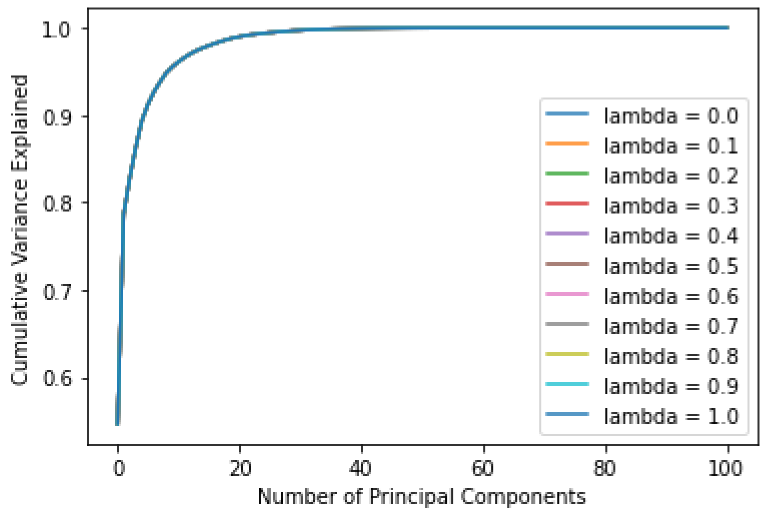
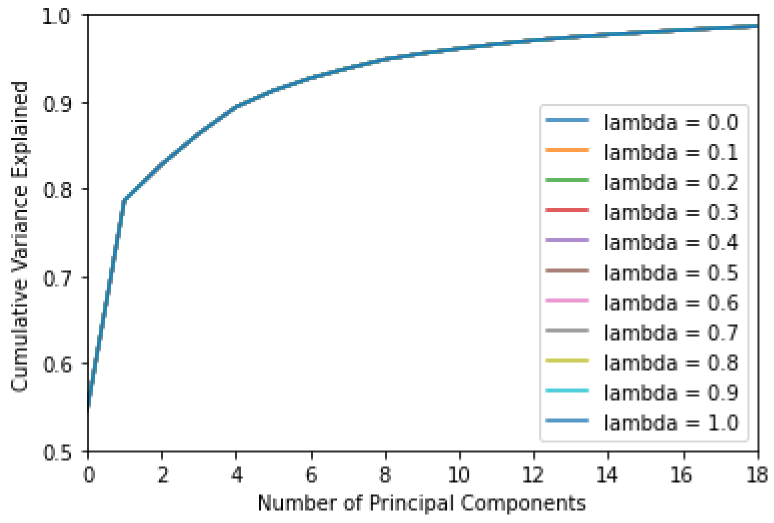
Problem 1:

After creating the functions for exponentially weighted covariance matrix and doing a PCA analysis on the outputs that resulted from the varying lambdas, it can be seen that they all converge towards 1 and take the shape of a logarithmic graph. As can be seen below, By around between 14 to 20 components, each of the graphs of nearly reached a value of 1. This tells us that A lot of variants can be captured from the initial 20 ish components. With regards to Lambda, even though it's hard to see within this graph, mathematically Lambda should cause a decrease in the rate that the cumulative variance as explained by the initial components.

Here is the full graph:



Here is a zoomed in version:



Problem 2:

The code for Chol_psd and near_psd is inside the github homework 3 jupyter notebook

When comparing Higham’s to near_psd function: Higham’s can be found to be much more accurate than near psd where higham was three times as accurate as near psd. The trade off with higham’s compared to near_psd is that near_psd is 5 times faster than highams. When I played around with the code in changed n=500 to n=2000, higham took 30 seconds compared to near_psd taking around 5 seconds. Below are the results from n=500.

Outputs from Frobenius Norm function in order to compare higham and near psd:

Frobenius norm differential between sigma and Higham's corrected matrix: 10.601955731358144

Frobenius norm differential between sigma and near_PSD corrected matrix: 32.056372036273515

Time taken by Higham's method: 0.9053726196289062

Time taken by near_PSD method: 0.17799949645996094

If we're trying to think about when to use One over the other, an example of when to use near_psd is when speed matters the most such as high frequency trades that's and you need to scan entire market where correlations are off from what they historically be.Highams could be used when you need to get the answer right and you only have one shot to get it right.

Problem 3:

Outputs:

Frobenius norms: [0.00022045809159205326, 0.00021448526990145762, 0.0002143581297283025, 0.0002217757829211889, 0.0001847297700137534, 0.00017678742443359588, 0.00018586666199652514, 0.00018837545114129767]

Run times: [0.09327054023742676, 0.08716940879821777, 0.0885460376739502, 0.08570671081542969, 0.09600257873535156, 0.0899965763092041, 0.08099985122680664, 0.08863496780395508]

Order of Sims:

1. Covar_pearson, no PCA, explained variance = None
2. Covar_pearson, PCA explained variance = .5
3. Covar_pearson, PCA explained variance = .75
4. Covar_pearson, PCA explained variance = 1
5. Cov_exp_weighted, no PCA, explained variance = None

6. Cov_exp_weighted, PCA explained variance = .5
7. Cov_exp_weighted, PCA explained variance = .75
8. Cov_exp_weighted, PCA explained variance = 1

After running The Sims, it can be found that the simulations that use PCA, Cov_exp_weighted we're slightly faster and more accurate than functions that use PCA, Pearson. for the functions that did not use PCA, it can still be seen that exponentially weighted covariance is more accurate than Pearson covariance.

overall with regards to runtime the functions that use PCA, exponentially weighted covariance all had faster run times then Pearson.

Fom an accuracy standpoint, explained variance both improved accuracy and speed. both speed and accuracy increased as explained variance increased what can possibly said to be linear. from a high overview it can be said that it's best to use explained variance with a value closer to 1 in order to achieve the best speed and accuracy.