Fintech545-project2
Name:Long Zhang
Date:2/11/2023

Problem 1

1.1
For generate exponentially weighted covariance, we follow the routine below:

$$\sigma_t^2 = \lambda\sigma_{t-1}^2 + (1 - \lambda)\left(x_{t-1} - \bar{x}\right)^2$$

Back substitution and setting the weight of a prior observation, we get:

$$w_{t-i} = (1 - \lambda)\,\lambda^{i-1}$$

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

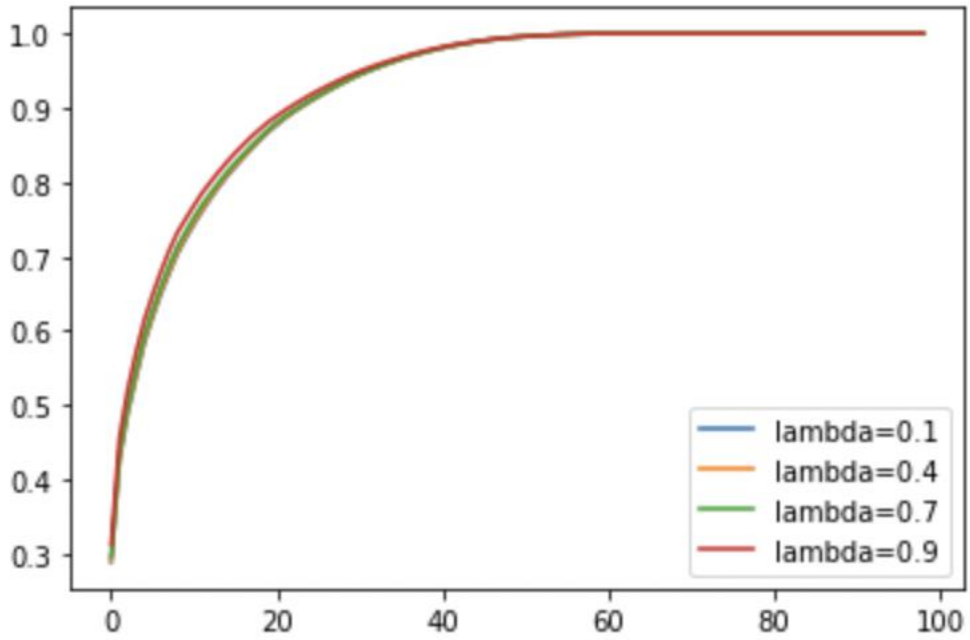
Variance and Covariance Estimators are then

$$\widehat{\sigma_t^2} = \sum_{=1}^{n} w_{t-i}\left(x_{t-i} - \bar{x}\right)^2$$

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

We import the data from file. Next define a function named populateWeights, which is used to generate weights. Then pass the output of populateWeights into exw function, follow the routine above and get the covariance matrix.

1.2
Implement PCA and exponentially weight matrix function, vary the lambda from 0 to 1 and set the number of simulation range from (1, 100), then plot the graph:



From the graph, we can see when lambda gets larger, the graph is flatter, which means the covariance is explained by a larger number of eigenvalues.

Problem2

2.1
Using the chol_psd() and near_psd() function from class, convert them into Python code.

2.2
Following the lecture note of Higham nearest psd algorithm:

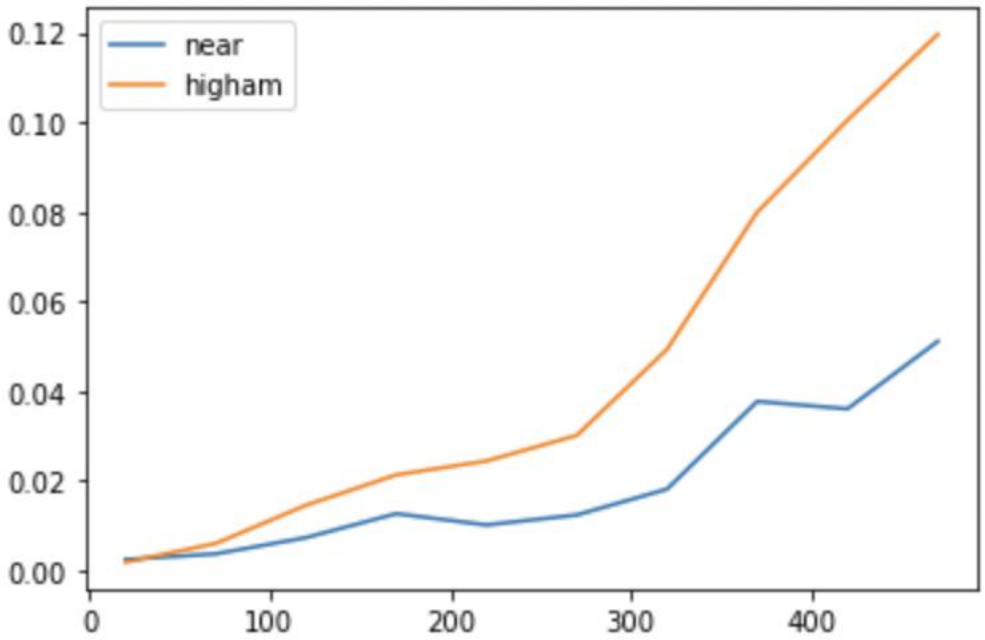
$$\Delta S_0 = 0,\ Y_0 = A,\ \gamma_0 = max\ Float$$
$$Loop\ k \in 1 \ldots max\ Iterations$$
$$R_k = Y_{k-1} - \Delta S_{k-1}$$
$$X_k = P_S(R_k)$$
$$\Delta S_k = X_k - R_k$$
$$Y_k = P_U(X_k)$$
$$\gamma_k = \gamma(Y_k)$$
$$if\ \gamma_k - \gamma_{k-1} < tol\ then\ break$$

Combined with function np.linalg.norm(df, ord = ‘fro’), which implement Frobenius norm, find the nearest psd of the input matrix.

2.3
First we generate a non psd matrix. After implementing near_psd() and higham_psd(), check the fixed matrix by a defined is_pos_def() function. The is_pos_def() function set the eigenvalues threshold as -1e-8. The output confirms that the two fixed matrix are both psd.

2.4
Comparing two fixed matrix with Frobenius norm, the output is shown below:
Frobenius_higham=0.06364303890468953
Frobenius-near=0.6275226557619892
Obviously, higham psd Frobenius norm is much smaller than near_psd Frobenius norm, which indicated Higham psd method is more accurate than near_psd function.

The runtime pragh is shown below:



As N increase, near_psd method keep faster than Higham method, which means near_psd method is more efficient.

2.5
Pros:near_psd method is more efficient and Higham method is more accurate
Cons:near_psd method is less accurate and Higham methos is less efficient

While we are dealing with large size data, we probably choose near_psd method, and when we hope to get more accurate data, we can use Higham method.

Problem3

3.1
Define simulate_multivariate_normal() function, set a parameter use_pca as an option to apply PCA. The default is set to False, which means does not use PCA. If we set use_pca=True, then we check the percentage correctness first (0, 100) and then implement PCA.

3.2
With four combination:
1.person correlation and var();
2.pearson correlation and exponentially weight variance;
3.exponentially weight covariance and correlation;
4.exponentially correlation and var().
We compute each covariance matrix respectively.

3.3
Define a run_time_4() function to record the runtime of direct simulation and 3 PCA simulation,using Frobenius Norm to check the accuracy, the out put is shown below:
1.person correlation and var();

```
Direct time: 0.14380192756652832 Frobenius Norm: 0.000253999441525778
PCA 1.0: 0.09244012832641602 Frobenius Norm: 0.006449095896888769
PCA 0.75: 0.09148216247558594 Frobenius Norm: 0.0016457901826718894
PCA 0.5: 0.08663797378540039 Frobenius Norm: 0.0033396542741153455
```

2.pearson correlation and exponentially weight variance:

```
Direct time: 0.1178741455078125 Frobenius Norm: 0.8674261065085048
PCA 1.0: 0.10222578048706055 Frobenius Norm: 0.00104294765581314
PCA 0.75: 0.0933692455291748 Frobenius Norm: 0.0010315530276428539
PCA 0.5: 0.09413003921508789 Frobenius Norm: 0.0010882147479793416
```

3.exponentially weight covariance and correlation:

```
Direct time: 0.1218869686126709 Frobenius Norm: 0.008748666125820866
PCA 1.0: 0.09900188446044922 Frobenius Norm: 0.00731800046011233
PCA 0.75: 0.08974790573120117 Frobenius Norm: 0.06764534004270853
PCA 0.5: 0.08454489707946777 Frobenius Norm: 0.138062725129939
```

4.exponentially correlation and var():

```
Direct time: 0.12064003944396973 Frobenius Norm: 0.00019022981826616932
PCA 1.0: 0.09585976600646973 Frobenius Norm: 0.00020651625684188786
PCA 0.75: 0.09497499465942383 Frobenius Norm: 0.0016176736904649884
PCA 0.5: 0.09337615966796875 Frobenius Norm: 0.0034650256727319726
```

From the data above, we can conclude that with lower percentage explained, the runtime is smaller but the accuracy is also lower. With Direct simulation, we have highest accuracy but longest runtime, with PCA 0.5 explained, we have shortest runtime but worst accuracy.