foam

January 5, 2025

1 Mean-Variance Model

1.1 Exercise 2.

- Task 0: Cleaning Data and Importing Libraries:

[ ]:
```
#pip install gurobipy
#pip install openpyxl
```

[ ]:
```
#import required libraries:
import gurobipy as gb
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

I used gurobipy library to optimize the portfolio, the other libraries are for data manipulations and plotting the graphs. We first need to make sure we correctly loaded the data.

[ ]:
```
df=pd.read_excel("/workspaces/FOAM/data/data.xlsx", sheet_name="Returns S&P Mib␣
 ↪30 ", index_col=0, parse_dates=True)
df.head(2)
```

We can drop the index name, and change the date format appropriately (back to the given format)

[ ]:
```
df.index.name=None
df.index = df.index.strftime('%b-%Y')
df.head(2)
```

Since the stocks are given in the range from A1 to A20, I prepared dictionary to replace them with their respective names. So that we can plot the efficient frontier including the real asset names.

[ ]:
```
stocks = [
    "AL", "AGL", "AUTO", "NTV", "BFI", "BIN", "BPM", "BPVN", "BNL", "BPU",␣
 ↪"BUL", "CAP",
    "EDN", "ENEL", "ENI", "FWB", "F", "FNC", "G", "ES"
] # we can later use this list to refer to the columns of the dataset.

stock_dict = {f"A{i+1}": stock for i, stock in enumerate(stocks)}
```

```
# now we can replace them
df = df.rename(columns=stock_dict)
df.head(2)
```

1.2 Exercise 2.1

- Task 1: Compute the expected returns of all assets in the market and the matrix of variances and covariances.
- Task 2: Formulate and solve the Markowitz model for finding the maximum possible expected return value for an efficient portfolio (ER_max)
- Task 3: Formulate and solve the Markowitz model to find the minimum possible expected return value for an efficient portfolio (ER_min).
- Task 4: Fix 5 different values for the portfolio expected return in the range (ER_min, ER_max), denoting them by , , , , .

1.2.1 Task 1: Compute the expected returns of all assets in the market and the matrix of variances and covariances

Expected returns are found as follows, since the data frame is already about the returns we can just take the `.mean()` (mean) for an asset for the whole period.

[ ]:
```
expected_returns = df[stocks].mean()
expected_returns.head(2)
```

Compute variance and covariance matrix with `.cov()` function, where main diagnal is variance and the rest are covariance values.

[ ]:
```
covariance_matrix = df[stocks].cov()
covariance_matrix.head(2)
```

---

2 Model Implementation

2.0.1 Task 2: Formulate and solve the Markowitz model for finding the maximum possible expected return value for an efficient portfolio (ER_{max})

Here, since we want to find ER_{max}, we need to set the objective function to `MAXIMIZE` the portfolio return.

[ ]:
```
# Create model
model_one = gb.Model()
```

```
# Add variables for portfolio weights
x = pd.Series(model_one.addVars(stocks, lb=0, name='X'), index=stocks)

# Compute portfolio variance (to use as constraint) and return (to use in␣
 ↪objective funtion)
portfolio_variance = covariance_matrix.dot(x).dot(x)
portfolio_return = expected_returns.dot(x)

# Add objective function: Maximize portfolio return
model_one.setObjective(portfolio_return, sense=gb.GRB.MAXIMIZE)

# Add budget constraint: Sum of weights equals 1
model_one.addConstr(x.sum() == 1, name="budget")

# Add portfolio variance constraint: Variance less than or equal to␣
 ↪sigma_squared
# Here I relaxed the constraint because if we fix the sigma, the model could␣
 ↪not find
# the optimal solution and the output was infeasable or unbound.
sigma_squared = 2
model_one.addConstr(portfolio_variance <= sigma_squared,␣
 ↪name="variance_constraint")

# Optimize model
model_one.optimize()

# Display results
if model_one.status == gb.GRB.OPTIMAL:
    model_one_weights = pd.Series({stock: x[stock].X for stock in stocks})
    print("Optimal Portfolio Weights:")
    print(model_one_weights)
    print(f"ER_max: {portfolio_return.getValue()}")
    print(f"Portfolio Variance: {portfolio_variance.getValue()}")
else:
    print("No feasible solution found.")
```

[ ]:
```
plt.figure(figsize=(10, 6), dpi=150)
model_one_weights.plot(kind="bar", color="cyan", edgecolor="skyblue")
plt.title(r"Portfolio Weights ($ER_{max}$)", fontsize=20)
plt.xlabel("Assets", fontsize=18)
plt.ylabel("Weights", fontsize=18)
plt.xticks(rotation=45, fontsize=18)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
# plt.savefig("/Users/javlon/Documents/GitHub/RedPill/FOAM/Analysis/
 ↪Portfolio_Weights_for_ER_max", dpi=300)
plt.show()
```

2.0.2 Task 3: Formulate and solve the Markowitz model to find the minimum possible expected return value for an efficient portfolio (ER_min).

But in this task, we are asked to find ER_{min}, so we need to set the objective function to `MINIMIZE` the portfolio return.

[ ]:
```
model_two = gb.Model()

x = pd.Series(model_two.addVars(stocks, lb=0, name='X'), index=stocks)

portfolio_variance = covariance_matrix.dot(x).dot(x)
portfolio_return = expected_returns.dot(x)

# everything is the same as above except this part, where maximize is replaced␣
 ↪with minimize
model_two.setObjective(portfolio_return, sense=gb.GRB.MINIMIZE)

model_two.addConstr(x.sum() == 1, name="budget")

sigma_squared = 2
model_two.addConstr(portfolio_variance <= sigma_squared,␣
 ↪name="variance_constraint")

model_two.optimize()

if model_two.status == gb.GRB.OPTIMAL:
    model_two_weights = pd.Series({stock: x[stock].X for stock in stocks})
    print("Optimal Portfolio Weights:")
    print(model_two_weights)
    print(f"ER_min: {portfolio_return.getValue()}")
    print(f"Portfolio Variance: {portfolio_variance.getValue()}")
else:
    print("No feasible solution found.")
```

[ ]:
```
plt.figure(figsize=(10, 6), dpi=150)
model_two_weights.plot(kind="bar", color="orangered", edgecolor="red")
plt.title(r"Portfolio Weights ($ER_{min}$)", fontsize=20)
plt.xlabel("Assets", fontsize=18)
plt.ylabel("Weights", fontsize=18)
plt.xticks(rotation=45, fontsize=18)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
# plt.savefig("/Users/javlon/Documents/GitHub/RedPill/FOAM/Analysis/
 ↪Portfolio_Weights_for_ER_min", dpi=300)
plt.show()
```

2.0.3 Task 4: Fix 5 different values for the portfolio expected return in the range (ER_min, ER_max), denoting them by , , , , .

We have already found values for ER_{min} ( ) and ER_{max} ( ), we now need to another 3 values for μ's so that there will be 5 equally spaced portfolio returns.

[ ]:
```
ER_min = -0.009408867882700435
ER_max = 0.03348577185008574

mu_values = np.linspace(ER_min, ER_max, 5)
mu1, mu2, mu3, mu4, mu5 = mu_values

print(f"mu1: {mu1:.8f}")
print(f"mu2: {mu2:.8f}")
print(f"mu3: {mu3:.8f}")
print(f"mu4: {mu4:.8f}")
print(f"mu5: {mu5:.8f}")
```

2.1 Exercise 2.2

- Solve model (MwV) formulated in 1.1 for each possible value $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5$. Denote by $P_{\mu 1}, P_{\mu 2}, P_{\mu 3}, P_{\mu 4}, P_{\mu 5}$ the optimal portfolios so found.

So above, when we find the $ER_{min} and ER_{max}$, we input the portfolio variance as constraint and the portfolio return as objective function, here we switch the places.

[ ]:
```
# Define the list of mu values
mu_values = [mu1, mu2, mu3, mu4, mu5]

# Create a dictionary to store results
m3_results = {}

# List to store data for plotting the efficient frontier
m3_efficient_frontier = []

for i, mu in enumerate(mu_values, 1):
    # Create a new model
    m3 = gb.Model()

    # Add variables for portfolio weights
    x = pd.Series(m3.addVars(stocks, lb=0, name='X'), index=stocks)

    # Compute portfolio variance and return
    portfolio_variance = covariance_matrix.dot(x).dot(x)
    portfolio_return = expected_returns.dot(x)

    # Add objective function: Minimize portfolio variance
    m3.setObjective(portfolio_variance, sense=gb.GRB.MINIMIZE)
```

```
    # Add budget constraint: Sum of weights equals 1
    m3.addConstr(x.sum() == 1, name="budget")

    # Add portfolio return constraint
    m3_trc = m3.addConstr(portfolio_return == mu, name="return_constraint")

    # Optimize model
    m3.optimize()

    # Store results if feasible solution found
    if m3.status == gb.GRB.OPTIMAL:
        m3_weights = pd.Series({stock: x[stock].X for stock in stocks})
        m3_portfolio_variance_value = portfolio_variance.getValue()

        # Store results in a dictionary
        m3_results[f"P_mu_{i}"] = {
            "mu_value": mu,
            "m3_weights": m3_weights,
            "m3_portfolio_variance": m3_portfolio_variance_value,
        }

        # Add data to the efficient frontier list
        m3_efficient_frontier.append({
            "mu": mu,
            "m3_variance": m3_portfolio_variance_value,
            "m3_weights": m3_weights,
        })
    else:
        m3_results[f"P_mu_{i}"] = "No feasible solution found"

# Display results
for key, result in m3_results.items():
    if isinstance(result, dict):
        print(f"\nResults for {key}:")
        print(f"mu value: {result['mu_value']:.8f}")
        print(f"{key}:")
        print(result['m3_weights'])
        print(f"Portfolio Variance: {result['m3_portfolio_variance']:.8f}")
    else:
        print(f"\nResults for {key}: {result}")
```

[ ]:
```
# Plotting the results
for key, result in m3_results.items():
    if isinstance(result, dict):
        m3_weights = result["m3_weights"]
        mu_value = result["mu_value"]
```

```
        # Create a bar plot for the weights
        plt.figure(figsize=(10, 6), dpi=150)
        m3_weights.plot(kind="bar", color="skyblue", edgecolor="black")
        plt.title(f"$P_{{\\mu_{key[-1]}}}$ ($\mu$={result['mu_value']:.4f})",␣
↪fontsize=20)
        plt.xlabel("Assets", fontsize=18)
        plt.ylabel("Weight", fontsize=18)
        plt.xticks(rotation=45, fontsize=18)
        plt.tight_layout()

        # Save the plot
        # plt.savefig(f"P_mu{key[-1]}_weights.png")
        plt.show()
```

2.2 Exercise 2.3

- For each possible value $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5$, solve model (MwV) with the additional constraint formulated in 1.2 by fixing $k = 2$. Denote by $PC_{\mu_1}, PC_{\mu_2}, PC_{\mu_3}, PC_{\mu_4}, PC_{\mu_5}$ the optimal portfolios so found.

[ ]:
```
# Define the list of mu values
mu_values = [mu1, mu2, mu3, mu4, mu5]

# Create a dictionary to store results
m4_results = {}

# List to store data for plotting the efficient frontier
m4_efficient_frontier = []

#fix the cardinality constraint
k = 2

for i, mu in enumerate(mu_values, 1):
    # Create a new model
    m4 = gb.Model()

    # Add variables for portfolio weights
    x = pd.Series(m4.addVars(stocks, lb=0, name='X'), index=stocks)

    # Introduce binary variables to indicate if a stock is selected
    y = pd.Series(m4.addVars(stocks, vtype=gb.GRB.BINARY, name='Y'),␣
↪index=stocks)

    # Compute portfolio variance and return
    portfolio_variance = covariance_matrix.dot(x).dot(x)
    portfolio_return = expected_returns.dot(x)
```

```
    # Add objective function: Minimize portfolio variance
    m4.setObjective(portfolio_variance, sense=gb.GRB.MINIMIZE)

    # Add budget constraint: Sum of weights equals 1
    m4.addConstr(x.sum() == 1, name="budget")

    # Add portfolio return constraint
    m4_trc = m4.addConstr(portfolio_return == mu, name="return_constraint")

    # Add cardinality constraint
    for stock in stocks:
        cardinality_constraint = m4.addConstr(x[stock] <= y[stock],␣
 ↪name=f"selection_{stock}")
    m4.addConstr(y.sum() == k, name="cardinality_constraint")

    # Optimize model
    m4.optimize()

    # Store results if feasible solution found
    if m4.status == gb.GRB.OPTIMAL:
        m4_weights = pd.Series({stock: x[stock].X for stock in stocks})
        m4_portfolio_variance_value = portfolio_variance.getValue()

        # Store results in a dictionary
        m4_results[f"PC_mu_{i}"] = {
            "mu_value": mu,
            "m4_weights": m4_weights,
            "m4_portfolio_variance": m4_portfolio_variance_value,
        }

        # Add data to the efficient frontier list
        m4_efficient_frontier.append({
            "mu": mu,
            "m4_variance": m4_portfolio_variance_value,
            "m4_weights": m4_weights,
        })
    else:
        m4_results[f"PC_mu_{i}"] = "No feasible solution found"

# Display results
for key, result in m4_results.items():
    if isinstance(result, dict):
        print(f"\nResults for {key}:")
        print(f"mu value: {result['mu_value']:.8f}")
        print(f"{key}:")
        print(result['m4_weights'])
        print(f"Portfolio Variance: {result['m4_portfolio_variance']:.8f}")
```

```
    else:
        print(f"\nResults for {key}: {result}")
```

[ ]:
```
# Plotting the results
for key, result in m4_results.items():
    if isinstance(result, dict):
        m4_weights = result["m4_weights"]
        mu_value = result["mu_value"]

        # Create a bar plot for the weights
        plt.figure(figsize=(10, 6), dpi=150)
        m4_weights.plot(kind="bar", color="skyblue", edgecolor="black")
        plt.title(f"$PC_{{\\mu_{key[-1]}}}$ ($\mu$={result['mu_value']:.4f} and␣
 ↪k=2)", fontsize=20)
        plt.xlabel("Assets", fontsize=18)
        plt.ylabel("Weight", fontsize=18)
        plt.xticks(rotation=45, fontsize=18)
        plt.tight_layout()

        # Save the plot
        # plt.savefig(f"PC_mu{key[-1]}_weights.png")
        plt.show()
```

[ ]:
```
# Extract data for plotting
m4_mu_values_plot = [entry["mu"] for entry in m4_efficient_frontier]
m4_variances_plot = [entry["m4_variance"] for entry in m4_efficient_frontier]
m3_mu_values_plot = [entry["mu"] for entry in m3_efficient_frontier]
m3_variances_plot = [entry["m3_variance"] for entry in m3_efficient_frontier]

stock_stds = df[stocks].std()
stock_returns = df[stocks].mean()


plt.figure(figsize=(10, 6), dpi=150)
plt.plot(m4_variances_plot, m4_mu_values_plot, marker="o", color='red',␣
 ↪label="Efficient Frontier ($PC_μ$)")
plt.plot(m3_variances_plot, m3_mu_values_plot, marker="o", color='blue',␣
 ↪label="Efficient Frontier ($P_μ$)")
# Visualize stocks

plt.xlabel("Standard Deviation ($\sigma$)", fontsize=18)
plt.ylabel("Expected Return ($\mu$)", fontsize=18)
plt.title("Efficient Frontier ($PC_μ$ and $P_μ$)", fontsize=20)
plt.legend()
plt.grid()
plt.show()
```