Project 01

By Xingjian Yin, NetID as: xy179

##Problem 1:

1)

Mean: 0.05019795790476916

Variance: 0.010322143931072109

Skewness: 0.1204447119194402

Kurtosis: 0.2229270674503816

Simply used the packages provided by pandas.

2)

I would rather use normal distribution, because the sample size is large. (Usually, the larger the sample size, the better it fits normal distribution than t-distribution)

3)

Based on AIC and BIC and QQ plot, the normal distribution better fits the data, to a minor extent.

Normal Distribution: AIC = -131.586728836508, BIC = -1721.7712182785438

T-Distribution: AIC = -1731.4183689195443, BIC = -1716.6951030825978

##Problem 2:

1)

The pandas package naturally computes the pairwise covariance.

```
----------------------------------------------------------------------------------------------------
Pairwise Covariance Matrix:
          x1        x2        x3        x4        x5
x1  1.470484  1.454214  0.877269  1.903226  1.444361
x2  1.454214  1.252078  0.539548  1.621918  1.237877
x3  0.877269  0.539548  1.272425  1.171959  1.091912
x4  1.903226  1.621918  1.171959  1.814469  1.589729
x5  1.444361  1.237877  1.091912  1.589729  1.396186
----------------------------------------------------------------------------------------------------
```

2)

Used an “np.all(eigenvalues>=0)” method

```
----------------------------------------------------------------------------------------------------
Is the covariance matrix positive semi-definite? False
----------------------------------------------------------------------------------------------------
```

3)

For Higham method:

Firstly, compute the covariance and correlation matrix of the data.

Secondly, implement Projection1 and Projection 2.

Thirdly, follow the pseudocode.

(Some insights:

To avoid some negative eigenvalues, I used the “eigenvalues[eigenvalues<0]=1e-8”, instead of 0;

Don’t forget C=B@B.T is the nearest correlation PSD, still needs to be multiplied by std_deviations.)

For R & J method:

Simply decomposition and rescaling from the start, easy.

----

iteration: 48

```
Nearest Positive Semi-Definite Matrix using Higham's method:
[[1.47048437 1.33236075 0.88437762 1.62760181 1.3995556 ]
 [1.33236075 1.25207795 0.619028   1.45060409 1.21445034]
 [0.88437762 0.619028   1.272425   1.07684649 1.05965831]
 [1.62760181 1.45060409 1.07684649 1.81446921 1.57792822]
 [1.3995556  1.21445034 1.05965831 1.57792822 1.39618646]]

Nearest Positive Semi-Definite Matrix using Rebonato and Jackel's method:
[[1.47048437 1.3270091  0.84258339 1.62446381 1.36483264]
 [1.3270091  1.25207795 0.55542072 1.43310934 1.16590627]
 [0.84258339 0.55542072 1.272425   1.05278908 1.06042417]
 [1.62446381 1.43310934 1.05278908 1.81446921 1.54499251]
 [1.36483264 1.16590627 1.06042417 1.54499251 1.39618646]]
```

----

4)

Use data.dropna() to fetch the overlapping data.

```
Covariance Matrix using only overlapping data:
          x1        x2        x3        x4        x5
x1  0.418604  0.394054  0.424457  0.416382  0.434287
x2  0.394054  0.396786  0.409343  0.398401  0.422631
x3  0.424457  0.409343  0.441360  0.428441  0.448957
x4  0.416382  0.398401  0.428441  0.437274  0.440167
x5  0.434287  0.422631  0.448957  0.440167  0.466272
```

5)

The differences between the original and the Higham's method covariance matrices are:

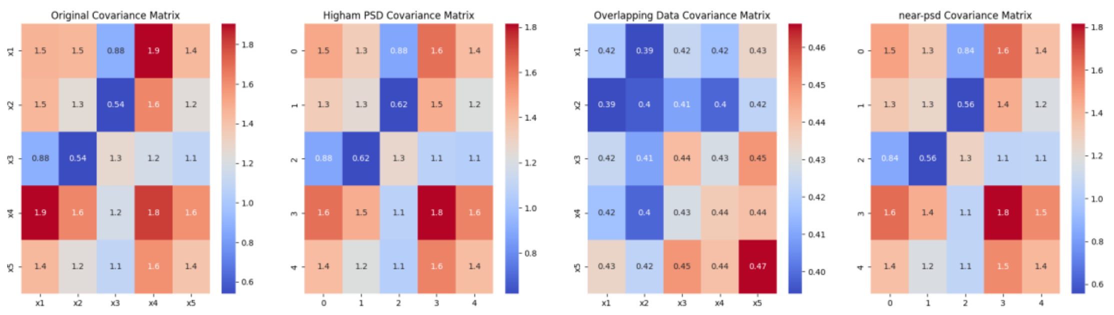
1. The C answers uses more data that D, which is more accurate

2. The Higham's method converges more and is the closest

3. The Rebonato and Jackel's method is the simplest and fastest, which leaves out the negative eigenvalues

4. The overlapping uses less data, so the covariance is rather small

Here is a graph.



##Problem 3

1)

Applies the Multivariate_normal Function.

```
----------------------------------------------------------------------------------------------------
Mean of the data:
x1      0.046002
x2      0.099915
dtype: float64
Covariance matrix of the data:
              x1          x2
x1    0.010162    0.004924
x2    0.004924    0.020284
Fitted multivariate normal distribution:
Mean:
```

```
[0.04600157 0.09991502]
Covariance matrix:
[[0.0101622  0.00492354]
 [0.00492354 0.02028441]]
```

----

2)

Method 1: Formula

$$\mu = \mu_2 + \Sigma_{12}(\Sigma_{11})^{-1}(a - \mu_1)$$
$$\Sigma = \Sigma_{22} - \Sigma_{12}(\Sigma_{11})^{-1}\Sigma_{12}$$

Method 2: OLS

$$x_2|x_1 = \beta_0 + \beta_1 \times x_1$$
$$var(x_2|x_1) = var(x_2) - \beta_1 \times cov(x_1, x_2)$$

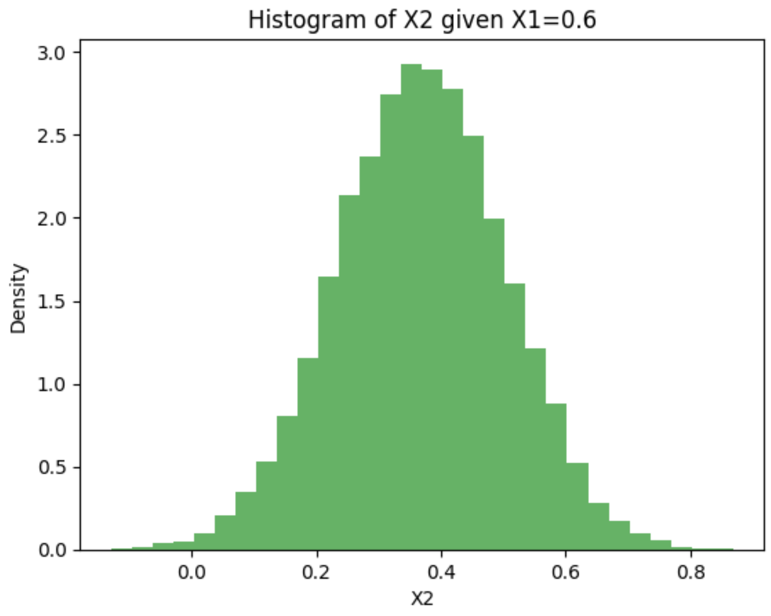
Where $\beta_1 = \frac{cov(x_1, x_2)}{var(x_1)}$; $\beta_0 = mean(x_2) - \beta_1 \times mean(x_1)$

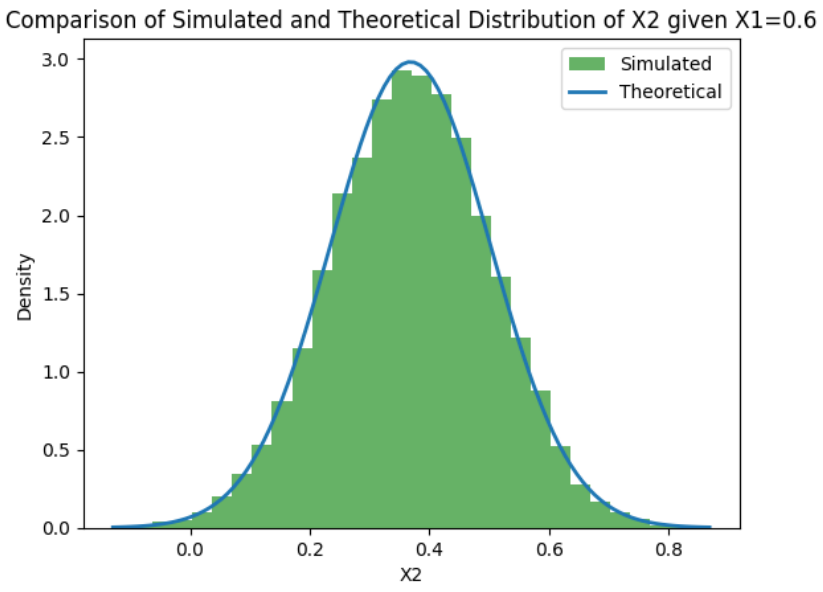
(Note here that we have a given x1 in the problem)

3)

Simulation is simply to generate 10000 samples of norm vector (1×1 here). Then compute the X2 from that and consider the X2's distribution.

```
Sample Mean: 0.3687
Sample Variance: 0.0180
```

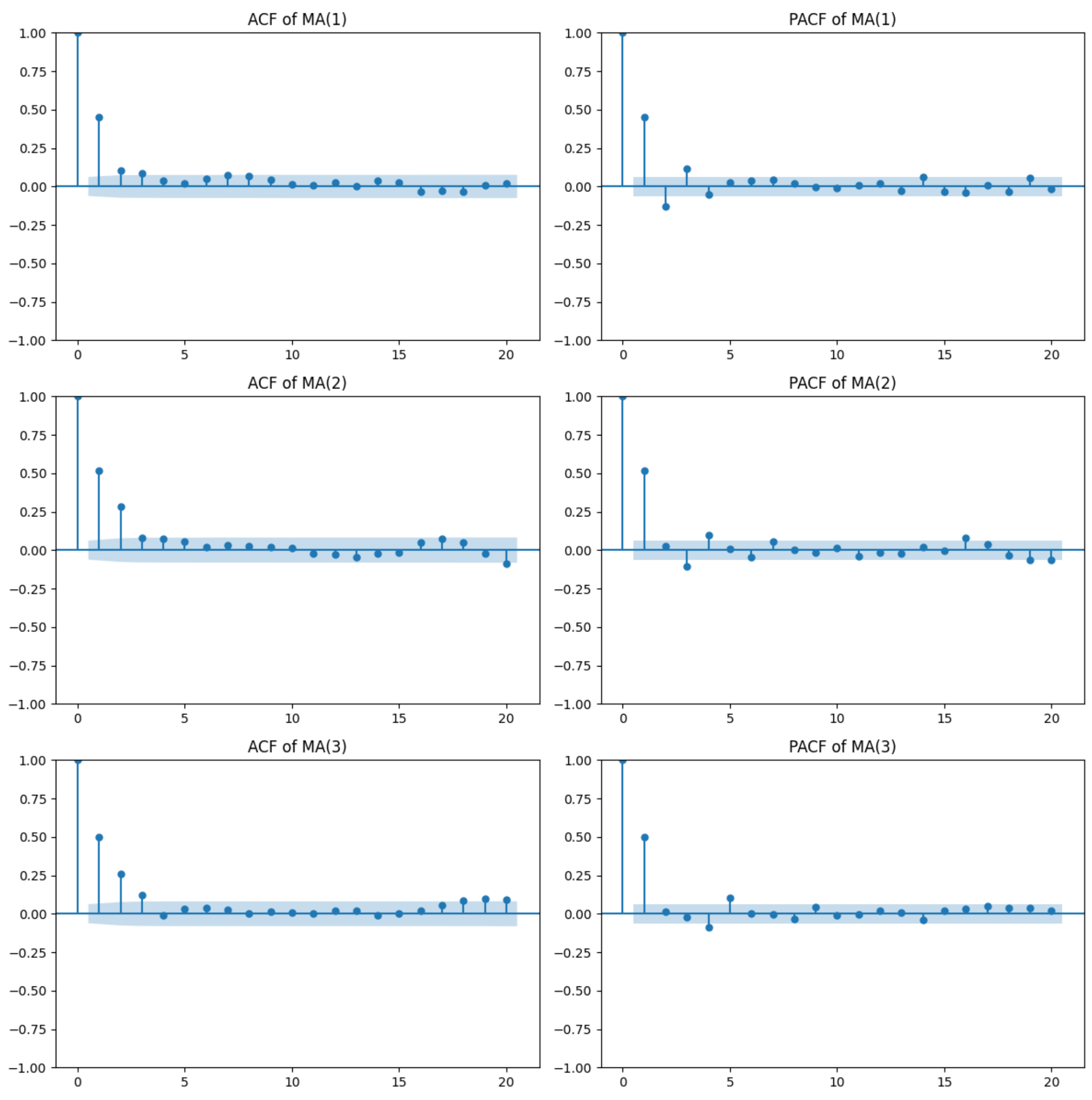




Problem 4

1)

USE ARMAProcess to simulate the process.



For the MA(1) process, the ACF showed a significant spike only at lag 1, with negligible autocorrelations at higher lags, which is exactly what theory predicts for an MA(1) process. The PACF, however, decayed gradually rather than showing a sharp cutoff.

For the MA(2) process, the ACF showed significant spikes at lags 1 and 2, with negligible autocorrelations at higher lags, which is exactly what theory predicts for an MA(2) process. The PACF, however, decayed gradually rather than showing a sharp cutoff.

For the MA(3) process, the ACF showed significant spikes at lags 1, 2, and 3, with negligible autocorrelations at higher lags, which is exactly what theory predicts for an MA(3) process. The PACF, however, decayed gradually rather than showing a sharp cutoff.

----

2)

USE ARMAProcess to simulate the process

AR(1) Process:

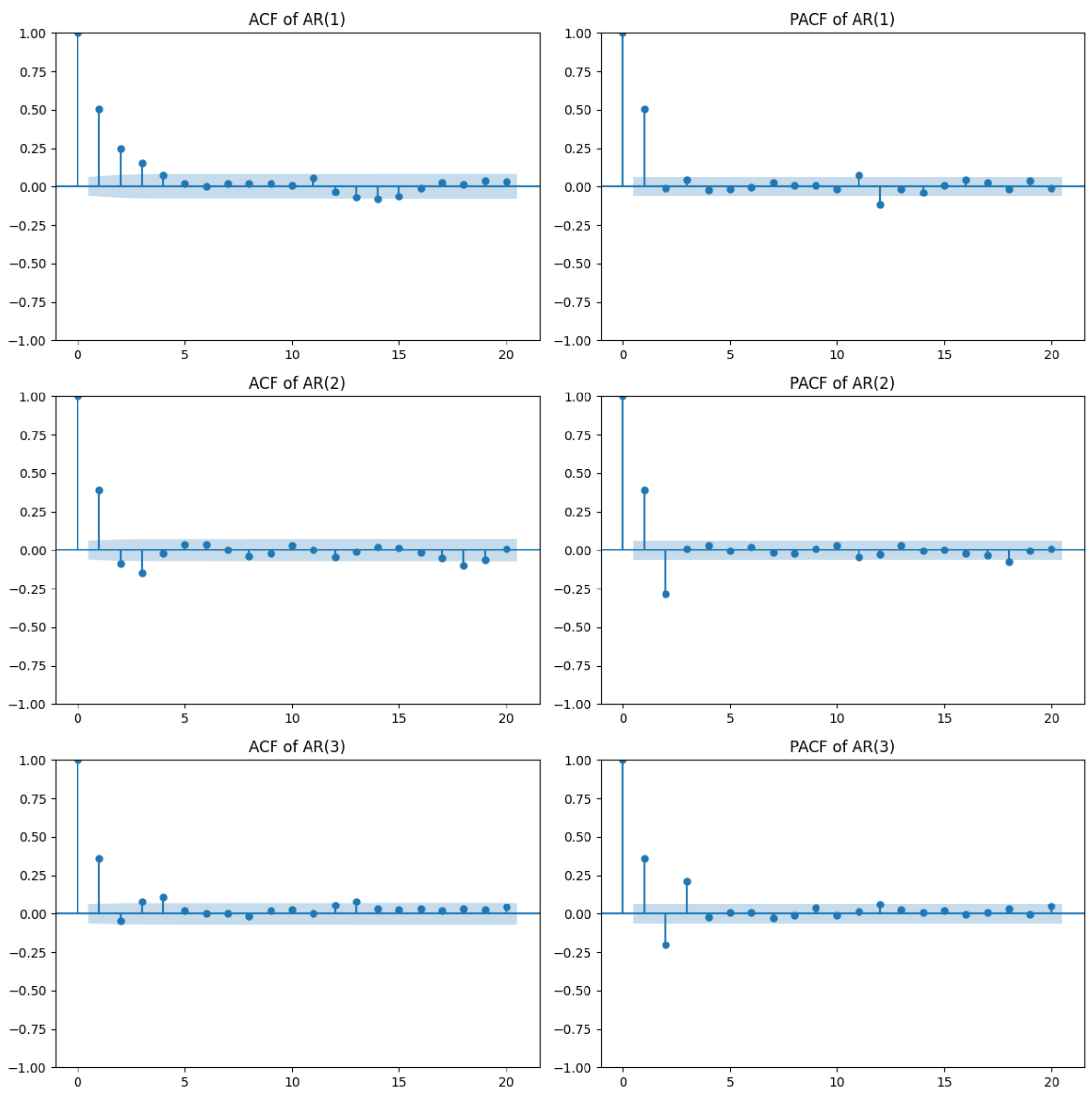
The ACF plot for the AR(1) process shows a high autocorrelation at lag 1 that decays gradually as the lag increases. In contrast, the PACF plot exhibits a significant spike only at lag 1, with the autocorrelations at higher lags quickly falling within the confidence bounds. This is characteristic of an AR(1) process.

AR(2) Process:

For the AR(2) process, the ACF decays gradually, reflecting the influence of both lag 1 and lag 2. The PACF plot displays significant spikes at lags 1 and 2, and the values beyond lag 2 drop off sharply. This clear cutoff in the PACF after lag 2 confirms the AR(2) structure.

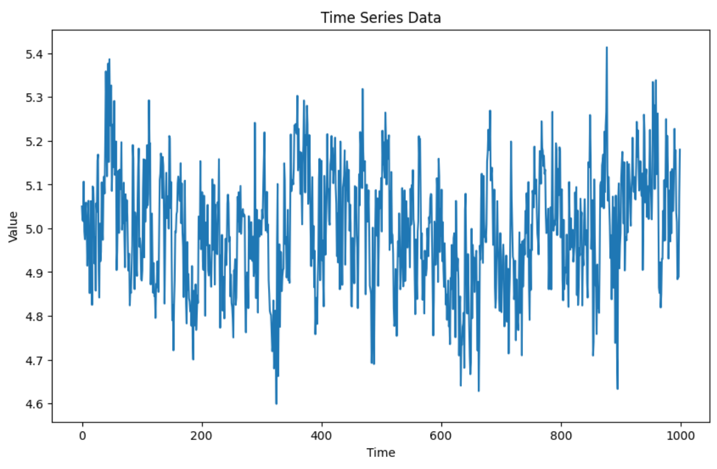
AR(3) Process:

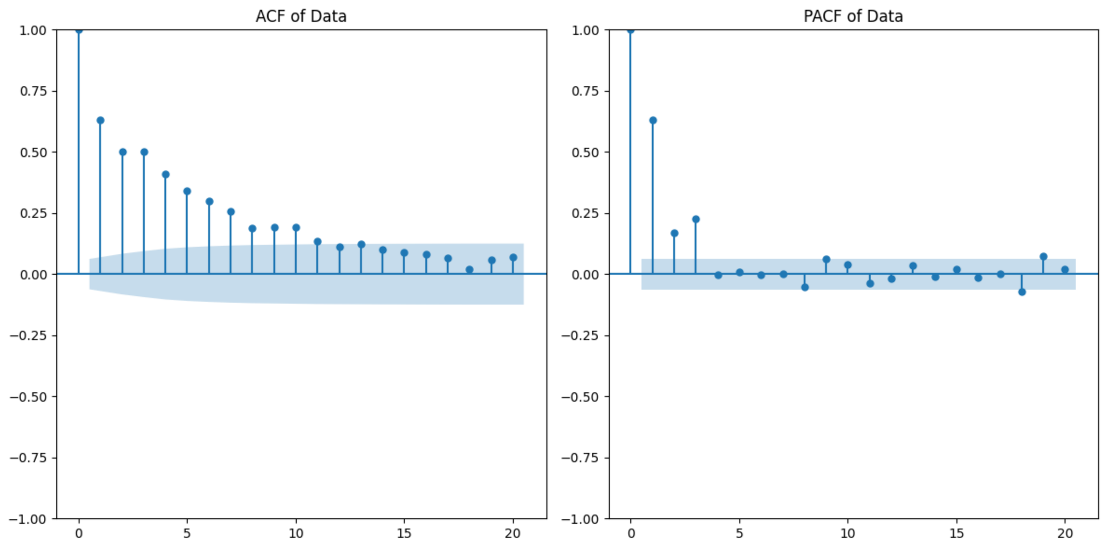
In the AR(3) simulation, the ACF again decays gradually, although the pattern may be slightly more complex (potentially with some oscillatory behavior, depending on the coefficients). The PACF shows significant spikes at lags 1, 2, and 3, with lags beyond 3 being insignificant. This pattern is exactly what we expect for an AR(3) process.



3)

Similar, choose the lag where a significant level drops.





According to the PACF plot, I would use AR(3). Because the significance of the lags drops off after lag 3.

---

4)

Fit all models and choose the smallest AIC.

Best model based on AIC: (3, 0, 0) with AIC: -1746.281720902659

ARIMA model	AIC
(3, 0, 0)	-1746.281720902659
(1, 0, 0)	-1669.0892673454036
(2, 0, 0)	-1696.091685495691
(0, 0, 1)	-1508.9270332399133
(0, 0, 2)	-1559.250931870883
(0, 0, 3)	-1645.132968952571

Problem 5

1)

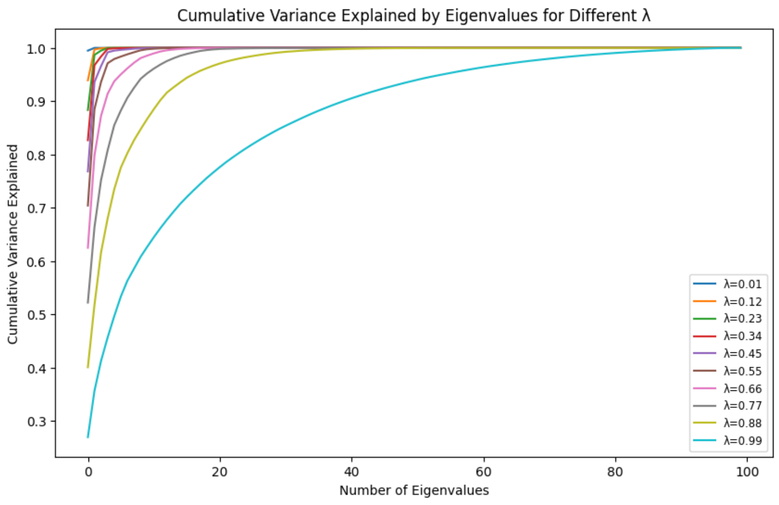
Strictly follow the pseudocode and instructions in the slides. Use matrix to help compute the ewm.

(Use a (data1-mean1) @ np.diag(weights) @ (data2-mean2))

(See in Problem5.ipynb for full matrix of this question)

2)

Use linspace to vary lambda and form a list of cumsum to draw the graph.



3）

We have plotted the cumulative variance explained charts for different values of λ. From these plots, we can make the following observations:

For larger λ values:

The cumulative variance explained curve is relatively flat, which indicates that historical data has a significant influence on the calculation of the covariance matrix. As a result, the eigenvalue distribution may be more uniform.

For smaller λ values:

The cumulative variance explained curve is relatively steep, implying that recent data has a greater impact on the calculation of the covariance matrix. Consequently, the eigenvalue distribution may be more concentrated.

Problem 6

1)

The answer to this question is closely related to what we have done in Problem 2. Use the Higham PSD for Σ, and Cholesky for L, and

$$X_S = L \ @\ Z + \mu$$

See in Problem6.ipynb for full matrix.

2)

I manually implement the functions to PCA here, instead of using the packages from sklearn. It causes a lot of trouble to sort the std_devs, eigenvalues, eigenvecs, means, and reversed_sort the final dataframe of PCA_cov(). For other parts, it is simple to implement.

$$X_S = B \times \left(diag\left(P_{stdev_{sorted}}\right)\right) \times Z + \mu_{sorted}$$

Where B is the result of principle eigenvals relative vectors.
It projects a smaller but principle vectors into all of the dimensions.
See in Problem6.ipynb for full matrix.

3)

The norm of PCA method to the original matrix is significantly larger than that of Cholesky because it neglected many variables.

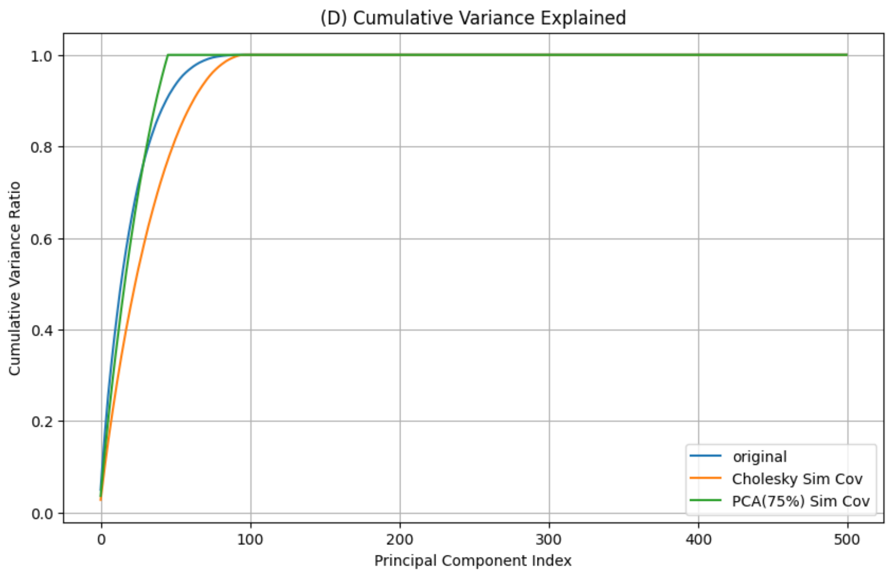
Frobenius norm (Cholesky): 0.26095954764200724

Frobenius norm (PCA): 0.24318133622037832

----

4)

As shown, the PCA graph is the most concentrated, and the Cholesky graph is the most scattered. They put different weights on every component.



5)

Method	Time Taken
Cholesky	2.3
PCA	0.7

6)

Cholesky: More accurate(sometimes not) theoretically and accounts for more variables, but slower and takes too much noise

PCA: Faster, less noise, but less accurate.

Thanks for your reading!   :)