Week 3 Project Solutions

Peilin Luo

Problem 1

To generate a series of weights and cumulative weights, I wrote the function *populate_weights*. Then I wrote the function *populate_exp_weighted_cov_matrix* to calculate the exponentially weighted covariance matrix. The function *conduct_pca* would return all the positive eigenvalues of the input matrix in descending order.

Plotting the exponential weights and cumulative exponential weights for different λ, we can conclude that as λ decreases, periods that are more recent are given more weights. From Figure 1 to Figure 4, the lines become steeper.

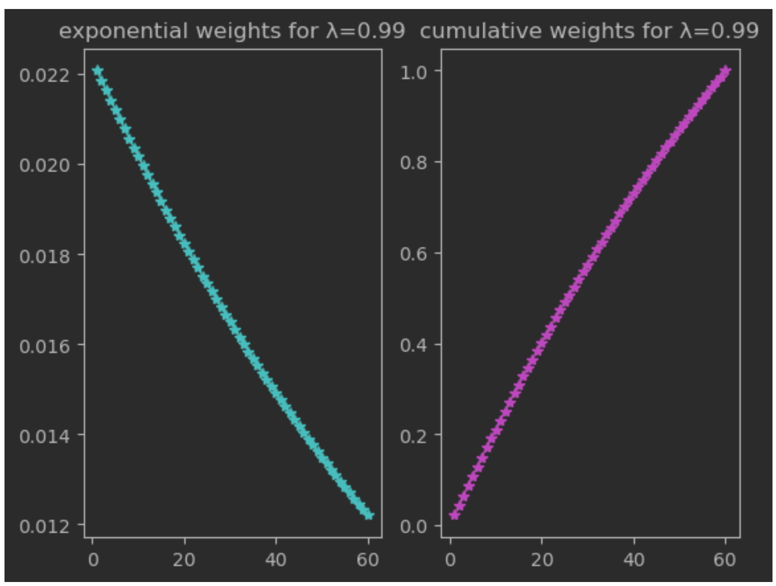

Figure 1

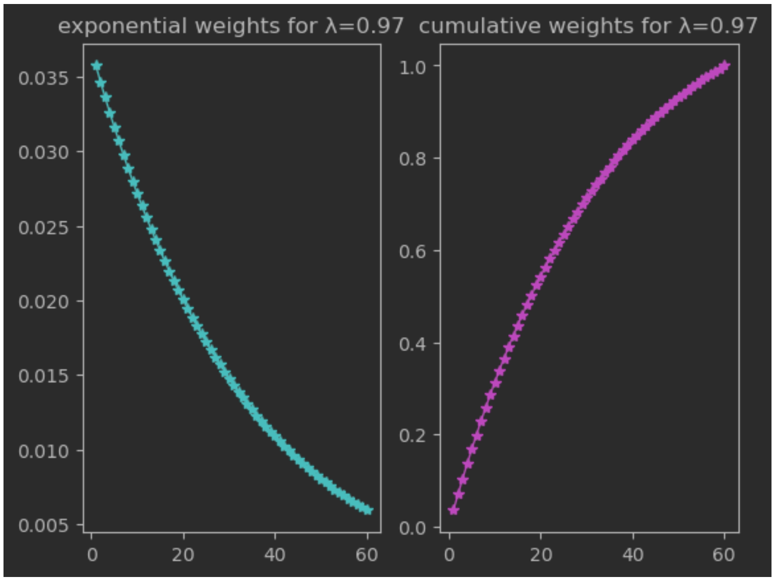

Figure 2

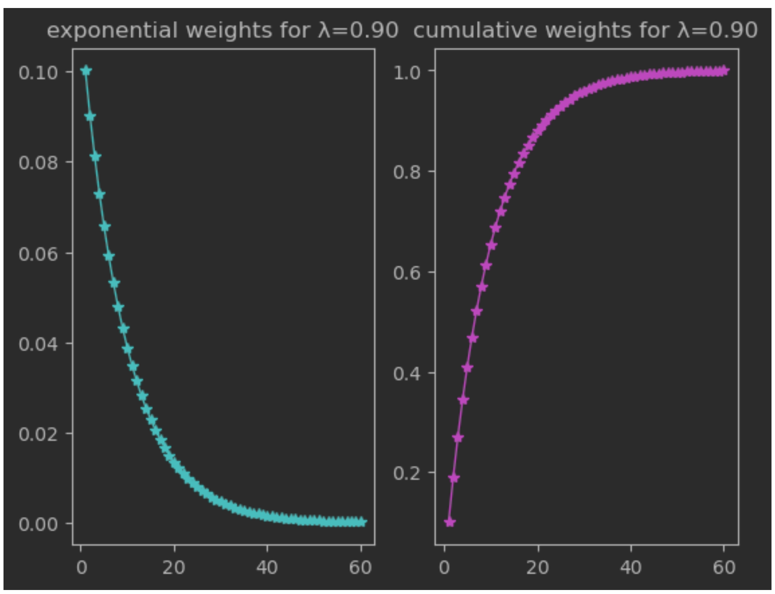

Figure 3

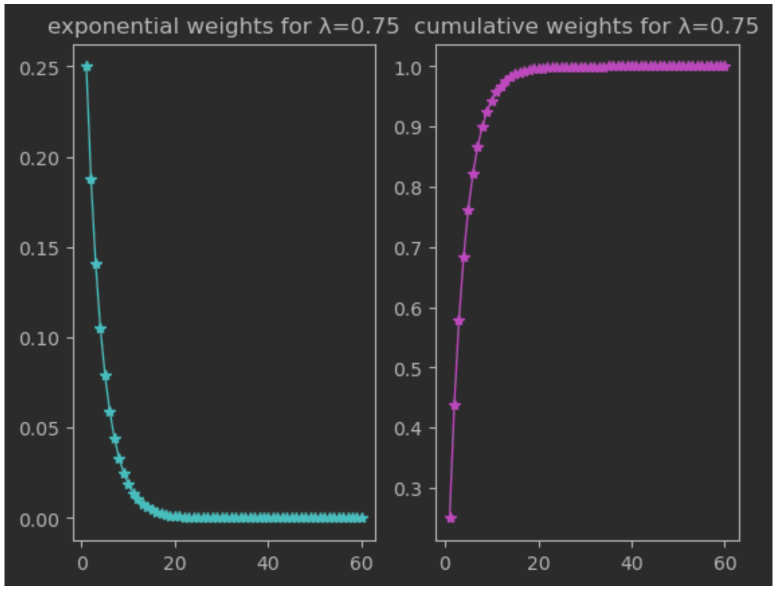

Figure 4

Using PCA and plotting the cumulative variance explained by each eigenvalue for each λ chosen, we can see from Figure 5 that as λ decreases, the line becomes steeper, just like the cumulative weight line in Figures 1 to 4. From table 1, we can conclude that **as λ decreases, the number of positive eigenvalues of the covariance matrix decreases**. At the same time, from Figure 5, the **top N eigenvalues could explain more cumulative variance**, given N is a constant integer.

Table 1

λ	number of positive eigenvalues
0.99	59
0.97	59
0.90	59
0.75	42
0.50	20

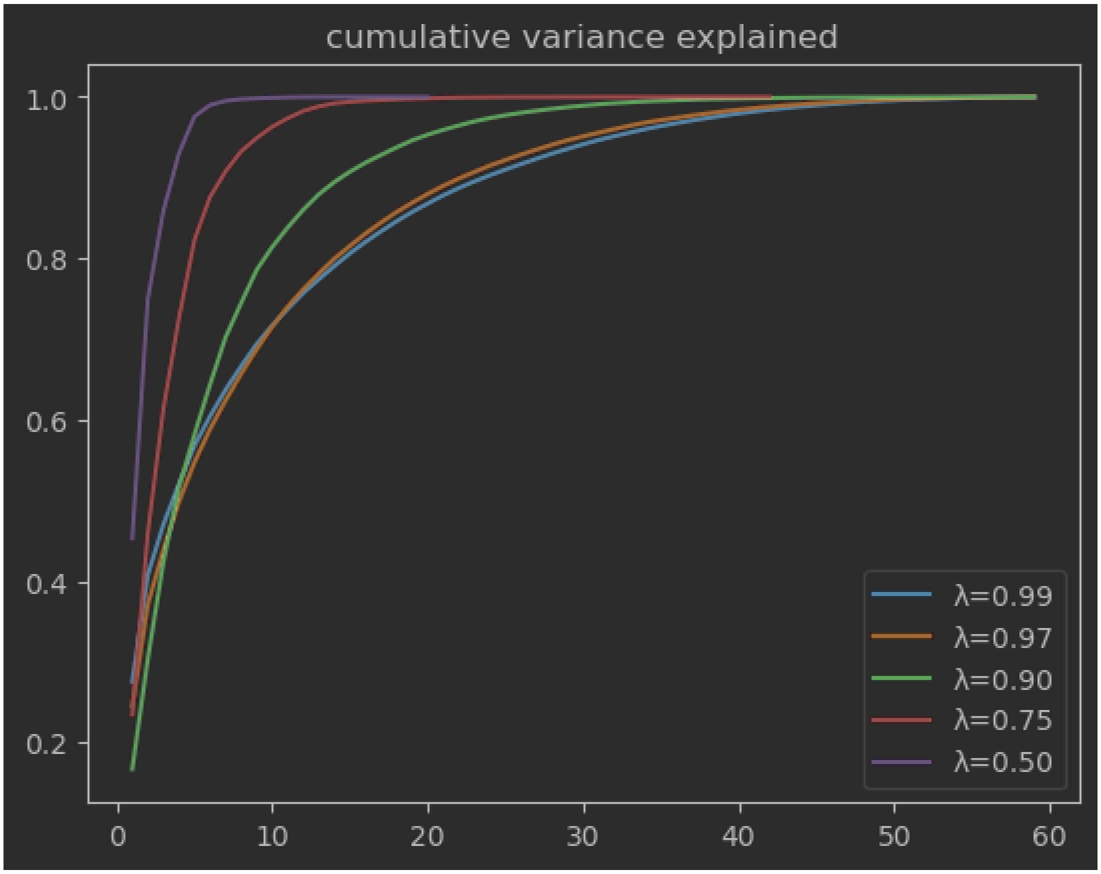

Figure 5

Problem 2

For an N×N non-PSD matrix, I implemented *near_psd* method and *Higham's* method to fix the matrix to PSD. First, I initialized **N to be 500**. The fixed matrices of near_psd and Higham are both PSD now, with no eigenvalue significantly negative. For the near_psd method, the Frobenius norm is 0. 393785, while for Higham's method, the Frobenius norm is 0.008037. For the near_psd method, the run time is 0.073873s, while for Higham's method, the run time is 3.391604s.

Then I increased the N and recorded the results in Tables 2 and 3. Comparing the run time and the Frobenius norm between near_psd and Higham, we can conclude that **near_psd takes much less time to run**, showing higher efficiency. However, **Higham's method has lower Frobenius norms than near_psd**, indicating that Higham's method is more accurate.

Table 2 Run Time

N	Near_psd	Higham’s
500	0.073873s	3.391604s
1000	0.218971s	11.37476s
2000	1.183893s	75.13262s

Table 3 Frobenius Norm

N	Near_psd	Higham’s
500	0. 393785	**0.008037**
1000	0.792974	**0.008152**
2000	1.591348	**0.008210**

Since Higham’s method conducts interpolation of the first projection and second projection, and for each loop, the work-in-progress fixed matrix will move back a little to make sure not deviating too far from the original matrix, Higham’s method will get a fixed matrix which is more accurate. From Table 3, we can also see that **as N increases, the Frobenius norm of Higham basically keeps the same**.

Therefore, for a large matrix, I would choose to use Higham’s method to ensure the accuracy of my simulation result, even if it takes time. For a small matrix, near_psd will be a good choice since it is simple and fast.

Problem 3

To conduct simulations for 4 different matrices, I generated a covariance matrix, an exponentially weighted covariance matrix, a Pearson correlation + exponentially weighted variance, and an exponentially weighted correlation + variance.

I got the following results using 4 different methods listed in the table to simulate 25,000 draws from each covariance matrix. Comparing the Frobenius norm in Table 5, we can conclude that **direct simulation has basically the smallest Frobenius norm**, showing high accuracy. And **as the percentage explained by PCA decreases, the Frobenius norm increases**, showing more deviation from the original dataset.

From Table 4, we can see that **direct simulation always takes the longest time to run**, indicating low efficiency. And **as the percentage explained by PCA decreases, the run time decreases**.

Additionally, comparing direct simulation and PCA with 100% explained, these two methods have similar Frobenius norms while direct simulation always takes longer to run. This may indicate that **PCA with 100% explained is a better way than the direct simulation** method. **PCA with 100% explained drops some useless variables** for us but keeps the accuracy.

Table 4 Run Time (s)

Method	Cov	Ew cov	Cor + ew var	Ew cor + var
Direct sim	0.046350	0.048518	0.047865	0.046816
PCA 100%	0.030961	0.031333	0.033239	0.032252
PCA 75%	0.017960	0.014035	0.013882	0.015244
PCA 50%	0.010261	0.011131	0.009741	0.012132

Table 5 Frobenius norm

Method	Cov	Ew cov	Cor + ew var	Ew cor + var
Direct sim	0.0000000649	0.0000000422	0.0000000505	0.0000000461
PCA 100%	0.0000000746	0.0000000463	0.0000000451	0.0000000474
PCA 75%	0.0000026809	0.0000028235	0.0000025509	0.0000030966
PCA 50%	0.0000111436	0.0000104645	0.0000108957	0.0000134805

In total, when choosing between simulation methods, we need to trade off between time to run and accuracy. When we choose accuracy, the simulation will take a longer time to run. And when we choose speed, we will sacrifice accuracy.