Problem 1:

Routine for calculating an exponentially weighted covariance matrix:

- Read the DailyReturn.csv file with pandas package, drop the date column
- Write the weight function that take lambda as input, return a numpy array with length t, which is 60 in our data. each term is calculated by the following formula. Then normalize it.

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

- Write the exponential weighted function that take float X, Y as inputs, use the weighted array created last step

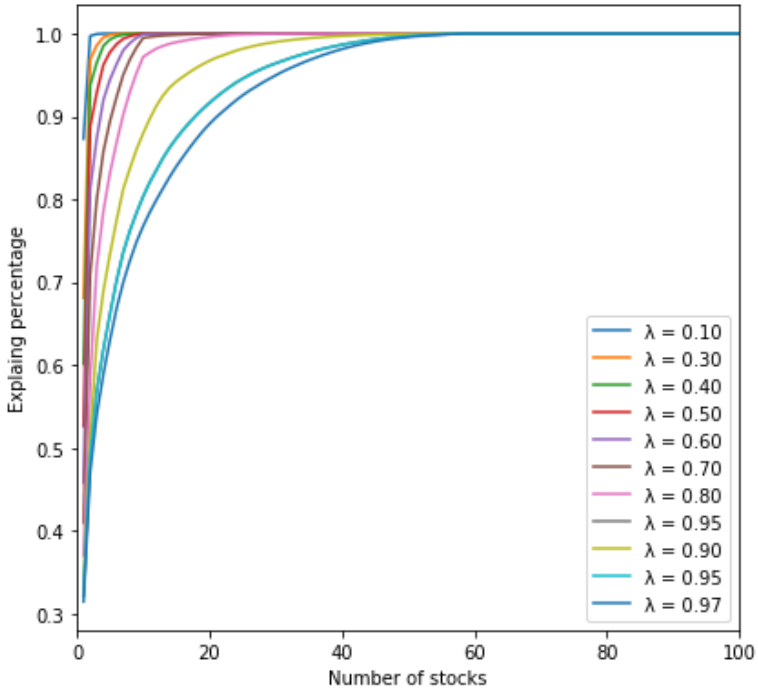
$$\widehat{cov(x, y)} = \sum_{i=1}^{n} w_{t-i}\left(x_{t-i} - \bar{x}\right)\left(y_{t-i} - \bar{y}\right)$$

- Initialize a correlation matrix and call the function we created last step with inputs (i, j), which represent all combination of stocks in our data. Fill the function returned value in the matrix.
- Print our covariance matrix as following:

```
[[8.25839218e-05 1.13157307e-04 1.32658493e-04 ... 1.15015729e-04
  6.51973440e-05 8.28276770e-05]
 [1.13157307e-04 2.61179609e-04 2.20642626e-04 ... 1.14994673e-04
  2.48273209e-05 8.33597068e-05]
 [1.32658493e-04 2.20642626e-04 3.29737461e-04 ... 5.98170234e-05
  2.51233125e-05 7.07802358e-05]
 ...
 [1.15015729e-04 1.14994673e-04 5.98170234e-05 ... 7.30215981e-04
  2.67284635e-04 2.02408061e-04]
 [6.51973440e-05 2.48273209e-05 2.51233125e-05 ... 2.67284635e-04
  2.96819677e-04 5.56068297e-05]
 [8.28276770e-05 8.33597068e-05 7.07802358e-05 ... 2.02408061e-04
  5.56068297e-05 2.54613778e-04]]
```

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

- Write the Principal Component Analysis (PCA) simulation function that take matrix and a n value as inputs, where n value represents the number of eigenvalues we want to use. It will return the explanation percentages.
- Initialize our lambdas from 0 to 1.
- Initial an independent variable x from 1 to 100. Use our pca function to calculate corresponding Y values.
- Iterating all lambda lists to plots all X and Y we generated as following:



This plot shows that the lower lambda we choose, the faster the curve will converge to 1, which means fewer nodes were taken to complete the covariance matrix. In other word, we will take account of too little number of stocks to represent the population, which could be biased. In contrary, the higher lambda means we need moderate numbers of stocks to explain the population. For lambda = 0.95, we need about 58 stocks to explain one hundred percent of the population.

Problem 2

Implement chol_psd(), near_psd(), and Higham's method function in python:

- Create a non-psd correlation matrix A that is 500x500
- First test matrix A cannot be loaded in chol_psd function because it is a non_psd function
- Build the Higham function that take non psd matrix as input, and use iteration to reconstruct the matrix until there is no negative eigenvalue.
- Take the output of higham (A) and near_psd (A). Then, apply chol_psd function to the outputs, respectively.
- By examine no error occurs, we are confident now matrix A become positive semi-definite.
- Write a Frobenius norm function as this formula:

$$||A||_F = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2$$

- Calculate the norm difference between the two methods and the original matrix A

```
Frobenius norm difference of near_psd: 0.0555
Frobenius norm difference of higham: 4.499e-06
```

  This result of Higham havs a lower difference between it and the original matrix.

- Test runtime by time module in python:

```
Near_psd takes: 0.07626986503601074
Higham takes:  0.11855340003967285
```

- Now make the non-psd matrix with size of 2000x2000, repeat the last two processes:

```
Frobenius norm difference of near_psd: 0.0559
Frobenius norm difference of higham: 1.142e-06
Time for near_psd: 4.539523601531982
Time for higham: 4.865905284881592
```

- Let n = 5000, repeat the process:

```
Frobenius norm difference of near_psd: 0.056
Frobenius norm difference of higham: 4.803e-07
Time for near_psd: 79.2311098575592
Time for higham: 81.81068062782288
```

From the Frobenius norm results, we can see that Higham has a better performance than that of near_psd, It generate a covariance matrix with accuracy of 10^-6, In contrary, near_psd takes a little shorter time to get the result.

In conclusion, to get more accuracy data, we prefer to use Higham's method. And the time spans of two methods are approximately the same. Near_psd could be a slightly faster, but I believe it is worthy to do that for a much smaller deviation.

Problem 3

Implement a multivariate normal simulation

- Generate correlation matrix and variance vectors to form different covariance matrix:
  - Matrix 1. Use unweighted correlation and unweighted variance
  - Matrix 2. Use unweighted correlation and weighted variance
  - Matrix 3. Use weighted correlation and weighted variance
  - Matrix 4. Use weighted correlation and unweighted variance

  To find these four values:
    - First, create exponential weighted variance function by formula:

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

    - Calculate the weighted covariance as we did in problem 1.
    - Divide weighted covariance by weighted variance to get the weighted correlation
    - Use python build-in function to get the unweighted data
- Combine these to form 4 difference covariance matrices.
- Write pca simulation function that input Matrix, nsim, and percentage explained.
- Input PCA percentage explained and get the different results

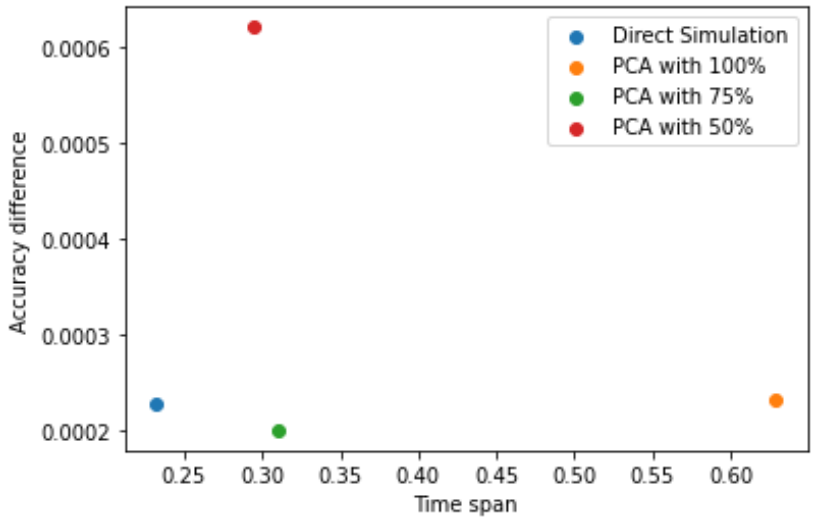
Simulate 25,000 draws from each covariance matrix using:

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

Plot Deviation V.S. time span graph to visualize the trade-off

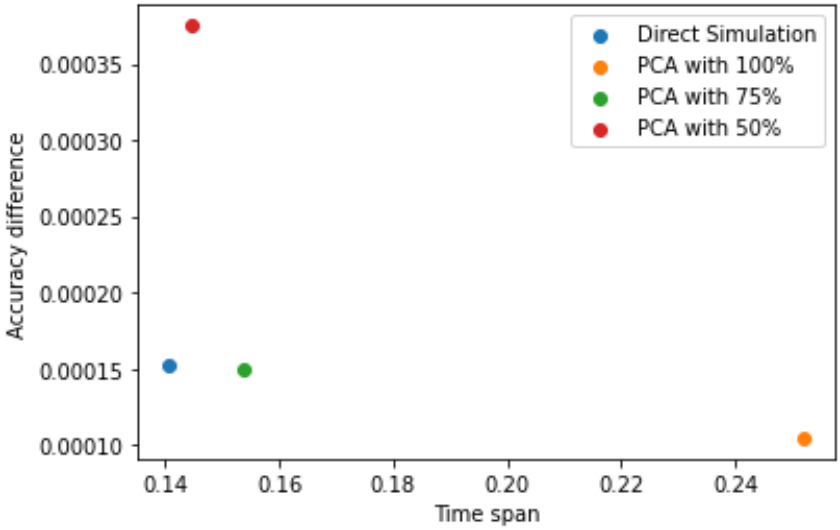
- Matrix 1:

```
Direct Simulation time: 0.12745
Norm difference for direct simulation: 0.00018084
PCA with 100% explained time: 0.3355
Norm difference for PCA with 100% explained: 0.00020611
PCA with 75% explained time: 0.16183
Norm difference for PCA with 75% explained: 0.00025579
PCA with 50% explained time: 0.12805
Norm difference for PCA with 50% explained: 0.00064118
```



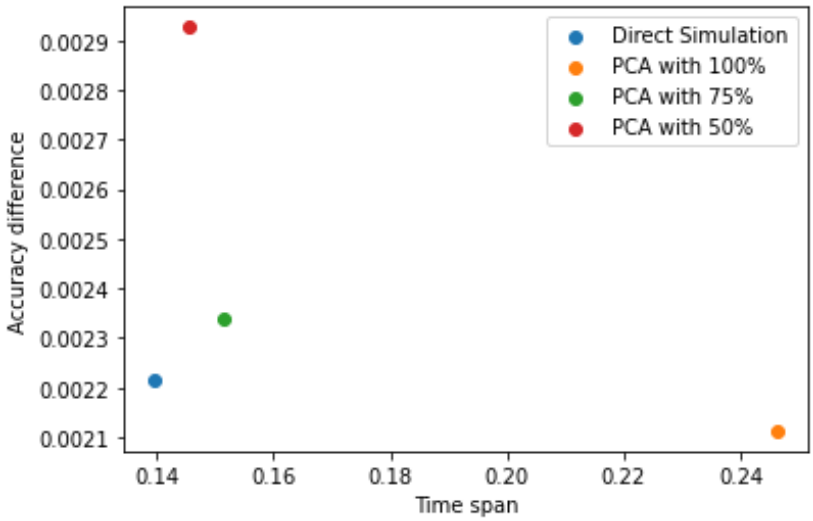
- Matrix 2:

```
Direct Simulation time: 0.15102
Norm difference for direct simulation: 4.194e-05
PCA with 100% explained time: 0.27412
Norm difference for PCA with 100% explained: 0.00010286
PCA with 75% explained time: 0.15914
Norm difference for PCA with 75% explained: 6.3245e-05
PCA with 50% explained time: 0.14627
Norm difference for PCA with 50% explained: 0.00056408
```



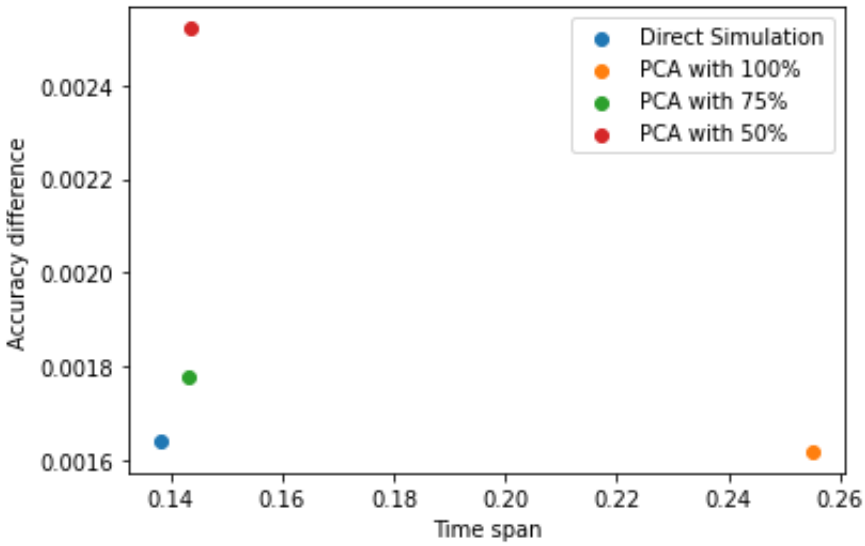
- Matrix 3:

```
Direct Simulation time: 0.13951
Norm difference for direct simulation: 0.00083671
PCA with 100% explained time: 0.25673
Norm difference for PCA with 100% explained: 0.00078314
PCA with 75% explained time: 0.13967
Norm difference for PCA with 75% explained: 0.00075406
PCA with 50% explained time: 0.17655
Norm difference for PCA with 50% explained: 0.00017183
```



- Matrix 4:

```
Direct Simulation time: 0.12752
Norm difference for direct simulation: 0.00056461
PCA with 100% explained time: 0.23987
Norm difference for PCA with 100% explained: 0.00061352
PCA with 75% explained time: 0.19683
Norm difference for PCA with 75% explained: 0.00063142
PCA with 50% explained time: 0.14576
Norm difference for PCA with 50% explained: 0.00022125
```



We can see for each matrix, the higher PCA explanation percentage, the longer time will take, and the lower deviation the simulation will generate.

From the plots, we can deduct that Direct simulation could be the relative reliable one. It stays with a relative low time span and low deviation except for Matrix 2.

For PCA simulations, we can see generally the 50% explanation PCA have the worst error deviation but the least time span. 100% explanation PCA have generally the longest time span and great accuracy. 75% explanation PCA perform moderately between the formers.