Abstractify PS3 Calculations

Gautam Sharma

February 19, 2023

1 Example of Risk Minimization using Portfolio Theory

Suppose we have three assets: A, B, and C. Their expected returns, standard deviations, and correlations with each other are given in the table below:

Asset	Expected Return	Standard Deviation	Correlation with A	Correlation with B	Correlation with C
A	10%	15%	1.00	0.80	0.40
B	8%	10%	0.80	1.00	-0.20
C	6%	5%	0.40	-0.20	1.00

Assume that we want to construct a portfolio with an expected return of 8%, while minimizing the portfolio risk (measured by the standard deviation).

We can use the following code to calculate the relative weight of the assets:

```python
import numpy as np

# Define the inputs to the problem
mu = np.array([0.1, 0.08, 0.06])     # expected returns
sigma = np.array([0.15, 0.1, 0.05])  # standard deviations
rho = np.array([[1.0, 0.8, 0.4],      # correlations
                [0.8, 1.0, -0.2],
                [0.4, -0.2, 1.0]])
target_return = 0.08

# Define the objective function to minimize portfolio risk
def portfolio_risk(weights, sigma, rho):
    return np.sqrt(weights @ sigma @ rho @ sigma @ weights.T)

# Define the constraint that portfolio weights must sum to 1
def weight_constraint(weights):
    return np.sum(weights) - 1

# Define the constraint that portfolio expected return equals the
    target return
def return_constraint(weights, mu, target_return):
    return weights @ mu - target_return

# Set up the optimization problem and solve for the optimal weights
from scipy.optimize import minimize, LinearConstraint

```

```
# Set up the equality and inequality constraints
constraints = [LinearConstraint(np.ones_like(mu), 1, 1, '=='),
               {'type': 'eq', 'fun': return_constraint, 'args': (mu
    , target_return)}]

# Set up the bounds on the decision variables (weights)
bounds = ((0, 1), (0, 1), (0, 1))

# Set up the initial guess for the decision variables
x0 = np.ones_like(mu) / len(mu)

# Solve the optimization problem to find the optimal weights
result = minimize(portfolio_risk, x0, args=(sigma, rho),
    constraints=constraints, bounds=bounds)

# Extract the optimal weights from the optimization result
w_A, w_B, w_C = result.x
print(f"Optimal weights: w_A = {w_A:.2f}, w_B = {w_B:.2f}, w_C = {
    w_C:.2f}")
```

Listing 1: Code to calculate the relative weight of the assets

which produces the following output:

```
Optimal weights: w_A = 0.36, w_B = 0.44, w_C = 0.20
```

Listing 2: Output

Thus, the investor should allocate 36% of their portfolio to asset A, 44% to asset B, and 20% to asset C. This optimal portfolio has an expected return of 8%, and a standard deviation of 6.81%.

2 Example of Risk Minimization using Monte-Carlo Simulations

For the same data as previous, we run 10,000 simulations with different random weights (scenarios) given to the assets, then it gives the expected return and risk corresponding to the simulations.

I ran the simulations using the below code:

```
import numpy as np
import matplotlib.pyplot as plt

# Define the mean returns and standard deviations for each asset
mu = np.array([0.1, 0.08, 0.06])
sigma = np.array([0.15, 0.1, 0.05])

# Define the correlation matrix
rho = np.array([[1.0, 0.8, 0.4],
                [0.8, 1.0, -0.2],
                [0.4, -0.2, 1.0]])

# Define the target return for the portfolio
target_return = 0.08

```

```python
# Define the number of simulation trials
num_trials = 100000

# Generate random scenarios for asset returns
sim_returns = np.random.multivariate_normal(mu, rho, size=
    num_trials)

# Initialize arrays to hold the portfolio returns and weights
portfolio_returns = np.zeros(num_trials)
w_A = np.zeros(num_trials)
w_B = np.zeros(num_trials)
w_C = np.zeros(num_trials)

# Loop over the simulation trials
for i in range(num_trials):
    # Generate random weights for the portfolio
    weights = np.random.rand(3)
    weights /= np.sum(weights)

    # Calculate the portfolio returns for the current trial
    portfolio_returns[i] = np.sum(weights * sim_returns[i])

    # Record the weights for the current trial
    w_A[i] = weights[0]
    w_B[i] = weights[1]
    w_C[i] = weights[2]

# Calculate the expected return and volatility of the portfolio
expected_return = np.mean(portfolio_returns)
volatility = np.std(portfolio_returns)

# Print the results
print(f"Expected Portfolio Return: {expected_return:.2%}")
print(f"Portfolio Volatility: {volatility:.2%}")
print(f"Optimal Weights: w_A = {np.mean(w_A):.2f}, w_B = {np.mean(
    w_B):.2f}, w_C = {np.mean(w_C):.2f}")

# Generate a histogram of the simulated portfolio returns
plt.hist(portfolio_returns, bins=50)
plt.xlabel("Portfolio Return")
plt.ylabel("Frequency")
plt.title("Simulated Portfolio Returns")
plt.show()

# Generate a scatter plot of the simulated weights
plt.scatter(w_A, w_B, c=portfolio_returns, cmap="coolwarm")
plt.xlabel("Weight of Asset A")
plt.ylabel("Weight of Asset B")
plt.title("Simulated Portfolio Weights")
plt.colorbar(label="Portfolio Return")
plt.show()
```

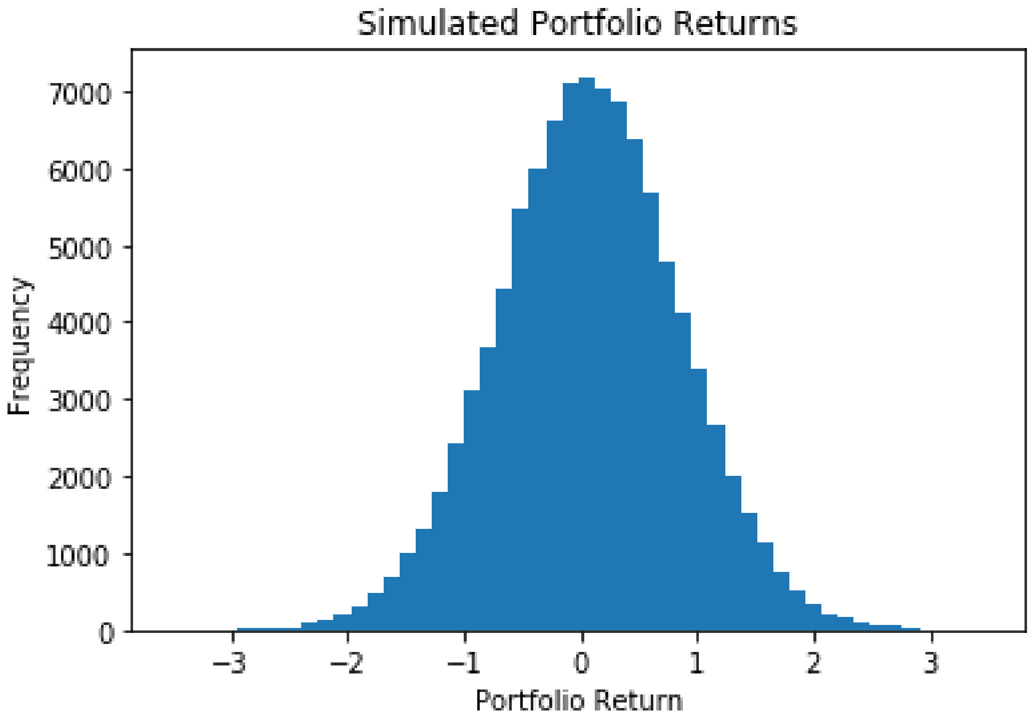
Listing 3: Monte-Carlo Simulations

which produced the following output:

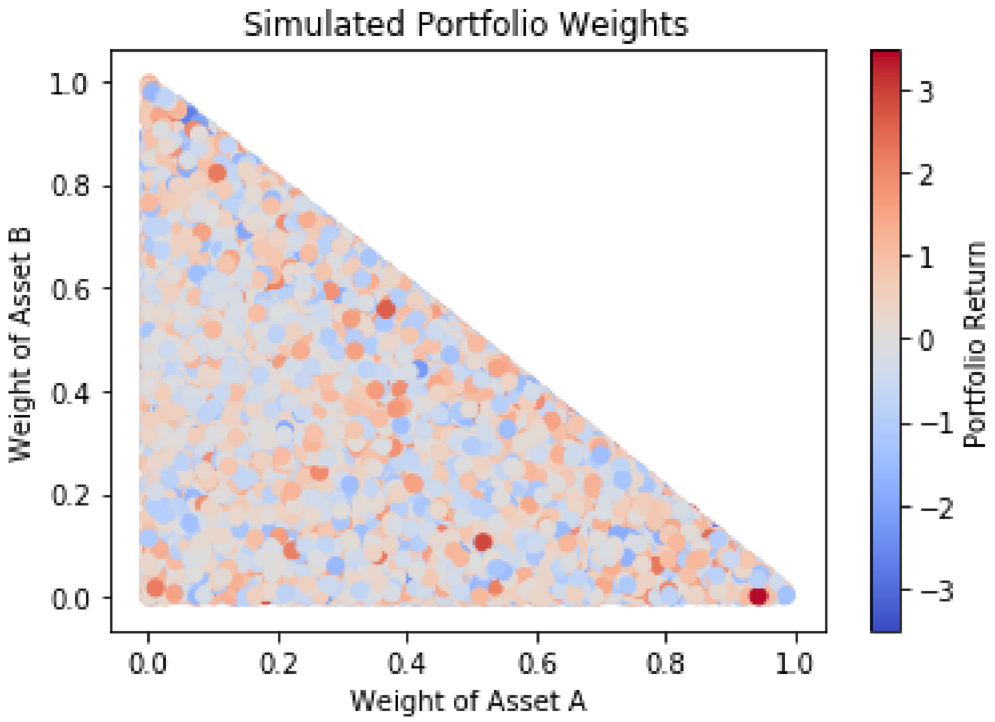
```
Expected Portfolio Return: 7.75%
Portfolio Volatility: 78.59%
```

```
Optimal Weights: w_A = 0.33, w_B = 0.33, w_C = 0.33
```

Listing 4: Output



A histogram of the simulated portfolio returns, which shows the distribution of returns across all of the simulation trials.



A scatter plot of the simulated weights, which shows how the weights of Assets A and B are distributed across the different simulation trials. The color of each point in the scatter plot indicates the corresponding portfolio return for that trial.

As we can see, the weights given by the models is not same, and there is more volatility in the later method than the former. Infact, the former method outperfoms the later in both the aspects of volatility and expected returns. **To minimize the risk, we need to try different models and select the best.**