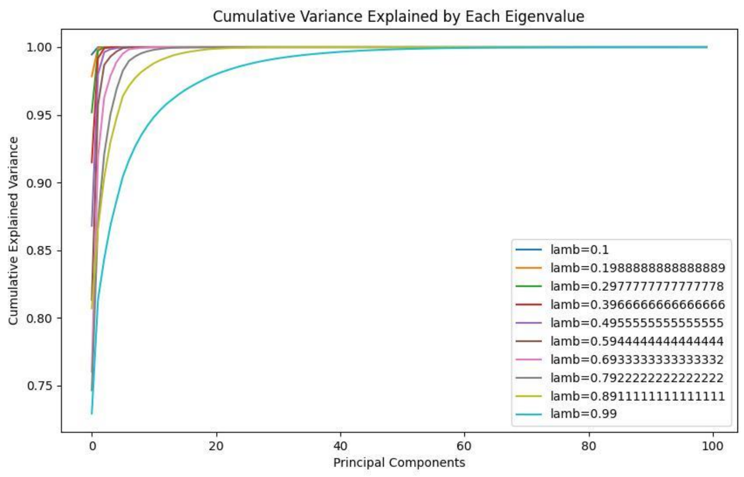
Week03 project
Wenqi Cai

Problem1

For two assets X and Y, the exponentially weighted covariance at time can be computed as:

$$\mathrm{Cov}_{\mathrm{exp}}(X,Y)_t = \sum_{i=0}^{n-1} \lambda^i \cdot (X_{t-i} - \bar{X}) \cdot (Y_{t-i} - \bar{Y})$$

By using PCA formula, we can plot the cumulative variance explained by each eigenvalue with different input parameter, which is lambda. I chose different lambda to see the differences of the cumulative variance explained curve the cumulative variance explained curves.

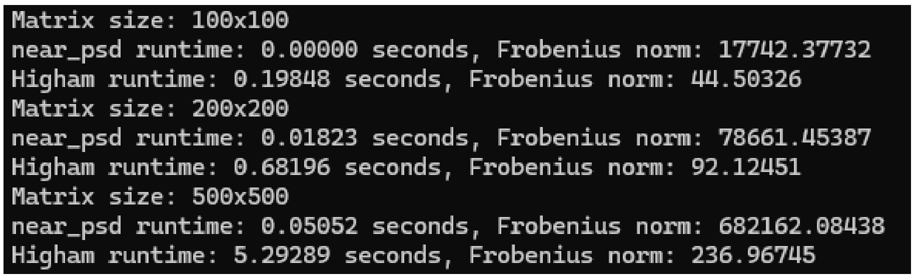


From the graph, we can see that smaller values of λ lead to a faster change in cumulative variance. Smaller λ values focus more on recent fluctuations, so the first few principal components in PCA capture more short-term variations. In contrast, larger λ values distribute weights over past data, causing the variance explained in PCA to be more evenly spread, reflecting long-term trends.

Problem2

From the notes and the code provided, I can implement these two functions using Python(Details can be seen in my cod file).

I compared the two methods by generating non-PSD matrices of different sizes and processing them using both methods. The chart below shows the result of Python:



```
Matrix size: 100x100
near_psd runtime: 0.00000 seconds, Frobenius norm: 17742.37732
Higham runtime: 0.19848 seconds, Frobenius norm: 44.50326
Matrix size: 200x200
near_psd runtime: 0.01823 seconds, Frobenius norm: 78661.45387
Higham runtime: 0.68196 seconds, Frobenius norm: 92.12451
Matrix size: 500x500
near_psd runtime: 0.05052 seconds, Frobenius norm: 682162.08438
Higham runtime: 5.29289 seconds, Frobenius norm: 236.96745
```

By checking the eigenvalues of the new matrix generated by near_psd() and Higham(), I found that the eigenvalues for them are all positive. Based on the results, These two functions are implemented successfully.
There are some differences between them:
  a) Runtime: The near_psd method consistently runs much faster than the Higham method, especially as the matrix size increases.
  b) Frobenius Norm: The Higham method achieves significantly smaller Frobenius norms compared to near_psd, indicating that it adjusts the matrix to a much closer approximation of the original matrix.

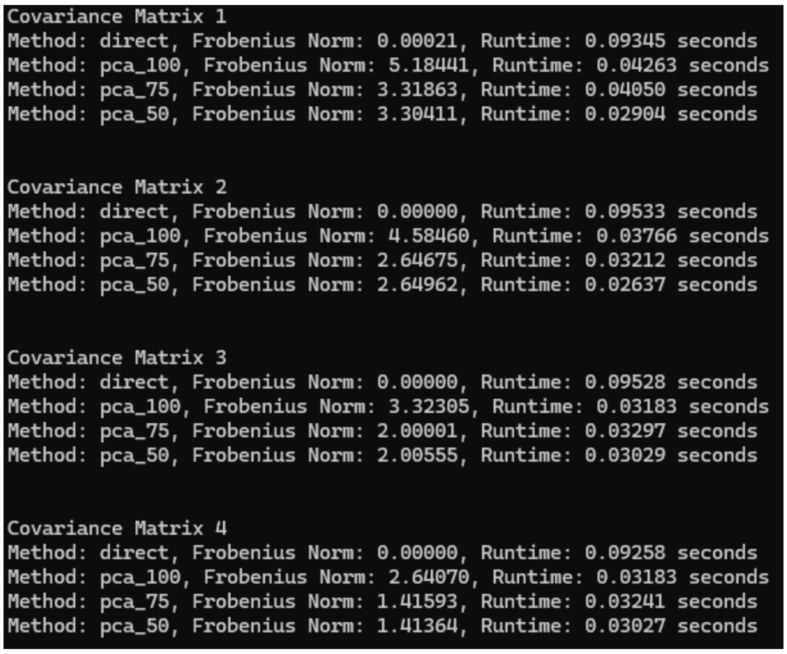
Comparison:

a) Pros of near_psd: It is much faster and may be suitable for cases where performance is critical and the exact precision of the adjustment is less important.
b) Pros of Higham: It provides a much more accurate correction, though at the cost of longer runtimes, making it more suitable when precision is the priority.

Problem3

we are going to simulate 25000 times using:
#cov1: Pearson correlation + Pearson variance
#cov2: Pearson correlation + Exponentially weighted variance
#cov3: Exponentially weighted covariance matrix
#cov4: Exponentially weighted covariance with Pearson variance

The table below shows the result of Python:



```
Covariance Matrix 1
Method: direct, Frobenius Norm: 0.00021, Runtime: 0.09345 seconds
Method: pca_100, Frobenius Norm: 5.18441, Runtime: 0.04263 seconds
Method: pca_75, Frobenius Norm: 3.31863, Runtime: 0.04050 seconds
Method: pca_50, Frobenius Norm: 3.30411, Runtime: 0.02904 seconds


Covariance Matrix 2
Method: direct, Frobenius Norm: 0.00000, Runtime: 0.09533 seconds
Method: pca_100, Frobenius Norm: 4.58460, Runtime: 0.03766 seconds
Method: pca_75, Frobenius Norm: 2.64675, Runtime: 0.03212 seconds
Method: pca_50, Frobenius Norm: 2.64962, Runtime: 0.02637 seconds


Covariance Matrix 3
Method: direct, Frobenius Norm: 0.00000, Runtime: 0.09528 seconds
Method: pca_100, Frobenius Norm: 3.32305, Runtime: 0.03183 seconds
Method: pca_75, Frobenius Norm: 2.00001, Runtime: 0.03297 seconds
Method: pca_50, Frobenius Norm: 2.00555, Runtime: 0.03029 seconds


Covariance Matrix 4
Method: direct, Frobenius Norm: 0.00000, Runtime: 0.09258 seconds
Method: pca_100, Frobenius Norm: 2.64070, Runtime: 0.03183 seconds
Method: pca_75, Frobenius Norm: 1.41593, Runtime: 0.03241 seconds
Method: pca_50, Frobenius Norm: 1.41364, Runtime: 0.03027 seconds
```

In simulation, I calculated the covariance of the simulated values and compared it to the input covariance matrices using the Frobenius Norm. The results showed that the "direct" method had the lowest Frobenius Norm values, indicating it produced the most accurate covariance estimates.

When looking at the run times, the "direct" method was generally slower than the PCA methods. However, PCA with 100% explained variance had the highest Frobenius Norm, meaning it was less accurate but faster than the "direct" method.

When we look at the trade-offs, we see that as the percentage of explained variance decreases, the run time becomes shorter, but the accuracy also drops. This is similar to the challenge we encountered in Problem 2, where we had to balance efficiency and accuracy.

In situations like high-frequency trading, where quick decisions are essential, we might prioritize efficiency even if it means losing some accuracy. However, if we’re working on projects that don’t have tight deadlines and require more precise results, we would choose a more accurate method. Ultimately, the choice depends on whether speed or accuracy is more critical for the task at hand.