Project Week 03

Yuemin Tang

Fintech 545

Problem 1

Use the stock returns in DailyReturn.csv for this problem.

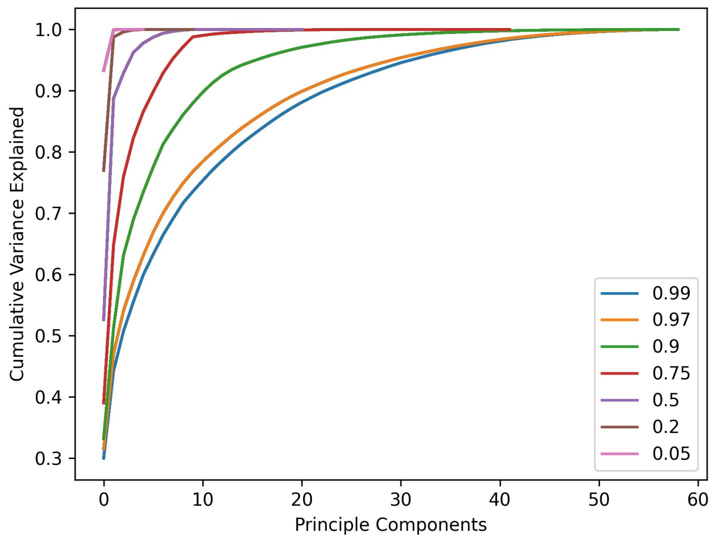
Create a routine for calculating an exponentially weighted covariance matrix.

Vary λ ∈ (0, 1). Use PCA and plot the cumulative variance explained by each eigenvalue for each λ.

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

To calculate the exponentially weighted covariance matrix, I first calculated the weight for a prior observation with the formula which provides a weighted estimator over time. Since the time interval for the dataset is not infinite, these weights are normalized so that they sum to 1 by dividing each of them by the sum of weights. Plugging these weights into the covariance formula will provide us with the exponentially weighted covariance matrix. To calculate the cumulative variance explained by each eigenvalue for each λ chosen from (0, 1), I used the python function to get the eigenvalue and eigenvector for the covariance matrix. To reduce the dimensionality, non-positive eigenvalues are removed since larger eigenvalues have more impact on the total system. Then I used the formula for % of variance explained to get the cumulative variance using PCA. Figure 1 shows the cumulative variance explained by each eigenvalue for each λ chosen.

Figure 1



We can see that as the value of λ increases, the curve for the explained variance is getting less concave and becomes smoother. This shows that a larger λ needs more data to reach the required explained variance. Therefore the cumulative variance explained will continue to change as the number of eigenvalues included increases, which means that the weight for each eigenvalue is less diverse.

Problem 2

Copy the chol_psd(), and near_psd() functions from the course repository – implement in your programming language of choice.

Implement Higham's 2002 nearest psd correlation function.

Generate a non-psd correlation matrix that is 500x500.

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

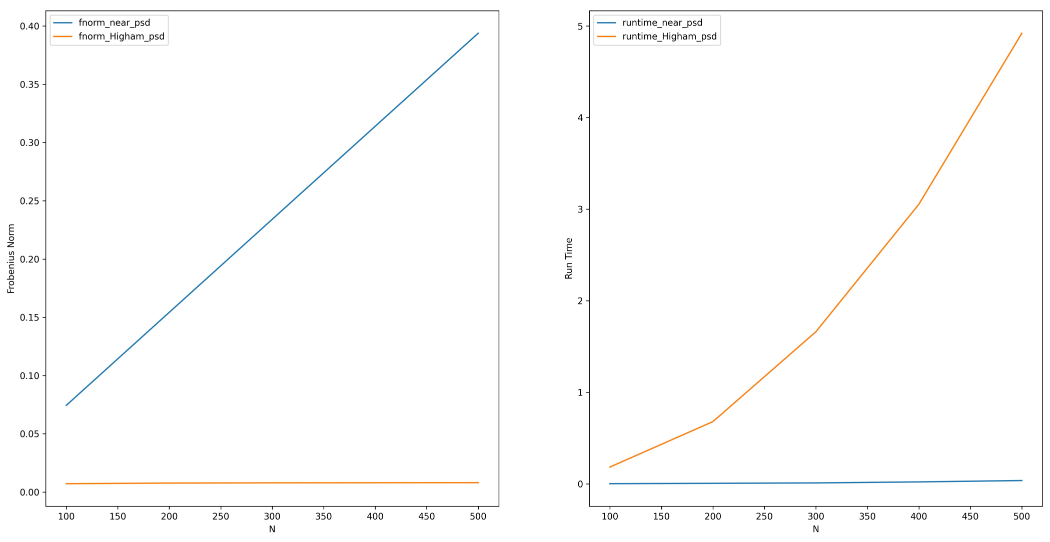
Compare the results of both using the Frobenius Norm.

Compare the run time. 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.

First, I implemented the chol_psd(), and near_psd() functions from the course repository using python. Then I implemented Higham's 2002 nearest psd correlation function, including the function for Frobenius Norm, first projection, and a second projection. To test these functions, I generated a non-psd correlation matrix that is 500x500 and implemented the near_psd() and Higham's method to fix the matrix. To confirm whether the matrix is now PSD, I checked if all eigenvalues of the matrix are now greater than or equal to 0. The result showed that these two methods successfully convert the matrix to a PSD one. To compare the result for the two methods, I used a non-psd correlation matrix with sizes 100, 200, 300, 400, and 500 and fixed them with near_psd() and Higham's method. Each time, I recorded their Frobenius Norm and run time. These data are plotted and the result is shown in Figure 2.

Figure 2



We can see that the Frobenius Norm for Higham's method is kept at a low level as the size of the input matrix increases. While for the near_psd() method, the Frobenius Norm increases linearly as the size of the input matrix increases. For the run time, the near_psd() method maintains a small run time as the size increases. While for Higham's method, the run time starts at a similar level to that of near_psd(), and increases in an exponential shape as the size increases. This means that as the size of the input matrix increases, Higham's method will have higher accuracy and slower runtime, while the near_psd() method will have lower accuracy and faster runtime. When choosing between these two methods, I will be indifferent when the matrix is small and choose Higham's method for better accuracy or near_psd() method for faster speed when the matrix is large.

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.

Generate a correlation matrix and variance vector 2 ways: Standard Pearson correlation/variance; Exponentially weighted $\lambda = 0.97$. Combine these to form 4 different covariance matrices.

Simulate 25,000 draws from each covariance matrix using: Direct Simulation; PCA with 100% explained; PCA with 75% explained ; PCA with 50% explained.

Calculate the covariance of the simulated values. Compare the simulated covariance to it's input matrix using the Frobenius Norm.

Compare the run times for each simulation.

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

First, I implemented a multivariate normal simulation that allows for simulation directly from a covariance matrix or using PCA with an optional parameter for % variance explained. Then I generated 4 different covariance matrices by using a combination of Standard Pearson correlation/ variance and Exponentially weighted correlation/variance with a λ of 0. 97. To test for the 4 covariance matrix, I simulated 25,000 draws from each covariance matrix using the method of directly from a covariance matrix and the method of PCA with variance explained of 100%, 75, and 50%. To compare these simulated covariances, I calculated the Frobenius Norm and runtime for each simulation of the 4 covariance matrices. The calculated data is shown in Table 1 and Table 2. The plotted norm and runtime are shown in Figure 3.

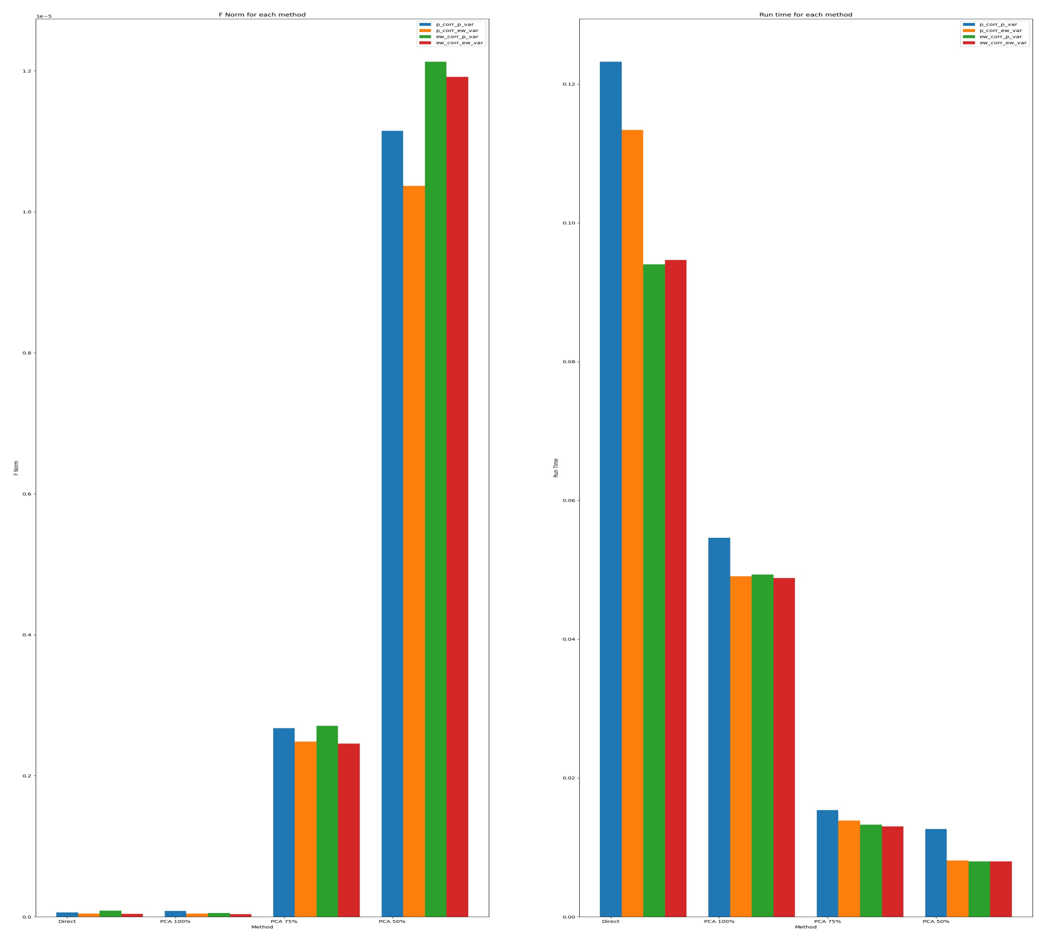
Table 1

(F Norm)	p_corr_p_var	p_corr_ew_var	ew_corr_p_var	ew_corr_ew_var
Direct	6.19832137744449E−08	4.39005996224252E−08	8.53817146135444E−08	3.93210480919464E−08
PCA 100%	8.45677546884457E−08	4.40466304127791E−08	5.55511951367945E−08	3.6304398811788E−08
PCA 75%	2.67622426122907E−06	2.48588463292653E−06	2.70888878132329E−06	2.45561135045445E−06
PCA 50%	1.11505833539819E−05	1.03668798896539E−05	1.21302353537161E−05	1.19154434607878E−05

Table 2

(Run Time)	p_corr_p_var	p_corr_ew_var	ew_corr_p_var	ew_corr_ew_var
Direct	0.131713151931763	0.118016004562378	0.101308107376099	0.0879981517791748
PCA 100%	0.0503048896789551	0.0479199886322022	0.0480062961578369	0.0476779937744141
PCA 75%	0.0139288902282715	0.0133187770843506	0.0113871097564697	0.0116300582885742
PCA 50%	0.00763416290283203	0.00943613052368164	0.00796794891357422	0.00730586051940918

Figure 3



We can see that the direct simulation from the covariance matrix has an overall lower Frobenius Norm and higher runtime than that of the PCA simulation. For the PCA simulation, as the % variance explained decreases, the runtime also decreases a lot, but the Frobenius Norm is increasing significantly. When the variance explained equals 100%, the PCA simulation is similar to a direct simulation and they have a very similar Frobenius Norm to the direct simulation and PCA 100% has a faster runtime. This shows that as the % variance explained increases, the accuracy for PCA increases with a tradeoff of increasing runtime. When we are seeking accuracy, we can choose PCA with 100% variance explained since it has similarly high accuracy and a faster runtime than the direct simulation. When we want lower runtime, we can reduce the % variance explained for the PCA simulation to achieve the goal.