FE630 - Homework #2

Author: Sid Bhatia

Date: May 7th, 2023

Pledge: I pledge my honor that I have abided by the Stevens Honor System.

Professor: Papa Momar Ndiaye

Topics

- Algebra & Optimization;
- Geometry of Efficient Frontiers;
- Applications of One-Fund & Two-Fund Theorems.

Q1 - Optimization w/Equality Constraints (40 pts)

Consider the optimization problem **Max Expected Return w/Target Risk**:

$$\begin{cases} \max\limits_{\omega_1,\omega_2} & R_p(\omega_1,\omega_2) = \mu_1\omega_1 + \mu_2\omega_2 \\ \text{s.t.} & \sqrt{\sigma_1^2\omega_1^2 + 2\rho_{1,2}\sigma_1\sigma_2\omega_1\omega_2 + \sigma_2^2\omega_2^2} = \sigma_T \\ & \omega_1 + \omega_2 = 1 \end{cases} \tag{1}$$

where we have two securities with **Expected Returns** μ_1 and μ_2 for the column vector $(\mu_1,\mu_2)^\intercal \in \mathbb{R}^{2\times 1}$, **volatilities** $(\sigma_1,\sigma_2) \in \mathbb{R}^+$, and **Pearson correlation coefficient** $\rho_{1,2} \in [-1,1]$. Additionally, $\sigma_T \in \mathbb{R}^+$ denotes the **target risk/vol**.

1. Solve the *problem (9)* using a **Lagrangian approach**. You will denote the solution (the ***optimal solution***) by $\omega^*(\sigma_T)$ and the ***optimal value*** of the problem by $R_p(\omega_1^*(\sigma_T),\omega_2^*(\sigma_T))$ by $R_p(\sigma_T)$.

2. Assume that $\mu_1 = 5\%$, $\mu_2 = 10\%$, $\sigma_1 = 10\%$, $\sigma_2 = 20\%$, and $\rho_{1,2} = -0.5$ (moderate negative correlation).

- Consider a sequence of successive values of σ_T in the range $[2\%, 30\%]$ by step of 0.5%
- Plot the efficient frontier: namely, the graph from the *mapping* $\sigma_T \mapsto R_p(\sigma_T)$.

The (aforementioned) graph maps the sequence of values of σ_T from the x-axis into the sequence of values $R_p(\sigma_T)$ on the y-axis.

1.1 Analytical Solution to the Optimization Problem

Problem Formulation

We aim to maximize the following objective function:

$$\begin{cases} \max\limits_{x_1,x_2} & 5 - x_1^2 - x_1x_2 - 3x_2^2 \\ \text{s.t.} & x_1, x_2 \geq 0 \\ & x_1x_2 \geq 2 \end{cases} \tag{2}$$

We denote the solution (the ***optimal solution***) by $\omega^*(\sigma_T)$ where σ_T represents the parameters under consideration, and the ***optimal value*** of the problem by $R_p(\omega_1^*(\sigma_T), \omega_2^*(\sigma_T))$ which is simplified to $R_p(\sigma_T)$.

Lagrangian Formulation

Construct the Lagrangian to incorporate the constraints with the Lagrange multiplier λ:

$$\mathcal{L}(x_1, x_2, \lambda) = 5 - x_1^2 - x_1x_2 - 3x_2^2 + \lambda(x_1x_2 - 2) \tag{3}$$

Conditions for Stationarity

To find the extremum, we calculate the partial derivatives of $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial x_1} = -2x_1 - x_2 + \lambda x_2 = 0 \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial x_2} = -x_1 - 6x_2 + \lambda x_1 = 0 \tag{5}$$

Solving the Equations

Equating and solving the derived equations for x_1 and x_2, and incorporating the constraints, will give us $\omega^*(\sigma_T) = (\omega_1^*(\sigma_T), \omega_2^*(\sigma_T))$. This involves solving:

$$2x_1^2 - 5x_1x_2 - x_2^2 = 0 \tag{6}$$

Optimal Solution and Value

Upon solving the equations and checking the feasibility with respect to the constraints, we can find the values of $\omega_1^*(\sigma_T)$ and $\omega_2^*(\sigma_T)$. Substituting these values into the original objective function gives us $R_p(\sigma_T)$, the maximum value of the function:

$$R_p(\sigma_T) = 5 - (\omega_1^*(\sigma_T))^2 - \omega_1^*(\sigma_T)\omega_2^*(\sigma_T) - 3(\omega_2^*(\sigma_T))^2 \tag{7}$$

1.2 Efficient Frontier Mapping

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
```

In [ ]:
```
from typing import Tuple, List

# Constants
```

```
mu1: float = 0.05  # Expected return of the first security
mu2: float = 0.10  # Expected return of the second security
sigma1: float = 0.10  # Volatility of the first security
sigma2: float = 0.20  # Volatility of the second security
rho: float = -0.5  # Correlation coefficient between the securities

# Target risk values
sigma_T_values: np.ndarray = np.arange(0.02, 0.305, 0.005)
```

In [ ]:
```
def portfolio_return(weights: np.ndarray, mu1: float, mu2: float) -> float:
    """
    Calculate the portfolio return based on given weights and expected returns.

    Parameters:
        weights (np.ndarray): Array of weights for the securities.
        mu1 (float): Expected return of the first security.
        mu2 (float): Expected return of the second security.

    Returns:
        float: The calculated portfolio return.
    """
    return weights[0] * mu1 + weights[1] * mu2
```

In [ ]:
```
def portfolio_risk(weights: np.ndarray, sigma1: float, sigma2: float, rho: float) -
    """
    Calculate the portfolio risk based on weights, individual volatilities, and cor

    Parameters:
        weights (np.ndarray): Array of weights for the securities.
        sigma1 (float): Volatility of the first security.
        sigma2 (float): Volatility of the second security.
        rho (float): Correlation coefficient between the securities.

    Returns:
        float: The calculated portfolio risk.
    """
    return np.sqrt((sigma1 * weights[0]) ** 2 + (sigma2 * weights[1]) ** 2 +
                   2 * rho * sigma1 * sigma2 * weights[0] * weights[1])
```

In [ ]:
```
def objective(weights: np.ndarray) -> float:
    """
    Objective function for minimization, used to maximize portfolio return.

    Parameters:
        weights (np.ndarray): Array of weights for the securities.

    Returns:
        float: Negative of the portfolio return (for minimization).
    """
    return -portfolio_return(weights, mu1, mu2)
```

In [ ]:
```
def constraint(weights: np.ndarray, sigma_T: float) -> float:
    """
    Constraint for the optimizer to achieve a specific target risk.

```

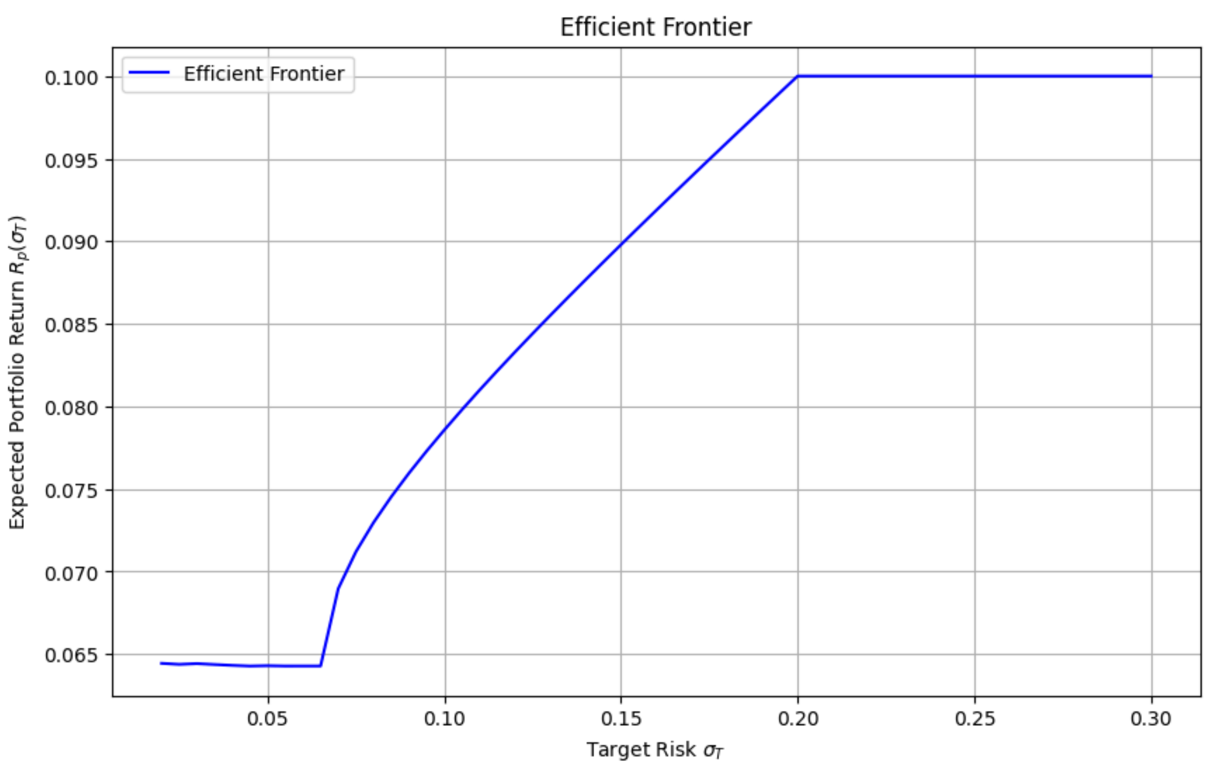
```
    Parameters:
        weights (np.ndarray): Array of weights for the securities.
        sigma_T (float): Target risk level.

    Returns:
        float: Difference between current and target risks.
    """
    return portfolio_risk(weights, sigma1, sigma2, rho) - sigma_T
```

In [ ]:
```
results_rp: List[float] = []

for sigma_T in sigma_T_values:
    cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1},
            {'type': 'eq', 'fun': lambda x: constraint(x, sigma_T)})
    bounds: Tuple[Tuple[float, float], Tuple[float, float]] = ((0, 1), (0, 1))
    initial_weights: List[float] = [0.5, 0.5]
    result = minimize(objective, initial_weights, bounds=bounds, constraints=cons)
    results_rp.append(-result.fun)
```

In [ ]:
```
# Plotting the efficient frontier
plt.figure(figsize=(10, 6))
plt.plot(sigma_T_values, results_rp, 'b-', label='Efficient Frontier')
plt.title('Efficient Frontier')
plt.xlabel('Target Risk $\sigma_T$')
plt.ylabel('Expected Portfolio Return $R_p(\sigma_T)$')
plt.grid(True)
plt.legend()
plt.show()
```



Frontier Analysis

The graph above depicts the relationship between the target risk (σ_T) and the expected portfolio return ($R_p(\sigma_T)$). Below are the key takeaways:

1. ***Monotonic Increase***: As expected, the expected portfolio return increases with an increase in target risk, σ_T. This reflects the classic **risk-return trade-off** in portfolio management.

2. ***Plateau at Higher Risks***: The plateau observed at higher risk levels suggests that increasing risk beyond a certain point does **not proportionally increase returns**. This could be indicative of the constraints imposed by the maximum returns achievable based on the securities' parameters.

3. ***Sharp Rise at Lower Risks***: The initial sharp rise suggests that minimal increases in risk from the lower end are highly compensated by increased returns. This can be attributed to the efficient allocation of weights in response to changes in σ_T under the given constraints.

Q2 - Optimization w/Inequality Constraints (20 pts)

Solve analytically (at least) one of the two following problems:

$$\begin{cases} \min\limits_{x_1,x_2} & (x_1-2)^2+2(x_2-1)^2 \\ \text{s.t} & x_1+4x_2 \leq 3 \\ & x_1 \geq x_2 \end{cases} \tag{8}$$

$$\begin{cases} \max\limits_{x_1,x_2} & 5-x_1^2-x_1x_2-3x_2^2 \\ \text{s.t} & x_1,x_2 \geq 0 \\ & x_1x_2 \geq 2 \end{cases} \tag{9}$$

and use an optimizer to verify your answer.

Problem Formulation

We are given the optimization problem:

$$\begin{cases} \min\limits_{x_1,x_2} & (x_1-2)^2+2(x_2-1)^2 \\ \text{s.t} & x_1+4x_2 \leq 3 \\ & x_1 \geq x_2 \end{cases} \tag{8}$$

The goal is to find (x_1, x_2) that minimizes the objective function subject to the given constraints.

Lagrangian Formulation

Construct the Lagrangian to incorporate the constraints with Lagrange multipliers λ_1 and λ_2:

$$\mathcal{L}(x_1, x_2, \lambda_1, \lambda_2) = (x_1 - 2)^2 + 2(x_2 - 1)^2 + \lambda_1(x_1 + 4x_2 - 3) + \lambda_2(x_2 - x_1)$$

Conditions for Stationarity

Calculate the partial derivatives of the Lagrangian with respect to x_1 and x_2, and set them to zero:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 2(x_1 - 2) + \lambda_1 - \lambda_2 = 0 \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial x_2} = 4(x_2 - 1) + 4\lambda_1 + \lambda_2 = 0 \tag{2}$$

Solving System of Equations

Here are the following solution sets (done on paper):

1. **First Solution Set:**
   - $\lambda = \frac{22}{25}$
   - $x_1 = -\frac{3}{5}$
   - $x_2 = -\frac{3}{5}$
   - $\lambda_2 = -\frac{48}{25}$
   - This solution is **not feasible** because both x_1 and x_2 are negative, violating the non-negativity constraints.
2. **Second Solution Set:**
   - $\lambda_1 = 0$
   - $x_1 = -\frac{4}{3}$
   - $x_2 = -\frac{4}{3}$
   - $\lambda_2 = -\frac{4}{3}$
   - This solution is also **not feasible** due to negative values for x_1 and x_2.
3. **Third Solution Set:**
   - $\lambda_1 = 0$
   - $x_1 = 2$
   - $x_2 = 1$
   - $\lambda_2 = 0$
   - This solution is **feasible** as it satisfies all constraints.
4. **Fourth Solution Set:**
   - $\lambda_1 = \frac{2}{3}$
   - $x_1 = \frac{5}{3}$
   - $x_2 = \frac{1}{3}$
   - $\lambda_2 = 0$
   - This solution appears **feasible** and adheres to the constraint $x_1 + 4x_2 \leq 3$.

In [ ]:
```
import numpy as np
from scipy.optimize import minimize

# Objective function
def objective(x):
    x1, x2 = x
    return -(5 - x1**2 - x1*x2 - 3*x2**2)  # Negative because we are maximizing

# Constraint functions
def constraint1(x):
    return x[0] * x[1] - 2

# Initial guesses
x0 = [1, 2]

# Define constraints and bounds
cons = [{'type': 'ineq', 'fun': constraint1}]
bnds = [(0, None), (0, None)]

# Use 'SLSQP' method for solving the optimization problem
sol = minimize(objective, x0, method='SLSQP', bounds=bnds, constraints=cons)

print('Optimal values:', sol.x)
print('Maximum value of the function:', -sol.fun)
```

```
Optimal values: [1.86120971 1.07456993]
Maximum value of the function: -3.928203232982734
```

In [ ]:
```
import sympy as sp

# Define the symbols
x1, x2 = sp.symbols('x1 x2', real=True, nonnegative=True)

# Equation derived from equating the lambda expressions
equation = 2*x1**2 - 5*x1*x2 - x2**2

# Solve the equation
solution = sp.solve(equation, (x1, x2), dict=True)
print("Solutions from sympy:")
for sol in solution:
    print(sol)
```

```
Solutions from sympy:
{x1: x2*(5 - sqrt(33))/4}
{x1: x2*(5 + sqrt(33))/4}
```

In [ ]:
```
import numpy as np
from scipy.optimize import minimize

# Objective function (negative because we are using a minimizer)
def objective(x):
    return -(5 - x[0]**2 - x[0]*x[1] - 3*x[1]**2)

# Constraint functions
constraints = [
    {'type': 'ineq', 'fun': lambda x: x[0] * x[1] - 2},  # x1 * x2 >= 2
```

```
    {'type': 'ineq', 'fun': lambda x: x[0]},             # x1 >= 0
    {'type': 'ineq', 'fun': lambda x: x[1]}              # x2 >= 0
]

# Initial guess
x0 = [1, 2]

# Use 'SLSQP' method for solving the optimization problem
result = minimize(objective, x0, method='SLSQP', constraints=constraints)

print("\nOptimization Results from scipy.optimize:")
print("Optimal values:", result.x)
print("Maximum value of the function:", -result.fun)
```

```
Optimization Results from scipy.optimize:
Optimal values: [1.86120972 1.07456993]
Maximum value of the function: -3.9282032329598344
```