Jorge Velásquez/Andrés Gonzalez

Portfolio Optimization

Nonlinear Programming

Master of Operation Research - U

FISICC, Galileo

PhD. Alberth Alvarado

Portfolio Investment Problem

The investment portfolio problem consists of determining how capital should be invested in different options, seeking to reduce risk while maximizing returns. The complexity of the problem lies in deciding the investment allocations because there may be many assets to invest, and the degree of benefit each asset can provide must be considered. Therefore, it is worth highlighting the following two key points to be addressed in this project:

- **Variability in returns.**
- **Risk mitigation and acceptance**

The Markowitz model is proposed to solve the problem. The model aims to find the optimal combination of investments within a set of predefined options that minimizes the risk incurred when the investor desires a given return. This model uses the mean variance of the known returns of the investments to determine the risk of obtaining unexpected results.

The Nonlinear Programming problem posed by the Markowitz model is as follows**:**

$$\begin{aligned} \text{mín} \quad & x^T \Sigma x \\ \text{s.a.} \quad & \mu^T x \geq R \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1, \ldots, n. \end{aligned}$$

Where vector x represents the ratio of the capital to be invested in each of the available options, ∑ is the covariance matrix of the average rates of return for each of the stocks, the vector μ contains the geometric mean of the rates of return for each of the options, and R is the minimum expected return on the investment.

The covariance matrix is defined as follows:

$$\Sigma_{ij} = \begin{cases} \sigma_i^2 & \text{si } i = j, \\ E[(r_i - \mu_i)(r_j - \mu_j)] & \text{si } i \neq j. \end{cases}$$

$$Let\ r = \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}, \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix}, therefore\ \ r - \mu = \begin{bmatrix} r_1 - \mu_1 \\ \vdots \\ r_n - \mu_n \end{bmatrix}, (r - \mu)^T = [r_1 - \mu_1 \quad \cdots \quad r_n - \mu_n]$$

$Given\ the\ matrix\ A = [a_{ij}], the\ notation\ is\ described\ as\ E[A] = [E(a_{ij})]$

$Therefore\ the\ covariance\ matrix\ can\ be\ written\ as$:

$$\Sigma = E[(r - \mu)(r - \mu)^T]$$

$To\ determine\ if\ \Sigma\ is\ positive\ we\ develop\ y^T\Sigma y$:

$$y^T\Sigma y = y^T E[(r-\mu)(r-\mu)^T]y$$
$$= E[y^T(r-\mu)(r-\mu)^T y] \quad (1)\ by\ linearity\ property\ of\ mean$$

$(r-\mu)\ as\ a\ scalar$:

$$y^T(r-\mu) = (r-\mu)^T y \qquad (2)\ by\ vector\ multiplication\ properties$$

$$y^T\Sigma y = E[(r-\mu)^T y(r-\mu)^T y] = E[((r-\mu)^T y)^2] \geq 0$$

$\boldsymbol{therefore\ \Sigma\ is\ positive\ semidefinite\ and\ x\hat{}T\ \Sigma x\ is\ convex}$.

In addition, the problem constraints represent a half-space, a hyperplane, and positive orthants, respectively. These are convex sets, and therefore the problem is convex.

Due to the above, we determine that we are dealing with a classic restricted quadratic problem and can use quadratic optimization tools.

Algorith implementation

To provide the optimal solution within an expected performance range, Python was used as the development tool. The main modules used for this project are:

- **NumPy / Pandas: Data loading and processing.**
- **cvxpy: Solving convex optimization problems.**
- **Matplotlib: Data visualization.**

By implementing the risk minimization algorithm, investment portfolio distribution solutions will be provided, allowing for the selection of an acceptable performance frontier with a scalable design.

Specialized Python modules were used to solve the optimization problem. NumPy and Pandas were key for data transformation and manipulation. It was decided to work with these two modules for data entry and manipulation because they have extensive documentation, support, and optimization for large amounts of data. For processing and solving, the CvxPy module was used. CvxPy is a library for solving convex optimization problems. This tool allows problems and constraints to be modeled and solved using an efficient and simple method. cvxpy was chosen because it is a specific library for solving convex problems, including quadratic problem-solving algorithms, and is a dynamic, fast tool capable of handling large amounts of data.

To be more specific, we use the CvxPy quad_form() function, which is specifically designed to detail the objective function of a quadratic problem. It expects a PSD matrix and then specifies the constraints and the problem objective, then solves it using the Problem() and solve() functions.

The CvxPy library internally determines the algorithm to use for solving problems based on the characteristics of the problem. When solving our problem, it was discovered that the library uses the OSQP solver (Operator Splitting Quadratic Program Solver), which is specifically designed for solving quadratic problems and uses a variant of the ADMM (Alternating Direction Method of Multipliers). This method is an iterative algorithm that decomposes complex problems into simpler subproblems and then iteratively adjusts the properties of the dual and primal problems. The OSQP solver is characterized by its efficiency and scalability and typically has a linear or sublinear convergence rate, depending on the characteristics of the problem and the number of iterations.

The CvxPy library was chosen after determining the convexity of the problem and finding examples of specific quadratic problem solving. Based on our own experience, the library is very easy to use, and we found that simple and intuitive functions can solve very complex problems. We also found a good amount of documentation and a large community where we can find applications for more complex problems.

Data Analysis

The program was executed using the data provided for this project (Data1.csv and Data2.csv), and the target return range was requested at the time of execution. To display the results obtained, a return range between [6.5%, 10.5%] was used, and to observe the behavior in detail, 50 steps were used to illustrate the optimization behavior.

The results obtained from running the program are: The efficient frontier graph and a table detailing the target return, minimum risk, and the proportion of investment in the different portfolio options. The efficient frontier is obtained by graphing the minimum values the portfolio can take for different return levels.

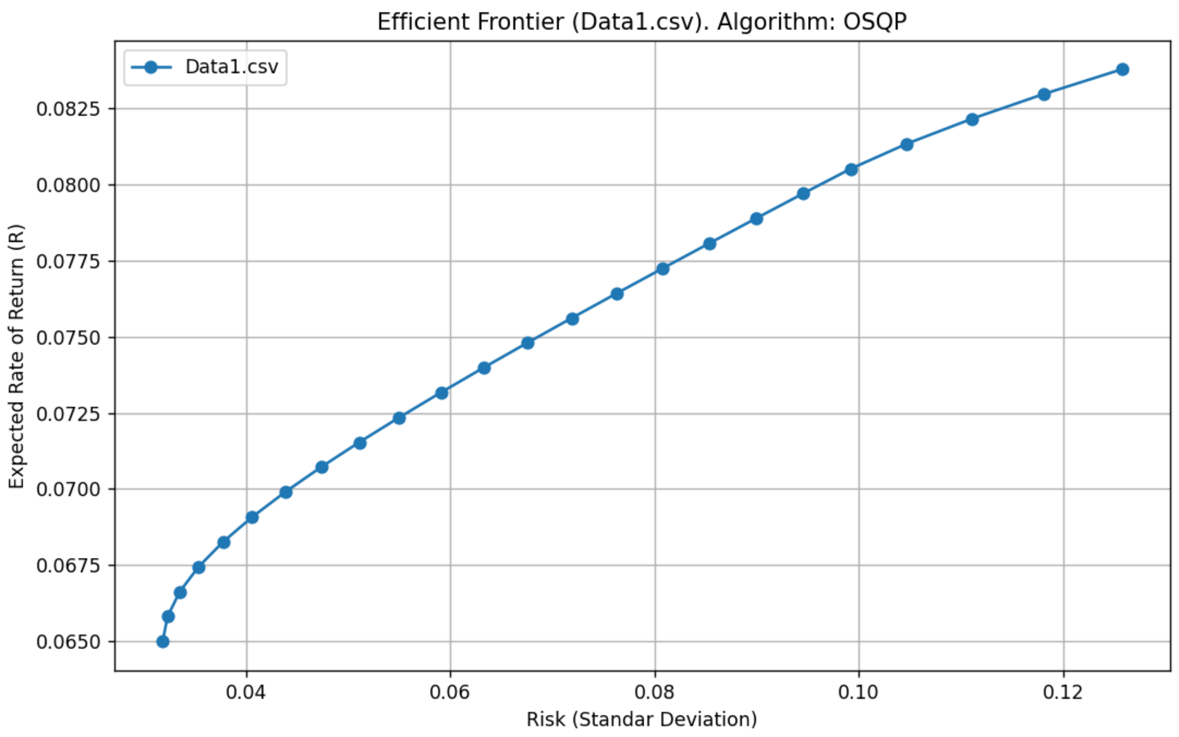
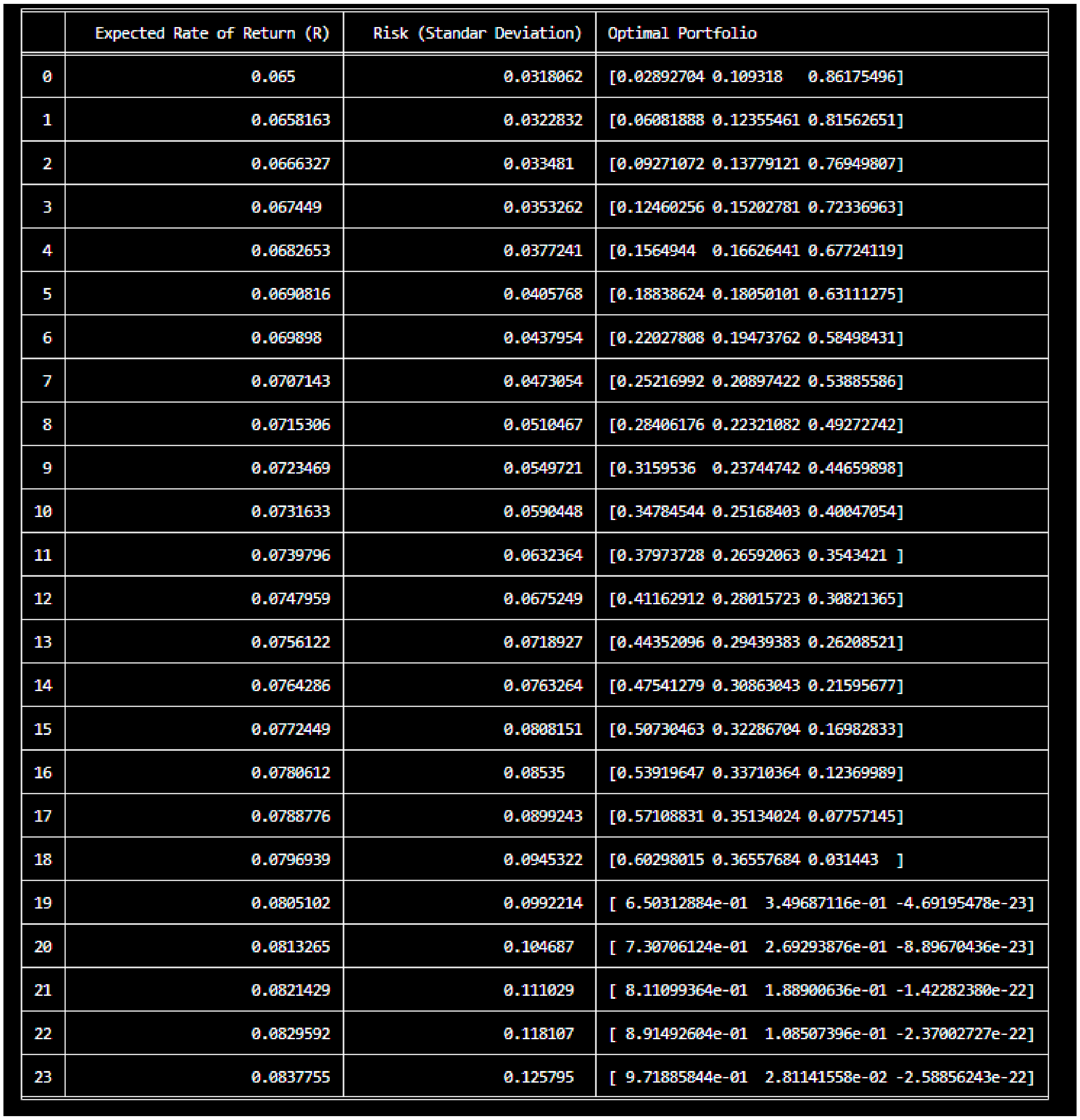


Image 1: Efficient frontier of the Data1.csv portfolio



	Expected Rate of Return (R)	Risk (Standar Deviation)	Optimal Portfolio
0	0.065	0.0318062	[0.02892704 0.109318 0.86175496]
1	0.0658163	0.0322832	[0.06081888 0.12355461 0.81562651]
2	0.0666327	0.033481	[0.09271072 0.13779121 0.76949807]
3	0.067449	0.0353262	[0.12460256 0.15202781 0.72336963]
4	0.0682653	0.0377241	[0.1564944 0.16626441 0.67724119]
5	0.0690816	0.0405768	[0.18838624 0.18050101 0.63111275]
6	0.069898	0.0437954	[0.22027808 0.19473762 0.58498431]
7	0.0707143	0.0473054	[0.25216992 0.20897422 0.53885586]
8	0.0715306	0.0510467	[0.28406176 0.22321082 0.49272742]
9	0.0723469	0.0549721	[0.3159536 0.23744742 0.44659898]
10	0.0731633	0.0590448	[0.34784544 0.25168403 0.40047054]
11	0.0739796	0.0632364	[0.37973728 0.26592063 0.3543421]
12	0.0747959	0.0675249	[0.41162912 0.28015723 0.30821365]
13	0.0756122	0.0718927	[0.44352096 0.29439383 0.26208521]
14	0.0764286	0.0763264	[0.47541279 0.30863043 0.21595677]
15	0.0772449	0.0808151	[0.50730463 0.32286704 0.16982833]
16	0.0780612	0.08535	[0.53919647 0.33710364 0.12369989]
17	0.0788776	0.0899243	[0.57108831 0.35134024 0.07757145]
18	0.0796939	0.0945322	[0.60298015 0.36557684 0.031443]
19	0.0805102	0.0992214	[6.50312884e-01 3.49687116e-01 -4.69195478e-23]
20	0.0813265	0.104687	[7.30706124e-01 2.69293876e-01 -8.89670436e-23]
21	0.0821429	0.111029	[8.11099364e-01 1.88900636e-01 -1.42282380e-22]
22	0.0829592	0.118107	[8.91492604e-01 1.08507396e-01 -2.37002727e-22]
23	0.0837755	0.125795	[9.71885844e-01 2.81141558e-02 -2.58856243e-22]

Table 1: Minimum risk results of the Data1.csv portfolio

The efficient frontier graph divides the plane into two areas. The area above the graph represents infeasible returns, that is, returns that are impossible to achieve at a given risk, while the area below the frontier determines suboptimal returns that can be achieved at different risk levels. We can clearly observe the quadratic nature of the problem and how, as risk increases, we expect higher returns. We can also see how each time we increase the investment risk, the expected return decreases.

As can be seen in both the graph and the data, the calculation stops around an 8.38% return, while the data expected to observe it up to 10.5%. This is because the problem entered an infeasible region, meaning that this specific portfolio cannot achieve a return greater than 8.38%.

Looking at the data, we can also conclude that the first option (Bonds) generates the highest returns, although it carries a higher risk, since the higher the expected return, the higher the investment percentage. The opposite is true for the third option (Money Market), since the higher the expected return, the lower the investment percentage. It's worth mentioning that the extremely small negative investment values shown in the table are floating-point approximations made by the program, so for all purposes, their value shouldbe considered 0.

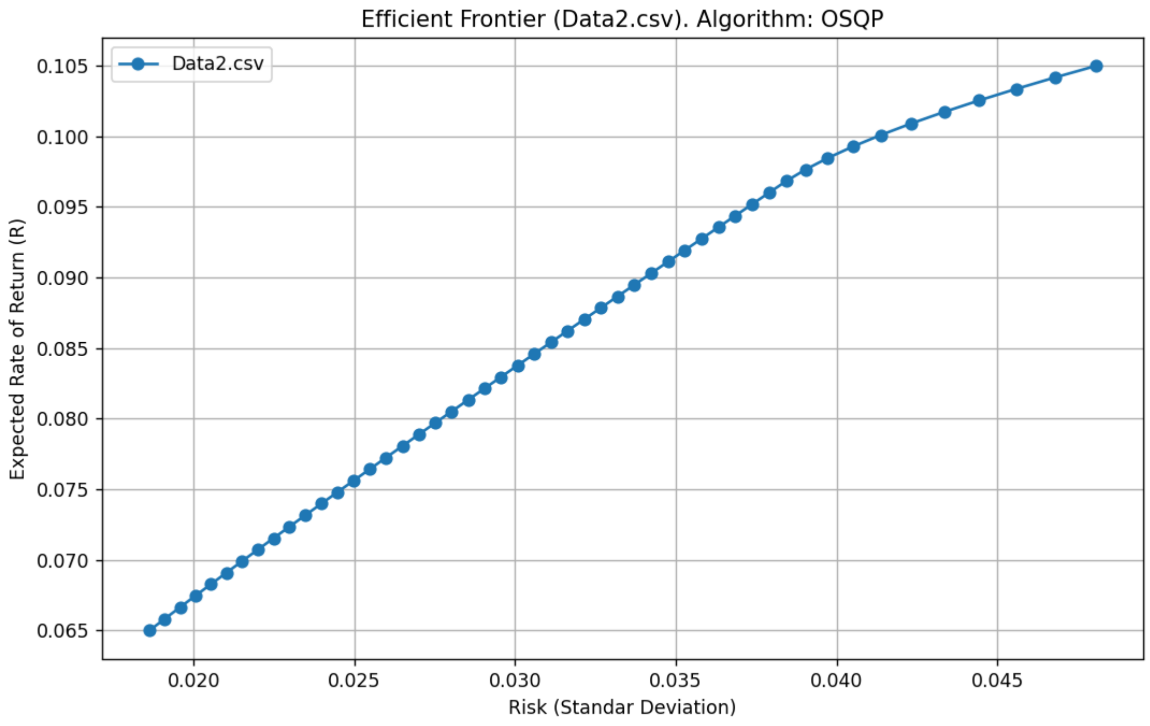
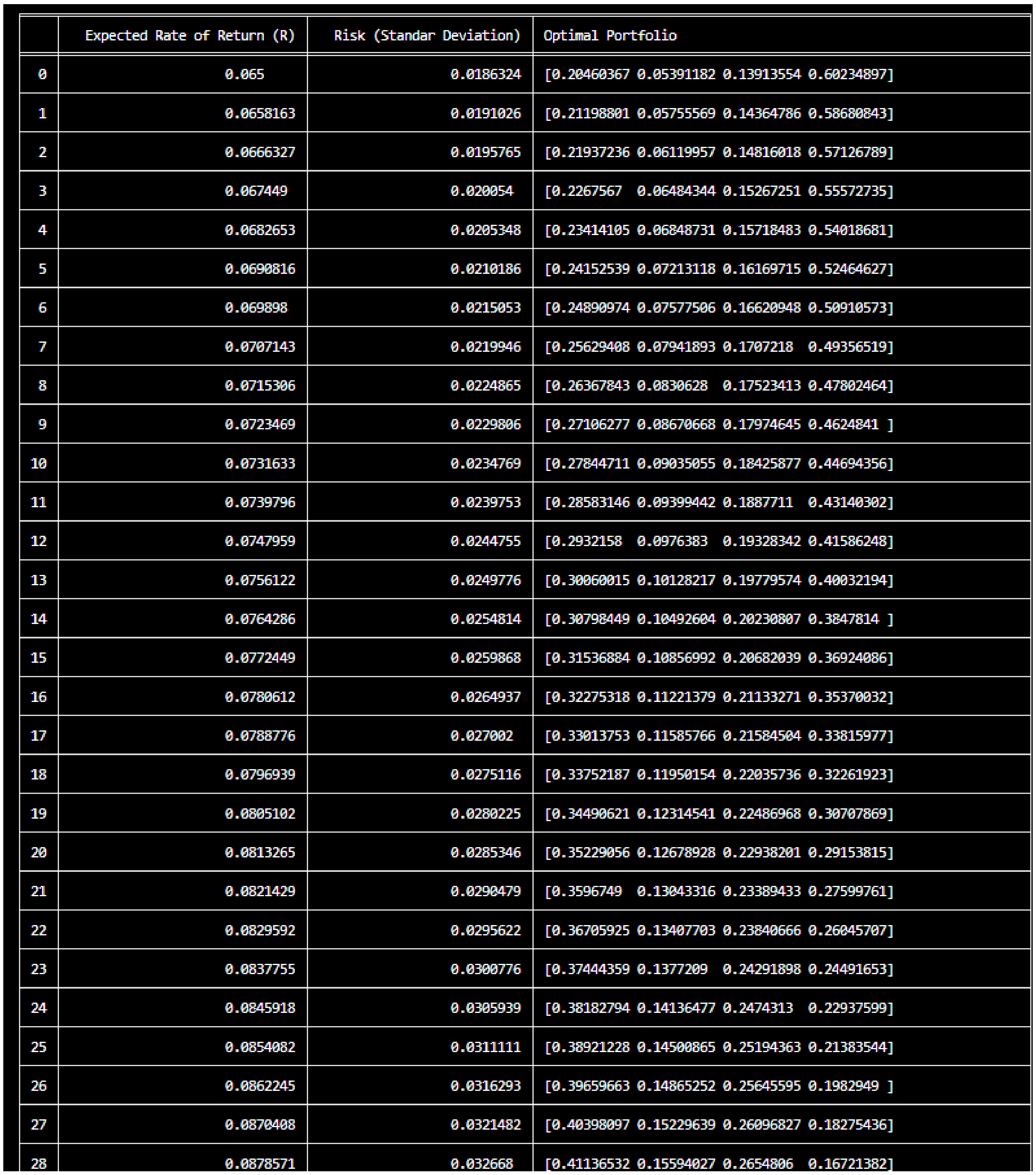
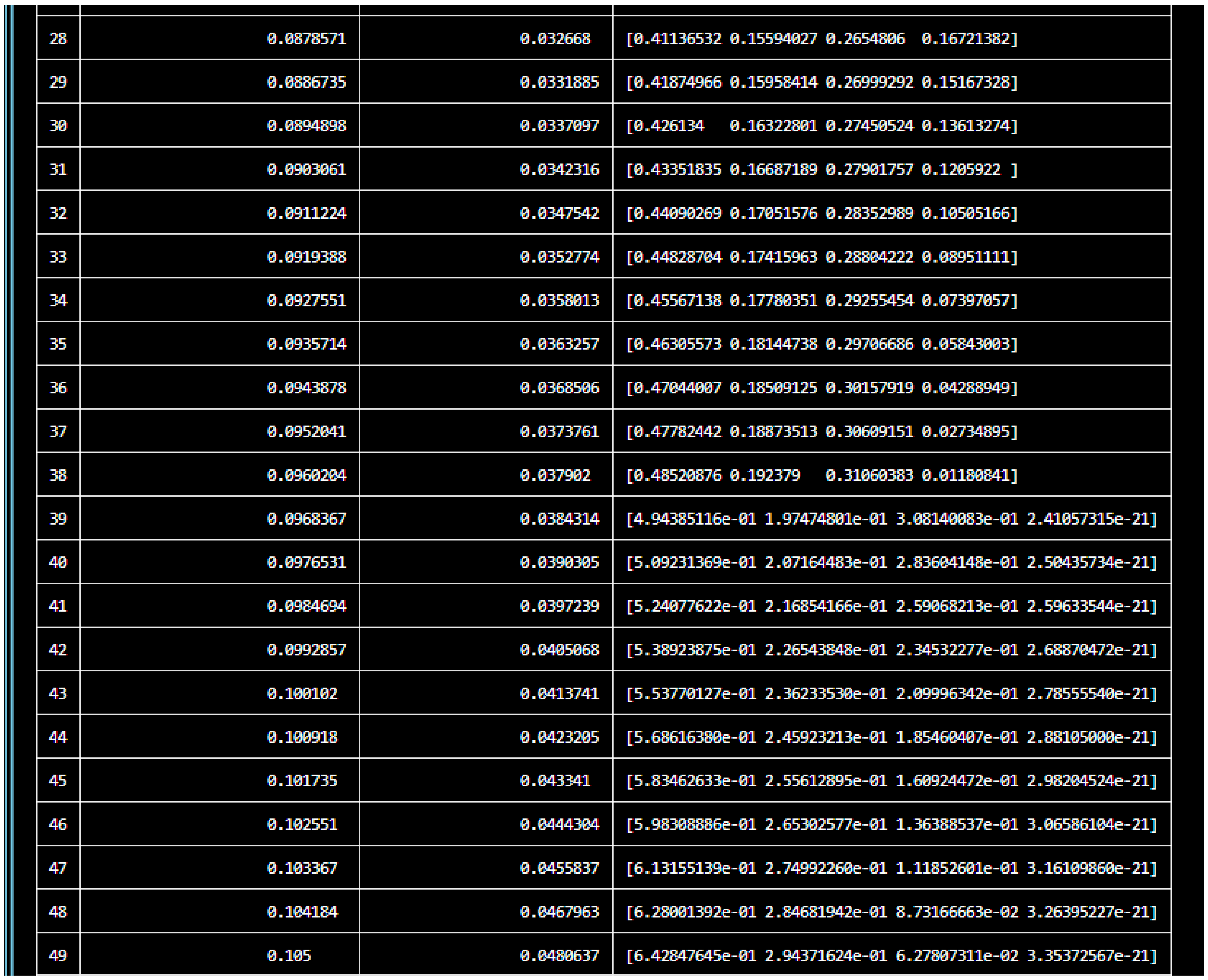


Image 2: Efficient frontier of the Data2.csv portfolio



	Expected Rate of Return (R)	Risk (Standar Deviation)	Optimal Portfolio
0	0.065	0.0186324	[0.20460367 0.05391182 0.13913554 0.60234897]
1	0.0658163	0.0191026	[0.21198801 0.05755569 0.14364786 0.58680843]
2	0.0666327	0.0195765	[0.21937236 0.06119957 0.14816018 0.57126789]
3	0.067449	0.020054	[0.2267567 0.06484344 0.15267251 0.55572735]
4	0.0682653	0.0205348	[0.23414105 0.06848731 0.15718483 0.54018681]
5	0.0690816	0.0210186	[0.24152539 0.07213118 0.16169715 0.52464627]
6	0.069898	0.0215053	[0.24890974 0.07577506 0.16620948 0.50910573]
7	0.0707143	0.0219946	[0.25629408 0.07941893 0.1707218 0.49356519]
8	0.0715306	0.0224865	[0.26367843 0.0830628 0.17523413 0.47802464]
9	0.0723469	0.0229806	[0.27106277 0.08670668 0.17974645 0.4624841]
10	0.0731633	0.0234769	[0.27844711 0.09035055 0.18425877 0.44694356]
11	0.0739796	0.0239753	[0.28583146 0.09399442 0.1887711 0.43140302]
12	0.0747959	0.0244755	[0.2932158 0.0976383 0.19328342 0.41586248]
13	0.0756122	0.0249776	[0.30060015 0.10128217 0.19779574 0.40032194]
14	0.0764286	0.0254814	[0.30798449 0.10492604 0.20230807 0.3847814]
15	0.0772449	0.0259868	[0.31536884 0.10856992 0.20682039 0.36924086]
16	0.0780612	0.0264937	[0.32275318 0.11221379 0.21133271 0.35370032]
17	0.0788776	0.027002	[0.33013753 0.11585766 0.21584504 0.33815977]
18	0.0796939	0.0275116	[0.33752187 0.11950154 0.22035736 0.32261923]
19	0.0805102	0.0280225	[0.34490621 0.12314541 0.22486968 0.30707869]
20	0.0813265	0.0285346	[0.35229056 0.12678928 0.22938201 0.29153815]
21	0.0821429	0.0290479	[0.3596749 0.13043316 0.23389433 0.27599761]
22	0.0829592	0.0295622	[0.36705925 0.13407703 0.23840666 0.26045707]
23	0.0837755	0.0300776	[0.37444359 0.1377209 0.24291898 0.24491653]
24	0.0845918	0.0305939	[0.38182794 0.14136477 0.2474313 0.22937599]
25	0.0854082	0.0311111	[0.38921228 0.14500865 0.25194363 0.21383544]
26	0.0862245	0.0316293	[0.39659663 0.14865252 0.25645595 0.1982949]
27	0.0870408	0.0321482	[0.40398097 0.15229639 0.26096827 0.18275436]
28	0.0878571	0.032668	[0.41136532 0.15594027 0.2654806 0.16721382]

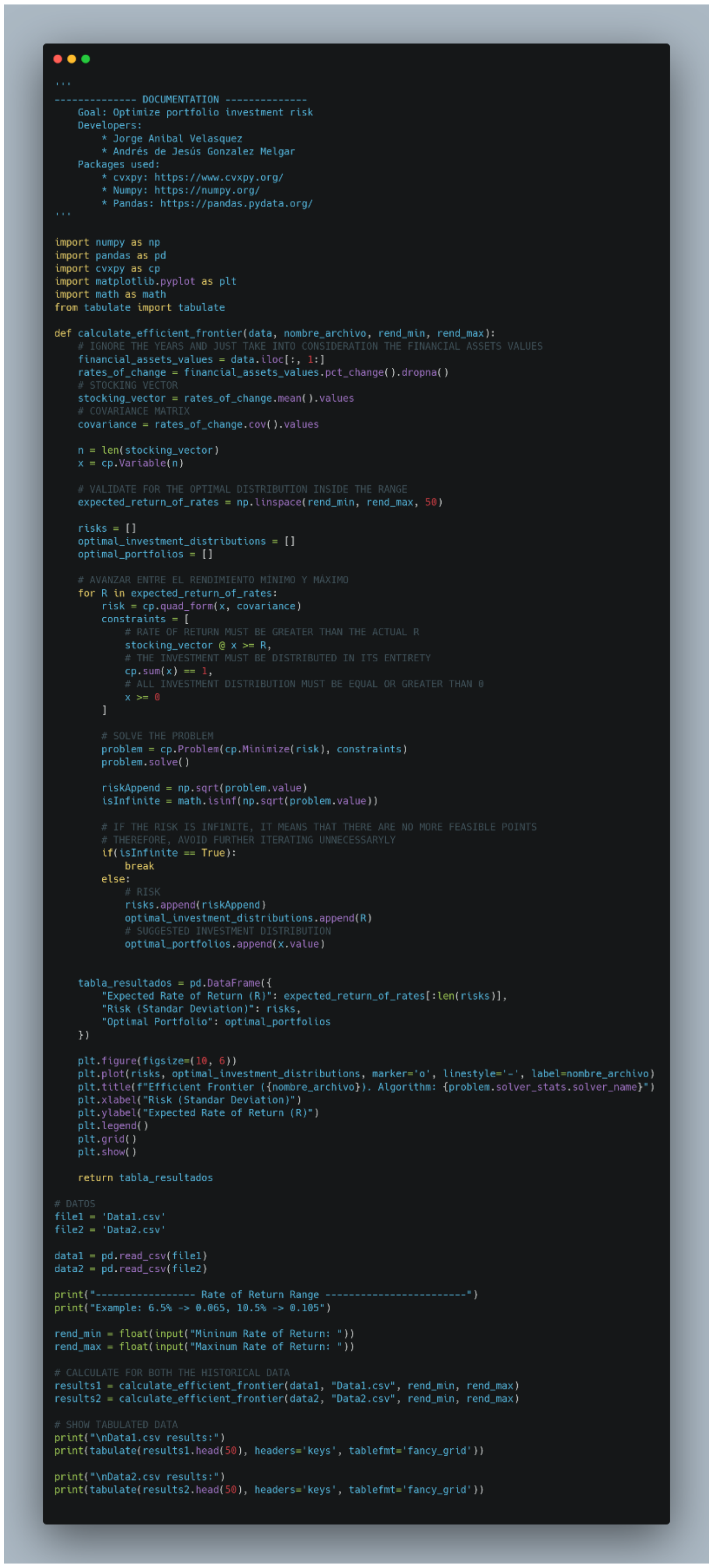


28	0.0878571	0.032668	[0.41136532 0.15594027 0.2654806 0.16721382]
29	0.0886735	0.0331885	[0.41874966 0.15958414 0.26999292 0.15167328]
30	0.0894898	0.0337097	[0.426134 0.16322801 0.27450524 0.13613274]
31	0.0903061	0.0342316	[0.43351835 0.16687189 0.27901757 0.1205922]
32	0.0911224	0.0347542	[0.44090269 0.17051576 0.28352989 0.10505166]
33	0.0919388	0.0352774	[0.44828704 0.17415963 0.28804222 0.08951111]
34	0.0927551	0.0358013	[0.45567138 0.17780351 0.29255454 0.07397057]
35	0.0935714	0.0363257	[0.46305573 0.18144738 0.29706686 0.05843003]
36	0.0943878	0.0368506	[0.47044007 0.18509125 0.30157919 0.04288949]
37	0.0952041	0.0373761	[0.47782442 0.18873513 0.30609151 0.02734895]
38	0.0960204	0.037902	[0.48520876 0.192379 0.31060383 0.01180841]
39	0.0968367	0.0384314	[4.94385116e-01 1.97474801e-01 3.08140083e-01 2.41057315e-21]
40	0.0976531	0.0390305	[5.09231369e-01 2.07164483e-01 2.83604148e-01 2.50435734e-21]
41	0.0984694	0.0397239	[5.24077622e-01 2.16854166e-01 2.59068213e-01 2.59633544e-21]
42	0.0992857	0.0405068	[5.38923875e-01 2.26543848e-01 2.34532277e-01 2.68870472e-21]
43	0.100102	0.0413741	[5.53770127e-01 2.36233530e-01 2.09996342e-01 2.78555540e-21]
44	0.100918	0.0423205	[5.68616380e-01 2.45923213e-01 1.85460407e-01 2.88105000e-21]
45	0.101735	0.043341	[5.83462633e-01 2.55612895e-01 1.60924472e-01 2.98204524e-21]
46	0.102551	0.0444304	[5.98308886e-01 2.65302577e-01 1.36388537e-01 3.06586104e-21]
47	0.103367	0.0455837	[6.13155139e-01 2.74992260e-01 1.11852601e-01 3.16109860e-21]
48	0.104184	0.0467963	[6.28001392e-01 2.84681942e-01 8.73166663e-02 3.26395227e-21]
49	0.105	0.0480637	[6.42847645e-01 2.94371624e-01 6.27807311e-02 3.35372567e-21]

Table 2: Minimum risk results of the Data2.csv portfolio

As we can see, a 10.5% return is possible in this portfolio. As in the previous case, the Money Market is the safest investment but provides the lowest return. We can also get an idea of how much they contribute to performance by observing that the first (S&P) and third options (Bonds) increase as higher returns are expected. It is also interesting to observe how the efficient frontier behaves almost linearly up to a 9.6% return, at which point investment in the Money Market is no longer recommended and the risk-return ratio is affected. This can be especially useful for making recommendations to different types of clients, showing the turning points in the slopes of returns and risks.

Appendices



```python
'''
-------------- DOCUMENTATION --------------
    Goal: Optimize portfolio investment risk
    Developers:
        * Jorge Anibal Velasquez
        * Andrés de Jesús Gonzalez Melgar
    Packages used:
        * cvxpy: https://www.cvxpy.org/
        * Numpy: https://numpy.org/
        * Pandas: https://pandas.pydata.org/
'''

import numpy as np
import pandas as pd
import cvxpy as cp
import matplotlib.pyplot as plt
import math as math
from tabulate import tabulate

def calculate_efficient_frontier(data, nombre_archivo, rend_min, rend_max):
    # IGNORE THE YEARS AND JUST TAKE INTO CONSIDERATION THE FINANCIAL ASSETS VALUES
    financial_assets_values = data.iloc[:, 1:]
    rates_of_change = financial_assets_values.pct_change().dropna()
    # STOCKING VECTOR
    stocking_vector = rates_of_change.mean().values
    # COVARIANCE MATRIX
    covariance = rates_of_change.cov().values

    n = len(stocking_vector)
    x = cp.Variable(n)

    # VALIDATE FOR THE OPTIMAL DISTRIBUTION INSIDE THE RANGE
    expected_return_of_rates = np.linspace(rend_min, rend_max, 50)

    risks = []
    optimal_investment_distributions = []
    optimal_portfolios = []

    # AVANZAR ENTRE EL RENDIMIENTO MÍNIMO Y MÁXIMO
    for R in expected_return_of_rates:
        risk = cp.quad_form(x, covariance)
        constraints = [
            # RATE OF RETURN MUST BE GREATER THAN THE ACTUAL R
            stocking_vector @ x >= R,
            # THE INVESTMENT MUST BE DISTRIBUTED IN ITS ENTIRETY
            cp.sum(x) == 1,
            # ALL INVESTMENT DISTRIBUTION MUST BE EQUAL OR GREATER THAN 0
            x >= 0
        ]

        # SOLVE THE PROBLEM
        problem = cp.Problem(cp.Minimize(risk), constraints)
        problem.solve()

        riskAppend = np.sqrt(problem.value)
        isInfinite = math.isinf(np.sqrt(problem.value))

        # IF THE RISK IS INFINITE, IT MEANS THAT THERE ARE NO MORE FEASIBLE POINTS
        # THEREFORE, AVOID FURTHER ITERATING UNNECESSARYLY
        if(isInfinite == True):
            break
        else:
            # RISK
            risks.append(riskAppend)
            optimal_investment_distributions.append(R)
            # SUGGESTED INVESTMENT DISTRIBUTION
            optimal_portfolios.append(x.value)


    tabla_resultados = pd.DataFrame({
        "Expected Rate of Return (R)": expected_return_of_rates[:len(risks)],
        "Risk (Standar Deviation)": risks,
        "Optimal Portfolio": optimal_portfolios
    })

    plt.figure(figsize=(10, 6))
    plt.plot(risks, optimal_investment_distributions, marker='o', linestyle='-', label=nombre_archivo)
    plt.title(f"Efficient Frontier ({nombre_archivo}). Algorithm: {problem.solver_stats.solver_name}")
    plt.xlabel("Risk (Standar Deviation)")
    plt.ylabel("Expected Rate of Return (R)")
    plt.legend()
    plt.grid()
    plt.show()

    return tabla_resultados

# DATOS
file1 = 'Data1.csv'
file2 = 'Data2.csv'

data1 = pd.read_csv(file1)
data2 = pd.read_csv(file2)

print("----------------- Rate of Return Range ------------------------")
print("Example: 6.5% -> 0.065, 10.5% -> 0.105")

rend_min = float(input("Mininum Rate of Return: "))
rend_max = float(input("Maxinum Rate of Return: "))

# CALCULATE FOR BOTH THE HISTORICAL DATA
results1 = calculate_efficient_frontier(data1, "Data1.csv", rend_min, rend_max)
results2 = calculate_efficient_frontier(data2, "Data2.csv", rend_min, rend_max)

# SHOW TABULATED DATA
print("\nData1.csv results:")
print(tabulate(results1.head(50), headers='keys', tablefmt='fancy_grid'))

print("\nData2.csv results:")
print(tabulate(results2.head(50), headers='keys', tablefmt='fancy_grid'))
```

References

- Pandas Development Team. (n.d.). *Pandas: Python Data Analysis Library*. Retrieved December 9, 2024, from https://pandas.pydata.org/
- CVXPY Developers. (n.d.). *CVXPY: A Python-embedded modeling language for convex optimization problems*. Retrieved December 9, 2024, from https://www.cvxpy.org/
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). *Array programming with NumPy*. *Nature*, *585*(7825), 357–362. Retrieved December 9, 2024, from https://numpy.org/
- Estrategias de Inversión. (n.d.). *Modelo de Markowitz*. Retrieved December 9, 2024, from https://www.estrategiasdeinversion.com/herramientas/diccionario/mercados/modelo-de-markowitz-t-240