Week 3 Report

Howard Jin

9/23/2023

Problem 1

Procedure:

1. **Data Loading and Preprocessing:**
   - Loaded the "DailyReturn.csv" file containing stock returns for 100 large US stocks and the ETF, SPY.
   - Dropped the first column ' to focus on the returns of the stocks.
2. **Exponentially Weighted Covariance Matrix Calculation:**
   - Implemented a routine to calculate the exponentially weighted covariance matrix for varying λ values in the set $[0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99]$.
3. **Principal Component Analysis (PCA):**
   - For each calculated covariance matrix corresponding to a specific λ, performed PCA to determine the cumulative variance explained by each eigenvalue.
4. **Visualization:**
   - Plotted the results to visualize the cumulative variance explained for different λ.

Implementation:

- Defined functions `compute_ew_cov_matrix` for calculating the exponentially weighted covariance matrix and `perform_pca` for executing PCA and computing cumulative variance.
- The weights are calculated by the formula

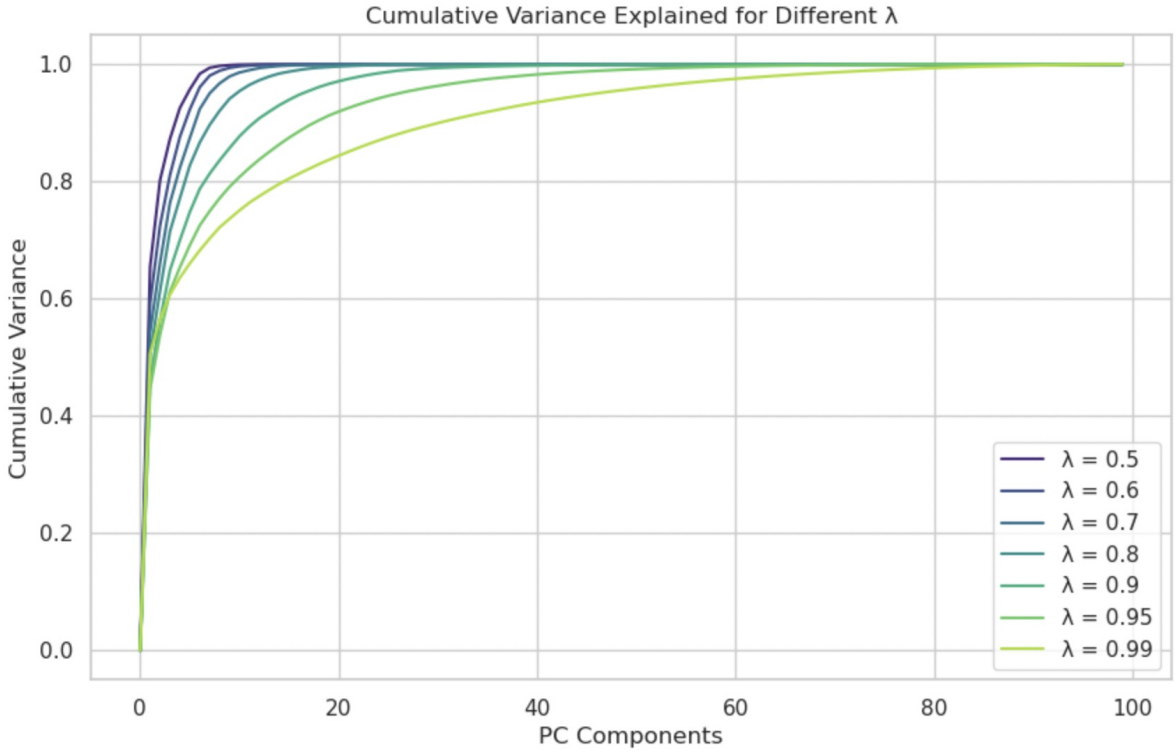
$$\hat{w_{t-i}} = \frac{w_{t-i}}{\sum_{j=1}^{n} w_{t-j}}$$

  and the corresponding exponentially weighted covariance between two stocks x and y is

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

- Iterated over different λ values to calculate the covariance matrices, perform PCA, and generate the plot illustrating the cumulative variance explained.

Conclusion:



As we saw from the graph, as the color of the curve gets lighter from purple to light green, the curve seems to have smaller slopes and converge to 1 slower. It suggests that lower λ values resulted in faster convergence of cumulative variance, indicating fewer components were needed to explain a significant proportion of the total variance, and higher λ values made the covariance matrix more responsive to recent market changes but potentially introduced more noise, requiring more components to capture the same level of information.

Thus, the choice of λ plays a crucial role in balancing the responsiveness of the covariance matrix to recent data against its stability. A smaller λ provides a more stable and less sensitive matrix, while a larger λ makes the matrix more sensitive to recent data fluctuations. Besides, the selection of λ has implications for dimensionality reduction, with varying λ values leading to different numbers of components needed to represent the original dataset effectively.

Problem 2

Procedure:

I first implemented the `chol_psd` and `near_psd` functions by translating the corresponding Julia functions in course repository. Then, I implemented the Higham's 2002 method for finding the nearest positive semi-definite (PSD) correlation matrix, which I named `higham_psd`. Subsequently, a non-PSD correlation matrix of dimensions 500x500 was generated using the provided code snippet. Both the `near_psd` and `higham_psd` methods were then applied to this non-PSD matrix to obtain PSD matrices. The results of both methods were compared using the Frobenius Norm, and the runtime of each method was analyzed across varying matrix sizes.

Implementation:

The `near_psd` and `higham_psd` functions were implemented in Python, ensuring that the diagonal elements of the resulting matrices were 1. The runtimes of both functions were plotted against increasing matrix sizes

to visualize their computational performance. The Frobenius Norm was used as a metric to measure the difference between the original non-PSD matrix and the matrices obtained from both methods.

Conclusion:

```
Matrix generated is PSD: False
Matrix fixed by near_psd is PSD: True
Matrix fixed by Highan is PSD: True
```

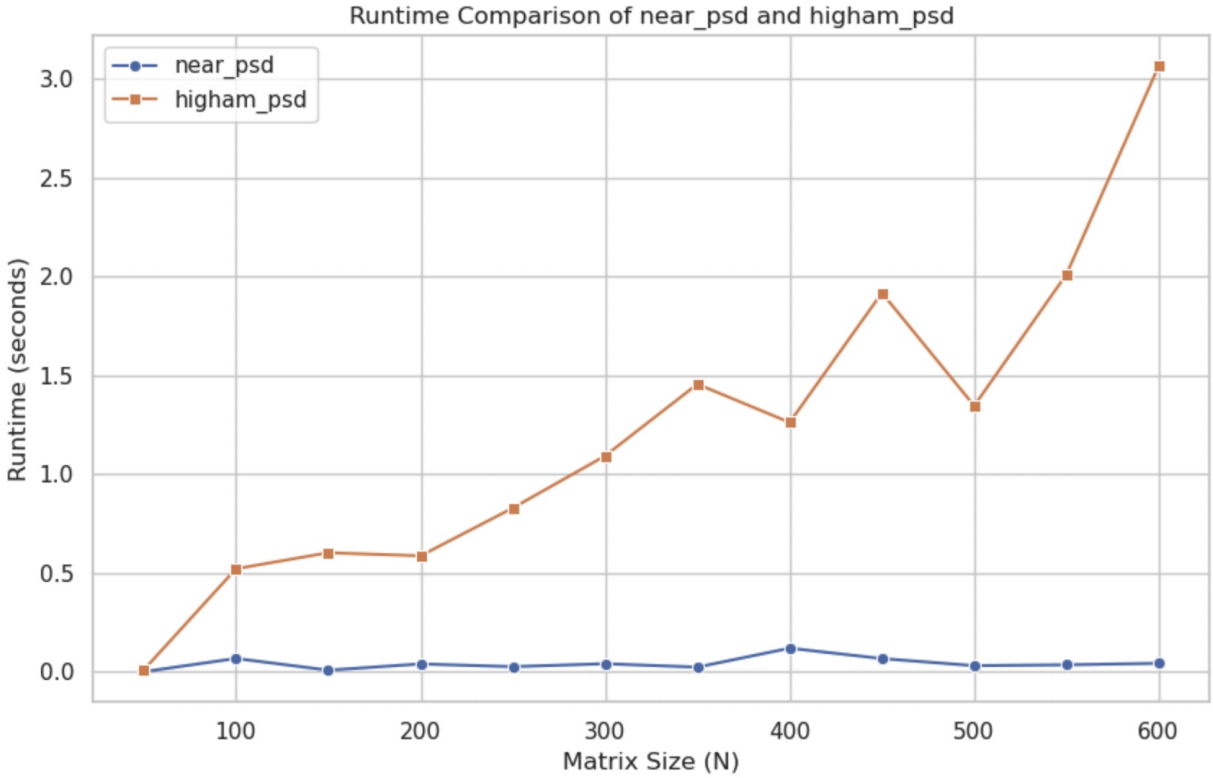
```
Runtime of near_psd: 0.4351673126220703 seconds

Runtime of higham_psd: 2.6808876991271973 seconds

Forbenium Norm of the difference between the near_psd matrix and the original matrix: 428.9109648458238

Forbenium Norm of the difference between the higham_psd matrix and the original matrix: 0.08964798746837777
```

After generating the 500x500 non-psd correlation matrix, I applied both functions to fix the matrix, finally obtained a PSD matrix. Also, the runtime of applying both methods are shown above, indicating that for the matrix size of 500, the runtime of higham_psd is greater than that of near_psd. In addition, the Forbenium norm of the difference between the matrix obtained from both method and the original matrix is provided as well.



Upon analyzing the results, it was observed that the `near_psd` method exhibited lower runtime compared to the `optimized_higham_psd` method, especially as the matrix size increased. This indicated a trade-off between computational efficiency and the method's ability to produce accurate PSD matrices. The `optimized_higham_psd` method, while more computationally intensive, might offer advantages in terms of achieving positive semi-definiteness. Therefore, the choice between these methods would depend on the specific requirements and constraints of the application, balancing the need for computational efficiency with the accuracy and characteristics of the resulting PSD matrix.

Problem 3

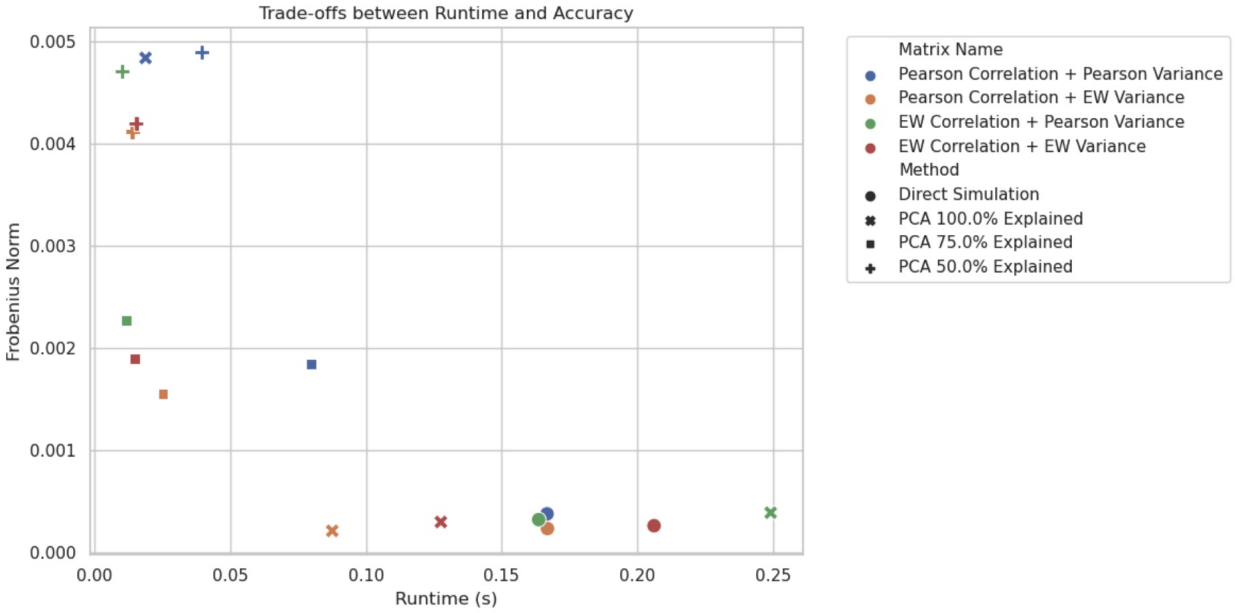
Procedure:

1. **Data Import and Preprocessing:**
   - Load the DailyReturn.csv file and preprocess it to remove any unnecessary columns.
   - Calculate the standard Pearson correlation matrix and variance vector.
   - Calculate the exponentially weighted correlation matrix and variance vector with $\lambda = 0.97$
2. **Covariance Matrix Formation:**
   - Form four different covariance matrices `pearson_pearson`, `pearson_expo`, `expo_pearson`, and `expo_expo` by combining the calculated correlation matrices and variance vectors in two ways, standard Pearson and exponentially weighted, interchangeably.
3. **Multivariate Normal Simulation:**
   - Implement a function for direct multivariate normal simulation using the Cholesky decomposition of the covariance matrix.
   - Implement a PCA-based simulation function that allows for simulating multivariate normal distributions with varying percentages of explained variance.
4. **Simulation and Comparison:**
   - For each of the four covariance matrices, simulate 25,000 draws using direct simulation and PCA-based simulation with 100%, 75%, and 50% variance explained.
   - Calculate the covariance of the simulated values.
   - Compare the simulated covariance matrix to the input covariance matrix using the Frobenius norm.
   - Record the runtime for each simulation method.
5. **Visualization and Analysis:**
   - Visualize the trade-offs between runtime and accuracy using scatter plots.

Implementation:

The implementation involved coding functions for direct multivariate normal simulation and PCA-based simulation. The `direct_sim` function used the Cholesky decomposition, ensuring the covariance matrix was positive semi-definite. The `pca_simulation` function employed eigen decomposition and allowed for varying levels of variance explained. A `simulate_and_compare` function was crafted to automate the simulations, comparisons, and record-keeping of results. The results were then visualized to depict the trade-offs between runtime and accuracy for different simulation methods and covariance matrices.

Conclusion:



The analysis of the trade-offs between runtime and accuracy for different simulation methods and covariance matrices revealed important insights:

- **Variance Explained:**
  - Reducing the percentage of variance explained in PCA simulations can significantly reduce runtime. However, it also results in a loss of accuracy, as indicated by a higher Frobenius norm.
- **Covariance Matrix Type:**
  - The type of covariance matrix used also influences the trade-offs. Different combinations of correlation matrices and variance vectors can result in varying levels of runtime and accuracy.
- **Optimal Trade-off:**
  - If accuracy is paramount, direct simulation or PCA with a high percentage of variance explained is preferable. However, if computational efficiency is more critical, PCA with reduced variance explained may be a suitable choice.