# Comparing Classical and Quantum Approaches to Mean-Variance Portfolio Optimization

Advik Chaudhary

January 2023

**Abstract**

[Working paper V4]. Portfolio optimization allows investors to diversify optimally, such that there are maximum returns while having minimum possible risk. This paper aims to compare binary portfolio optimization problems implemented via a classical and quantum approach (IBM Q). The paper will first provide some background of Modern Portfolio Theory, build a quadratic program, and convert that to a quadratic unconstrained binary optimization problem (QUBO). A quantum circuit will be created and data regarding time taken and weights outputted will be compared. The classical approach will be run via `Qiskit`'s `NumPyEigensolver` and quantum approach will use a `Variational Quantum Eigensolver` (VQE). The quantum approach (run on IBM's Nairobi, Lima, Oslo, and Belem quantum computers) was found to outperform the classical approach, especially at greater portfolio sizes.

## 1 Background and Modern Portfolio Theory

A portfolio is a collection of investments—a set of stocks, for instance—each with some certain allocation of a total budget. In general, stocks that are deemed risky also have a potential to make greater profit; there exists a trade-off between risk and return.

### 1.1 Returns and Risk

Returns essentially represent the change in a portfolio or an asset's value over a certain period (leading to a gain or loss in capital). Using a yearly time frame, we can calculate the expected yearly returns for each asset with:

$$R_i = \frac{\sum_d^D \frac{P_d - P_{d-1}}{P_{d-1}}}{D} \times 252 \tag{1.1}$$

where $D$ is the total number of trading days in the chosen dataset, and $P_d$ is the price at a given date. Note: 252 is the number of trading days in a year; multiplying this by the average daily percentage change of the dataset gives the expected annual return. To calculate the returns of the entire portfolio, the expected annualized returns of each asset is computed and is simply weighted by how much of the portfolio is allocated to it:

$$R_p = \sum_{i=0}^{n} R_i w_i \tag{1.2}$$

where $n$ is the number of assets. The overall risk of the portfolio is defined as its standard deviation; however, it is not simply the weighted sum of the standard deviations of each of the assets, but also factors in the correlation between assets. The variance of one particular asset is:

$$S_a = \frac{R_p - R_f}{\rho_p} \tag{1.3}$$

And the variance for a two-asset portfolio is:

$$\sigma_p^2 = w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + 2w_1w_2\sigma_1\sigma_2\rho_{1,2}$$

where $\rho_{1,2}$ is the correlation coefficient and $\text{Cov}_{1,2}$ is the covariance of the two assets. Since $\text{Cov}_{1,2} = \sigma_1\sigma_2\rho_{1,2}$ and $\text{Cov}_{1,1} = \sigma_1^2$,

$$\sigma_p^2 = w_1^2\text{Cov}_{1,1} + w_2^2\text{Cov}_{2,2} + 2w_1w_2\text{Cov}_{1,2} \tag{1.4}$$

Hence, we see the benefits of diversification: if two assets were to be less correlated, their covariance would be lower, decreasing the standard deviation of the portfolio, since losses are likely limited to that one asset. This can be generalized to a portfolio with n assets, in matrix form, as:

$$\sigma_p^2 = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix} \cdot \begin{bmatrix} \text{Cov}_{1,1} & \text{Cov}_{1,2} & \dots & \text{Cov}_{1,n} \\ \text{Cov}_{2,1} & \text{Cov}_{2,2} & \dots & \text{Cov}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}_{n,1} & \text{Cov}_{n,2} & \dots & \text{Cov}_{n,n} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \tag{1.5}$$

The square root of this expression, the $\sigma_p$, is the risk of the portfolio. This matrix-and-dot-product

form was how the risk was calculated in code, though an iterating sum could also be used.

# 2 Sharpe Ratio

An optimum portfolio is defined as one that provides the greatest return for the least risk (assuming that the investor is risk-averse, preferring to retain capital as opposed to achieving significant returns at the risk of losing capital). The returns-to-risk ratio is represented by the Sharpe ratio:

$$S_a = \frac{R_p - R_f}{\sigma_p} \tag{2.1}$$

where $R_p$ is the expected return of the entire portfolio based on historical performance, $R_f$ is the risk-free rate, and $\sigma_p$ is the standard deviation of the portfolio's returns as a measure of the portfolio's risk—how volatile it is. The risk-free rate is the rate of return on an asset that is deemed to have no risk of losing initial capital. Typically, the 10 Year Treasury Yield is used for this value (around 3.4% at time of writing).

## 2.1 Optimizing Portfolios

When several, randomly-generated, portfolios and their respective risks and returns are plotted, an "efficient" frontier forms—a curve on which points (representing different portfolios' risks and returns) have the highest possible returns for a given risk (see Fig. 1). These portfolios are simulated by assigning random weights to each of the stocks considered (e.g 20% of budget to AAPL). Then, using (1.2), (1.5), and (2.1), the returns, volatility, and sharpe ratio of the portfolio can be calculated respectively.

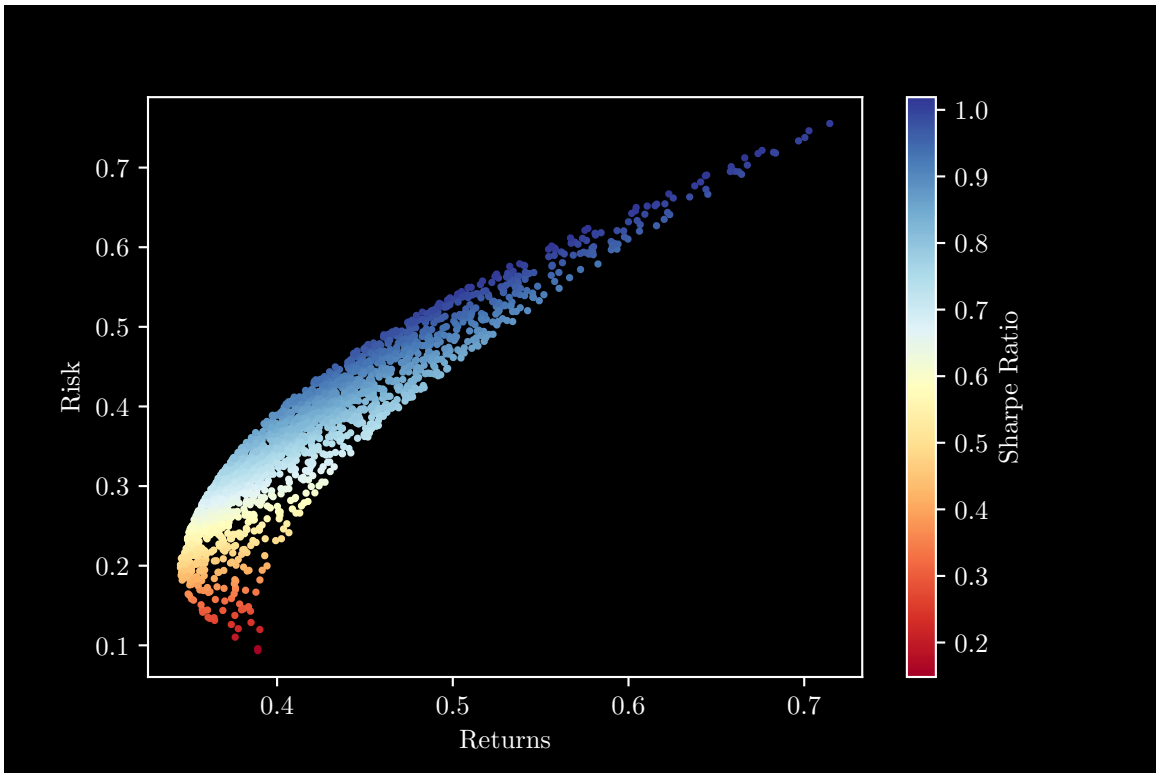


Figure 1: Returns vs standard deviations of 2,000 randomly generated portfolios using four assets with different weights. A frontier forms.

Portfolios within this edge are considered inefficient, as there exists another portfolio where for the same risk, a higher return is expected.

## 2.2 Mean-Variance Portfolio Optimization

To optimize a given portfolio, we can find a set of weights that would achieve the greatest possible returns for some given risk. One approach to this is the Minimum Volatility Portfolio, where a certain threshold for returns is set and the risk is minimized. For a simple two-asset portfolio, we can simply differentiate and find the minimum point for the risk function; differentiating with respect to $w_1$ ($w_2$ can be written as $1 - w_1$) and setting (1.4) to 0 gives:

$$w_1 = \frac{\sigma_2^2 - \text{Cov}_{1,2}}{\sigma_2^2 + \sigma_1^2 - 2\text{Cov}_{1,2}}$$

However, for a portfolio with three or more assets, this method will not work, since $w_2$, $w_3$ and other weights cannot be written in terms of $w_1$. Instead, the Lagrangian method can be used to optimize the weights, solved for using linear algebra.

$$\min_{w_1,w_2,\ldots,w_n} f(w_1,w_2,\ldots,w_n) = \sigma_p^2 = \begin{bmatrix} w_1 & w_2 & \ldots & w_n \end{bmatrix} \cdot \begin{bmatrix} \text{Cov}_{1,1} & \text{Cov}_{1,2} & \ldots & \text{Cov}_{1,n} \\ \text{Cov}_{2,1} & \text{Cov}_{2,2} & \ldots & \text{Cov}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}_{n,1} & \text{Cov}_{n,2} & \ldots & \text{Cov}_{n,n} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$$\text{subject to: } g(w_1,w_2,\ldots,w_n) = \sum_i^n w_i = 1$$

The Lagrangian of this would be:

$$L(w_1,w_2,\ldots w_n,\lambda) = f(w_1,w_2,\ldots,w_n) + \lambda[g(w_1,w_2,\ldots,w_n) - 1] \tag{2.2}$$

Giving the following system:

$$\begin{aligned}
\frac{\partial L(w_1,w_2,\ldots,w_n)}{\partial w_1} &= \frac{\partial f(w_1,w_2,\ldots,w_n)}{\partial w_1} - \lambda\frac{\partial g(w_1,w_2,\ldots,w_n)}{\partial w_1} = 0 \\
\frac{\partial L(w_1,w_2,\ldots,w_n)}{\partial w_2} &= \frac{\partial f(w_1,w_2,\ldots,w_n)}{\partial w_2} - \lambda\frac{\partial g(w_1,w_2,\ldots,w_n)}{\partial w_2} = 0 \\
&\vdots \\
\frac{\partial L(w_1,w_2,\ldots,w_n)}{\partial w_n} &= \frac{\partial f(w_1,w_2,\ldots,w_n)}{\partial w_n} - \lambda\frac{\partial g(w_1,w_2,\ldots,w_n)}{\partial w_n} = 0 \\
\frac{\partial L(w_1,w_2,\ldots,w_n)}{\partial \lambda} &= g(w_1,w_2,\ldots,w_n) - 1 = 0
\end{aligned}$$

Using this equation, we can directly find an optimized portfolio (found to have a risk of about 0.34 and returns of 0.17). Plotting this against the randomly generated portfolios, we generate the following:

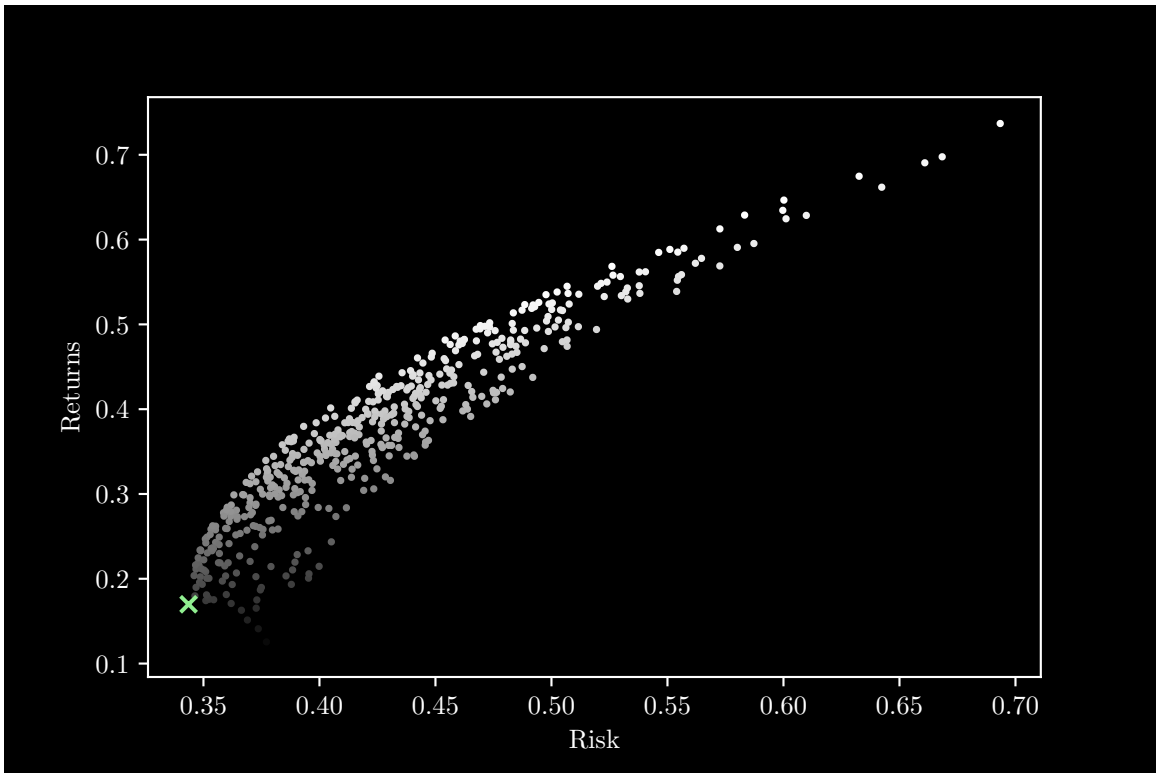


Figure 2: (Green) a portfolio with the minimum risk possible which generates returns, found through solving the Lagrangian for its minimum value

The process of computing partial derivatives and solving the system of equations can be automated with Python's `sympy` module. This approach can be applied to instead maximizing the returns for a given risk (MRP) or maximizing the sharpe ratio (MSRP). For instance, the Lagrangian can simply be modified to maximize the portfolio returns for a given risk instead of minimizing the overall risk:

$$L(w_1, w_2, \dots w_n, \lambda) = \sum_{i=0}^{n} R_i w_i + \lambda_1 \left[ W^T \cdot \sum_{i=0}^{n} \sum_{j=0}^{n} \text{Cov}_{i,j} \cdot W - \sigma_{\min} \right] + \lambda_2 \left[ \sum_{i=1}^{n} w_i - 1 \right] \tag{2.3}$$

where $W^T \cdot \sum_{i=0}^{n} \sum_{j=0}^{n} \text{Cov}_{i,j} \cdot W$ represents the variance of the portfolio, as found in (1.5) and the last two terms are the returns and budget constraints respectively.

The problem with this approach arises when a portfolio with a large number of assets are considered. The algorithm scales exponentially in time with the number of stocks considered for the portfolio.

The $n \times n$ covariance matrix, for instance, would increase by $2n+2$ terms (and $2n+3$ multiplications) for each additional asset and the differential equations would increase by one—more than doubling for each asset added. For such optimization problems, the quantum approach instead may be more productive.

# 3 Portfolio Optimization: Quantum Model

Since the quantum computer can output binary results for the optimization, the problem in (2.3) must be converted into a binary program, where the weights are now discrete ('1' for including the asset and '0' to not). This quadratic constrained binary optimization problem (QCBO) can be converted into an unconstrained binary optimization (QUBO) using Qiskit's `QuadraticProgramToQubo` (linear constraints will be converted to a penalty term, $(\sum_{i=1}^{n} w_i - B)^2$) as the problem can be easily translated to an Ising Hamiltonian (which can be executed on the quantum computer). Using the `MinimumEigenOptimizer` package, the QUBO can be solved with `VQE`, `QAOA`, or the classical equivalent, `NumpyMinimumEigensolver`. These are the methods that will be compared, using this same QUBO.

Since the constraints must be linear, the weight condition can be kept but the maximum return and minimum risk condition can be combined by minimizing the $risk - return$ value:

$$\min_{w_i \in \{0,1\}} = W^T \cdot \sum_{i=0}^{n} \sum_{j=0}^{n} \text{Cov}_{i,j} \cdot W - \sum_{i=0}^{n} R_i w_i + \left[ \sum_{i=1}^{n} w_i - B \right]^2 \tag{3.1}$$

Note that the constraint is now that the sum of the binary variables must sum to the number of desired stocks, $B$, to be chosen. This number is set to be $\lfloor n/2 \rfloor$—approximately half of the number of stocks considered.

The QUBO was formulated by first creating a constrained quadratic optimization problem using (3.1), computed using `Numpy` and `DOcplex`, then converted using `QuadraticProgramToQubo()`. The quadratic program before conversion was:

```
Problem name: 3 Portfolio Model

Minimize
    0.0005359374104806353*w_1^2 + 0.0007509577195231786*w_1*w_2
    + 0.0011029845003310 82*w_1*w_3 + 0.0006067495270254874*w_2^2
    + 0.0010501081688471282*w_2*w_3 + 0.0020738675716754237*w_3^2
    - 0.2821346671323079*w_1 - 0.08409120802875666*w_2 - 0.7899673827377993*w_3
```

```
Subject to
Linear constraints (1)
    w_1 + w_2 + w_3 == 1  'c0'

Binary variables (3)
    w_1 w_2 w_3
```

This was converted to an Ising Hamiltonian, and the eigenvalue was minimized using the Variational Quantum Eigensolver algorithm. For the quantum model, the quantum circuit was created and run for each QUBO (note that the maximum number of assets to choose from was limited by the qubits available, 7). The circuit was built with the RY and CNOT gates with the `TwoLocal` circuit (which allowed for qubit entanglement):

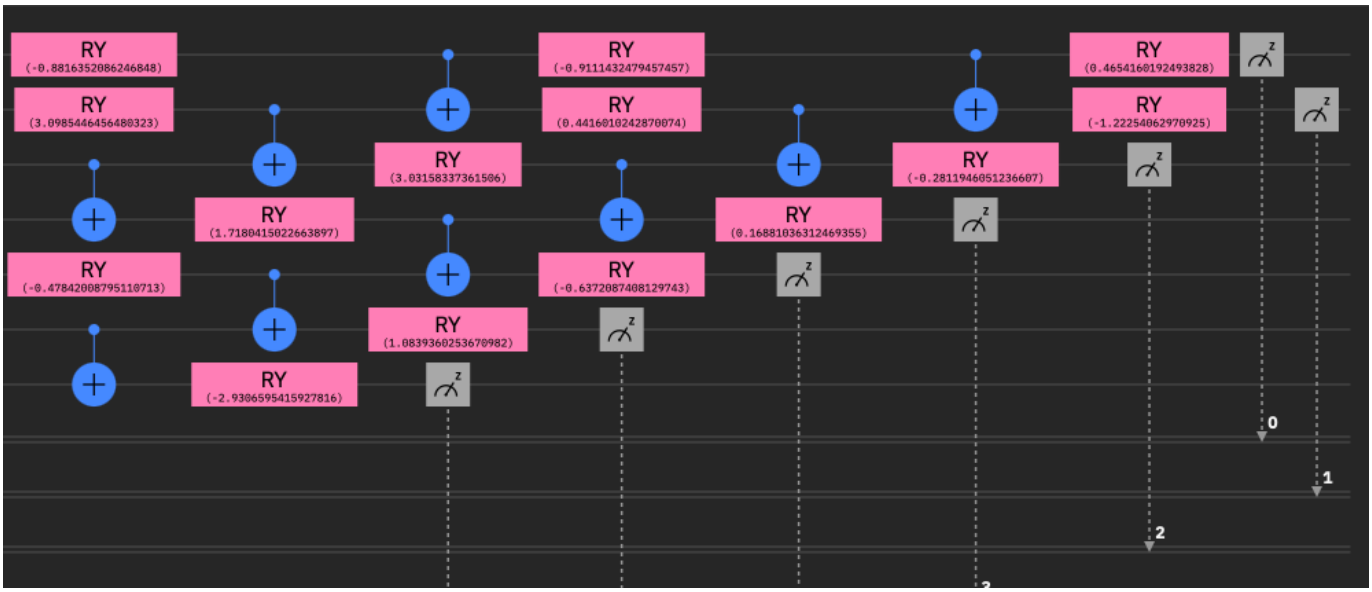


Figure 3: Quantum circuit (cropped) made for a 5-portfolio QUBO.

# 4 Data and Setup

## 4.1 Data and Portfolio Setup

For a portfolio of assets to be chosen out of $n$ stocks, the top $n$ stocks by purchase volume in Jan 2023 (for instance, 1st TSLA, 2nd AMZN...) were queried for. The different portfolio combinations are:

| Num Assets | To choose | Tickers to Choose From |
|---|---|---|
| 2 | 1 | TSLA; AMZN |
| 3 | 1 | TSLA; AMZN; AAPL |
| 4 | 2 | TSLA; AMZN; AAPL; BAC |
| 5 | 2 | TSLA; AMZN; AAPL; BAC; NVDA |
| 6 | 3 | TSLA; AMZN; AAPL; BAC; NVDA; MSFT |
| 7 | 3 | TSLA; AMZN; AAPL; BAC; NVDA; MSFT; GOOGL |

Table 1: The number of assets to select from, number of assets to be chosen, and the tickers of the assets to select from.

Note that both the number of assets to be selected and the number of total assets to choose from are being varied, as this would simulate a more realistic scenario, where an investor with a large budget would consider more stocks. Thus, the ratio of assets to select v.s. to choose from is constant and the number of assets to select is varied.

Using the `webreader` package, with `yfinance` as the source, the historical closing prices for the $n$ assets were obtained, from 1/1/2020 to 2023 (YTD). For the following, data for three stocks are shown, computing other results using the methodology in (1.2) and (1.5):

| Date | AAPL Closing Pr | AMZN Closing Pr | TSLA Closing Pr |
|---|---|---|---|
| 02-01-2020 | 73.561523 | 94.900497 | 28.684000 |
| 03-01-2020 | 72.846367 | 93.748497 | 29.534000 |
| 06-01-2020 | 73.426834 | 95.143997 | 30.102667 |
| 20-01-2023 | 137.869995 | 97.250000 | 133.419998 |
| 23-01-2023 | 141.110001 | 97.519997 | 143.750000 |

Table 2: Daily closing prices from `pandas datareader`, sourced from Yahoo! Finance.

## 4.2 Data Processing

From the closing prices, the various parameters required in (3.1) were computed (using the `pandas` module). The daily percentage change, for instance, was:

| Date | AAPL Pct Change | AMZN Pct Change | TSLA Pct Change |
|---|---|---|---|
| 02-01-2020 | – | – | – |
| 03-01-2020 | -0.972209 | -1.213903 | 2.963326 |
| 06-01-2020 | 0.796838 | 1.488557 | 1.925464 |
| 07-01-2020 | -0.470325 | 0.209162 | 3.880052 |

Table 3: Daily percentage change computed for the $n$ stocks for use in computing expected returns

By taking the mean of the daily percentage changes and multiplying it by the trading days, the expected yearly return can be found:

| AAPL Returns Yr | AMZN Returns Yr | TSLA Returns Yr |
|---|---|---|
| 28.4040% | 8.1278% | 78.9765% |

Table 4: The expected returns for each asset are computed.

For calculating risk, a covariance matrix is created for the $n$ stocks was found:

| | AAPL | AMZN | TSLA |
|---|---|---|---|
| AAPL | 931.892848 | 336.722714 | 2681.403412 |
| AMZN | 336.722714 | 809.700784 | 1400.422464 |
| TSLA | 2681.403412 | 1400.422464 | 9141.024566 |

Table 5: Sample covariance matrix for three stocks.

# 5 Results and Discussion

## 5.1 Truncated Raw Data

The results from the quantum computers are as follows.

| Num Assets | Weights | Risk - Returns | Probability | Time (s) |
|---|---|---|---|---|
| 2 | [1 1] | 1.007471 | 0.812250 | 2.9 |
| 3 | [0 1 0] | -0.080671 | 0.407500 | 2.9 |
| 4 | [0 1 0 1] | -0.867308 | 0.282750 | 3.4 |
| 5 | [0 1 0 1 0] | -0.618493 | 0.372250 | 3.6 |
| 6 | [0 1 1 1 1 0] | 2.100860 | 0.104000 | 6.2 |

Table 6: Results of simulating portfolios with 2–7 asset choices, run IBMQ Lima

Here is the raw data from a two asset selection portfolio run, for instance, showing the probabilities (the highest of which is the "chosen" weightage):

| Weights | Risk - Returns | Probability |
|---|---|---|
| [0 1] | -0.822658 | 1.000000 |
| [1 0] | -0.090228 | 0.000000 |
| [1 1] | 1.007494 | 0.000000 |
| [0 0] | 1.919325 | 0.000000 |

Table 7: Two asset portfolio run on IBMQ Belem.

For comparison, the classical results for this are:

| Weights | Risk - Returns | Probability |
|---|---|---|
| [0 1] | -0.822658 | 1.000000 |
| [1 1] | 1.007494 | 0.000000 |
| [1 0] | -0.090228 | 0.000000 |
| [0 0] | 1.919325 | 0.000000 |

Table 8: Two asset portfolio run on IBMQ Belem.

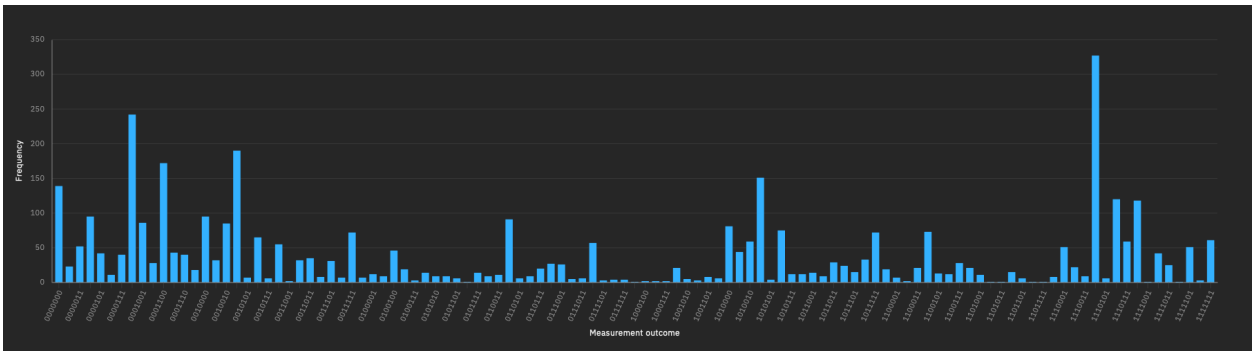


Figure 4: Histogram of weights and respective probabilities for a 7-portfolio QUBO, run on the IBM_Nairobi computer.

The "weights" are the binary decision variables representing whether or not to include the stocks for the $n$ sized portfolio in Table 1.

| Num Assets | Weights | Risk - Returns | Probability | Time (s) |
|---|---|---|---|---|
| 2 | [0 1] | -0.787689 | 1.000000 | 0.401655 |
| 3 | [0 0 1] | -0.787689 | 1.000000 | 1.202925 |
| 4 | [1 0 0 1] | -1.070087 | 1.000000 | 4.340321 |
| 5 | [1 0 0 0 1] | -1.070087 | 1.000000 | 4.698022 |
| 6 | [1 0 0 0 1 1] | -1.606107 | 1.000000 | 11.129465 |
| 7 | [1 0 0 0 0 1 1] | -1.606108 | 0.999999 | 15.058850 |

Table 9: Results from classical solution using `NumPyEigensolver`

## 5.2 Analysis and Further Discussion

As expected, the time taken to optimize the portfolios through the classical approach seemed to grow exponentially with the size of the portfolio.

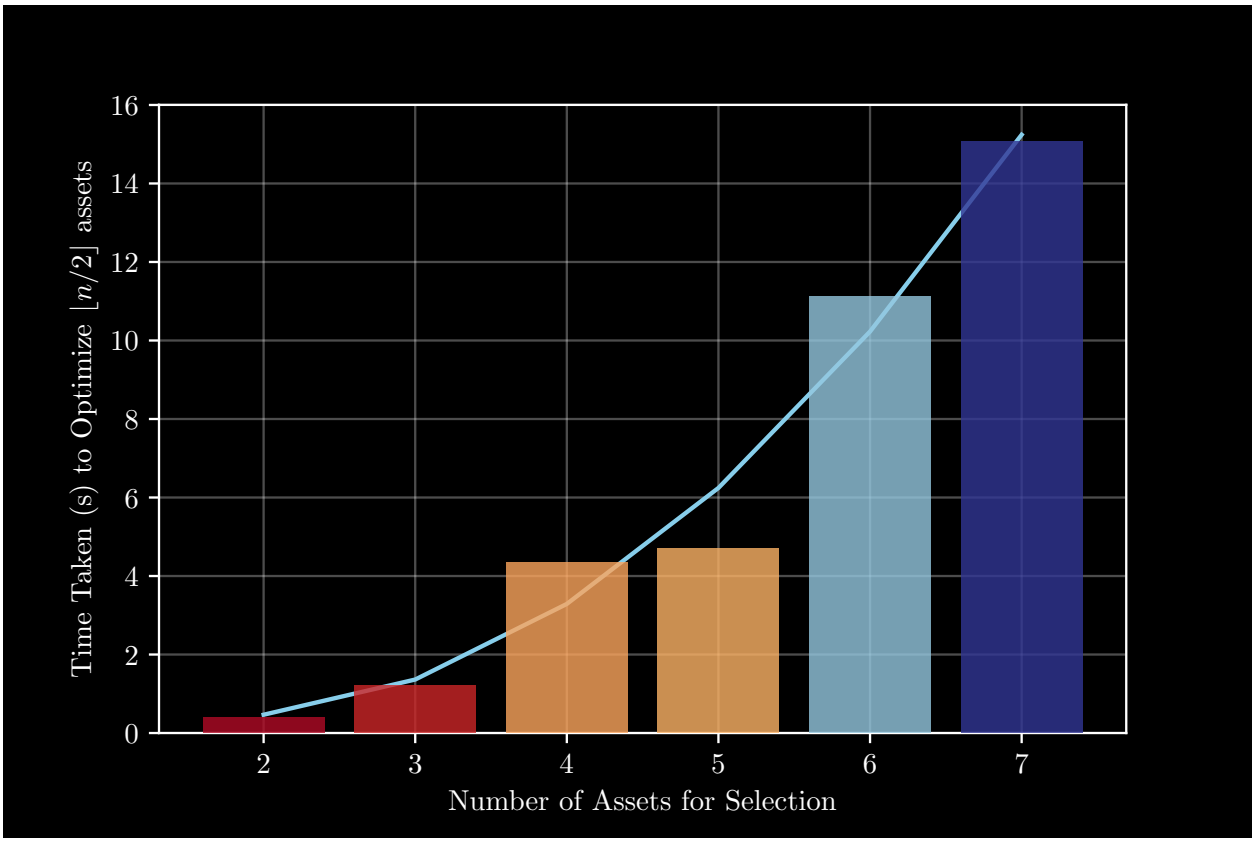


Figure 5: Time v.s. asset size for classical approach.

It seems that the time taken to solve the optimization problem in the quantum computer was relatively constant, or at least increasing more gradually than the classical version. Note that for the last portfolio run (with six assets), a 7-qubit quantum computer was used (IBM Oslo) since the previous 5-qubit computers cannot handle more than 5 selections. Based on previous tests, this computer has a lower performance and hence has not been included in the line fit.

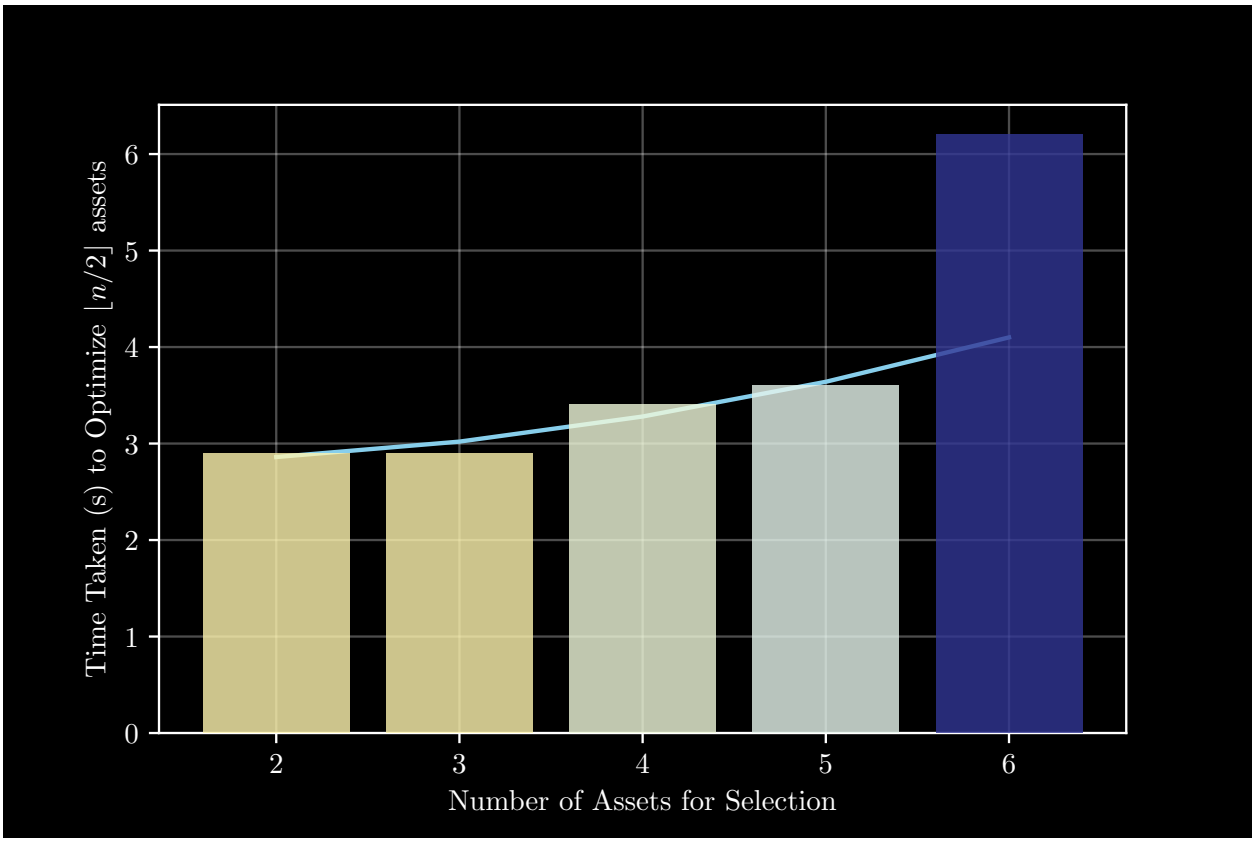


Figure 6: The time v.s. number of assets for the quantum approach. The six asset portfolio was not included for the best fit calculation.

However, some errors are evident in the quantum results—not finding the true minimum $risk - return$ value, as compared to the classical results. This is likely due to the limitations of the hardware—the error of measurement, which becomes especially significant for larger circuits that optimize larger portfolios. Note that the circuit was sent to different IBM quantum computers, depending on availability, and this variability could also affect results.

Further explorations can investigate portfolios with even larger assets, on the order of $10^2$ as it is in these large values that quantum algorithms outperform the classical, exponentially growing algorithms. As for reducing the error, the depth (number of layers) of the circuit could be reduced. Further, the gates in the circuit could be optimized such that fewer are required, reducing the error during measurement. The algorithm to find the minimum eigenvalue can also be changed, employing the Quantum Approximate Optimization Algorithm (QAOA) instead of the VQE. Additionally, other computing platforms, such as D-Wave could also be explored, with potentially different error minimizing techniques.

# 6 Bibliography


1. Cohen, J., Khan, A., & Alexander, C. (2020). Portfolio optimization of 40 stocks using the dwave quantum annealer. arXiv preprint arXiv:2007.01430.

2. Corporate Finance Institute. (2022, October 24). Modern portfolio theory (MPT). https://corporatefinancein management/modern-portfolio-theory-mpt/

3. Elsokkary, N., Khan, F. S., La Torre, D., Humble, T. S., & Gottlieb, J. (2017). Financial portfolio management using d-wave quantum optimizer: The case of abu dhabi securities exchange. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).

4. Glover, F., Kochenberger, G., & Du, Y. (2018). A tutorial on formulating and using QUBO models. arXiv preprint arXiv:1811.11538.

5. IBM cloud pak for data. (n.d.). IBM Cloud Pak for Data. https://dataplatform.cloud.ibm.com/exchange/public/entry/view/e9c830e3e54b41a6d42efbeb37050fa9

6. Keizer, J. (2021, July 14). Mean-variance portfolio theory. CFA, FRM, and Actuarial Exams Study Notes. https://analystprep.com/study-notes/actuarial-exams/soa/ifm-investment-and-financial-markets/mean-variance-portfolio-theory/

7. Khumalo, M. T., Chieza, H. A., Prag, K., & Woolway, M. (2022). An investigation of IBM quantum computing device performance on combinatorial optimisation problems. Neural Computing and Applications. https://doi.org/10.1007/s00521-022-07438-4

8. Offiong, A. I., Riman, H. B., & Eyo, E. E. (2016). Determining optimal portfolio in a three-asset portfolio mix in Nigeria. Journal of Mathematical Finance, 6(4), 524-540.

9. Owhadi-Kareshk, M., & Boulanger, P. (2021). Portfolio Optimization on Classical and Quantum Computers Using PortFawn. arXiv preprint arXiv:2112.08998.

10. Palmer, S., Sahin, S., Hernandez, R., Mugel, S., & Orus, R. (2021). Quantum portfolio optimization with investment bands and target volatility. arXiv preprint arXiv:2106.06735.

11. Pool, P. (2021, September 26). Cplex Python: Installation, API, and examples. Python Pool. https://www.pythonpool.com/cplex-python/

12. Qiskit Textbook. (n.d.). qiskit.org. https://qiskit.org/learn/

13. Risk averse: What it means, investment choices and strategies. (2003, November 25). Investopedia. https://www.investopedia.com/terms/r/riskaverse.asp

14. Sharpe ratio formula and definition with examples. (2003, November 26). Investopedia. https://www.investopedia.com/terms/s/sharperatio.asp

15. Tomkins, S., & De Sousa, R. (2020). Noise mitigation with delay pulses in the IBM quantum experience. 2020 IEEE International Conference on Quantum Computing and Engineering (QCE). https://doi.org/10.1109/qce49297.2020.00058

# 7 Appendix

## 7.1 Raw Data from IBM Q Runs

| Weights | Risk - Returns | Probability |
|---|---|---|
| [0 1] | -0.822658 | 1.000000 |
| [1 0] | -0.090228 | 0.000000 |
| [1 1] | 1.007494 | 0.000000 |
| [0 0] | 1.919325 | 0.000000 |

| Weights | Risk - Returns | Probability |
|---|---|---|
| [0 0 1] | -0.822658 | 1.000000 |
| [0 0 0] | 2.208318 | 0.000000 |
| [0 1 0] | -0.090228 | 0.000000 |
| [1 0 1] | 1.100700 | 0.000000 |
| [1 1 1] | 7.637232 | 0.000000 |
| [1 0 0] | -0.286066 | 0.000000 |
| [0 1 1] | 1.296487 | 0.000000 |
| [1 1 0] | 1.832775 | 0.000000 |

| Weights | Risk - Returns | Probability |
|---|---|---|
| [1 1 0 1] | 1.118248 | 0.196750 |
| [1 0 1 0] | -0.388486 | 0.183750 |
| [0 1 0 1] | -0.911832 | 0.172750 |
| [0 0 1 0] | 2.211300 | 0.152250 |
| [1 0 0 0] | 2.028223 | 0.046250 |
| [0 0 0 0] | 9.257158 | 0.038000 |
| [1 0 1 1] | 1.104915 | 0.036000 |
| [1 0 0 1] | -1.107619 | 0.028750 |
| [0 1 0 0] | 2.224061 | 0.028750 |
| [0 0 1 1] | -0.924984 | 0.022750 |
| [1 1 0 0] | -0.375543 | 0.022750 |
| [0 0 0 1] | 1.491631 | 0.020750 |
| [1 1 1 0] | 1.836732 | 0.018250 |
| [0 1 1 0] | -0.192812 | 0.015000 |
| [0 1 1 1] | 1.300538 | 0.009250 |
| [1 1 1 1] | 7.959766 | 0.008000 |

| Weights | Risk - Returns | Probability |
|---|---|---|
| [1 1 0 1 0] | 1.948359 | 0.128000 |
| [1 0 0 1 0] | -0.831547 | 0.073250 |
| [1 1 0 0 0] | -0.375543 | 0.065250 |
| [1 0 1 1 0] | 1.935005 | 0.063250 |
| [0 0 0 1 0] | 2.321597 | 0.049500 |
| [1 0 0 1 1] | 1.216923 | 0.043250 |
| [1 0 0 0 0] | 2.582175 | 0.040250 |
| [0 0 1 1 0] | -0.648902 | 0.039500 |
| [0 1 0 1 0] | -0.635729 | 0.036000 |
| [1 1 0 1 1] | 9.734366 | 0.035750 |
| [1 0 0 0 1] | -1.107619 | 0.034500 |
| [1 0 1 0 0] | -0.388486 | 0.033000 |
| [1 0 1 0 1] | 1.658866 | 0.030500 |
| [0 0 0 0 0] | 11.472964 | 0.029500 |
| [1 1 1 0 0] | 2.390684 | 0.026000 |
| [0 0 1 0 0] | 2.765251 | 0.025250 |
| [1 0 1 1 1] | 9.720621 | 0.022500 |
| [0 1 0 0 0] | 2.778013 | 0.021500 |
| [1 1 1 1 0] | 10.451799 | 0.019500 |
| [0 1 1 0 0] | -0.192812 | 0.017250 |
| [1 1 1 0 1] | 10.175572 | 0.017250 |
| [1 1 1 1 1] | 23.974952 | 0.016750 |
| [1 1 0 0 1] | 1.672199 | 0.015750 |
| [0 0 1 0 1] | -0.924984 | 0.015500 |
| [0 1 1 0 1] | 1.854489 | 0.014750 |
| [0 0 0 1 1] | -1.367519 | 0.014750 |
| [0 1 1 1 1] | 9.916225 | 0.014250 |
| [0 1 0 1 1] | 1.412691 | 0.013500 |
| [0 0 1 1 1] | 1.399127 | 0.012750 |
| [0 0 0 0 1] | 2.045583 | 0.012000 |
| [0 1 1 1 0] | 2.130659 | 0.011750 |
| [0 1 0 0 1] | -0.911832 | 0.007500 |

Note: runs 6 and 7 are not shown as there were simply too many combinations.

## 7.2 Loop for Optimization on Quantum Computer

```
    #QUBO Looped
from qiskit.utils import algorithm_globals, QuantumInstance
from qiskit.algorithms import QAOA, NumPyMinimumEigensolver, VQE
from qiskit_optimization.algorithms import (
    MinimumEigenOptimizer,
    RecursiveMinimumEigenOptimizer,
```

```
    SolutionSample,
    OptimizationResultStatus,
)
from qiskit_optimization import QuadraticProgram
from qiskit.visualization import plot_histogram
from typing import List, Tuple
import numpy as np

out_max = pd.DataFrame(columns=['Num Assets','Weights', 'Risk - Returns
    ','Probability', 'Time'])
c_0 = 0

def index_to_selection(i, num_assets):
    s = "{0:b}".format(i).rjust(num_assets)
    x = np.array([1 if s[i] == "1" else 0 for i in reversed(range(
        num_assets))])
    return x

    from qiskit_optimization.translators import from_docplex_mp
    from qiskit_optimization.converters import QuadraticProgramToQubo
    from docplex.mp.model import Model

for num_tick in range(2, 7 + 1):
    n=num_tick

# qubo = QuadraticProgram()
# qubo.binary_var("x")
# qubo.binary_var("y")
# qubo.binary_var("z")
# qubo.minimize(linear=[1, -2, 3], quadratic={("x", "y"): 1, ("x", "z")
    : -1, ("y", "z"): 2})
# print(qubo.prettyprint())

    ### GENERATE DATA
    tickers_csv = pd.read_csv('../yfinance_quotes.csv')['Symbol'][:n].
        to_list() # 10 largest by volume, 18 Jan
    out = pd.DataFrame(columns=['Weights', 'Risk - Returns','
        Probability'])
    #['Symbol'][:10].to_list() # 10 largest by market cap
```

```
#data=web.DataReader('GOOG',"yahoo",start=dt.datetime(2018, 1, 1),
    end = dt.datetime.today())
data = yf.Tickers(tickers=tickers_csv).history(start='2020-01-01')
    ['Close']
risk_free = yf.Ticker('^FVX').history()['Close'][-1]/100 #13 Week
    Treasury Bill (^IRX), 5 yr: ^FVX

data_norm = data/data.max()
returns_day = data.pct_change()
returns_yr = returns_day.mean() * 252 # mean times 252 trading days
     for annual
num_stocks = len(tickers_csv)
cov_matrix = returns_day.cov()

set_risk = 0.35 # std
n = len(tickers_csv)
n_chosen = len(tickers_csv)//2
returns_yr = np.array(returns_yr)
cov_matrix = np.array(cov_matrix)



model = Model("DC Model", float_precision=10)
weights = model.binary_var_dict([i for i in range(1, n+1)], name='w
    ')
weights_arr = np.array(list(weights.values()))
s1 = np.dot(np.dot(weights_arr.T, cov_matrix), weights_arr)
model.minimize(-np.dot(returns_yr, weights_arr)+s1)
model.add_constraint(np.sum(weights_arr) == n_chosen)

q_prog = from_docplex_mp(model)

qubo = QuadraticProgramToQubo().convert(q_prog)#quantum version
op, offset = qubo.to_ising()

algorithm_globals.random_seed = 10598
from qiskit import IBMQ #
provider = IBMQ.load_account() #
from qiskit.providers.ibmq import least_busy #
```

```
#backend = provider.get_backend('ibm_nairobi')
quantum_instance = QuantumInstance(
    least_busy(provider.backends(simulator=False, operational=True)
        ),
    seed_simulator=algorithm_globals.random_seed,
    seed_transpiler=algorithm_globals.random_seed,
)
qaoa_mes = VQE(quantum_instance=quantum_instance) #, initial_point
    =[0.0, 0.0]

qaoa = MinimumEigenOptimizer(qaoa_mes)  # using QAOA
#exact = MinimumEigenOptimizer(exact_mes)  # using the exact
    classical numpy minimum eigen solver

# exact_result = exact.solve(qubo)
#print(exact_result.prettyprint())

st = time.time()
result = qaoa.solve(qubo)
time_taken = time.time()-st

eigenstate = result.min_eigen_solver_result.eigenstate
eigenvector = eigenstate if isinstance(eigenstate, np.ndarray) else
    np.array(list(eigenstate.values())) #previously, eigenstate.
    to_matrix()
probabilities = np.abs(eigenvector) ** 2

i_sorted = reversed(np.argsort(probabilities))
c=0
for i in i_sorted:
    x = index_to_selection(i, n)
    value = QuadraticProgramToQubo().convert(q_prog).objective.
        evaluate(x)

    probability = probabilities[i]
    out.loc[c] = [x, value, probability]
    if c==0:
        out_max.loc[c_0]=[n, x, value, probability, time_taken]
        c_0+=1
```

```
            c+=1
        out.to_csv(f'./Data/quantum_{n}.csv', index=False, float_format
            ='%.6f')
out_max.to_csv(f'./Data/quantum_all.csv', index=False, float_format
    ='%.6f')
```