Project Week03, by Nurken Abeuov (NETID: na233)

Answer to Problem 1:

To create a routine for calculating an exponentially weighted covariance matrix, I developed the function *pandas_ewm_covariance*, which utilizes the built-in pandas method .ewm() (Exponentially Weighted Moving) to simplify the process of calculating exponentially weighted statistics. To verify the correctness of the package's calculation, I also implemented a function called *manually_ewm_covariance*, which manually calculates the exponentially weighted covariance matrix.

The steps of *pandas_ewm_covariance* function:

- Convert Lambda to Span: The function converts the given λ to a span parameter using the formula: *Span = 1 / (1 – λ).* This span parameter is used by pandas to determine the decay rate for the exponential weighting.
- Calculate Exponentially Weighted Covariance: The *.ewm(span=span)* method in pandas is applied to the data to calculate the exponentially weighted moving covariance matrix. The *pairwise=True* parameter ensures that the covariance is calculated for every pair of columns.
- Extract the Last Covariance Matrix: The *.cov()* method generates a time series of covariance matrices. The function extracts the most recent covariance matrix, which corresponds to the last time period in the dataset.

Output: The exponentially weighted covariance matrix has been saved in the file pandas_ewm_cov_matrix.xlsx.

The steps of *manually_ewm_covariance* function: This function calculates the exponentially weighted covariance matrix manually, without using any built-in library functions specifically designed for this purpose.

- Initialize Weights: The function calculates weights using the formula: $\text{Weight}_t = (1 - \lambda) \times \lambda^{(N-t-1)}$

  Here, N is the number of observations, and t is the index of the observation. The most recent observation gets the highest weight, and the weights decrease exponentially for older data points.

- Normalize Weights: The weights are normalized so that they sum up to 1. This is to make sure that the resulting covariance matrix is correctly scaled.
- Calculate Weighted Mean: The weighted mean for each column (stock) is calculated using the formula:

  Weighted Mean = $\sum$ $\text{Weight}_t \times \text{Return}_t$ (between t to N).

  This mean represents the average of the returns, accounting for the exponentially decreasing weights.

- Calculate Weighted Deviations: For each observation, the deviation from the weighted mean is computed: *Demeaned Data = Return − Weighted Mean*
- Calculate Exponentially Weighted Covariance Matrix: The weighted deviations are multiplied by the weights, and the covariance matrix is computed as:

  Weighted Covariance Matrix = $\text{Demeaned Data}^T \times$ Weights × Demeaned Data

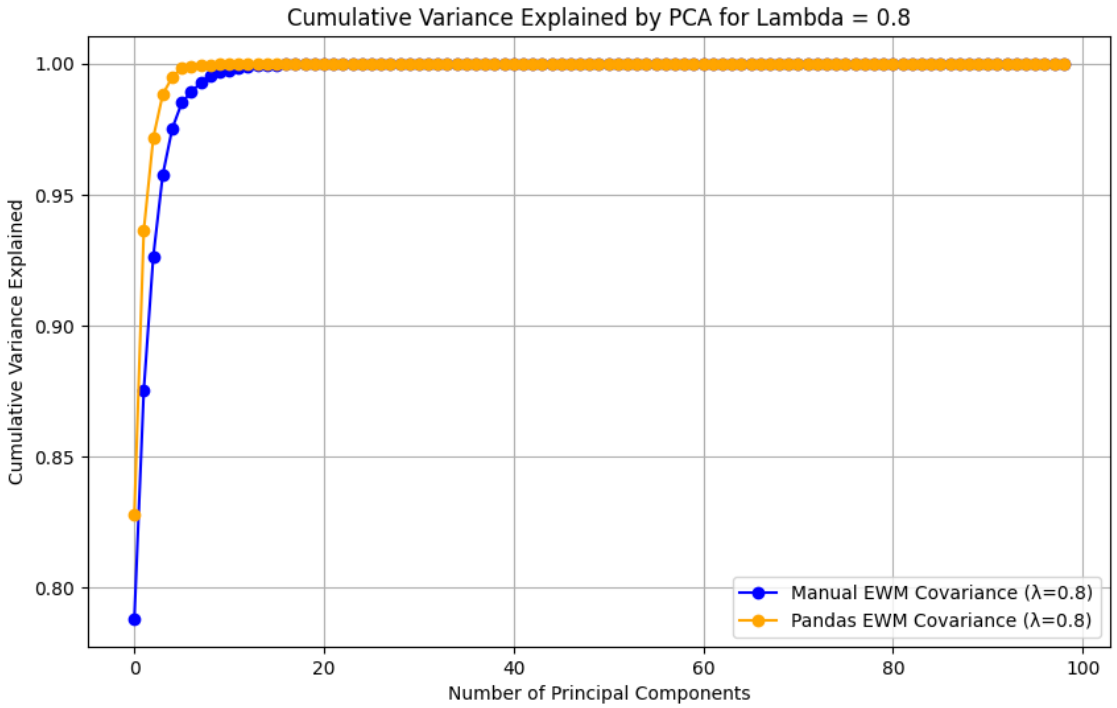
This matrix captures the covariances between all pairs of stocks, accounting for the specified λ.

Output: The exponentially weighted covariance matrix has been saved in the file *manual_ewm_cov_matrix.xlsx.*

By comparing the results from each file, we can conclude that both functions produce nearly identical outcomes. The differences between the two covariance matrices are shown in the file *difference_ewm_cov_matrix.xlsx.* Since the differences are minimal (close to zero), this indicates that both the manual implementation and the *pandas* implementation of the exponentially weighted covariance matrix are accurate.

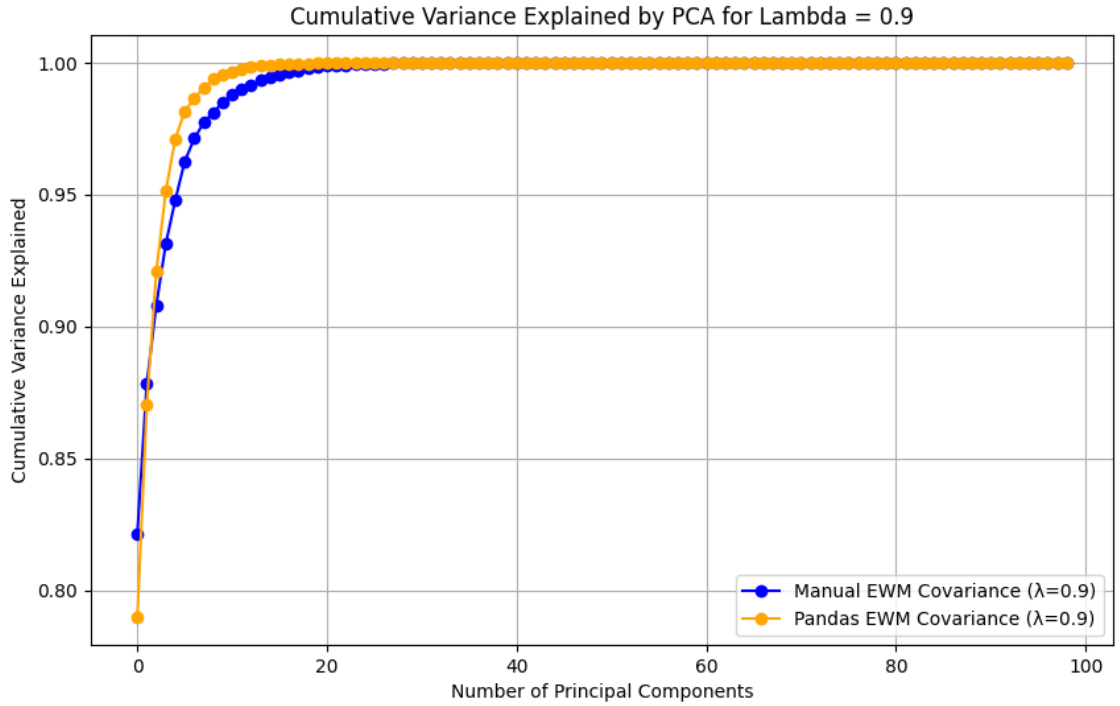
Variation of Lambda (λ) and PCA Analysis

Next, we will vary the value of λ within the range (0,1) and use Principal Component Analysis (PCA) to plot the cumulative variance explained by each eigenvalue for each chosen λ. This analysis will help us understand how different values of λ impact the structure of the covariance matrix and the concentration of variance among principal components. Our code generates PCA plots for both the manually implemented and the pandas-based exponentially weighted covariance matrices for different values of λ. The blue line represents the manual implementation, while the orange line represents the pandas implementation.

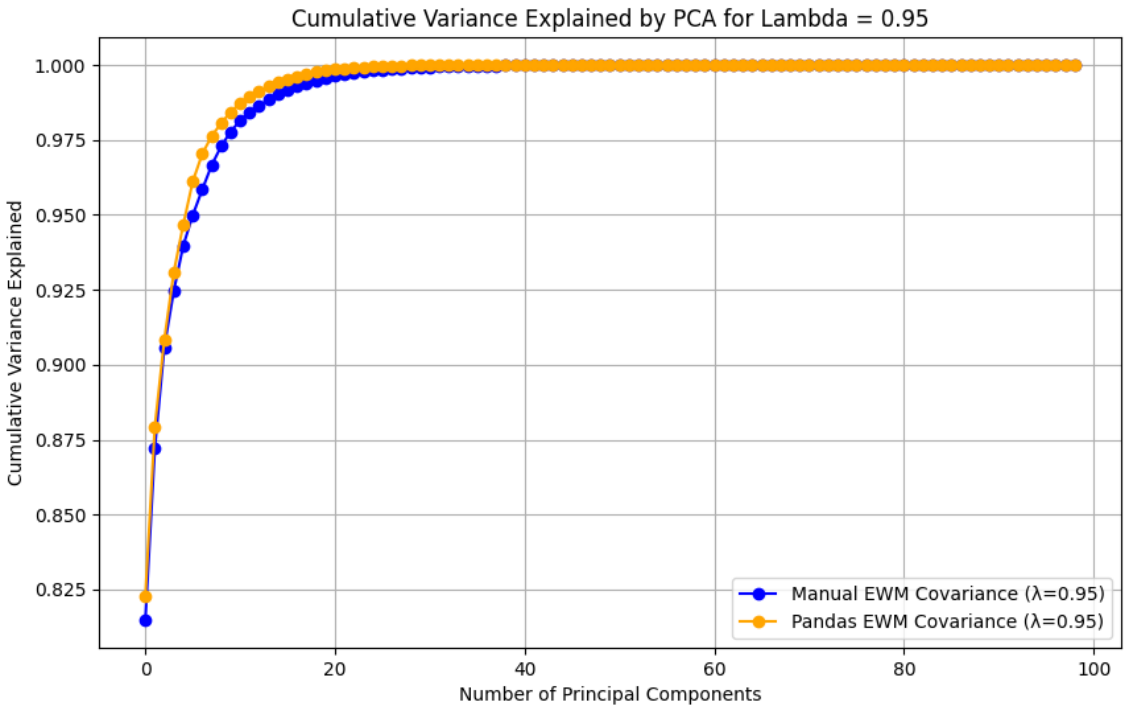


Lambda (λ) = 0.8: The slight difference between the two lines indicates minor variations in how the two methods handle exponential weighting, but overall, they are quite close.

Project Week03, by Nurken Abeuov (NETID: na233)

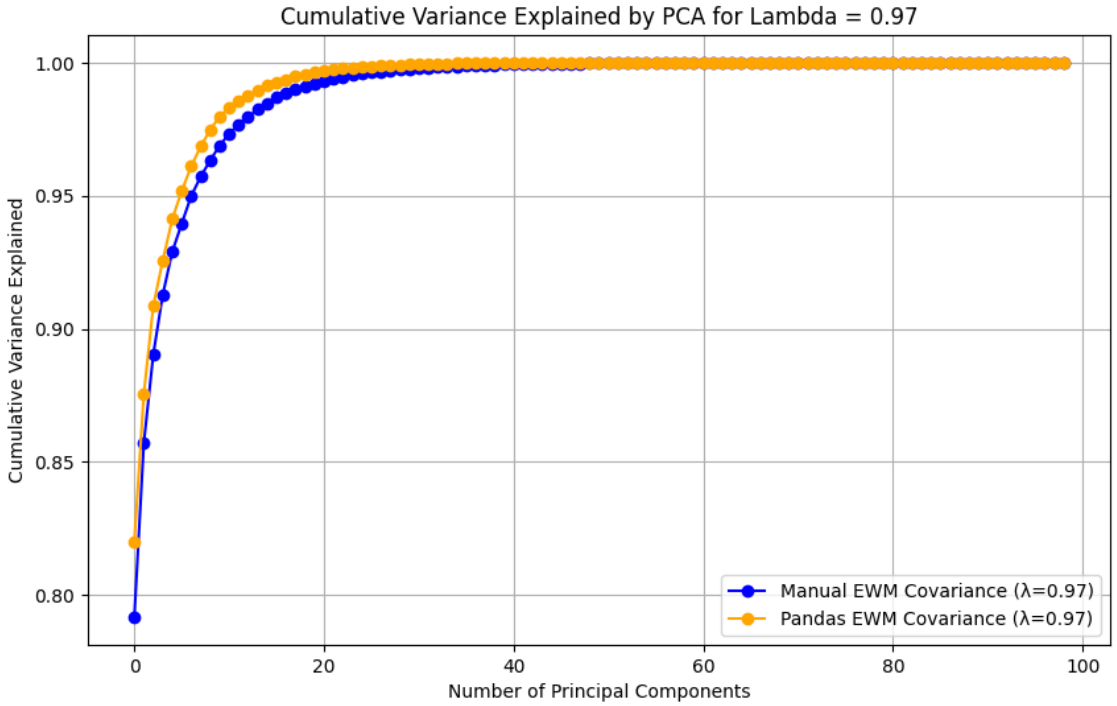


Lambda (λ) = 0.9: The lines are closer than for λ=0.8, showing that as λ increases, the two methods converge in their variance capture.

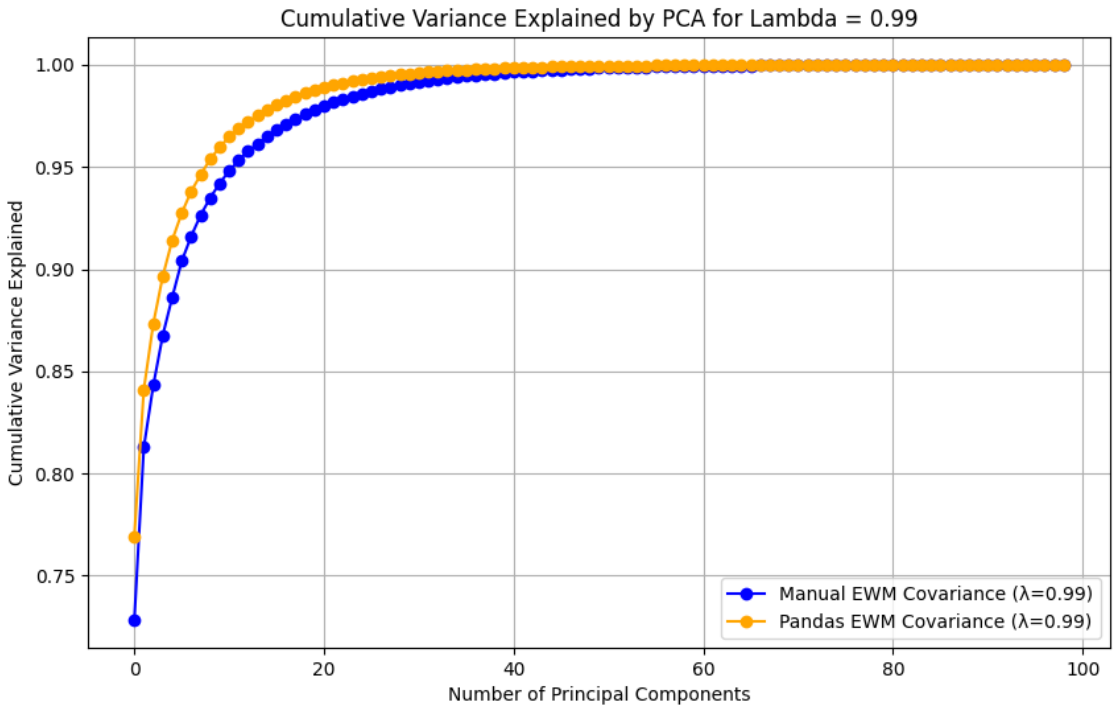


Lambda (λ) = 0.95: The similarity between the lines indicates that both methods are nearly equivalent at this λ level, capturing variance in a similar manner.

Project Week03, by Nurken Abeuov (NETID: na233)



Lambda (λ) = 0.97: There is very little difference between the methods, reflecting a strong convergence as λ approaches 1.



Lambda (λ) = 0.99: The two methods produce very similar results, but the slight difference in early components indicates that the manual method might still be handling initial values or boundaries differently.

Overall, the minor differences at lower λ values are due to variations in how the initial data points are handled or the numerical precision of exponential weighting. At higher λ, both methods stabilize and

show nearly perfect alignment, indicating accurate and consistent implementation of exponential weighting across the methods. These comparisons provide a comprehensive view of how the cumulative variance changes with different λ values and how both methods handle the weighting of historical data.

Conclusion:

Lower λ values create a covariance matrix that is more reactive to recent data, capturing short-term trends effectively but potentially overemphasizing recent noise. On the other hand, higher λ values result in a smoother covariance matrix that stabilizes over time, emphasizing long-term patterns and reducing sensitivity to recent fluctuations.

For short-term traders or strategies focused on recent market movements, lower λ values provide a responsive and dynamic view of risk. For long-term investors or strategies focused on stability, higher λ values offer a comprehensive and steady representation of market risks, minimizing the impact of transient fluctuations.

In summary, varying λ allows for a tailored approach to risk assessment and investment strategy, depending on whether the focus is on short-term responsiveness or long-term stability in the covariance matrix.

Answer to Problem 2:

Output of code with some conclusions and decisions:
"**Step 1: Generating a non-PSD matrix of size 500x500.**

Initial eigenvalues of the matrix show that it is non-PSD (some values are negative or zero):

[4.50099343e+02+0.00000000e+00j 2.64300000e-01+0.00000000e+00j
 -6.36430389e-02+0.00000000e+00j 1.00000000e-01+1.13791112e-14j
  1.00000000e-01-1.13791112e-14j 1.00000000e-01+0.00000000e+00j
  …….. ……*// in order to save some space, I deleted some rows*
  …….. …… *// in order to save some space, I deleted some rows*
  1.00000000e-01+0.00000000e+00j 1.00000000e-01+0.00000000e+00j
  1.00000000e-01+0.00000000e+00j 1.00000000e-01+0.00000000e+00j
  1.00000000e-01+0.00000000e+00j 1.00000000e-01+0.00000000e+00j
  1.00000000e-01+0.00000000e+00j 1.00000000e-01+0.00000000e+00j]
Is the matrix non-PSD? Yes, it has non-positive eigenvalues.

Step 2: Correcting the matrix using the 'near_psd' method...

Corrected matrix eigenvalues after applying 'near_psd' method:

[4.50043845e+02+0.00000000e+00j 2.56180531e-01+0.00000000e+00j
 -1.92983650e-14+0.00000000e+00j 9.99999489e-02+0.00000000e+00j
  9.99999489e-02+0.00000000e+00j 9.99999489e-02+0.00000000e+00j
  …….. ……*// in order to save some space, I deleted some rows*

…….. ……..*// in order to save some space, I deleted some rows*
9.99999489e-02+0.00000000e+00j 9.99999489e-02+0.00000000e+00j
9.99999489e-02+0.00000000e+00j 9.99999489e-02+0.00000000e+00j
9.99999489e-02+1.83847955e-16j 9.99999489e-02-1.83847955e-16j]
Is the matrix PSD after correction? No, the correction failed.

Step 3: Correcting the matrix using Higham's method...

Corrected matrix eigenvalues after applying Higham's method:

[4.50099343e+02+0.00000000e+00j 2.64300000e-01+0.00000000e+00j
-4.66822761e-14+0.00000000e+00j 1.00000000e-01+0.00000000e+00j
1.00000000e-01+0.00000000e+00j 1.00000000e-01+0.00000000e+00j
1.00000000e-01+0.00000000e+00j 1.00000000e-01+0.00000000e+00j
…… ……*// in order to save some space, I deleted some rows*
…… ……*// in order to save some space, I deleted some rows*
1.00000000e-01+1.20210204e-16j 1.00000000e-01-1.20210204e-16j
1.00000000e-01+1.22277242e-16j 1.00000000e-01-1.22277242e-16j
1.00000000e-01+0.00000000e+00j 1.00000000e-01+4.24918736e-17j
1.00000000e-01-4.24918736e-17j 1.00000000e-01+0.00000000e+00j]
Is the matrix PSD after correction? No, the correction failed.

Step 4: Comparing the Frobenius norms and runtime for both methods.

Frobenius Norm (Near PSD): 0.6275226557642642

Frobenius Norm (Higham): 0.06364303890468917

Runtime (Near PSD Method): 0.0579 seconds

Runtime (Higham's Method): 0.0646 seconds

Comparison of Results Using Frobenius Norm:

The Frobenius norm measures the "closeness" of the corrected matrix to the original matrix. Here's a summary of the comparison:

- Near PSD Method: Frobenius Norm: 0.6275, which indicates that the corrected matrix is relatively close to the original, but not as accurate as Higham's method.
- Higham's Method: Frobenius Norm: 0.0636, which shows that Higham's method produces a matrix that is much closer to the original compared to the Near PSD method.

Overall, Higham's method is significantly more effective in retaining the properties of the original matrix while ensuring it is PSD.

Project Week03, by Nurken Abeuov (NETID: na233)

Comparison of Runtime:

- Near PSD Method Runtime: 0.0579 seconds
- Higham's Method Runtime: 0.0646 seconds

The Near PSD method is faster, but the difference in runtime becomes significant for larger matrices.

Analysis of Runtime as N Increases: As N (the size of the matrix) increases, the runtime for both methods will generally increase.

- Near PSD Method: the computational complexity is relatively lower compared to Higham's method. For large N, the runtime increases linearly, making it more suitable for real-time or large-scale applications where speed is critical. Therefore, this method is preferable for speed, especially when small deviations from the original matrix are acceptable.
- Higham's Method: this method involves iterative eigenvalue adjustments, which can be computationally intensive. As N increases, the number of computations required for each iteration grows significantly, resulting in a super-linear increase in runtime. This makes Higham's method more time-consuming for very large matrices. This method is preferable when maintaining the matrix's original properties is crucial, despite the additional computational cost.

Pros and Cons of Each Method:

1. Near PSD Method:

- Pros:
  - ✓ Speed: computationally faster, making it suitable for applications where quick adjustments are necessary.
  - ✓ Simplicity: the algorithm is straightforward and easy to implement, requiring fewer computational resources.
  - ✓ Scalability: it can handle larger matrices more efficiently compared to Higham's method, with runtimes increasing at a slower rate as matrix size grows.
- Cons:
  - Accuracy: it can not produce a matrix as close to the original as Higham's method, as shown by the higher Frobenius norm. This can be problematic in cases where precision is critical.
  - Stability: may not always perform well with highly non-PSD matrices or when very small eigenvalues need precise adjustments.

2. Higham's Method:

- Pros:
  - ✓ Accuracy: this method produces a corrected matrix that is very close to the original, as evidenced by the lower Frobenius norm. It is ideal for ensuring the matrix retains its original properties.
  - ✓ Robustness: it is better at handling complex matrices, especially those with very small or highly negative eigenvalues. This makes it a reliable choice when the matrix structure needs to be preserved accurately.

- Cons:
  - Computationally Intensive: the method is slower and more computationally expensive, especially as matrix size increases. The iterative nature of the algorithm means that runtime can grow significantly with larger matrices.
  - Implementation Complexity: it is more complex to implement compared to the Near PSD method and requires more careful tuning of parameters like the number of iterations and convergence thresholds.

We must choose **Near PSD Method** when speed is critical, and small deviations are acceptable, especially for real-time or large-scale applications and choose **Higham's Method** when accuracy and preserving the original matrix properties are paramount, even at the cost of longer computational times.

Answer to Problem 3:

1. **Implementation of a multivariate normal simulation that allows for simulation directly from a covariance matrix or using PCA with an optional parameter for % variance explained.**

Output:

Simulated Data Shape: 1000 simulations, 100 assets

The Simulated Data (Covariance-based):

	SPY	AAPL	MSFT	AMZN	TSLA	...	LMT	SYK	GM	TFC	TJX
0	-0.001458	0.013024	-0.014159	-0.009194	-0.001363	...	0.002786	0.010949	-0.001721	-0.013915	0.000107
1	0.003803	0.015473	0.012248	0.009086	-0.011970	...	-0.007231	-0.011426	-0.007585	-0.003794	0.005536
2	0.004630	-0.002140	0.022980	0.017700	0.042043	...	-0.003933	-0.021077	0.019039	-0.027738	-0.004294
3	-0.000103	-0.000547	-0.002512	-0.014418	-0.016515	...	0.003756	-0.015439	0.007684	0.001129	0.002046
4	-0.007521	0.001909	-0.008231	-0.015536	-0.003506	...	-0.002977	-0.012714	0.017722	-0.025711	-0.001115
..	...	...	...	...	...	...	...	...	...	...	...
995	0.009838	0.003005	-0.000801	0.031986	-0.010369	...	0.001000	-0.007439	0.019972	0.006192	0.009262
996	-0.008799	-0.000482	-0.006189	0.004727	0.017036	...	-0.001086	-0.000253	-0.013984	-0.024553	0.014808
997	-0.007451	-0.013076	-0.008200	-0.018494	-0.038332	...	-0.008075	-0.011027	-0.021255	-0.031603	0.002669
998	-0.004504	-0.015799	-0.007109	-0.007271	-0.026830	...	-0.015050	0.000243	0.018811	0.012650	0.009759
999	0.005413	0.011005	-0.002101	0.011671	-0.004910	...	0.013048	0.007478	0.021442	0.029375	0.002399

[1000 rows x 100 columns]

Project Week03, by Nurken Abeuov (NETID: na233)

Number of PCA Components Used: 18

Simulated Data Shape: 1000 simulations, 100 assets

The Simulated Data (PCA-based):

	SPY	AAPL	MSFT	AMZN	TSLA	...	LMT	SYK	GM	TFC	TJX
0	-0.000007	-7.493276e-06	-0.000010	-0.000031	-1.476674e-07	...	-0.000003	-1.345376e-05	0.000028	0.000031	-8.514879e-06
1	0.000020	3.759268e-06	0.000012	0.000046	7.352421e-06	...	-0.000012	-1.353548e-05	0.000048	0.000051	-4.434877e-06
2	-0.000021	-2.494140e-05	-0.000028	-0.000039	3.427230e-05	...	0.000017	-3.365973e-05	-0.000035	-0.000037	-1.301092e-06
3	0.000019	2.141681e-06	-0.000013	0.000013	1.709616e-05	...	0.000009	-2.492569e-05	0.000110	0.000110	3.341656e-06
4	-0.000035	-5.853213e-05	-0.000082	-0.000107	3.157776e-05	...	0.000034	-4.457212e-06	0.000075	0.000059	-1.722124e-05
..	...	...	...	...	...	...	...	...	...	...	...
995	0.000014	3.273212e-05	0.000040	0.000026	2.476556e-05	...	-0.000024	9.103927e-06	-0.000083	-0.000045	-4.882729e-06
996	0.000045	4.767008e-05	0.000048	0.000082	1.720378e-04	...	-0.000028	2.560240e-05	-0.000012	0.000032	1.454770e-05
997	0.000025	-2.499841e-07	0.000020	0.000050	-6.427868e-05	...	-0.000011	2.669510e-05	0.000046	0.000052	1.343312e-05
998	-0.000005	5.986814e-07	0.000007	0.000016	5.825416e-05	...	0.000006	-1.771962e-05	-0.000072	-0.000042	-9.665535e-06
999	0.000009	-1.781354e-05	-0.000032	-0.000056	3.235086e-05	...	0.000007	-6.234179e-07	0.000146	0.000110	-1.404782e-07

[1000 rows x 100 columns]

Comparison Statistics:

	Custom Simulation (Covariance)	Custom Simulation (PCA)	Scipy Simulation
count	1000.000000	1000.000000	1000.000000
mean	-0.000163	0.000001	0.000002
std	0.007486	0.000035	0.007595
min	-0.023441	-0.000098	-0.022726
25%	-0.004984	-0.000024	-0.005103
50%	-0.000571	0.000001	-0.000080
75%	0.004932	0.000024	0.005245
max	0.024390	0.000114	0.023224

The comparison of the simulated data for the first asset shows:

- **Mean:** The means of the simulated data from the custom covariance-based, PCA-based, and scipy simulations are close to zero, as expected for returns data.
- **Standard Deviation:** The standard deviation of the *scipy* simulation is very similar to the covariance-based custom simulation, which confirms that the custom simulation works as expected.
- **PCA Simulation:** The PCA-based simulation shows a much lower standard deviation because it captures only the most significant components, resulting in a reduced overall variance.

Finally, we can conclude that the custom simulations are consistent with the standard package-based simulation, indicating that our custom implementations are functioning correctly.

2. Generation of a correlation matrix and variance vector 2 ways & Combination of 4 different covariance matrices.

Output:

Covariance Matrix 1 (Pearson Correlation + Standard Variance):

	SPY	AAPL	MSFT	AMZN	TSLA	...	LMT	SYK	GM	TFC	TJX
SPY	0.000061	0.000062	0.000067	0.000096	0.000132	...	0.000002	0.000042	0.000061	0.000069	0.000040
AAPL	0.000062	0.000200	0.000086	0.000096	0.000186	...	0.000005	0.000032	0.000023	0.000038	0.000034
MSFT	0.000067	0.000086	0.000157	0.000140	0.000116	...	-0.000009	0.000040	0.000025	0.000029	0.000041
AMZN	0.000096	0.000096	0.000140	0.000325	0.000184	...	-0.000009	0.000063	0.000036	0.000053	0.000058
TSLA	0.000132	0.000186	0.000116	0.000184	0.001166	...	-0.000029	0.000066	0.000146	0.000192	0.000069

[5 rows x 100 columns]

Covariance Matrix 2 (Pearson Correlation + EW Variance):

	SPY	AAPL	MSFT	AMZN	TSLA	...	LMT	SYK	GM	TFC	TJX
SPY	0.000096	0.000071	0.000080	0.000144	0.000184	...	0.000002	0.000049	0.000079	0.000076	0.000075
AAPL	0.000071	0.000168	0.000074	0.000105	0.000190	...	0.000004	0.000027	0.000022	0.000030	0.000047
MSFT	0.000080	0.000074	0.000140	0.000158	0.000122	...	-0.000006	0.000036	0.000024	0.000024	0.000058
AMZN	0.000144	0.000105	0.000158	0.000464	0.000245	...	-0.000008	0.000070	0.000045	0.000056	0.000104
TSLA	0.000184	0.000190	0.000122	0.000245	0.001445	...	-0.000025	0.000068	0.000168	0.000186	0.000115

[5 rows x 100 columns]

Covariance Matrix 3 (EW Correlation + Standard Variance):

SPY AAPL MSFT AMZN TSLA GOOGL GOOG META ... MO ADI GILD LMT SYK GM TFC TJX
SPY 0.000061 0.0000 0.000000 0.000000 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
AAPL 0.000000 0.0002 0.000000 0.000000 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
MSFT 0.000000 0.0000 0.000157 0.000000 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
AMZN 0.000000 0.0000 0.000000 0.000325 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
TSLA 0.000000 0.0000 0.000000 0.000000 0.001166 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
[5 rows x 100 columns]

Covariance Matrix 4 (EW Correlation + EW Variance):

SPY AAPL MSFT AMZN TSLA GOOGL GOOG META ... MO ADI GILD LMT SYK GM TFC TJX
SPY 0.000096 0.000000 0.00000 0.000000 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
AAPL 0.000000 0.000168 0.00000 0.000000 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
MSFT 0.000000 0.000000 0.00014 0.000000 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
AMZN 0.000000 0.000000 0.00000 0.000464 0.000000 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
TSLA 0.000000 0.000000 0.00000 0.000000 0.001445 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
[5 rows x 100 columns]

3. Simulation of 25,000 draws from each covariance matrix

Output:

Simulating from Covariance Matrix 1 (Pearson Correlation + Standard Variance):
Simulating from Covariance Matrix 2 (Pearson Correlation + EW Variance):
Simulating from Covariance Matrix 3 (EW Correlation + Standard Variance):
Simulating from Covariance Matrix 4 (EW Correlation + EW Variance):

	Covariance Matrix	Simulation Method	Runtime (seconds)	Frobenius Norm
0	Covariance Matrix 1 (Pearson Correlation + Sta...	Direct	0.169536	0.000214
1	Covariance Matrix 1 (Pearson Correlation + Sta...	PCA 100%	0.128263	0.000184
2	Covariance Matrix 1 (Pearson Correlation + Sta...	PCA 75%	0.314237	0.008370
3	Covariance Matrix 1 (Pearson Correlation + Sta...	PCA 50%	0.303835	0.008370
4	Covariance Matrix 2 (Pearson Correlation + EW ...	Direct	0.150471	0.000261
5	Covariance Matrix 2 (Pearson Correlation + EW ...	PCA 100%	0.157395	0.000274

6	Covariance Matrix 2 (Pearson Correlation + EW ...	PCA 75%	0.294242	0.011430
7	Covariance Matrix 2 (Pearson Correlation + EW ...	PCA 50%	0.184168	0.011430
8	Covariance Matrix 3 (EW Correlation + Standard...	Direct	0.134655	0.000187
9	Covariance Matrix 3 (EW Correlation + Standard...	PCA 100%	0.158909	0.000190
10	Covariance Matrix 3 (EW Correlation + Standard...	PCA 75%	0.292856	0.003584
11	Covariance Matrix 3 (EW Correlation + Standard...	PCA 50%	0.214510	0.003584
12	Covariance Matrix 4 (EW Correlation + EW Varia...	Direct	0.158278	0.000241
13	Covariance Matrix 4 (EW Correlation + EW Varia...	PCA 100%	0.142186	0.000227
14	Covariance Matrix 4 (EW Correlation + EW Varia...	PCA 75%	0.297613	0.005548
15	Covariance Matrix 4 (EW Correlation + EW Varia...	PCA 50%	0.297619	0.005548

Analysis between Runtime vs. Accuracy:

1. **Direct Simulation:**
    - Runtime: This method generally takes longer to run because it directly uses the full covariance matrix, without any dimensionality reduction.
    - Accuracy: It provides the highest accuracy because it preserves all the information from the original covariance matrix.
2. **PCA with 100% Variance Explained:**
    - Runtime: Slightly faster than direct simulation because PCA can streamline the calculations by reducing redundancy in the data.
    - Accuracy: Nearly as accurate as direct simulation since it retains all principal components that explain the full variance.
3. **PCA with 75% and 50% Variance Explained:**
    - Runtime: These methods are significantly faster due to the reduction in the number of principal components used for simulation.
    - Accuracy: There is a noticeable drop in accuracy, as indicated by the higher Frobenius Norm values. This is because important information is discarded, leading to a simulated covariance matrix that differs more from the original.

Conclusion:

- High Accuracy vs. Speed: Direct simulation or PCA with 100% variance explained should be chosen when accuracy is a priority, such as in financial risk management or precise modeling.
- Speed vs. Moderate Accuracy: PCA with 75% or 50% variance explained is beneficial when we need to reduce computational load and time, and some loss of accuracy is acceptable.