Problem 1

Question:

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 λ chosen.

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

Solution:

To solve this problem, I defined three functions to calculate the exponential weights, exponentially weighted covariance matrix, and perform the PCA. The weights and covariances are calculated based on the following formula,

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

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

After calculating the covariance matrix, I calculated the eigenvalues and eigenvectors using *np.linalg.eigh* in *numpy*, sorting the eigenvectors by the eigenvalues, normalized them, and calculated the cumulative sums. Figure 1 shows the plotted results with the chosen lambda values in (0.15, 0.99) with 0.1 space gap. When the lambda value is small, indicating that the past does not have too much influence on the present, thus the cumulative variance curve increases quickly as the lags or number of components increases. On the other hand, the variance curve is much smoother when lambda values increases.

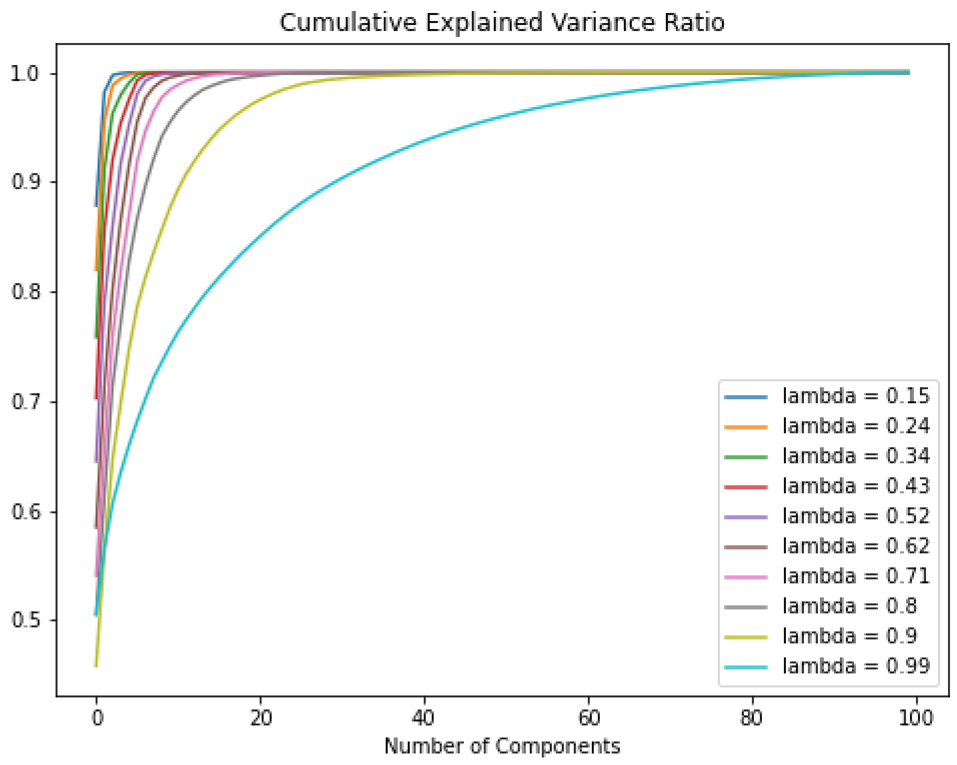

Figure 1: Cumulative Explained Variance Ratios From PCA Using Exponentially Weighted Covariance

Problem 2

Question:

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.

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.

Solution:

The chol_psd() and near_psd() are implemented by replicating the code from lecture notes, so is the Higham method. The code in the notebook confirms that both near_psd() and HIgham's method fixed the correlation matrix. The Frobenius norm using near_psd() is 0.6275, and 0.0896 using Higham's method, indicating that Higham's method is more accurate.

However, the run time for Higham's method can go much higher than the near_psd(). The graph below shows that the run time for Higham's method is higher than that for the near_psd() in general. While the time for two methods is close for smaller data—matrix dimensions less than 200x200, the run time for Higham's method grows exponentially and reaches to 7 seconds when the dimension is 1000x1000. On the other hand, the run time for near_psd() always stays below 1 second as the matrix size reaches 1000x1000.

As a result, if one is pursuing more accuracy or is rich in time, he may choose Higham's method. However, if time is short, the near_psd() can be used as a substitute.

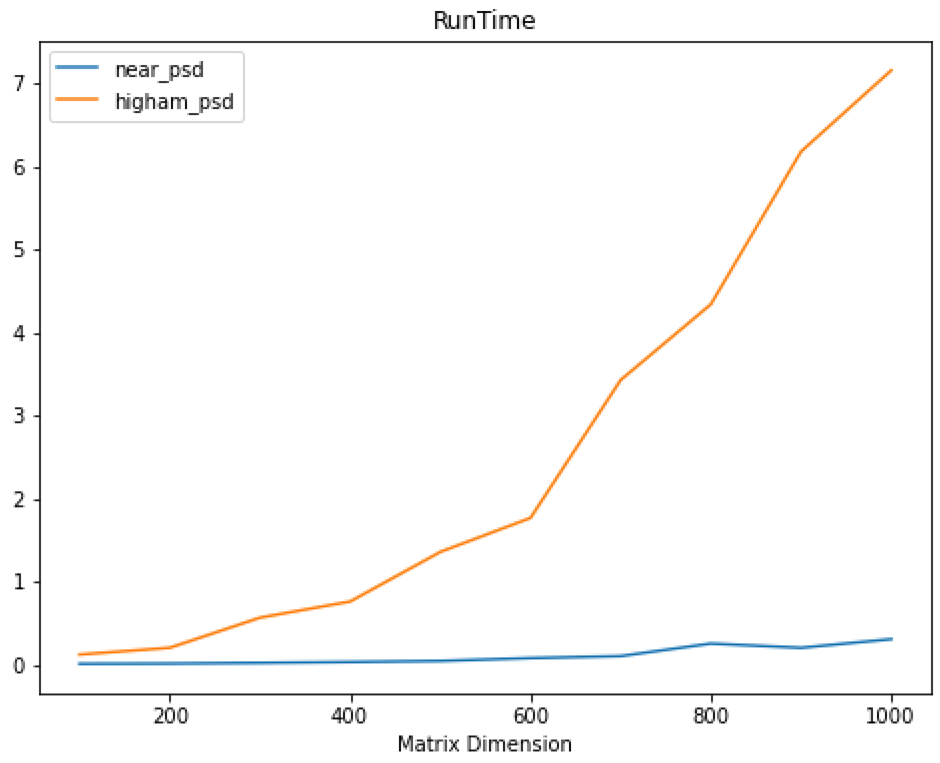

Figure 2: Run Time Comparison Between near_psd() and Higham's Method

Problem 3

Question:

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.

Solution:

For the PCA simulations, we look at the errors and runtime. The errors are small for direct simulation and 100% PCA. But as the percentage explained in PCA declines, the error becomes much larger. This is because the portion of features becomes smaller when the percentage explained in PCA gets lower, and therefore increases the error.

However, if more features are dropped—lower percentage explained simulated in PCA—the run time becomes much lower. The run time for 100% PCA and direct simulation are much higher than the 75% and 50% PCA.

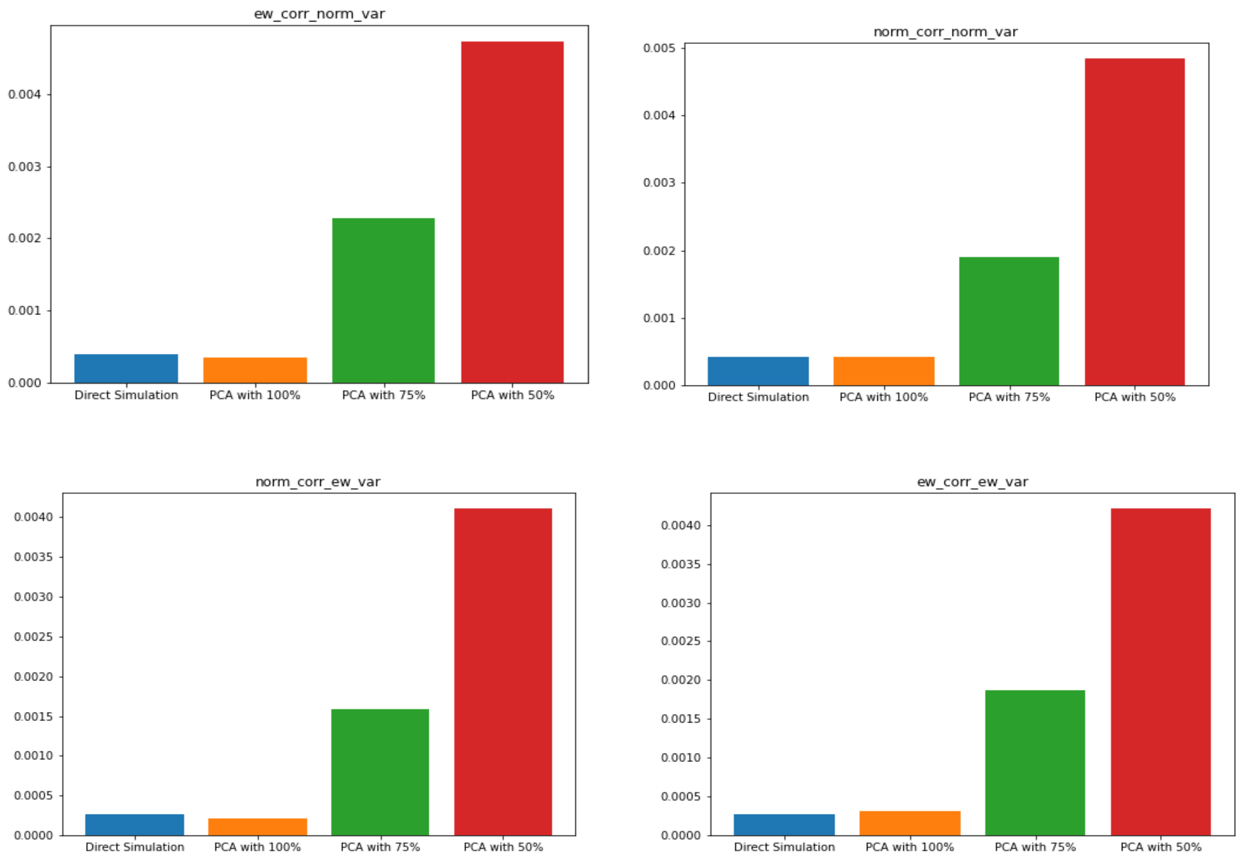

Figure 3: Covariance For Direct and PCA Simulations (50%, 70%, 100%)

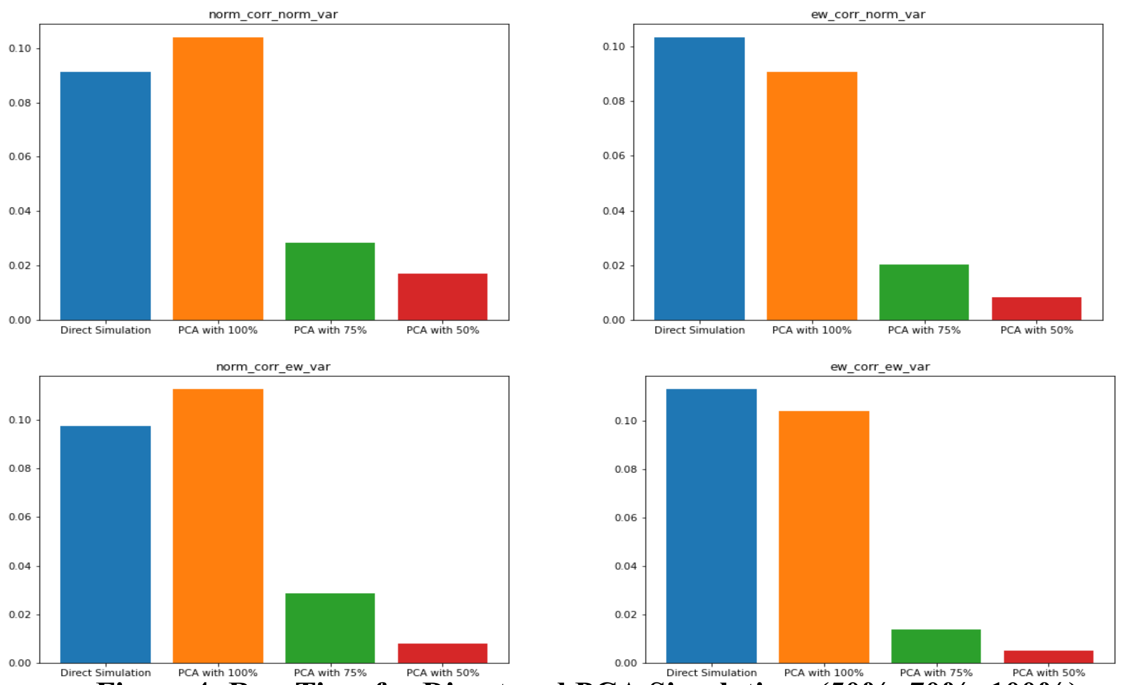

Figure 4: Run Time for Direct and PCA Simulations (50%, 70%, 100%)