# Week 3 Project

FINTECH 545

Jiwei Xia

# Problem 1

QUESTION: Use the stock returns in DailyReturen.csv for this problem. DailyReturen.csv contains returns for 100 large US stocks and as well as the ETF, SPY which tracks the S&P500.

Create a routine for calculating exponentially weighted covariance matrix.

Vary $\lambda \in (0, 1)$. Use PCA and plot the cumulative variance explained by each eigenvalue for each $\lambda$ chosen.

What does this tell us about values of $\lambda$ and the effect i has on the covariance matrix?

# Problem 1

1.1 Exponentially weighted covariance matrix

To calculate the exponentially weighted covariance matrix, we will first calculate the weights using this equation:

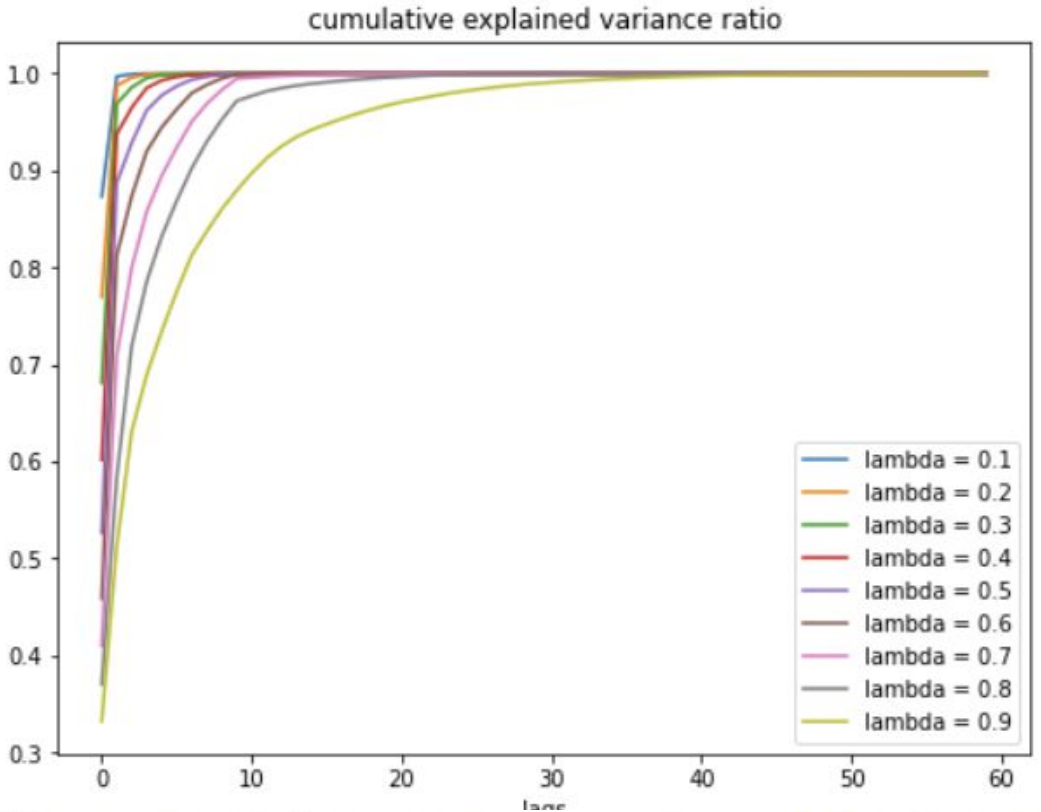
$$w_{t-1} = (1-\lambda)\lambda^{i-1}$$

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

Then we will multiply the error matrices with the weights to get the covariance matrix.

1.2 Calculate the PCA cumulative explained variance ratio.

# Problem 1




From the plot, we can see that when lambda is small, the exponentially weighted covariance matrix puts more weight on current data, and decays quickly as lags increase. As lambda becomes larger, the covariance matrix is more evenly distributed throughout the time duration, and decays slowly towards the maximum lag.

# Problem 2

QUESTION: Copy the chol_psd(), and near_psd() functions from the course repository = inplement in your programming language of the class.

Implement Higham's 2002 nearest psd correlation function.

Generate a non-psd correlation matrix that is 500x500. Use the code in class:

```
n = 500
sigma = fill(0.9, (n,n))
for i in 1:n:
  signma[i,i] = 1.0
and
sigma[1,2] = 0.07357
sigma[2,1] = 0.07357
```

Use near_psd() and Higham's method to fix the matrix. Confirm the matrix is now PSD.

Compare the results o both using the Frobenius Norm. Compare the run time between the two. How does the run time of each functin compare as N increases?

Based on the above, discuss the pros and cons of each method and when you would use each.

# Problem 2

2.1 Implement chol_psd

2.2 Implement near_psd

2.3 Implement Higham's 2002 nearest psd correlation function.

2.4 Use near_psd() and Higham's method to fix the matrix. Confirm the matrix is now PSD.

# Problem 2

```
def is_psd(matrix, tol=1e-7):
   return np.all(np.linalg.eigvals(matrix) >= -tol)
```

```
n = 500
sigma = np.matrix(np.full((n, n), 0.9))
np.fill_diagonal(sigma, 1)
sigma[0, 1] = 0.7357
sigma[1, 0] = 0.7357

near_psd_matrix = near_psd(sigma)
print(is_psd(near_psd_matrix))
higham_psd_matrix = higham_psd(sigma)
print(is_psd(higham_psd_matrix))
```

```
True
True
```

We can confirm that both matrices fixed by near_psd and higham_psd are now psd matrices.

# Problem 2

2.5 Compare the results of both using the Frobenius Norm. Compare the run time between the two. How does the run time of each function compare as N increases?

```
print(frobenius_norm(near_psd_matrix - sigma))
print(frobenius_norm(higham_psd_matrix - sigma))
```

```
0.6275226557679096
0.08964798746820993
```

By comparing the Frobenius Norm, we can see that the result from higham_psd function is more accurate than the one from near_psd.

# Problem 2

| Runtime | near_psd | higham_psd | ratio |
|---|---|---|---|
| 100 | 0.008479 | 0.059492 | 7 |
| 500 | 0.078750 | 1.415074 | 18 |
| 900 | 0.317703 | 6.716566 | 21 |

Looking at the run time, we can see that higham_psd takes much longer to compute than near_psd. As N increases, the ratio between the run times of these 2 functions would also increase.

Based on the results above, the near_psd method is less accurate, but runs very fast, especially when N is small. The higham_psd method is more accurate, but runs very slowly, and the run time increases even more as N increases.

When we have a strict requirement on accuracy and don't care about run time, we can use higham_psd. When the accuracy requirement is more relaxed, and the computational resource is more limited, especially on a smaller matrix, we would choos near_psd.

# Problem 3

Implement a multivariate normal simulation that allows for simulation directly from a covariance

matrix or using PCA with an optional parameter for % variance explained. If you have a library

that can do these, you still need to implement it yourself for this homework and prove that it

functions as expected.

# Problem 3

Generate a correlation matrix and variance vector 2 ways:

1. Standard Pearson correlation/variance (you do not need to reimplement the cor() and

var() functions).

2. Exponentially weighted $\lambda = 0.97$

# Problem 3

Combine these to form 4 different covariance matrices. (Pearson correlation + var()), Pearson correlation + EW variance, etc.)

Simulate 25,000 draws from each covariance matrix using:

1. Direct Simulation
2. PCA with 100% explained.
3. PCA with 75% explained.
4. PCA with 50% explained.

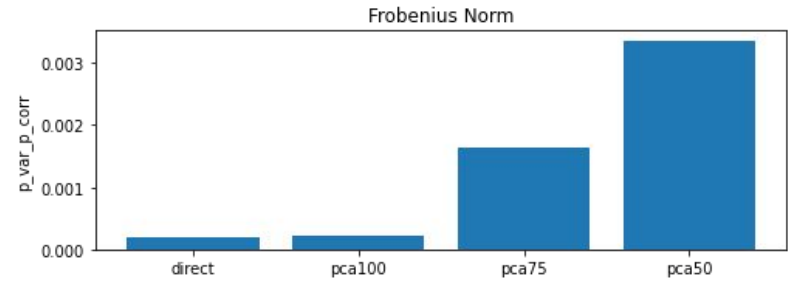
# Problem 3

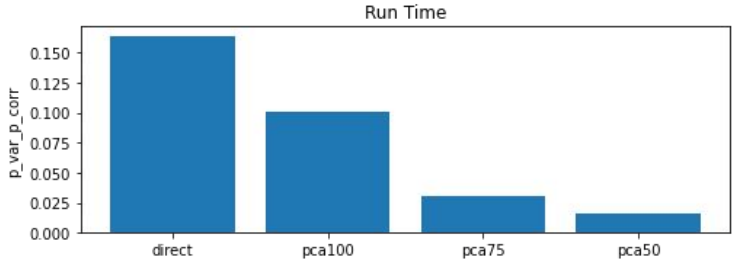
Calculate the covariance of the simulated values. Compare the simulated covariance to it’s

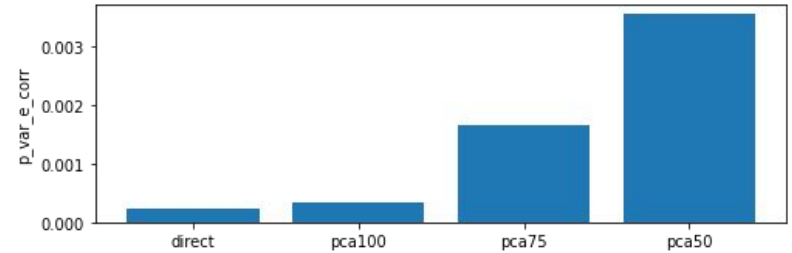
input matrix using the Frobenius Norm (L2 norm, sum of the square of the difference between

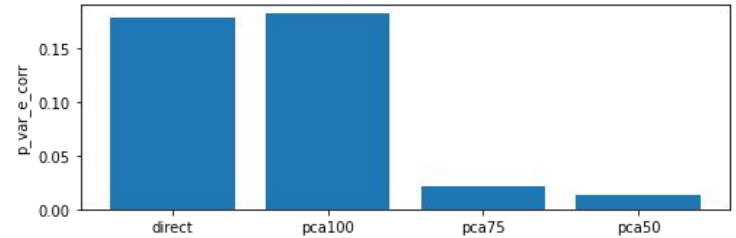
the matrices). Compare the run times for each simulation.

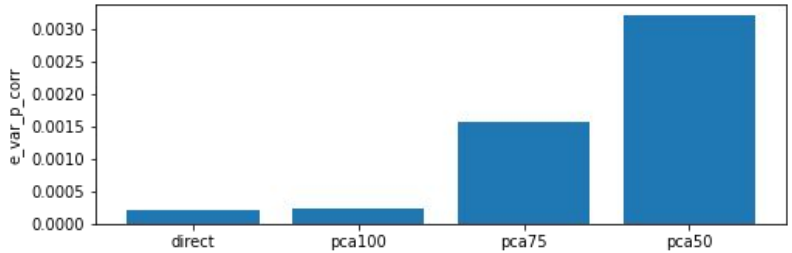
What can we say about the trade offs between time to run and accuracy.

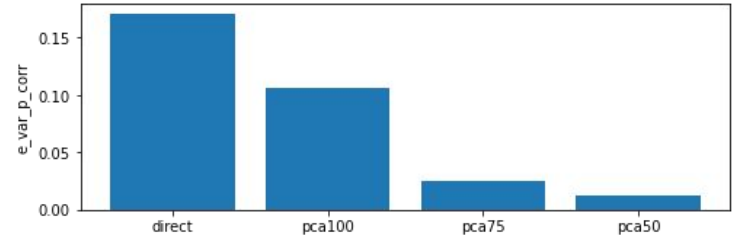

Frobenius Norm
p_var_p_corr
0.003
0.002
0.001
0.000
direct
pca100
pca75
pca50

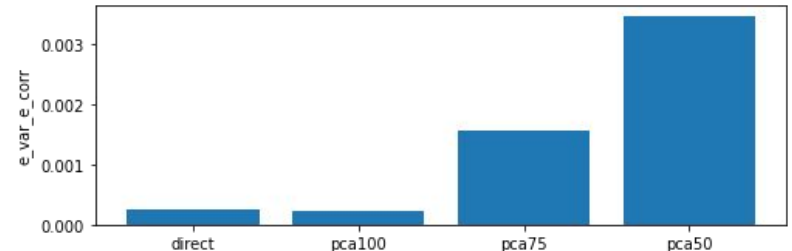

Run Time
p_var_p_corr
0.150
0.125
0.100
0.075
0.050
0.025
0.000
direct
pca100
pca75
pca50

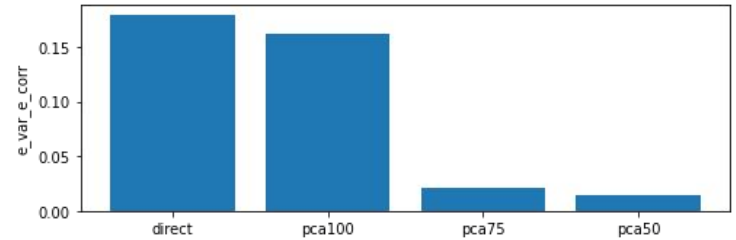

p_var_e_corr
0.003
0.002
0.001
0.000
direct
pca100
pca75
pca50


p_var_e_corr
0.15
0.10
0.05
0.00
direct
pca100
pca75
pca50


e_var_p_corr
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
direct
pca100
pca75
pca50


e_var_p_corr
0.15
0.10
0.05
0.00
direct
pca100
pca75
pca50


e_var_e_corr
0.003
0.002
0.001
0.000
direct
pca100
pca75
pca50


e_var_e_corr
0.15
0.10
0.05
0.00
direct
pca100
pca75
pca50

# Problem 3

From the results above, we can see that using PCA with decreasing explained percentage, the accuracy becomes worse and worse, while the run time becomes faster and faster. The run time benefit increases slower than the accuracy losses, so that the gain of lowering the PCA explained percentage is not very high. Only do it when computational resource is really limited.