Assignment - Week 03

Problem 1

Use the stock returns in `DailyReturn.csv` for this problem. `DailyReturn.csv` contains returns for 100 large US stocks 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. 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 the values of λ and its effect on the covariance matrix?

Solution

	SPY	AAPL	MSFT	AMZN	TSLA	GOOGL	GOOG	META	NVDA	BRK-B	...	PNC	MDLZ	MO	ADI	GILD
LMT	0.000185	0.000868	0.000056	0.000376	0.000406	0.000259	0.000264	0.000138	0.000856	-0.000060	...	0.000436	-3.858118e-05	0.000331	-0.000062	0.000277
SYK	-0.000017	-0.000130	0.000025	0.000013	0.000151	-0.000033	-0.000033	0.000053	-0.000164	0.000019	...	-0.000096	5.406144e-05	-0.000050	0.000025	-0.000021
GM	0.000061	0.000342	0.000041	0.000045	0.000038	0.000069	0.000068	0.000009	0.000301	-0.000013	...	0.000257	-4.567227e-05	0.000110	0.000013	0.000131
TFC	0.000124	0.000620	0.000108	0.000129	0.000265	0.000127	0.000126	0.000102	0.000489	-0.000015	...	0.000380	-2.888986e-05	0.000186	0.000053	0.000242
TJX	0.000034	0.000139	-0.000002	0.000110	0.000093	0.000062	0.000064	0.000025	0.000151	-0.000012	...	0.000074	9.531857e-10	0.000079	-0.000047	0.000058

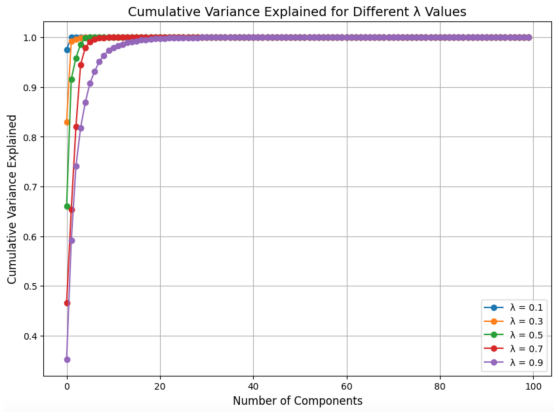
5 rows × 100 columns

I set λ to affect the weight of historical data on the overall volatility, with more recent data weighted more heavily. I normalized all weights to ensure they sum to one. The covariance matrix was then obtained.

```
              SPY      AAPL      MSFT      AMZN      TSLA     GOOGL      GOOG  \
248 LMT -0.000015  0.000007 -0.000012 -0.000044 -0.000060 -0.000022 -0.000023
    SYK  0.000027  0.000011  0.000030  0.000072  0.000078  0.000032  0.000033
    GM   0.000023  0.000037  0.000026  0.000035  0.000079  0.000023  0.000024
    TFC  0.000057 -0.000042  0.000038  0.000180  0.000251  0.000098  0.000102
    TJX  0.000034 -0.000064  0.000004  0.000079  0.000103  0.000059  0.000062

              META      NVDA     BRK-B  ...       PNC      MDLZ        MO  \
248 LMT -0.000008 -0.000074 -0.000008  ... -0.000002 -0.000021 -0.000006
    SYK  0.000016  0.000078  0.000014  ... -0.000010  0.000021  0.000006
    GM   0.000006  0.000041  0.000003  ... -0.000005 -0.000002  0.000007
    TFC  0.000021  0.000295  0.000030  ...  0.000023  0.000096  0.000026
    TJX  0.000005  0.000330  0.000019  ...  0.000005  0.000079  0.000021

              ADI      GILD       LMT       SYK       GM       TFC       TJX
248 LMT -0.000025  0.000021  0.000006 -0.000008 -0.000011 -0.000020 -0.000016
    SYK  0.000086 -0.000050 -0.000008  0.000034  0.000026  0.000013  0.000012
    GM   0.000095 -0.000021 -0.000011  0.000026  0.000061  0.000038  0.000005
    TFC  0.000047 -0.000071 -0.000020  0.000013  0.000038  0.000113  0.000050
    TJX  0.000009 -0.000065 -0.000016  0.000012  0.000005  0.000050  0.000095

[5 rows x 100 columns]
```

Using the pandas function `data.ewm(alpha=0.5).cov()`, I verified the accuracy of my manually calculated covariance matrix. The difference between

the two matrices was less than 10^{-4}, so I concluded that my calculation was correct.



The graph shows that with a small λ, the curve moves more steeply and flattens out with fewer principal components. This suggests that recent data is more volatile and contributes more to the overall variance, requiring fewer principal components to explain most of the variance.

```
Recent 30 Days Standard Deviation:
SPY    0.0119037713
AAPL   0.0137628610
MSFT   0.0139639933
AMZN   0.0245421447
TSLA   0.0430763794
          ...
LMT    0.0122140068
SYK    0.0149071324
GM     0.0250492323
TFC    0.0163816598
TJX    0.0165444878
Length: 100, dtype: float64

Previous 30 Days Standard Deviation:
SPY    0.0053881237
AAPL   0.0203070260
MSFT   0.0106120033
AMZN   0.0158121554
TSLA   0.0362550396
          ...
LMT    0.0061121290
SYK    0.0114567526
GM     0.0160331886
TFC    0.0158409777
TJX    0.0064955143
Length: 100, dtype: float64
```

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 of λ regulates 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.

When λ is 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.

When λ is 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 and implement them in your programming language of choice. Implement Higham's 2002 nearest PSD correlation function. Generate a non-PSD correlation matrix that is 500x500 using the code from class.

```
n=500
sigma = fill(0.9,(n,n))
for i in 1:n
    sigma[i,i]=1.0
end
sigma[1,2] = 0.7357
sigma[2,1] = 0.7357
```

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 and compare the run times as N increases. Discuss the pros and cons of each method and when you would use each.

Solution

```
Initial matrix shape: (500, 500)
Matrix shape after standardization: (500, 500)
Eigenvalues shape: (500,), Eigenvectors shape: (500, 500)
Corrected eigenvalues (first 10): [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
T matrix shape: (500, 500)
Output matrix shape (before inverse scaling): (500, 500)
near_psd Time: 0.27064 秒
Eigenvalues after near_psd correction (first 20): [-3.55719557e-14 -1.85702596e-15 -1.85163801e-15 -1.84179014e-15
 -1.75729832e-15 -1.74671236e-15 -1.73937336e-15 -1.71853229e-15
 -1.67169279e-15 -1.65056345e-15 -1.62619628e-15 -1.59847339e-15
 -1.58412735e-15 -1.58026327e-15 -1.52064622e-15 -1.49299268e-15
 -1.48214780e-15 -1.46823355e-15 -1.45142832e-15 -1.44236244e-15]
```

```
Eigenvalues shape: (500,), Eigenvectors shape: (500, 500)
Corrected eigenvalues (first 10): [1.e-08 1.e-08 1.e-08 1.e-08 1.e-08 1.e-08 1.e-08 1.e-08 1.e-08 1.e-08]
Higham Time: 0.23192 s
Eigenvalues after Higham correction (first 20): [9.99993712e-09 9.99999606e-09 9.99999749e-09 9.99999770e-09
 9.99999777e-09 9.99999783e-09 9.99999784e-09 9.99999786e-09
 9.99999794e-09 9.99999795e-09 9.99999799e-09 9.99999802e-09
 9.99999805e-09 9.99999808e-09 9.99999811e-09 9.99999814e-09
 9.99999817e-09 9.99999819e-09 9.99999824e-09 9.99999826e-09]
```

```
near_psd Frobenius norm: 146.62583, Time: 0.14494 s
Higham Frobenius norm: 19.53152, Time: 0.23192 s
```

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 $O(N\hat{3})$. 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 $O(N\hat{3})$, 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 from a covariance matrix or using PCA with an optional parameter for % variance explained. Generate a correlation matrix and variance vector in two ways:

1. Standard Pearson correlation/variance.
2. Exponentially weighted $\lambda = 0.97$.

Simulate 25,000 draws from each covariance matrix using:

- Direct simulation.

- PCA with 100% explained.
- PCA with 75% explained.
- PCA with 50% explained.

Compare the simulated covariance to the input matrix using the Frobenius Norm and compare the run times.

Solution

```
Simulated PCA Pearson 100% shape: (25000, 100)
Simulated PCA Pearson 75% shape: (25000, 100)
Simulated PCA Pearson 50% shape: (25000, 100)
```

```
Frobenius Norm - Direct Pearson: 7.952468191758981e-08
Frobenius Norm - Direct EW: 7.293827330947085e-09
```

```
Number of components to retain 100.0% variance: 100
Shape of selected eigenvalues: (100,)
Shape of selected eigenvectors: (100, 100)
Shape of simulated data: (25000, 100)
Number of components to retain 75.0% variance: 14
Shape of selected eigenvalues: (14,)
Shape of selected eigenvectors: (100, 14)
Shape of simulated data: (25000, 100)
Number of components to retain 50.0% variance: 4
Shape of selected eigenvalues: (4,)
Shape of selected eigenvectors: (100, 4)
Shape of simulated data: (25000, 100)
Simulated PCA Pearson 100% shape: (25000, 100)
Simulated PCA Pearson 75% shape: (25000, 100)
Simulated PCA Pearson 50% shape: (25000, 100)
```

```
Frobenius Norm - PCA Pearson 100%: 9.676758915795301e-08
Frobenius Norm - PCA Pearson 75%: 5.25237212359378e-07
Frobenius Norm - PCA Pearson 50%: 1.1081079736883782e-06
```

```
Number of components to retain 100.0% variance: 100
Shape of selected eigenvalues: (100,)
Shape of selected eigenvectors: (100, 100)
Shape of simulated data: (25000, 100)
PCA 100% variance simulation time: 0.24585 seconds
Number of components to retain 75.0% variance: 14
Shape of selected eigenvalues: (14,)
Shape of selected eigenvectors: (100, 14)
Shape of simulated data: (25000, 100)
PCA 75% variance simulation time: 0.28484 seconds
Number of components to retain 50.0% variance: 4
Shape of selected eigenvalues: (4,)
Shape of selected eigenvectors: (100, 4)
Shape of simulated data: (25000, 100)
PCA 50% variance simulation time: 0.07297 seconds
```

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.