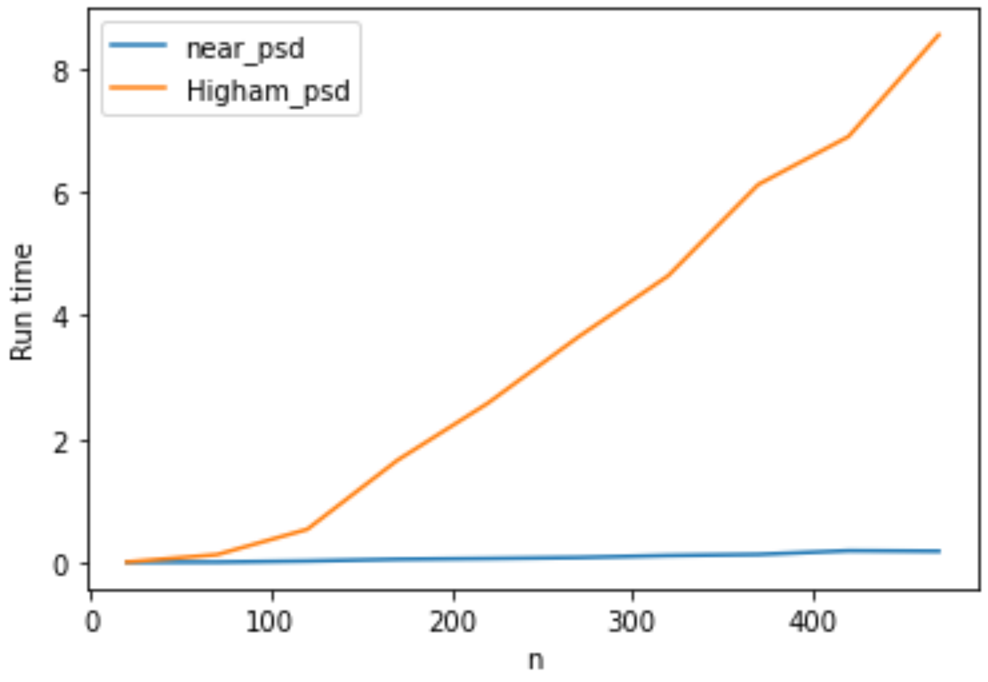
The N choose here is from 50 to 500 with steps of 50. Frobenius Norm of the result is the Frobenius norm of the difference between the input matrix and after its psd version. It is a scalar value that gives us information about how close the "near PSD matrix" is to the original input matrix in terms of magnitude. The smaller the Frobenius norm, the closer the two matrices are.

For the result of Frobenius norm we have:



As N increase, the Frobenius norm of Higham method keeps at a low level, which means the accuracy keeps at high level. While the Frobenius norm of near_psd method increase linearly with n, which means the accuracy is decrease at linear speed.

For the run time we have:



As N increase, the run time of near_psd method keeps at a low level, which means the efficiency keeps at high level. While the run time of Higham method increase with n, which means the efficiency is relatively low.

Problem 2.5

Based on the above, discuss the pros and cons of each method and when you would use each. There is no wrong answer here, I want you to think through this and tell me what you think.

Method	**Pros(as n increase)**	**Cons(as n increase)**
near_psd(R&J method)	**Run time does not increase much.**	**Result is far from original matrix.**
Higham	**Result is close to original matrix.**	**Run time increase a lot.**

when you would use each:

It is a balance between efficiency and accuracy.

When we want the run time to be slow such as we are doing the high frequency trading, we have to choose near_psd() to earn efficiency but sacrifice some accuracy at the same time. If we are doing some work not in hurry and need more accuracy, Higham method will be chosen.

Problem 3

Using DailyReturn.csv.

Implement a multivariate normal simulation that allows for simulation directly from a covariance matrix or using PCA with an optional parameter for % variance explained. If you have a library that can do these, you still need to implement it yourself for this homework and prove that it functions as expected.

Generate a correlation matrix and variance vector 2 ways:

1. Standard Pearson correlation/variance (you do not need to reimplement the cor() and var() functions).
2. Exponentially weighted $\lambda = 0.97$

Combine these to form 4 different covariance matrices. (Pearson correlation + var()), Pearson correlation + EW variance, etc.)

Simulate 25,000 draws from each covariance matrix using:

1. Direct Simulation
2. PCA with 100% explained.
3. PCA with 75% explained.
4. PCA with 50% explained.

Calculate the covariance of the simulated values. Compare the simulated covariance to it's input matrix using the Frobenius Norm (L2 norm, sum of the square of the difference between the matrices). Compare the run times for each simulation.

What can we say about the trade offs between time to run and accuracy.

Problem 3.1

A multivariate normal simulation is implemented that have parameter "method" which you can choose whether simulation directly from a covariance matrix or using PCA. And parameter explained_variance which is set to 1 default with an optional parameter for % variance explained.

Problem 3.2

Generate a correlation matrix and variance vector 2 ways:

1. Standard Pearson correlation/variance (you do not need to reimplement the cor() and var() functions).

2. Exponentially weighted $\lambda = 0.97$

Combine these to form 4 different covariance matrices. (Pearson correlation + var()), Pearson correlation + EW variance, etc.)

The four combinations are

1. Pearson correlation and var
2. Pearson correlation and EW variance
3. EW(cov+corr)
4. EW Corr +Var

Problem 3.3

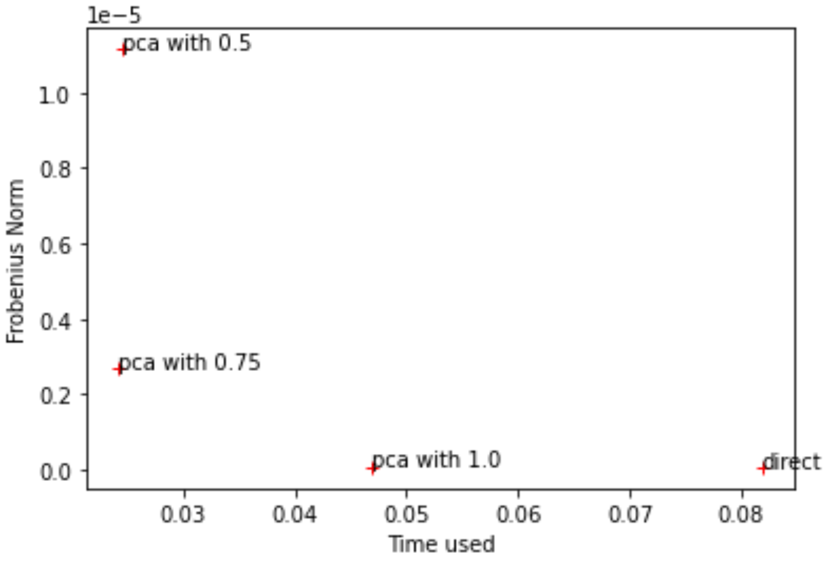
Simulate 25,000 draws from each covariance matrix using:

1. Direct Simulation
2. PCA with 100% explained.
3. PCA with 75% explained.
4. PCA with 50% explained.

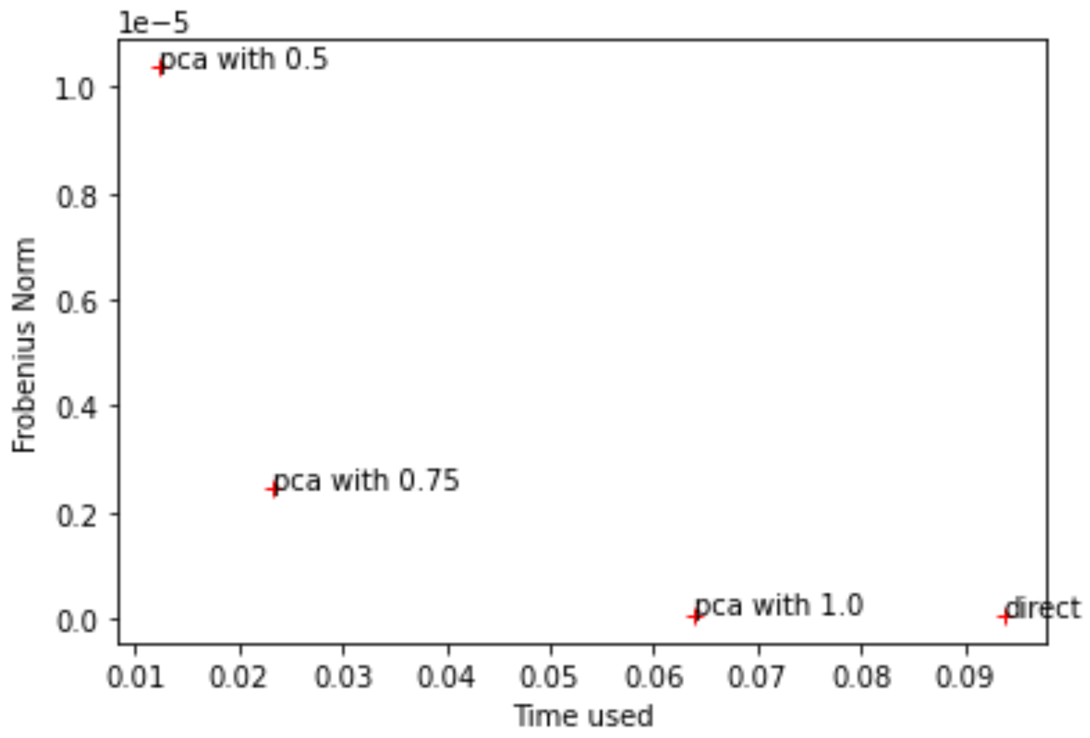
Calculate the covariance of the simulated values. Compare the simulated covariance to it's input matrix using the Frobenius Norm (L2 norm, sum of the square of the difference between the matrices). Compare the run times for each simulation.

What can we say about the trade offs between time to run and accuracy.

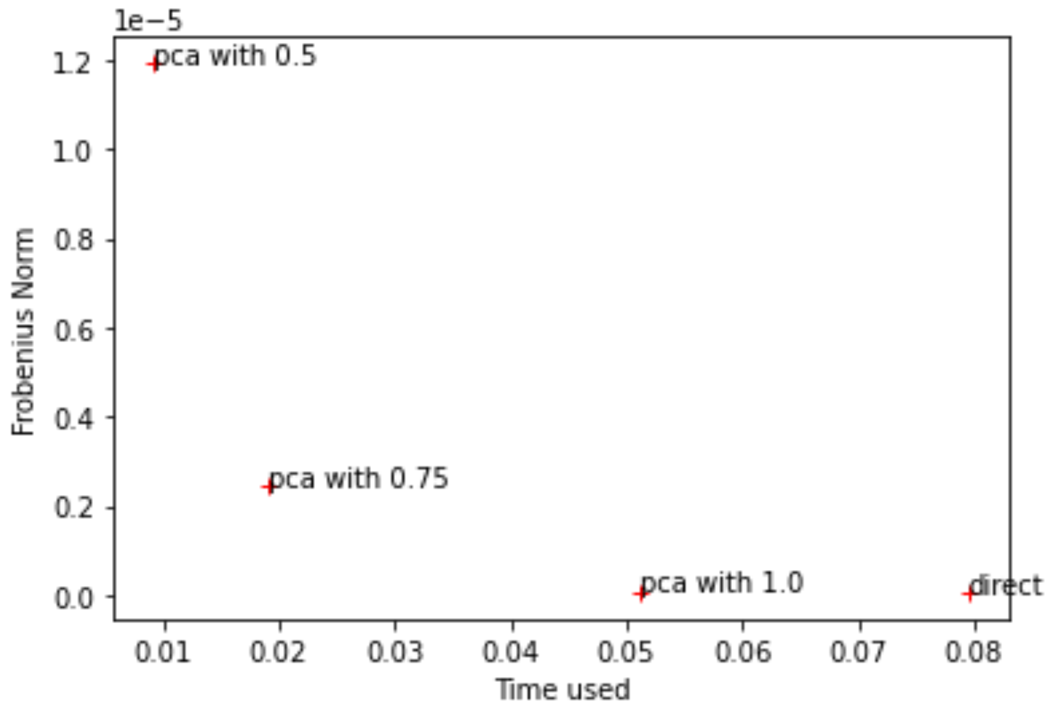
1. Pearson correlation and var.



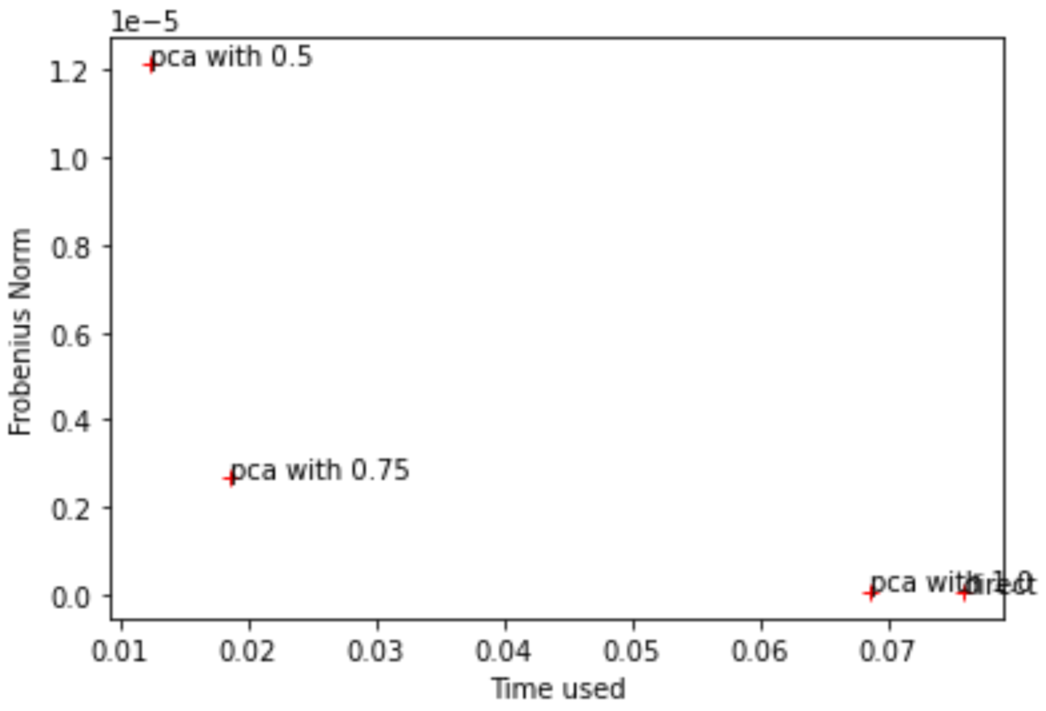
2. Pearson correlation and exponentially weighted var.



3. exponentially weighted(cov + var)



4. exponentially weighted cov + var



When direct method is compared with PCA with 100% explained, the accuracy are quite similar, but the efficiency of PCA with 100% explained is higher, as the time used is smaller.

As the level of % variance explained decrease, the run time is shorter while the accuracy is decreased. The tradeoff here is the same problem we faced in Problem 2.5: efficiency and accuracy. The accuracy drops dramatically after PCA with 75% explained. Is also support the conclusion found by Risk Metrics research that $\lambda = 0.97$ was the best value.