Evaluative Assignment - Efficient Frontier

For this assignment, write a program to find the efficient frontier - the optimal set of portfolios, given a universe of assets and their correlations. Please note the Academic Integrity section.

Part One

Write a program "efficient_frontier", which takes two command line arguments:

1. The name of a file with a universe of assets. This file is in CSV format with no header row. Each line contains the asset name, the average rate of return, and the standard deviation of the rate of return.
2. The name of the file containing the correlation matrix for the universe of assets.

Provided Data

As sample data, universe.csv contains the following assets along with their average(annualized) returns and standard deviations of those returns: international equities, commodities, real estate, investment-grade corporate bonds, inflation-linked, medium-term (7-10yr) US Treasury bonds, and short-term (1-3yr) US Treasury bonds. correlation.csv contains the asset correlations from March 2009 to June 2015. (This period comes right after the end of the credit crisis bear market.)

Efficient Frontier

The efficient frontier is a hyperbola representing the boundary of all possible optimal portfolios that can be created from the assets provided.

From these assets, we need to find the optimal (least risky – i.e. lowest volatility) portfolio for each level of expected portfolio return.

A portfolio has a vector of weights, indicating the portion of the portfolio invested in each asset. These weights sum to 1 (i.e., 100%). The program must estimate the optimal weights, i.e. minimum volatility, for each return level between 1% and 26% in 1% increments.

The formula for portfolio volatility is

$$\sigma_p = \sqrt{\sum_i \sum_j w_i w_j cov_{ij}} = \sqrt{\sum_i \sum_j w_i w_j \sigma_i \sigma_j \rho_{ij}}$$

Where

- w_i = weight of the *i*th asset in the portfolio
- cov_{ij} = covariance between the *i*th and *j*th assets
- σ_i = standard deviation of the rates of return for the *i*th asset
- ρ_{ij} = correlation between the *i*th and *j*th assets

For this step, print a comma-separated list of the return and the minimum volatility for that rate of return. In this part, assume that short sales are allowed (the weights can be negative).

For example, the values for 1-4% are: (expected output is on the right side)

Rate of Return	Volatility
1%	0.78%
2%	1.09%
3%	1.59%
4%	2.13%

```
ROR,volatility
1.0%,0.78%
2.0%,1.09%
3.0%,1.59%
4.0%,2.13%
```

Optimizing this problem is tricky, as it is a constrained quadratic optimization problem. You are free to use any method you like to implement the optimization, but additional details are provided in the document ("Optimization approach for efficient frontier") in the assignment folder.. Additionally, this document demonstrates how to utilize a linear system of equations to perform the optimization.

Part Two

In this step, you will add additional functionality to the program. If the option "-r" is passed as the first command line argument (i.e., after the executable name), you should implement a restricted optimization, i.e., use an additional constraint that no short sales are allowed. That is, all weights must be non-negative. Many different ways exist to check for the "-r", use one that works best for you.

Again, print a comma-separated list of the return and the minimum volatility for that rate of return.

When no short sales are allowed, the values for 1-4% are: (0.98% is the correct value, not 0.96%)

```
ROR,volatility
1.0%,0.98%
2.0%,1.20%
3.0%,1.71%
4.0%,2.28%
```

Solving Systems of Linear Equations

To solve a linear system of equations, you can use the Eigen package, a well-documented, well-tested, and frequently used linear algebra package for C++: http://eigen.tuxfamily.org.

Eigen Installation

Refer to the matrix assignment for notes on how to install and/or reference Eigen.

Implementation Notes

- As mentioned, the executable must be named “efficient_frontier”.
- You must provide a Makefile. Recommended C++ version: std=C++17
- The Makefile must have targets of “efficient_frontier” and “clean”.
- If the –r flag is present, it will be in the 2nd position of the arguments - the first (index 0) is the program name. The asset file will then be specified, followed by the correlation matrix file.
- Review the relationship between standard deviations, covariance, and correlation.
- If you plan to use Eigen, spend some time getting comfortable with it. Start with the “Getting Started” page: https://eigen.tuxfamily.org/dox/GettingStarted.html
- Think about what errors may exist in the data files. You do not need to perform any type of data correction. When you find errors, print an appropriate message to standard error and exit with a status code of “EXIT_FAILURE”. When checking for data errors, you should perform these checks as close to the initial parsing as possible. Do not rely on checking invalid data after processing.
- Specific hints on error-checking:
    - Is your '-r' option optional? It should be.
    - Do you check for two files as command line arguments?
    - Do you verify that each file exists?
    - Do you verify that each file has all the required values? For example, if it's an N by N matrix, does each of the N rows have N numeric values?
    - Do you accept spaces in asset names?
    - Do you verify that each input that should be numeric is numeric?
    - For this assignment, values that should be numeric but aren't (nulls, spaces, words, etc...) can be reported as errors.
- Look towards using more of an object-oriented design. Nouns typically specify classes/properties, while verbs represent potential behaviors.
    - An Asset has a name, average rate of return, and standard deviation.
    - A Portfolio has a list of weights and a list of Assets, from which the portfolio volatility and rate of return can be calculated. You can add more attributes and functions.
    - Other classes and files as needed to support parsing and optimization.

Unrestricted Notes

From the Optimization Approach notes in the assignment folder, we can find the optimal solution where the gradients are 0, so we have the following equation.

$$\begin{bmatrix} \sigma & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}$$

σ is the covariance matrix

A is a 2 by n matrix (where n = number of assets) that has the form: (from equations 6, 7, 10)

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ r_1 & r_2 & \dots & r_n \end{bmatrix}$$

This produces the following matrix

0.0978	0.0208	-0.0003	-0.0064	-0.0035	0.0400	-0.0005	1.0000	0.2651
0.0208	0.0436	-0.0002	-0.0040	-0.0002	0.0228	-0.0003	1.0000	-0.0064
-0.0003	-0.0002	0.0035	0.0030	0.0023	0.0016	0.0003	1.0000	0.0791
-0.0064	-0.0040	0.0030	0.0048	0.0031	-0.0047	0.0005	1.0000	0.0407
-0.0035	-0.0002	0.0023	0.0031	0.0033	-0.0014	0.0003	1.0000	0.0391
0.0400	0.0228	0.0016	-0.0047	-0.0014	0.0640	-0.0005	1.0000	0.2505
-0.0005	-0.0003	0.0003	0.0005	0.0003	-0.0005	0.0001	1.0000	0.0093
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000	0.0000
0.2651	-0.0064	0.0791	0.0407	0.0391	0.2505	0.0093	0.0000	0.0000

b is a column vector of the form: $\begin{bmatrix} 1 \\ r_p \end{bmatrix}$ (from the right-hand side of equations 6, 7, 10)

$\begin{bmatrix} 0 \\ b \end{bmatrix}$ =

0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
1.0000
R_p

Now, solve a linear system of equations of the form $Ax = b$.

x contains the vector of weights and the Lagrange multipliers.

The Eigen package, https://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html provides sample code.

Restricted Notes

While the optimization approach document presents using a projection step with the primal-dual gradient method, we can also continue to utilize the matrix equation approach. Once we have an initial vector of weights, we look to see where any weights are negative. If a negative weight exists (e.g., for the *i*th asset), we add a linear equation to the "Matrix", setting that variable to 0. This means a new row with a 1 in the *i*th position is added to the "Matrix". We also need to add a "0" row to the column vector on the right-hand side of the equation. As the number of columns in the "Matrix" must equal the number of rows in both x and b, we need to add the transpose of that new row onto the matrix as well. This process repeats until no negative weights exist or the weight is within a small value of 0 such that it will not have any factor in the portfolio volatility equation. Each weight that is negative should only be added once.

Approach

Compute x for the unrestricted case

While x has negative members (alternatively, $x_i < -\varepsilon$, where ε is a sufficiently small number):

Add a linear equation for each member x_i that is negative to constrain that number to 0

Recompute weights by solving the updated linear system of equations.

Example Data:

Suppose weights 1 and 3 were negative. The updated matrix and vector becomes: (rate = .1)

```
 0.0978  0.0208 -0.0003 -0.0064 -0.0035  0.0400 -0.0005  1.0000  0.2651  0.0000  0.0000
 0.0208  0.0436 -0.0002 -0.0040 -0.0002  0.0228 -0.0003  1.0000 -0.0064  1.0000  0.0000
-0.0003 -0.0002  0.0035  0.0030  0.0023  0.0016  0.0003  1.0000  0.0791  0.0000  0.0000
-0.0064 -0.0040  0.0030  0.0048  0.0031 -0.0047  0.0005  1.0000  0.0407  0.0000  1.0000
-0.0035 -0.0002  0.0023  0.0031  0.0033 -0.0014  0.0003  1.0000  0.0391  0.0000  0.0000
 0.0400  0.0228  0.0016 -0.0047 -0.0014  0.0640 -0.0005  1.0000  0.2505  0.0000  0.0000
-0.0005 -0.0003  0.0003  0.0005  0.0003 -0.0005  0.0001  1.0000  0.0093  0.0000  0.0000
 1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  0.0000  0.0000  0.0000  0.0000
 0.2651 -0.0064  0.0791  0.0407  0.0391  0.2505  0.0093  0.0000  0.0000  0.0000  0.0000
 0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 0.0000  0.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000


0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
1.0000
0.1000
0.0000
0.0000
```

Grader

Unlike previous assignments, we will not provide direct results of the test cases. As this assignment forms part of your final evaluation for the course, you will be expected to create your test cases. The assignment grader will run your test cases against your submission and the reference implementation. You need to create a "testcases.txt" file that has two sections:

- *#error*: lists any test cases that should generate an error message and exit code.
- *#success*: lists any test cases that should run normally. The outputs from the two implementations will be compared.

You may create additional files (e.g. a file with bad asset data) as part of your submission.

Sample testcases.txt:

```
#error
universe_err.csv correlation.csv
universe.csv correlation_err1.csv
universe.csv correlation_err2.csv
universe.csv correlation_err3.csv

#success
universe.csv correlation.csv
```

Your testcases.txt will not be graded. The grading breakdown will be 95% for test cases generated by the instructors and 5% for a code review.

Submission

Submit your Makefile, all associated code files, testcases.txt, and any associated data files to the Gradescope assignment. For the data files, you do need to include the provided data files if they are part of your test cases.

Academic Integrity

Note: This is an evaluative assignment! By submitting this assignment, you agree that you have adhered to the following rules:

- Your work must be your own!
- You may NOT consult with other students about:
  - high-level approaches,
  - how to implement your algorithm in code, or
  - how to debug your code.
    (Basically, you may not discuss anything particular to this assignment).
- You may NOT look at another student's code, nor show your code to anyone else.
- You are responsible for keeping your code private.
- You may not look for solutions to this or similar problems online.
- You may not use code from any other source besides what is listed here.
- You may not use any AI assistive technology (e.g., ChatGPT)
- You MAY consult *Programming: Principles and Practices Using C++*.
- You MAY consult any docable page provided for this class.
- You MAY consult cppreference.com, cplusplus.com, eigen.tuxfamily.org
- You MAY consult any C++ book on the O'Reilly Learning Platform
- You MAY consult notes you wrote in your notebook.
- You MAY consult the man pages.
- You MAY ask the professor or TAs for clarification on the assignment.

If you use code from one of the approved sources, you should reference that source in a comment just prior to the function or code section. This includes any prior assignments.