**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.

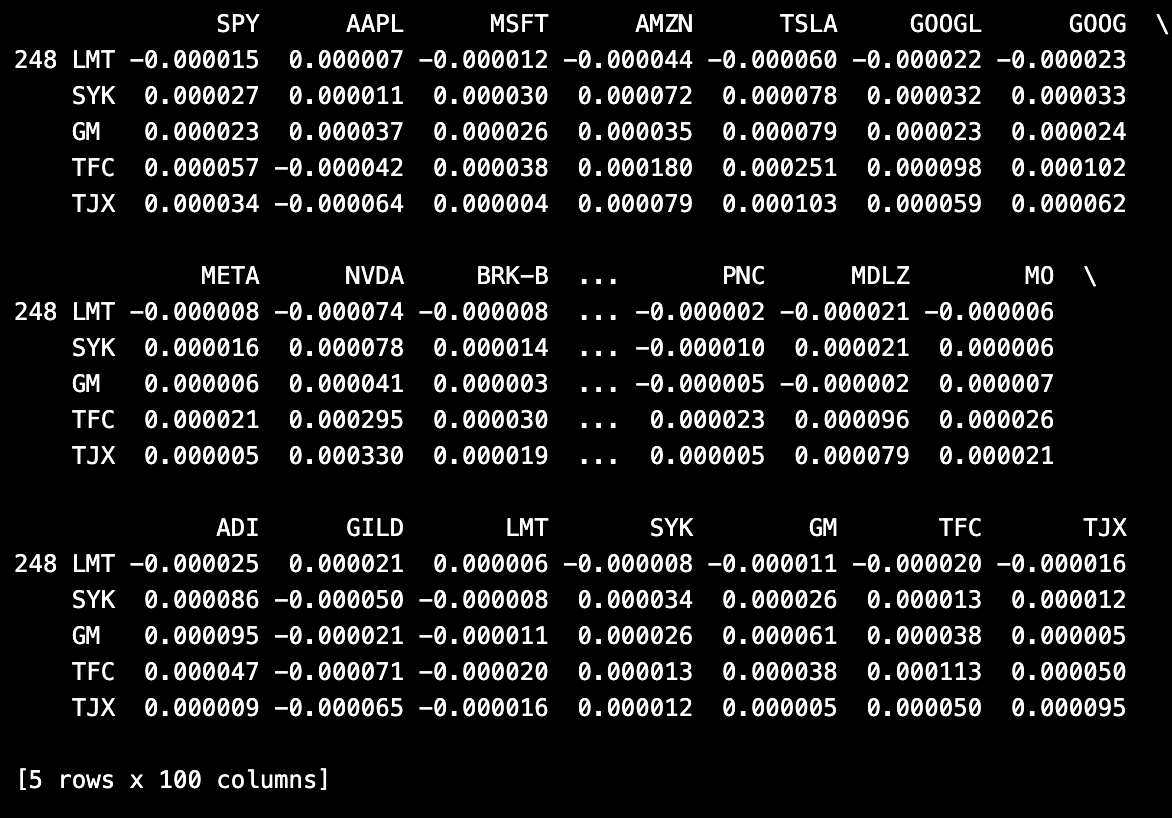
Create a routine for calculating an exponentially weighted covariance matrix. If you have a package that calculates it for you, verify that it calculates the values you expect. This means you still have to implement it.

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

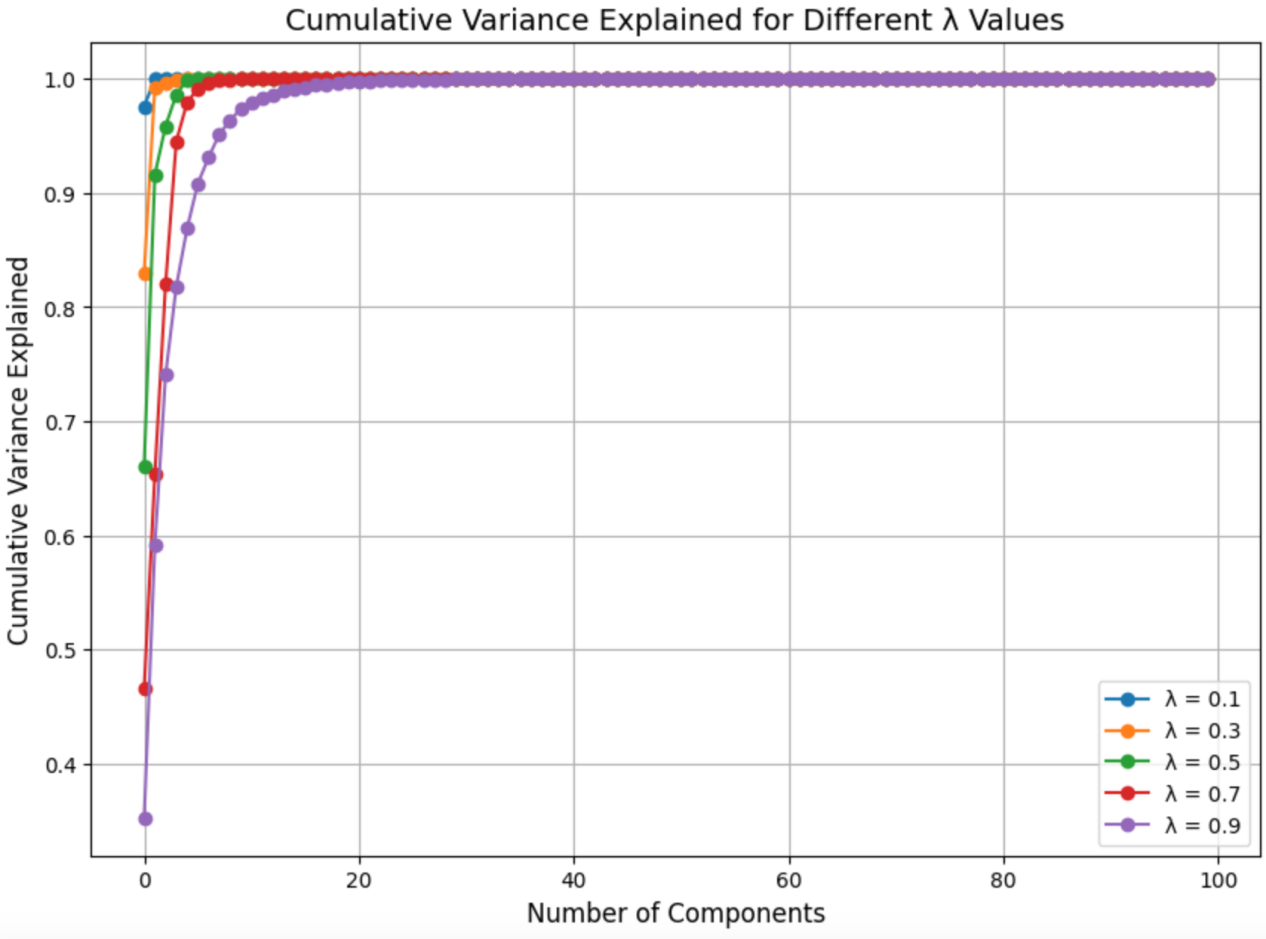
What does this tell us about values of λ and the effect it has on the covariance matrix?



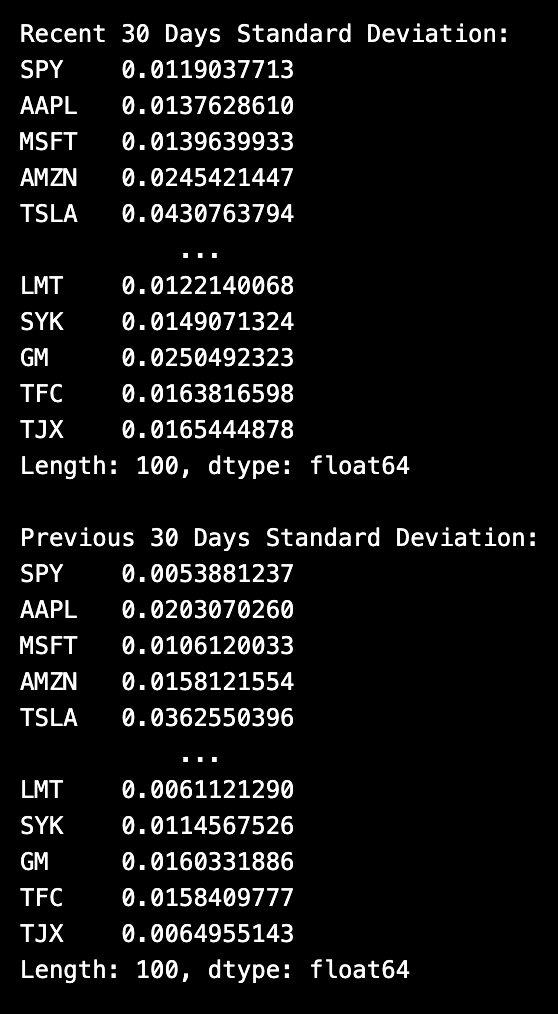
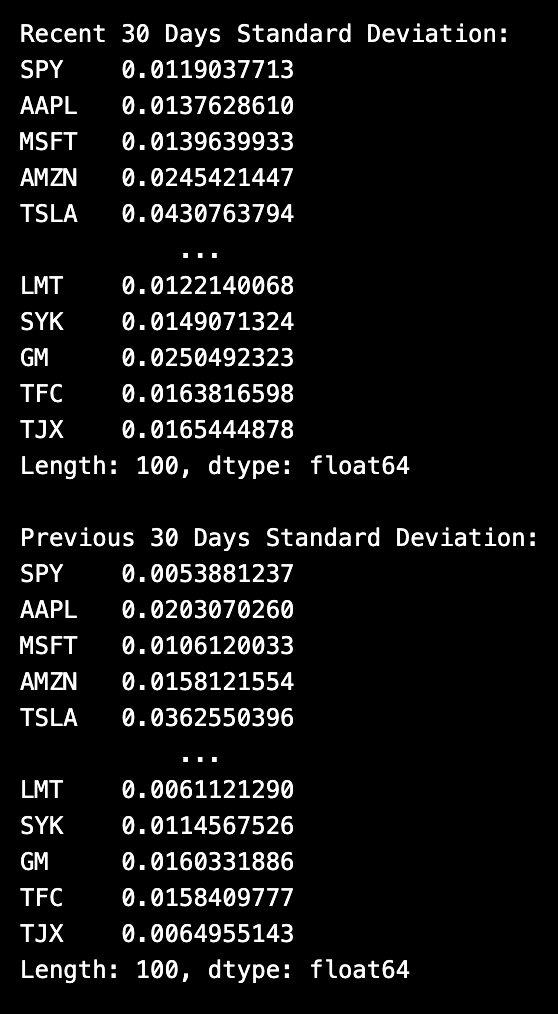
I set lambda to the effect of historical data on the overall volatility, with the more recent data weighted more heavily, and normalized all weights in order to ensure that the weights sum to one. Finally the covariance matrix is obtained.



I found out that the pandas package has a function data.ewm(alpha=0.5).cov() that does covariance matrix calculations, so I used it to verify the accuracy of my manually calculated covariance matrix. After calculating it, I found that the difference between these two matrices is less than 10^4, which can be regarded as equal. So I think my covariance matrix calculation is correct.



The graph shows that the curve with a small lambda moves more steeply and will flatten out when there are fewer principal components. So I thought this might be because the recent data is more volatile and contributes more to the overall variance, so only a few principal components are needed to explain most of the variance.



After calculating the variance, I found that it was indeed the recent data that had more variance, indicating that the picture I plotted was correct.

In summary, the value ofλregulates the distribution of weights between historical and recent data in the covariance matrix. It determines whether the historical data is dominant or the recent data is dominant.

Whenλis small, the covariance matrix is more biased towards explaining recent volatility. Therefore, when the recent data are more volatile, a few principal components can explain most of the variation in the data, and the cumulative variance explained curve will level off more quickly.

Whenλis large, the covariance matrix is smoother and relies more on smooth fluctuations in the historical data. The variance distribution may be more dispersed and more principal components are needed to explain the data variation

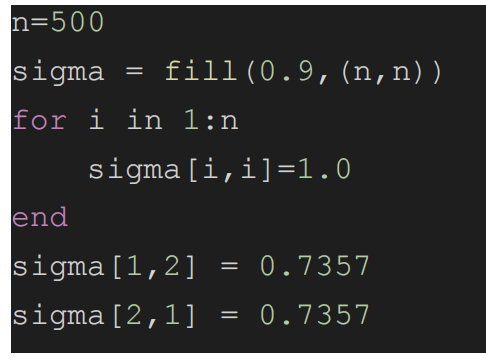
**Problem 2**

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:



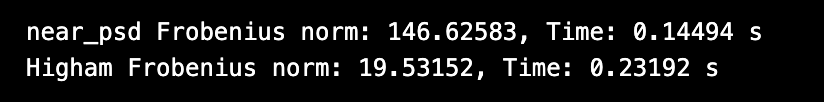
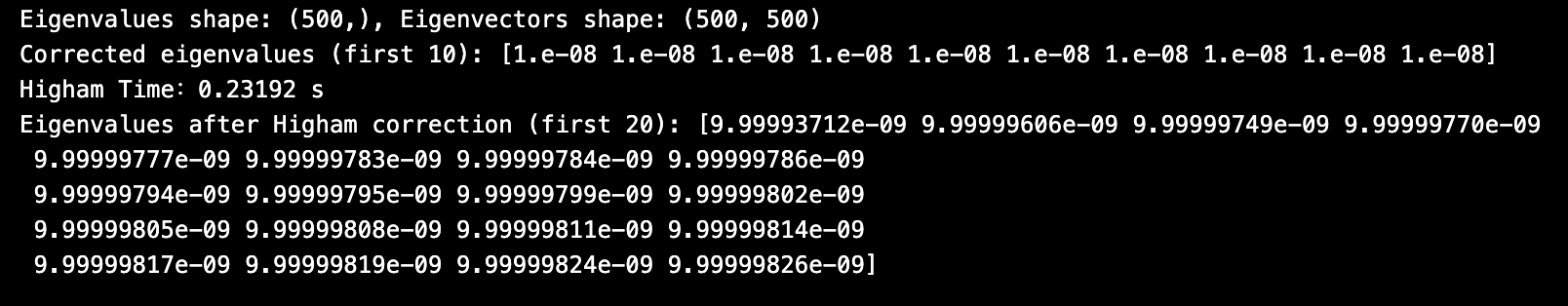
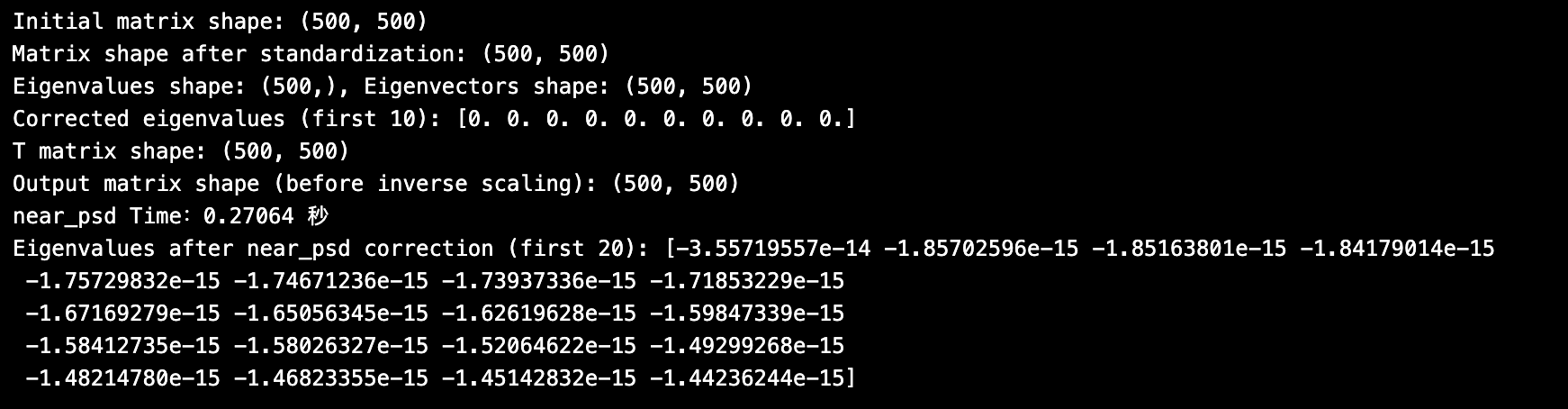
Use near\_psd() and Higham’s method to fix the matrix. Confirm the matrix is now PSD.

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.



From the computational results, the Higham method shows less difference between the corrected matrix and the original matrix, and the correction is faster. Therefore, the Higham method may be a better choice when accurate results are required and speed is high.

When the running time increases with N, the near\_psd method relies on eigenvalue decomposition and some matrix multiplication and normalization operations, which have a time complexity of about O(N^3). Therefore, as N increases, the running time tends to grow in an approximately cubic relationship.

The Higham method also relies on eigenvalue decomposition, but it is more efficient in handling negative eigenvalues and ensuring PSD correction. Due to the efficient eigenvalue processing and matrix reconstruction, the computational complexity of the Higham method is also O(N^3), but the constant factor is relatively low, so in practice, the Higham method is usually faster than the near\_psd method for the same matrix size.

Overall, for small matrices or simple implementations, the near\_psd method may be a suitable choice. For large matrices or cases with high accuracy requirements, the Higham method is a superior choice, as it not only provides more accurate results, but also maintains a relatively fast running speed at larger scales.

**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 λ = 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

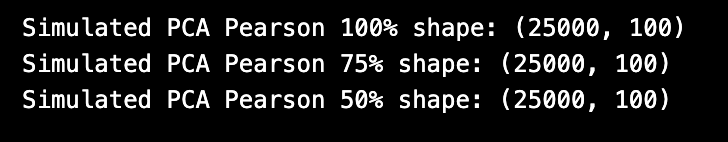
2) PCA with 100% explained.

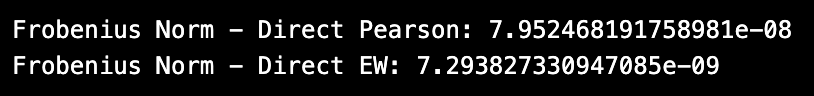
3) PCA with 75% explained.

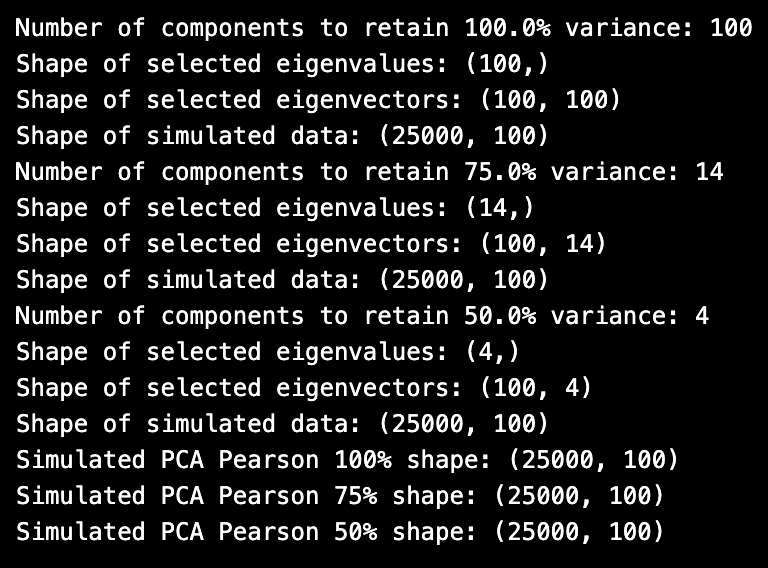
4) PCA with 50% explained.

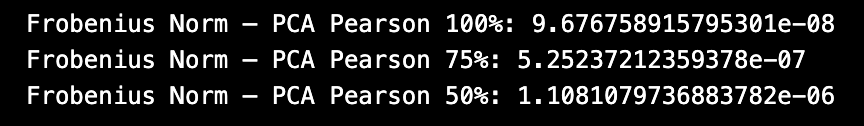
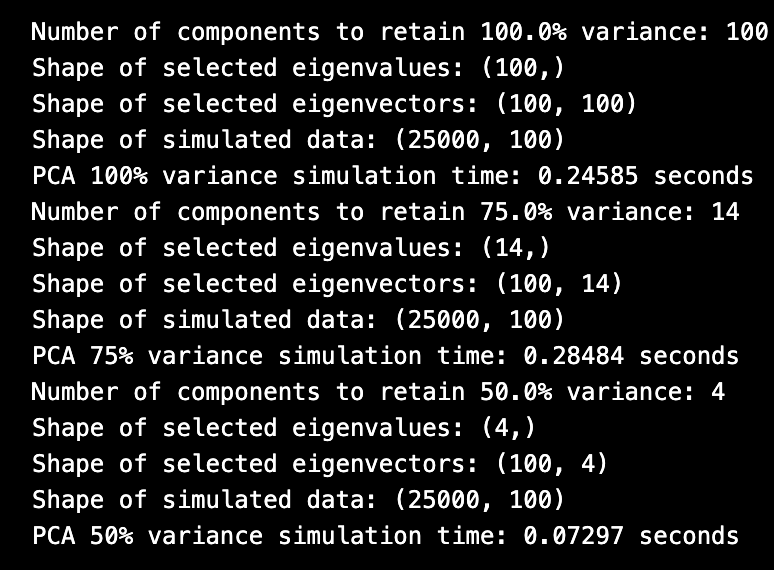
Calculate the covariance of the simulated values. Compare the simulated covariance to it’s input matrix using the Frobenius Norm (L2 norm, sum of the square of the difference between the matrices). Compare the run times for each simulation.

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









PCA (Principal Component Analysis):

PCA simulations were generated for three different explanatory rates (100%, 75%, 50%). The shape of the data for each simulation was also (25,000, 100), but the number of features used decreased as the explanation rate decreased, with 100% using 100 features, 75% using 14 features, and 50% using 4 features. In terms of time, the PCA simulation time decreases as the variance interpretation rate decreases, with the 50% variance simulation taking only about 0.07 seconds, while the 100% interpretation rate simulation took 0.24 seconds. This suggests that using fewer principal components (lower variance explained) can significantly speed up the simulation.

Comparison of Frobenius norm and simulation accuracy:

The Frobenius norm values show that the difference between the simulated covariance matrix and the original covariance matrix gradually increases as the PCA interpretation rate decreases:

100% variance: the error is the smallest (about 9.68e-08), i.e. the simulated data are closest to the original covariance matrix.

75% variance: the error increases slightly (about 5.25e-07), but is still relatively accurate.

50% variance: a significant increase in error (~1.10e-06), but the simulated data no longer retains the properties of the original covariance matrix well due to the use of only 4 principal components.

Accuracy vs runtime:

The PCA simulation with 100% explanation rate is almost indistinguishable from the direct simulation, with a small Frobenius paradigm and minimal error.

The simulation loses accuracy as the interpretation rate decreases, but the runtime is significantly shorter.

Direct simulation is computationally intensive despite its accuracy, whereas PCA simulation can be accelerated by reducing the dimensionality and is suitable for scenarios with different accuracy requirements.