Sid Bhatia - FE630 ~ Midterm (PDF)

March 25, 2024

0.0.1 FE630 - Midterm Project

Author: Sid Bhatia

Date: March 25th, 2023

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

Professor: Papa Momar Ndiaye

Question 1. (15 pts) The supplied `data.zip` file contains 30 space-delimited text files that contain price and volume data for 30 companies. Each row of each file contains date, opening price, closing price, high price, low price, volume, and adjusted closing price (last column). You will need that data for question 1.

Write a program `processdata` to:

1. Read all daily price files;
2. Create a price matrix `P` by aligning the data's dates and placing the adjusted closing prices side-by-side in columns;
3. From the `P matrix`, create a matrix of simple (not logarithmic) daily returns `R`;
4. Compute the vector of average daily returns mu for the companies using the `mean` function (do not use loops);
5. Compute the covariance matrix `Q` from the return matrix using the `cov` function; and
6. Save the return vector `mu` and the covariance matrix `Q` in the native format for your programming language in a file called `inputs.ext`, where `ext` is the appropriate extension for a binary file in your language.

[ ]:
```python
import pandas as pd
import numpy as np
import os
from typing import List

def processdata(data_dir: str = 'data') -> None:
    """
    Processes stock price data to compute and save the average daily return␣
    ↪vector and covariance matrix.

    This function reads stock price data from text files, each containing data␣
    ↪for a company, then:
```

```
    1. Creates a price matrix with adjusted close prices,
    2. Calculates the daily return matrix,
    3. Computes the vector of average daily returns for each company,
    4. Computes the covariance matrix of the return matrix,
    5. Saves the average daily returns vector and the covariance matrix to␣
↪binary files.

    Parameters:
    - data_dir (str): The directory containing the stock price files. Default␣
↪is 'data'.

    Returns:
    - None. The function saves two files: 'inputs_mu.pkl' and 'inputs_Q.pkl'␣
↪with the results.
    """
    # List to store the adjusted close price data for each company.
    price_data: List[pd.Series] = []

    # Loop through each file in the specified directory.
    for file in os.listdir(data_dir):
        if file.endswith('.txt'):
            filepath = os.path.join(data_dir, file)
            # Read data, assuming space-separated values without an explicit␣
↪header.
            df = pd.read_csv(filepath, sep=' ', header=None,
                             names=['Date', 'Open', 'Close', 'High', 'Low',␣
↪'Volume', 'Adj Close'])
            # Set date as the index for easy alignment later.
            df.set_index('Date', inplace=True)
            # Append the adjusted close price series to the list.
            price_data.append(df['Adj Close'])

    # Concatenate all the adjusted close prices side-by-side, aligning by date.
    P = pd.concat(price_data, axis=1)
    P.sort_index(inplace=True)  # Ensure the dates are in order.

    # Calculate daily returns by comparing each price with the previous day's␣
↪price.
    R = P.pct_change().dropna()  # Drop the first row since its percentage␣
↪change is undefined.

    # Calculate the vector of average daily returns for each company.
    mu = R.mean(axis=0)

    print(mu)
```

```
    # Calculate the covariance matrix of the daily returns.
    Q = R.cov()

    print(Q)

    # Save the vector of average daily returns and the covariance matrix as␣
 ↪binary files.
    mu.to_pickle('inputs_mu.pkl')
    Q.to_pickle('inputs_Q.pkl')

processdata()
```

```
Adj Close    0.000225
Adj Close    0.000473
Adj Close    0.000945
Adj Close    0.000540
Adj Close   -0.000340
Adj Close    0.000563
Adj Close   -0.000295
Adj Close    0.000339
Adj Close    0.001099
Adj Close    0.000425
Adj Close    0.001033
Adj Close    0.001010
Adj Close   -0.000270
Adj Close    0.000669
Adj Close    0.000521
Adj Close    0.000637
Adj Close    0.000240
Adj Close    0.000290
Adj Close    0.000705
Adj Close    0.000468
Adj Close    0.000901
Adj Close    0.000491
Adj Close    0.000233
Adj Close    0.000139
Adj Close    0.000595
Adj Close    0.001255
Adj Close    0.000200
Adj Close    0.000226
Adj Close    0.000022
Adj Close   -0.000107
dtype: float64
           Adj Close  Adj Close  Adj Close  Adj Close  Adj Close  Adj Close
Adj Close   0.000342   0.000079   0.000080   0.000097   0.000104   0.000069  \
Adj Close   0.000079   0.000144   0.000066   0.000088   0.000062   0.000051
Adj Close   0.000080   0.000066   0.000169   0.000076   0.000058   0.000056
Adj Close   0.000097   0.000088   0.000076   0.000215   0.000075   0.000073
```

Adj Close	0.000104	0.000062	0.000058	0.000075	0.000166	0.000059
Adj Close	0.000069	0.000051	0.000056	0.000073	0.000059	0.000190
Adj Close	0.000100	0.000056	0.000050	0.000074	0.000077	0.000054
Adj Close	0.000084	0.000060	0.000058	0.000073	0.000061	0.000058
Adj Close	0.000075	0.000071	0.000066	0.000080	0.000055	0.000059
Adj Close	0.000076	0.000063	0.000062	0.000074	0.000077	0.000062
Adj Close	0.000058	0.000062	0.000057	0.000069	0.000050	0.000047
Adj Close	0.000103	0.000057	0.000066	0.000079	0.000076	0.000088
Adj Close	0.000064	0.000041	0.000050	0.000059	0.000053	0.000065
Adj Close	0.000088	0.000063	0.000059	0.000069	0.000068	0.000076
Adj Close	0.000047	0.000057	0.000053	0.000051	0.000045	0.000045
Adj Close	0.000083	0.000083	0.000071	0.000136	0.000072	0.000071
Adj Close	0.000039	0.000044	0.000037	0.000038	0.000027	0.000035
Adj Close	0.000046	0.000041	0.000043	0.000044	0.000041	0.000039
Adj Close	0.000076	0.000065	0.000066	0.000070	0.000064	0.000061
Adj Close	0.000046	0.000053	0.000047	0.000048	0.000044	0.000052
Adj Close	0.000076	0.000063	0.000056	0.000071	0.000071	0.000077
Adj Close	0.000055	0.000055	0.000046	0.000055	0.000043	0.000048
Adj Close	0.000037	0.000043	0.000037	0.000039	0.000030	0.000039
Adj Close	0.000052	0.000043	0.000034	0.000040	0.000043	0.000040
Adj Close	0.000054	0.000055	0.000049	0.000069	0.000047	0.000045
Adj Close	0.000072	0.000052	0.000062	0.000079	0.000053	0.000058
Adj Close	0.000073	0.000068	0.000078	0.000072	0.000068	0.000059
Adj Close	0.000050	0.000052	0.000041	0.000040	0.000040	0.000034
Adj Close	0.000044	0.000041	0.000039	0.000041	0.000031	0.000034
Adj Close	0.000089	0.000053	0.000049	0.000066	0.000072	0.000051

	Adj Close	Adj Close	Adj Close	Adj Close	…	Adj Close	
Adj Close	0.000100	0.000084	0.000075	0.000076	…	0.000076	\
Adj Close	0.000056	0.000060	0.000071	0.000063	…	0.000063	
Adj Close	0.000050	0.000058	0.000066	0.000062	…	0.000056	
Adj Close	0.000074	0.000073	0.000080	0.000074	…	0.000071	
Adj Close	0.000077	0.000061	0.000055	0.000077	…	0.000071	
Adj Close	0.000054	0.000058	0.000059	0.000062	…	0.000077	
Adj Close	0.000141	0.000060	0.000053	0.000068	…	0.000062	
Adj Close	0.000060	0.000132	0.000064	0.000062	…	0.000065	
Adj Close	0.000053	0.000064	0.000148	0.000058	…	0.000067	
Adj Close	0.000068	0.000062	0.000058	0.000130	…	0.000057	
Adj Close	0.000049	0.000053	0.000066	0.000052	…	0.000061	
Adj Close	0.000068	0.000065	0.000049	0.000060	…	0.000082	
Adj Close	0.000053	0.000052	0.000050	0.000061	…	0.000068	
Adj Close	0.000061	0.000057	0.000060	0.000063	…	0.000098	
Adj Close	0.000047	0.000050	0.000052	0.000052	…	0.000049	
Adj Close	0.000075	0.000071	0.000073	0.000074	…	0.000077	
Adj Close	0.000037	0.000036	0.000044	0.000038	…	0.000040	
Adj Close	0.000037	0.000038	0.000044	0.000047	…	0.000048	
Adj Close	0.000056	0.000059	0.000061	0.000066	…	0.000059	
Adj Close	0.000052	0.000048	0.000046	0.000045	…	0.000051	

Adj Close	0.000062	0.000065	0.000067	0.000057	…	0.000228
Adj Close	0.000050	0.000048	0.000058	0.000058	…	0.000048
Adj Close	0.000043	0.000036	0.000045	0.000042	…	0.000044
Adj Close	0.000038	0.000038	0.000043	0.000043	…	0.000046
Adj Close	0.000049	0.000053	0.000059	0.000054	…	0.000051
Adj Close	0.000059	0.000053	0.000063	0.000060	…	0.000061
Adj Close	0.000057	0.000059	0.000068	0.000066	…	0.000057
Adj Close	0.000039	0.000044	0.000044	0.000044	…	0.000052
Adj Close	0.000030	0.000038	0.000042	0.000037	…	0.000046
Adj Close	0.000101	0.000059	0.000054	0.000064	…	0.000057

	Adj Close	Adj Close	Adj Close	Adj Close	Adj Close	Adj Close	
Adj Close	0.000055	0.000037	0.000052	0.000054	0.000072	0.000073	\
Adj Close	0.000055	0.000043	0.000043	0.000055	0.000052	0.000068	
Adj Close	0.000046	0.000037	0.000034	0.000049	0.000062	0.000078	
Adj Close	0.000055	0.000039	0.000040	0.000069	0.000079	0.000072	
Adj Close	0.000043	0.000030	0.000043	0.000047	0.000053	0.000068	
Adj Close	0.000048	0.000039	0.000040	0.000045	0.000058	0.000059	
Adj Close	0.000050	0.000043	0.000038	0.000049	0.000059	0.000057	
Adj Close	0.000048	0.000036	0.000038	0.000053	0.000053	0.000059	
Adj Close	0.000058	0.000045	0.000043	0.000059	0.000063	0.000068	
Adj Close	0.000058	0.000042	0.000043	0.000054	0.000060	0.000066	
Adj Close	0.000053	0.000041	0.000038	0.000050	0.000057	0.000056	
Adj Close	0.000043	0.000029	0.000039	0.000047	0.000056	0.000057	
Adj Close	0.000041	0.000036	0.000041	0.000047	0.000062	0.000060	
Adj Close	0.000056	0.000039	0.000045	0.000044	0.000049	0.000055	
Adj Close	0.000057	0.000045	0.000038	0.000046	0.000059	0.000054	
Adj Close	0.000063	0.000044	0.000049	0.000067	0.000075	0.000070	
Adj Close	0.000039	0.000046	0.000033	0.000039	0.000036	0.000039	
Adj Close	0.000037	0.000034	0.000031	0.000039	0.000040	0.000036	
Adj Close	0.000050	0.000043	0.000039	0.000053	0.000058	0.000067	
Adj Close	0.000065	0.000041	0.000039	0.000041	0.000052	0.000047	
Adj Close	0.000048	0.000044	0.000046	0.000051	0.000061	0.000057	
Adj Close	0.000110	0.000044	0.000039	0.000047	0.000059	0.000050	
Adj Close	0.000044	0.000085	0.000038	0.000044	0.000042	0.000040	
Adj Close	0.000039	0.000038	0.000088	0.000036	0.000035	0.000041	
Adj Close	0.000047	0.000044	0.000036	0.000086	0.000053	0.000054	
Adj Close	0.000059	0.000042	0.000035	0.000053	0.000179	0.000064	
Adj Close	0.000050	0.000040	0.000041	0.000054	0.000064	0.000110	
Adj Close	0.000042	0.000037	0.000065	0.000036	0.000030	0.000043	
Adj Close	0.000038	0.000043	0.000035	0.000041	0.000045	0.000037	
Adj Close	0.000048	0.000042	0.000041	0.000047	0.000056	0.000055	

	Adj Close	Adj Close	Adj Close
Adj Close	0.000050	0.000044	0.000089
Adj Close	0.000052	0.000041	0.000053
Adj Close	0.000041	0.000039	0.000049
Adj Close	0.000040	0.000041	0.000066

```
Adj Close   0.000040   0.000031   0.000072
Adj Close   0.000034   0.000034   0.000051
Adj Close   0.000039   0.000030   0.000101
Adj Close   0.000044   0.000038   0.000059
Adj Close   0.000044   0.000042   0.000054
Adj Close   0.000044   0.000037   0.000064
Adj Close   0.000040   0.000048   0.000051
Adj Close   0.000042   0.000034   0.000060
Adj Close   0.000040   0.000034   0.000052
Adj Close   0.000046   0.000035   0.000060
Adj Close   0.000041   0.000041   0.000047
Adj Close   0.000050   0.000044   0.000070
Adj Close   0.000035   0.000036   0.000035
Adj Close   0.000032   0.000033   0.000038
Adj Close   0.000041   0.000038   0.000058
Adj Close   0.000043   0.000036   0.000047
Adj Close   0.000052   0.000046   0.000057
Adj Close   0.000042   0.000038   0.000048
Adj Close   0.000037   0.000043   0.000042
Adj Close   0.000065   0.000035   0.000041
Adj Close   0.000036   0.000041   0.000047
Adj Close   0.000030   0.000045   0.000056
Adj Close   0.000043   0.000037   0.000055
Adj Close   0.000100   0.000034   0.000040
Adj Close   0.000034   0.000084   0.000032
Adj Close   0.000040   0.000032   0.000112

[30 rows x 30 columns]
```

Question 2. (15 pts) Write a function called `port` that uses standard quadratic programming libraries that will:

- Take the set of input parameters `mu` (mean vector μ), `Q` (covariance matrix Q), and `tau` (risk tolerance τ), and return vector h that maximizes the following utility function U defined by:

$$U(h) = -\frac{1}{2}h^T Q h + \tau h^T \mu$$

subject to the constraints

$$0 \leq h_i \leq 0.1 \text{ for all } i, \text{ and}$$

$$\sum_{i=1}^{n} h_i = h^T e = 1$$

where n is the number of securities in the portfolio.

[ ]:
```
import cvxpy as cp

def port(mu: np.ndarray, Q: np.ndarray, tau: float) -> np.ndarray:
    """
```

```
    Solves the portfolio optimization problem to maximize the utility function
    under given constraints using quadratic programming.

    Parameters:
    - mu (np.ndarray): The mean return vector for the securities.
    - Q (np.ndarray): The covariance matrix of the securities' returns.
    - tau (float): The risk tolerance parameter of the utility function.

    Returns:
    - h (np.ndarray): The optimized portfolio weights vector.
    """

    # Number of securities
    n = mu.shape[0]

    ## print(n)

    # Portfolio weights variable
    h = cp.Variable(n)

    ## print(h)

    # Define utility function to maximize.
    utility = -0.5 * cp.quad_form(h, Q) + tau * cp.matmul(h.T, mu)

    ## print(utility)

    # Constraints: 0 <= h_i <= 0.1 for all i, and sum(h_i) == 1
    constraints = [0 <= h, h <= 0.1, cp.sum(h) == 1]

    ## print(constraints)

    # Problem definition
    problem = cp.Problem(cp.Maximize(utility), constraints)

    # Solve the problem.
    problem.solve()

    # Return the optimized portfolio weights.
    return h.value
```

Question 3. (15 pts) Write a program called `frontier` that will:

1. Load in your programming environment the data stored in the `inputs.ext`;
2. Create a sequence `TAU` containing numbers from zero to 0.5 in steps of 0.001;
3. Run through a loop for each value of your `TAU` sequence to

- Find the optimum portfolio with the given `m`, `Q`, and `tau` selected from `TAU`;

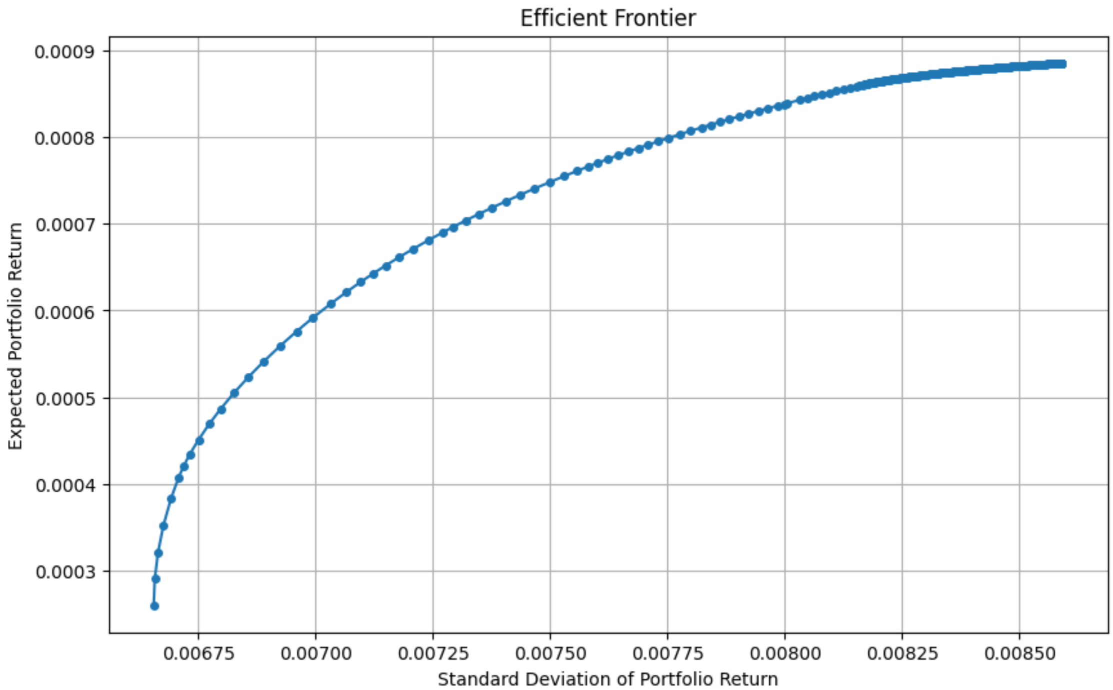
- Compute the optimum portfolio's expected return and standard deviation of return;
- Store the portfolio return and standard deviation.

4. After completing the loop, plot the efficient frontier.

[ ]:
```python
import matplotlib.pyplot as plt

def load_inputs(mu_path: str = 'inputs_mu.pkl', Q_path: str = 'inputs_Q.pkl')␣
↪-> (np.ndarray, np.ndarray): # type: ignore
    """
    Loads the mean returns vector and covariance matrix from the specified␣
↪files.

    Parameters:
    - mu_path (str): Path to the file containing the mean returns vector.
    - Q_path (str): Path to the file containing the covariance matrix.

    Returns:
    - tuple: A tuple containing the mean returns vector and the covariance␣
↪matrix.
    """
    mu = pd.read_pickle(mu_path).values
    Q = pd.read_pickle(Q_path).values
    return mu, Q

def compute_metrics(mu: np.ndarray, Q: np.ndarray, h: np.ndarray) -> (float,␣
↪float): # type: ignore
    """
    Computes the expected return and standard deviation of the portfolio.

    Parameters:
    - mu (np.ndarray): The mean return vector for the securities.
    - Q (np.ndarray): The covariance matrix of the securities' returns.
    - h (np.ndarray): The portfolio weights.

    Returns:
    - tuple: A tuple containing the expected return and standard deviation of␣
↪the portfolio.
    """
    expected_return = np.dot(mu.T, h)
    std_dev = np.sqrt(np.dot(h.T, np.dot(Q, h)))
    return expected_return, std_dev

def frontier():
    """
    Plots the efficient frontier by optimizing portfolios over a range of risk␣
↪tolerance values using the `port` function.
```

```
    Loads mean returns vector and covariance matrix, then calculates and plots␣
↪the efficient frontier.
    """
    # Load the mean returns vector and covariance matrix from saved files.
    mu, Q = load_inputs()
    TAU = np.arange(0, 0.501, 0.001)  # Range of risk tolerance values
    returns = []  # List to store expected returns of optimized portfolios
    std_devs = []  # List to store standard deviations of optimized portfolios

    for tau in TAU:
        # Use the previously defined port() function for portfolio optimization.
        h_opt = port(mu, Q, tau)
        # Compute expected return and standard deviation for the optimized␣
↪portfolio.
        ret, std = compute_metrics(mu, Q, h_opt)
        # Store the results.
        returns.append(ret)
        std_devs.append(std)

    # Plot the efficient frontier.
    plt.figure(figsize=(10, 6))
    plt.plot(std_devs, returns, 'o-', markersize=4)
    plt.title('Efficient Frontier')
    plt.xlabel('Standard Deviation of Portfolio Return')
    plt.ylabel('Expected Portfolio Return')
    plt.grid(True)
    plt.show()
```

[ ]:
```
frontier()
```


Efficient Frontier
Expected Portfolio Return
Standard Deviation of Portfolio Return
0.0009
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.00675
0.00700
0.00725
0.00750
0.00775
0.00800
0.00825
0.00850