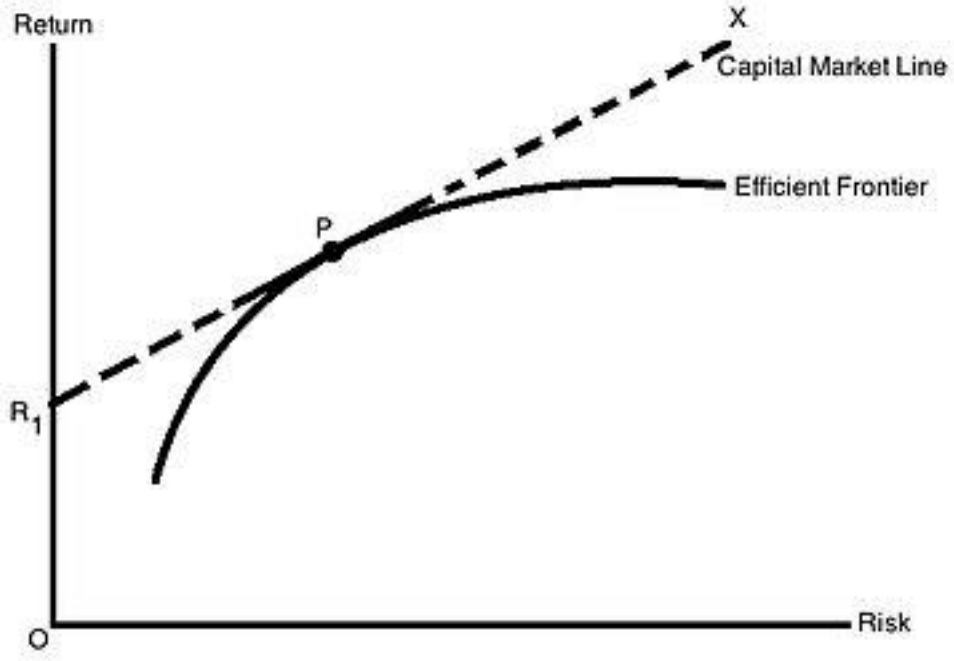
SF1811 HOME ASSIGNMENT 2
MARKOWITZ PORTFOLIO PROBLEM

Geoffrey Lau Chung Yin

Introduction

Markowitz model is a portfolio optimization model devised by Harry Markowitz in 1952. The purpose of the model is to assist in selecting the most efficient portfolio by analysing the portfolios' expected return and standard deviation, showing the investors of how to reduce their risk and maximize returns for the risk assumed with the efficient frontier plotted in the mean-variance model. This theory is the foundation to modern portfolio theory, and helped Markowitz to win the Nobel memorial prize in economic science in 1990.



The assignment is to simulate the mean-variance model using quadratic programming functions in programming language, and observe the effects to the model with changes in constraints. The program I wrote for this assignment is in python programming language as the university I exchanged from does not provide MATLAB for students. The program will first generate a random correlation matrix C and a vector μ from the translation of the given code. The program then set up constraints in matrices format, which is then used for the function solvers.qp() in the package cvxopt, which is equivalent to the function quadprog in MATLAB. The optimal solution is then stored and used to calculate mean and standard deviation, finally plotting into a mean-variance model using package matplotlib to observe the efficient frontier.

Explanation of implementation

`cvxopt.solvers.qp(P, q [, G, h [, A, b [, solver [, initvals ] ] ] ])`

Solves the pair of primal and dual convex quadratic programs

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

According to cvxopt documentation, we have to first setup the following equations at hand to the format as above.

$$\min x^T Cx$$
$$s.t.\ \mu^T x = r$$
$$e^T x = 1, e = (1, \dots, 1)^T$$
$$x \geq 0$$

We have to find the matrix P,q,G,h,A and b in order for the function to work. For P we can figure that it is equal to the matrix C, which is already provided by the given code. As for q, we will set a zero matrix with the size 1 by n, as there is no constant in this question. This two matrices will stay the same for all the upcoming exercises.

For matrices G and h, we will need to change our inequality equation of x as follow,

$$-x \leq 0$$
$$-Ix \leq 0$$

As x is a 1 by n matrix, therefore we need an identity matrix at the front, which means,

$$G = -I, with\ size\ of\ n * n$$
$$h = (0, \dots, 0), with\ size\ of\ n * 1$$

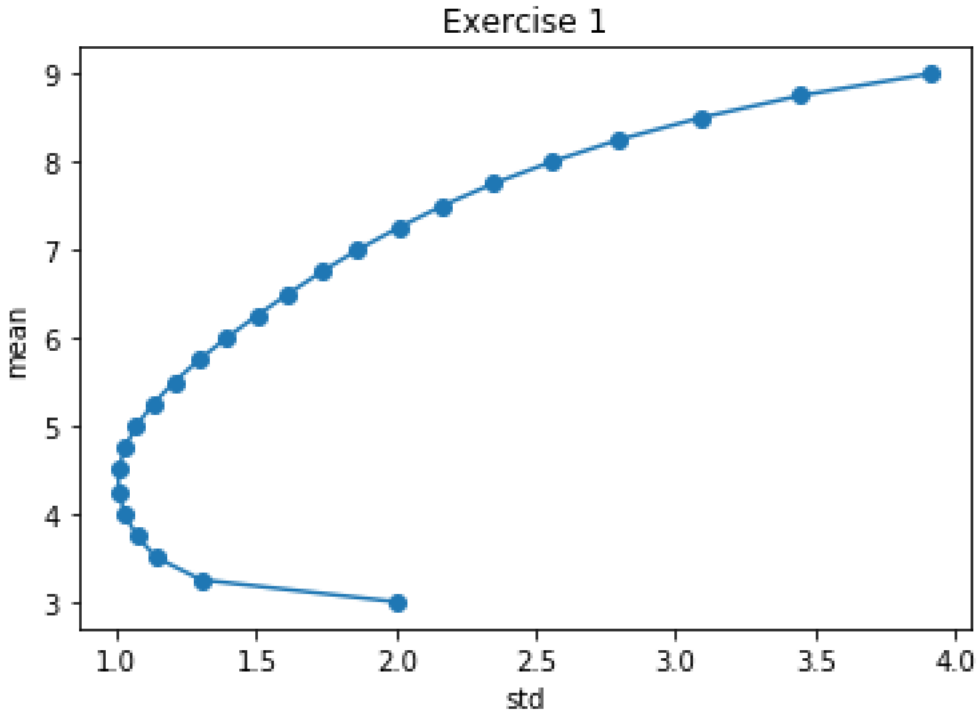
For A and b matrices, we will combine two equations into one matrix.

$$\begin{pmatrix} \mu \\ e \end{pmatrix}^T x = \begin{pmatrix} r_i \\ 1 \end{pmatrix}, i = 1, \dots, 25$$
$$A = \begin{pmatrix} \mu \\ e \end{pmatrix}^T, with\ size\ 2 * n$$
$$b = \begin{pmatrix} r_i \\ 1 \end{pmatrix}, with\ size\ 2 * 1$$

Matrices μ and e is given while r is a 1 by 25 matrix with ascending values from 3 to 9 with steps of 0.25. Note only one value of r is used in matrix b, as we will repeat the solvers.qp() function for all 25 r values in order to generate the curve. The resulted 1 by n values for x in each round is first stored in a list, which then it is processed into standard deviation and mean, separately stored in another two different lists, named sig and mean in the program respectively. The calculation for mean and standard deviation is as follows,

$$\sigma(x) = \sqrt{x^T Cx}$$
$$\mu(x) = \mu^T x$$

Finally using matplotlib package, a graph is plotted with mean as y-axis and standard deviation as x-axis. The following diagram is the graph plotted using the program. The above implementation is labelled as exercise 1 in the code commentation.



For exercise 2, the constraint has now changed into

$$\min x^T Cx$$
$$s.t.\ \mu^T x = r$$
$$e^T x \leq 1, e = (1, \dots, 1)^T$$
$$x \geq 0$$

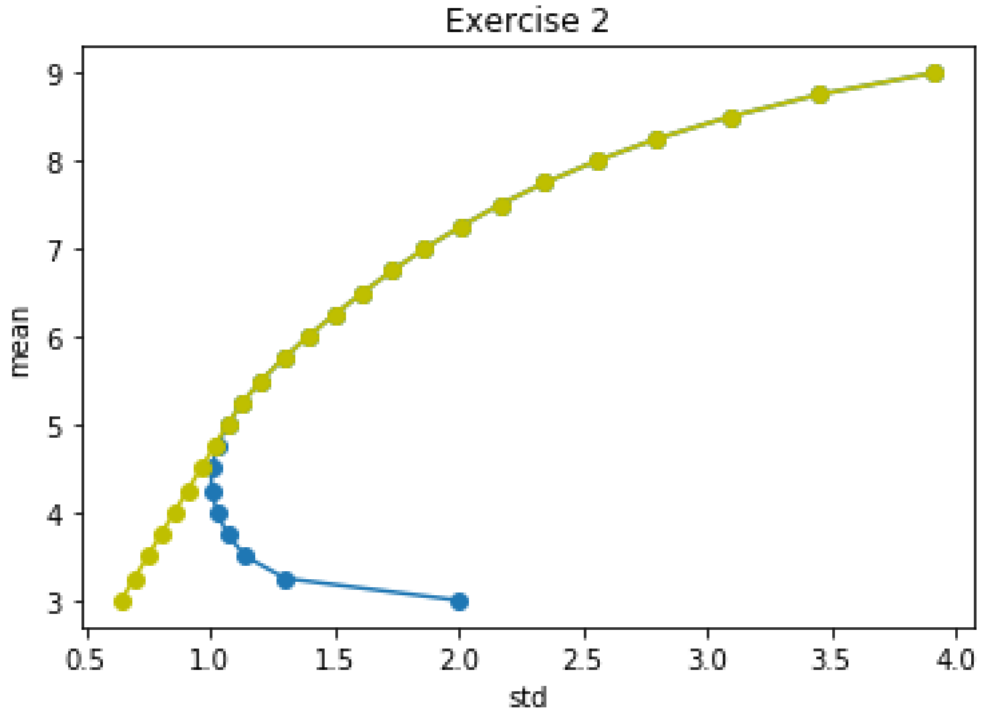
This changes G and h matrices into the following, as there is one more inequality statement than exercise 1.

$$-Ix \leq 0$$
$$e^T x \leq 1$$
$$\begin{pmatrix} -I \\ e \end{pmatrix} x \leq \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$G = \begin{pmatrix} -I \\ e \end{pmatrix}, with\ size\ of\ n+1\ *\ n$$
$$h = \begin{pmatrix} 0, \dots, 0 \\ 1 \end{pmatrix}, with\ size\ of\ n+1 * 1$$

This also changes A and b as there is now one less equality statement.

$$\mu^T x = r_i, i = 1, \dots, 25$$
$$A = \mu^T, with\ size\ 1 * n$$
$$b = r_i, with\ size\ 1 * 1$$

The following diagram is the graph plotted with the new matrix format. The blue line represents the frontier created by exercise 1, and the yellow line represents the frontier created from the new matrices. The above implementation is labelled as exercise 2 in the code commentation. As for the differences between the two frontiers, the mean stayed the same for all points while standard deviation begins to differ when mean is below 5.



The differences conveys that if the person is not necessary to invest all of his capital, which is portraited by the yellow curve, he would have the same level risk (standard deviation) if he aims for high level of return (mean), but he benefits by having relative lower level risk if he aims for low level of return, resembling to a risk-free market portfolio situation.

For exercise 3, the constraint has now changed into

$$\min x^T Cx$$
$$s.t.\ \mu^T x \geq r$$
$$e^T x = 1, e = (1, \dots, 1)^T$$
$$x \geq 0$$

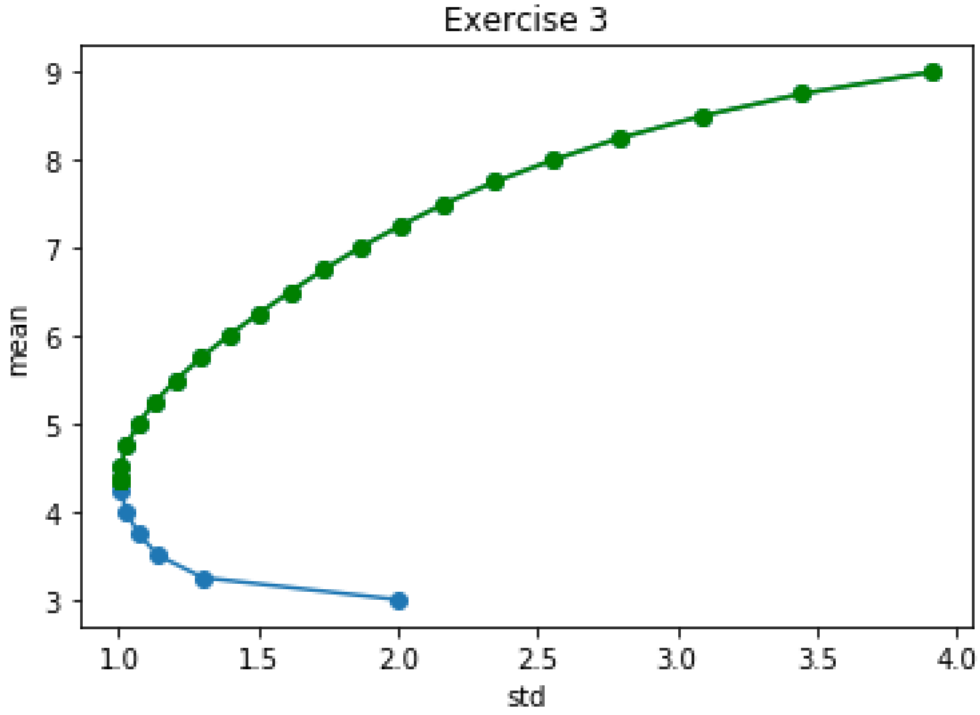
This changes G and h matrices into the following, as there is one more inequality statement than exercise 1.

$$-Ix \leq 0$$
$$-\mu^T x \leq r_i$$
$$\begin{pmatrix} -I \\ -\mu \end{pmatrix} x \leq \begin{pmatrix} 0 \\ r_i \end{pmatrix}$$
$$G = \begin{pmatrix} -I \\ -\mu \end{pmatrix}, with\ size\ of\ n+1\ *\ n$$
$$h = \begin{pmatrix} 0, \dots, 0 \\ r_i \end{pmatrix}, with\ size\ of\ n+1 * 1\ , i = 1, \dots, 25$$

This also changes A and b as there is now one less equality statement.

$$e^T x = 1$$
$$A = e^T, with\ size\ 1 * n$$
$$b = 1\ , with\ size\ 1 * 1$$

The following diagram is the graph plotted with the new matrix format. The blue line represents the frontier created by exercise 1, and the green line represents the frontier created from the new matrices. The above implementation is labelled as exercise 3 in the code commentation.



For difference in figures, the mean and standard deviation are all the same for all the points that is with mean above 4.5, where the new frontier stops abruptly. Checking with the program, it appears all the calculated x values with r lower than 4.5 as constraint has all equated with a mean of 4.5, which still satisfy the constraint of larger or equal to r when multiplied with μ, which represents. The difference in curve conveys that the green frontier is only representing the best possible expected return of the portfolio, as values on the blue curve shows that the lower the return below 4.5, actually the higher the risk.

For exercise 4, the constraint has now changed into

$$\min x^T Cx$$
$$s.t.\ \mu^T x = r$$
$$e^T x = 1, e = (1, \dots, 1)^T$$

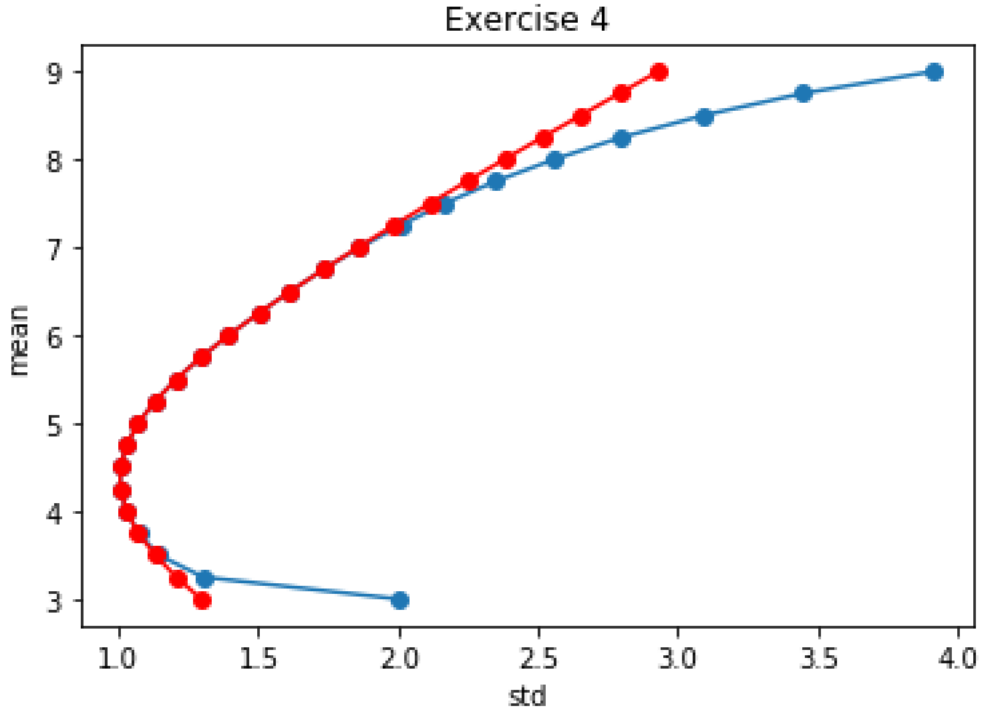
This changes G and h matrices into the following, as there is no more inequality statement. In this case we have to set both G and h to zero matrix.

$$G = (0, \dots, 0), with\ size\ of\ n * n$$
$$h = (0, \dots, 0), with\ size\ of\ n * 1$$

A and b matrices are same as exercise 1, as there are no changes in the equality constraints.

$$\begin{pmatrix} \mu \\ e \end{pmatrix}^T x = \begin{pmatrix} r_i \\ 1 \end{pmatrix}, i = 1, \dots, 25$$
$$A = \begin{pmatrix} \mu \\ e \end{pmatrix}^T, with\ size\ 2 * n$$
$$b = \begin{pmatrix} r_i \\ 1 \end{pmatrix}, with\ size\ 2 * 1$$

The following diagram is the graph plotted with the new matrix format. The blue line represents the frontier created by exercise 1, and the red line represents the frontier created from the new matrices. The above implementation is labelled as exercise 4 in the code commentation.



For difference in figures, the mean and standard deviation stayed the same, but only deviates when either mean is below 3.5 or above 7.25. The difference suggest that short selling allow portfolios that are aiming for very high or very low returns to have lesser risk, but would still have the same amount risk if the portfolio is not aiming for the extremes in returns. This conveys that short selling usually can reduce risk of investment. However, short selling might be dangerous as x values can be negative due to lack of constraint of x must be bigger than 0, meaning there might be loss in some individual parts of the asset.

In conclusion, the above exercises have shown that optimization is crucial when deciding a good market portfolio. Having the ability to save a part of the capital instead of fully investing, and having the ability to short sell can significantly reduce the risk while attempting to maximize returns. However, in the case of short sell, further analysis is recommended as it might induce other problems within the assets.

Appendix

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec  4 23:14:49 2019

@author: geoffreylau
"""
## initialization
import numpy as np
import cvxopt as cv
import matplotlib.pyplot as plt

## the following code is equivalent to the matlab code provide in the file
## hw_2_data.m

n = 8
corr = np.zeros((n,n))
for i in range (n):
   for j in range (n):
      corr[i][j] = pow(-1,abs(i-j))/(abs(i-j)+1)
sigma = np.zeros((n,1))
mu = np.zeros((n,1))
sigma[0] = 2
mu[0] = 3
for i in range(0,n-1):
   sigma[i+1] = sigma[i] + 2 * np.random.random_sample()
   mu[i+1] = mu[i] + 1
D = np.diagflat(sigma)
C2 = D @ corr @ D
C = 0.5 * (C2 + np.transpose(C2))

## end of translation of given matlab code

##stopping cvxopt showing optimization progress
cv.solvers.options['show_progress'] = False

## Exercise 1 begins
## setting up the 25 values for r, from 3 to 9 with steps of 0.25
r = np.linspace(3.,9.,25)
## formatting matrix into cvxopt matrix, preparing for qp function
u = cv.matrix(mu)
p = cv.matrix(C)
q = cv.matrix([0.0 for i in range(n)])
G = - cv.matrix(np.eye(n))
h = cv.matrix(0.0, (n ,1))
```

```
A = cv.matrix([[1.0 for i in range(n)],[u]]).trans()
## sig and mean stores the standard deviation and mean of the 25 values
sig = np.zeros(len(r))
mean = np.zeros(len(r))
## running qp for 25 times, storing into sig and mean array each time
for i in range(len(r)):
   b = cv.matrix([1.0,r[i]])
   temp = cv.solvers.qp(p,q,G,h,A,b)
   temp2 = np.matrix(temp['x'])
   sig[i] = np.sqrt(temp2.transpose() @ C @ temp2)
   mean[i] = mu.transpose() @ temp2

plt.plot(sig,mean,'o-')
plt.ylabel('mean')
plt.xlabel('std')
plt.title('Exercise 1')
plt.show()
## Excerise 1 ends

## Exercise 2 begins

## reformatting matrix with the changes in constraints

e = cv.matrix(1.0,(1,n))
G2 = cv.matrix([G,e],(n+1,n))
h2 = cv.matrix([h,1.0],(n+1,1))
A2 = cv.matrix([u]).trans()
## sig2 and mean2 stores the standard deviation and mean of the 25 values
sig2 = np.zeros(len(r))
mean2 = np.zeros(len(r))
for i in range(len(r)):
   b2 = cv.matrix([r[i]])
   temp = cv.solvers.qp(p,q,G2,h2,A2,b2)
   temp2 = np.matrix(temp['x'])
   sig2[i] = np.sqrt(temp2.transpose() @ C @ temp2)
   mean2[i] = mu.transpose() @ temp2
## plotting the original exercise with this exercise
plt.plot(sig,mean,'o-')
plt.plot(sig2,mean2,'yo-')
plt.ylabel('mean')
plt.xlabel('std')
plt.title('Exercise 2')
plt.show()
## Exercise 2 ends

## Exercise 3 begins
```

```
## reformatting due to constraint changes
G3 = cv.matrix([G,-u.trans()],(n+1,n))
A3 = e
b3 = cv.matrix(1.0)
sig3 = np.zeros(len(r))
mean3 = np.zeros(len(r))
## sig3 and mean3 stores the standard deviation and mean of the 25 values
for i in range(len(r)):
   h3 = cv.matrix([h,-r[i]],(n+1,1))
   temp = cv.solvers.qp(p,q,G3,h3,A3,b3)
   temp2 = np.matrix(temp['x'])
   sig3[i] = np.sqrt(temp2.transpose() @ C @ temp2)
   mean3[i] = mu.transpose() @ temp2
## plotting the original exercise with this exercise
plt.plot(sig,mean,'o-')
plt.plot(sig3,mean3,'go-')
plt.ylabel('mean')
plt.xlabel('std')
plt.title('Exercise 3')
plt.show()
## Exercise 3 ends

## Exercise 4 begins
## reformatting due to constraint changes
G4 = cv.matrix(0.0,(n,n))
sig4 = np.zeros(len(r))
mean4 = np.zeros(len(r))
## sig4 and mean4 stores the standard deviation and mean of the 25 values
for i in range(len(r)):
   b = cv.matrix([1.0,r[i]])
   temp = cv.solvers.qp(p,q,G4,h,A,b)
   temp2 = np.matrix(temp['x'])
   sig4[i] = np.sqrt(temp2.transpose() @ C @ temp2)
   mean4[i] = mu.transpose() @ temp2
## plotting the original exercise with this exercise
plt.plot(sig,mean,'o-')
plt.plot(sig4,mean4,'ro-')
plt.ylabel('mean')
plt.xlabel('std')
plt.title('Exercise 4')
plt.show()
## Exercise 4 ends
```