CF969-7-SU-CO
Big-Data for Computational Finance
Academic Year: 2023/24
Assignment 1
Deadline: Check FASER

Before you begin

Please refer to the Student's handbook on the School's Policy on Plagiarism and Late Submission.

The two deliverables below must be uploaded on FASER by the deadline as independent items, i.e., not bundled together in a zip file.

Assignment – Purpose

You are asked to formulate Markowitz model for portfolio optimisation as a quadratic optimization problem, and solve it using Python, for example using Gurobipy or any other appropriate solver such as qpsolvers[1]. I repeat the model below.

Model

The model can briefly be described as follows. Assume we have some capital and n different assets. A fraction x_j of the total capital is invested in asset j, for $j = 1, \ldots, n$. The annual (or daily) return of each asset is modelled as a random variable ξ_j

The random variables can be correlated and it is assumed that the expected values

$\mu_j = E[\xi_j]$ and the covariance matrix $C_{ij} = E[(\xi_i - \mu_i)(\xi_j - \mu_j)]$ are known, for any $i, j = 1, \ldots, n$. The annual return of an asset is defined as the relative change of the asset price in a year. For given expected returns $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$, given correlation matrix $\mathbf{C}$ and a given expected return of the portfolio, $r = \mu^T\boldsymbol{x}$, the Markowitz portfolio optimization problem is then to determine the vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ that minimises the variance of the portfolio, i.e.

$$\begin{aligned} \text{minimise} \quad & \boldsymbol{x}^T C\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{\mu}^T\boldsymbol{x} = r \\ & \boldsymbol{e}^T\boldsymbol{x} = 1, \text{ where } \mathbf{e} = (1, \ldots, 1)^T \\ & \boldsymbol{x} \geq \mathbf{0}. \end{aligned}$$

You can use any solver (e.g., Gurobi, gurobipy, qpsolvers etc) for solving a quadratic program.

[1] https://pypi.org/project/qpsolvers/

<u>Problems to be solved</u>

Task 1 (25%)

Solve the above problem for the following 29 different values of the right hand side *r*:

r = 2.00, 2.25, 2.50, 2.75, 3.00, 3.25, 3.50, 3.75, …, 8.50, 8.75, 9.00.

Save the obtained 29 values of $\sigma(\boldsymbol{x}) = \sqrt{\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x}}$ and $\mu(\boldsymbol{x}) = \boldsymbol{\mu}^T\boldsymbol{x}$ in two vectors. Plot these vectors in a figure showing σ on the horizontal and μ on the vertical axis. Use the following code for generating your vector $\boldsymbol{\mu}$ and matrix ***C***. Note that the generated data uses input of random numbers, so that the values of the matrix ***C*** will vary with each run. Note also that you are expected to replace variables dig1 and dig2 below with the appropriate registration number digits

```
import numpy as np
import random
n = 10
# replace dig1 with the second-to-last digit of your registration number
# replace dig2 with the last digit of your registration number
dig1 = 0
dig2 = 0
dummyrepetitions = 10*dig1+dig2
for _ in range(dummyrepetitions):
      dummy = random.uniform(0,1)
Corr = np.array([[0]*n for _ in range(n)], dtype = float)
for i in range(n):
      for j in range(n):
            Corr[i][j] = (-1)**abs(i-j)/(abs(i-j)+1)
ssigma = np.array([[0]*1 for _ in range(n)], dtype = float)

# mmu is the vector μ in the assignment
mmu = np.array([[0]*1 for _ in range(n)], dtype = float)
ssigma[0] = 2
mmu[0] = 3
for i in range(n-1):
      ssigma[i+1] = ssigma[i] + 2*random.uniform(0,1)
      mmu[i+1] = mmu[i] + 1
ddiag = np.array([[0]*n for _ in range(n)], dtype = float)
np.fill_diagonal(ddiag, ssigma)
C2 = np.matmul(np.matmul(ddiag,Corr), ddiag)

# C is the matrix C in the assignment
C = 0.5*(C2 + C2.T)
```

Task 2 (25%)

Repeat Task 1 but modified as follows. Assume that it is not necessary to invest the whole capital, and that the not invested fraction $1 - \sum_j x_j$ of the whole capital can be saved without any return and without any "risk". Justify why this situation can be modelled by simply changing the constraint

$$\boldsymbol{e}^T\boldsymbol{x} = 1, \text{ where } \mathbf{e} = (1, \ldots, 1)^T$$

in the initial problem to

$$\boldsymbol{e}^T\boldsymbol{x} \leq 1, \text{ where } \mathbf{e} = (1, \ldots, 1)^T$$

Then comment on the difference between your obtained new figure and your figure from Task 1.

Task 3 (25%)

Repeat Task 1 but now modified as follows. Assume that the constraint

$$\boldsymbol{\mu}^T\boldsymbol{x} = r$$

in the initial problem is changed to

$$\boldsymbol{\mu}^T\boldsymbol{x} \geq r$$

Explain what this change implies and explain the difference between your obtained new figure and your figure from Task 1.

Task 4 (25%)

Repeat Task 1 but now modified as follows. Assume that so called "short selling" is possible. This essentially means that it is possible to borrow assets today and immediately sell them for today's market price. Then, at a future date agreed upon, (e.g., one year from today) the borrowed assets must be bought back, for the market price at that future date, and returned to the lender.

Motivate that this situation can be modelled by simply removing the constraints

$$\boldsymbol{x} \geq \boldsymbol{0}$$

in the initial problem. Then comment briefly on the difference between your obtained new figure and your figure from Task 1.

What am I grading you on?

On your ability to model and solve optimisation problems, such as those addressed during the first part of the module and on your ability to formulate your insights in a concise way into a report. Your ability to implement and solve the tasks is only half of the challenge. Your interpretation of the findings matters equally.

Deliverables for Assignment 1:

1. A document discussing each task, presenting the relevant plots, and commenting on each task's results. It should explains the problem faced at each task, refer to and describe the theoretical background and present the results.

   This document should be either a pdf file or a Python notebook file.

2. The source code, with a clear separation between the different tasks. E.g., a Python notebook file in case you are using gurobipy or another Python solver such as qpsolvers, the Gurobi file in case you are using Gurobi.