Mandatory Assignment 1

Jakob Espinoza, Jan Zeeberg og Anders Bruun

2024-02-03

1 Imports

```
# imports
import numpy as np
import pandas as pd
import yfinance as yf
import scipy.optimize as sco
import matplotlib.pyplot as plt
```

2 Problem 1

Follow the exercise sets to download daily adjusted prices for all constituents of the Dow Jones 30 index for the period from January 1st, 2000 until December 31st, 2023 from Yahoo!Finance. Remove all tickers with no continuous trading history for the entire sample (hint: you should end up with $= 27$ assets). Compute monthly returns for each of the tickers.

```
# get down jones tickers
djones_data = pd.read_excel(r'holdings-daily-us-en-dia.xlsx', skiprows=4)
djones_ticks = list(djones_data['Ticker'].dropna())

# collect adjusted prices
rawdata = pd.DataFrame(yf.download(
    tickers=djones_ticks,
    start="2000-01-01",
    end="2023-12-31"
))
adj_close_hist = rawdata['Adj Close'].copy()
adj_close_hist.columns = djones_ticks

# remove non-continuos assets
```

```
prices = adj_close_hist.dropna(axis=1, how='any')
djones_ticks = list(prices.columns)
print(f'Number of assets in cleaned dataset: {len(prices.columns)}\n')

# calc monthly returns for each asset
prices.index = pd.to_datetime(prices.index)
monthly_returns = prices.resample('ME').last().pct_change()
monthly_returns = monthly_returns.dropna()
```

```
Number of assets in cleaned dataset: 27
```

3 Problem 2

Compute the sample mean and the variance-covariance matrix Σ of the monthly returns.[1] Which of the individual assets delivered the highest Sharpe ratio (assume the risk-free rate is zero) during the sample period?

```
# Compute the sample means
mu = monthly_returns.mean()

# Compute the variance-covariance matrix
Sigma = monthly_returns.cov()

# Calculate Sharpe ratio for each asset
sharpe_ratios = mu / monthly_returns.std()

# Identify the asset with the highest Sharpe ratio
max_sharpe_tick = sharpe_ratios.idxmax()
max_sharpe = sharpe_ratios.max()

print(f"The asset with the highest Sharpe ratio is {max_sharpe_tick} with a Sharpe ratio o
```

```
The asset with the highest Sharpe ratio is CSCO with a Sharpe ratio of 0.27
```

4 Problem 3

Define a function compute_efficient_frontier which takes at least two inputs: a $\times$ variance-covariance matrix Sigma_est and a vector mu_est.

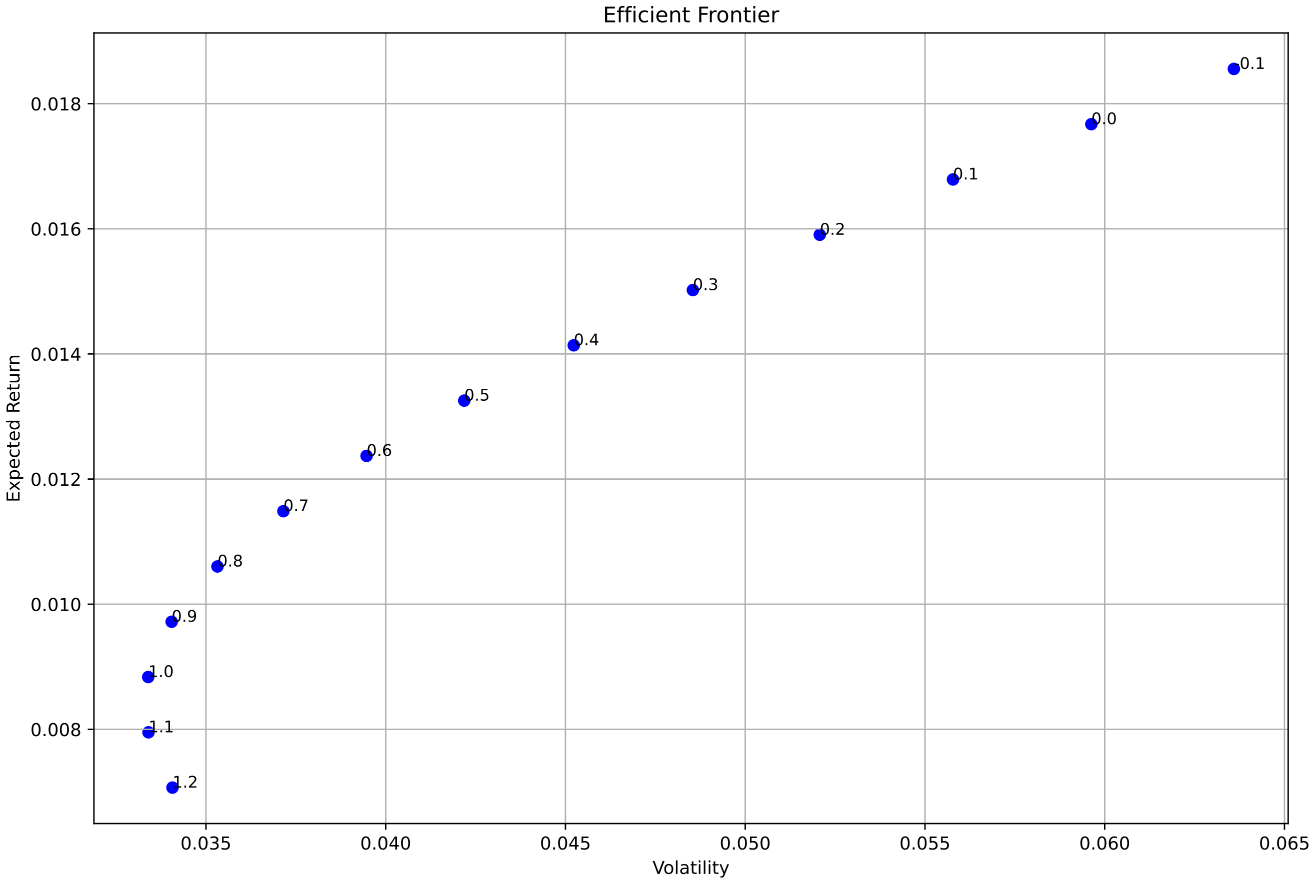
```
'Func for computing efficient frontier'
def compute_efficient_frontier(Sigma_est, mu_est):

    # Number of assets
    N = len(mu_est)

    # Objective function for minimum variance portfolio
    def min_variance(weights):
        return weights.T @ Sigma_est @ weights

    # Constraints
    weights_sum_to_one = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}

    # Initial guess for weights
    initial_guess = np.ones(N) / N

    # Bounds for weights
    bounds = tuple((0, 1) for asset in range(N))

    # Optimization for minimum variance portfolio
    min_var_result = sco.minimize(min_variance, initial_guess, method='SLSQP', bounds=bound
    omega_mvp = min_var_result.x

    # Compute expected return of MVP
    mvp_return = mu_est @ omega_mvp

    # Objective function for efficient portfolio
    def eff_portfolio(weights):
        return -mu_est @ weights  # We want to maximize return, hence the negative

    # Additional constraint for efficient portfolio to have double the MVP's return
    return_constraint = {'type': 'eq', 'fun': lambda weights: mu_est @ weights - 2 * mvp_r

    # Optimization for efficient portfolio
    eff_var_result = sco.minimize(eff_portfolio, initial_guess, method='SLSQP', bounds=bou
    omega_eff = eff_var_result.x

    # Compute range of portfolio weights using two-fund theorem
    c_values = np.arange(-0.1, 1.21, 0.1)  # Range of c values from -0.1 to 1.2
    portfolios = []

    for c in c_values:
        omega_c = c * omega_mvp + (1 - c) * omega_eff
        portfolios.append(omega_c)

    # Create a DataFrame to return the results
```

```
    df = pd.DataFrame(portfolios, columns=[f'Asset {i+1}' for i in range(N)])
    df['c'] = c_values

    return df[['c'] + [f'Asset {i+1}' for i in range(N)]]
```

5 Problem 4

Use the output of the function compute_efficient_frontier(Sigma_est, mu_est, …) to visualize the theoretically optimal efficient frontier in a diagram with volatility on the x-axis and expected returns on the y-axis based on the true parameters Σ and

```
# get efficient_frontier df
efficient_frontier_df = compute_efficient_frontier(Sigma, mu)

# define assets in the df
assets = [f'Asset {i+1}' for i in range(len(mu))]

# Initialize returns and volatilit lists
expected_returns = []
volatilities = []

# Compute expected return and volatility for each value of c
for index, row in efficient_frontier_df.iterrows():
    omega_c = row[assets].values  # Portfolio weights
    expected_return = np.dot(omega_c, mu)  # Expected return
    volatility = np.sqrt(np.dot(omega_c.T, np.dot(Sigma, omega_c)))  # Volatility

    expected_returns.append(expected_return)
    volatilities.append(volatility)

# Plotting the efficient frontier with c's as labels for eahc datapoint
plt.figure(figsize=(12, 8))
for i, txt in enumerate(efficient_frontier_df['c']):
    plt.scatter(volatilities[i], expected_returns[i], c='blue', marker='o')
    plt.text(volatilities[i], expected_returns[i], f'{txt:.1f}', fontsize=9)

plt.title('Efficient Frontier')
plt.xlabel('Volatility')
plt.ylabel('Expected Return')
plt.grid(True)
plt.show()
```




```r
knitr::purl("MA1.qmd")
```