

# **Introduction**

# Motivation

Financial Systems are complex, strongly correlated and difficult to predict, full of optimization problems, Monte Carlo sampling, stochastic differential equations, machine learning…

Quantum Mechanics principles native to Quantum Computing, make it suitable for implementing financial systems, also inline with business objectives, as reduced execution time due to Quantum Advantage can also ensure more savings.

| **Problem** | **High Level Solution** |
| --- | --- |
| Which assets should be included in an optimized portfolio, and how should one change its composition according to the market? | Classical: Optimization models  **Quantum optimization**  **Quantum-Inspired Optimization** |
| How to detect opportunities in the different assets in the market, and take profit by trading them? | Classical: Machine learning  **Quantum machine learning** |
| How to estimate the risk of a portfolio, a company, or even the whole financial system? | Classical: Monte Carlo  **Quantum amplitude estimation Quantum Monte Carlo** |

Project Overview

Financial portfolio optimization is the problem of optimal allocation of a fixed budget to a collection of assets (commodities, bonds, securities etc.) which produces random returns over time.

| **Portfolio Optimization** | | |
| --- | --- | --- |
| Goals | Inputs | Output |
| • Maximize returns  • Minimize risk  • Stay within budget | • Uniform random historical price data  • Budget  • Risk tolerance | • A portfolio representing a list of investments and the expected return |

The goal in this project is to provide a solution framework which deals with the Portfolio Optimization problem. The framework will be implemented using Quantum Machine Learning algorithms on the IBM Qiskit SDK platform. The framework will be trained and tested on different stock trading data.

The dataset consists of historical (2015 – 2020) trading data for 5 stocks from US Exchange Market:

* IBM (IT Industry)
* Pfizer (Healthcare / Pharmacy)
* Exxon Mobil Corp. (Oil & Gas )
* Bank of America (Finance / Banking)
* Tesla (Automobile / Technology)

In order to build a portfolio optimizer, 5 assets for stock data are based on lesser correlation . For study three different approaches are being used for comparison:

**Non-Machine Learning :** Google Sheets Default Solver

**Classical Machine Learning :** Principal Component Analysis, Long Short-Term Memory & SciPy.

**Quantum Machine Learning:** Quantum approximate optimization algorithm adaption of Conditional Value at Risk

**Non-Machine Learning**

## Excel tools and efficient frontier theory theory

Solver is a Microsoft Excel add-in program that we can use for what-if analysis. With the help of Solver we can find an optimal (maximum or minimum) value for a formula in one cell — called the objective cell — subject to constraints, or limits, on the values of other formula cells on a worksheet. Solver works with a group of cells, called decision variables or simply variable cells that are used in computing the formulas in the objective and constraint cells. Solver adjusts the values in the decision variable cells to satisfy the limits on constraint cells and produce the result you want for the objective cell.

#### Portfolio Expected Return

The expected return of a portfolio is calculated by multiplying the weight of the asset by its return and summing the values of all the assets together. To introduce a forward looking estimate, probability may be introduced to generate and incorporate features in business and economy.



##### Expected Returns of selected assets for project with weights

|  | **XOM** | **BAC** | **IBM** | **PFE** | **TSLA** |
| --- | --- | --- | --- | --- | --- |
| **Expected Return** | -0.04% | 0.04% | 0.00% | 0.03% | 0.18% |
| **Weights** | 0.0% | 11.9% | 0.0% | 28.1% | 60.0% |

| **Portfolio Return** | 0.12% |
| --- | --- |

#### Portfolio Variance

Portfolio variance is used as the measure of risk in this model. A higher variance will indicate a higher risk for the asset class and the portfolio. The formula is expressed as



##### Variances of selected assets for project with weights

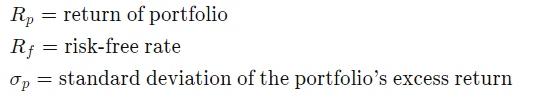
|  | **XOM** | **BAC** | **IBM** | **PFE** | **TSLA** |
| --- | --- | --- | --- | --- | --- |
| **Variance** | 0,03% | 0,04% | 0,03% | 0,02% | 0,12% |
| **Weights** | 0,0% | 11,9% | 0,0% | 28,1% | 60,0% |

| **Portfolio Risk** | 0,05% |
| --- | --- |

#### Sharpe Ratio

The Sharpe ratio measures the return of an investment in relation to the risk-free rate (Treasury rate) and its risk profile. In general, a higher value for the Sharpe ratio indicates a better and more lucrative investment. Thus, if comparing two portfolios with similar risk profiles, given all else equal, it would be better to invest in the portfolio with a higher Sharpe Ratio.





**Sharpe Ratio of Project Portfolio:** 0.032

**Efficient frontier theory**

In modern portfolio theory, the efficient frontier (or portfolio frontier) is an investment portfolio which occupies the "efficient" parts of the risk–return spectrum. Formally, it is the set of portfolios which satisfy the condition that no other portfolio exists with a higher expected return but with the same standard deviation of return (i.e., the risk). The efficient frontier was first formulated by Harry Markowitz in 1952.

This plot measures risk vs returns and is used to select the most optimum portfolio to invest into after considering the risk profile and the characteristics of the investor. The efficient frontier is essentially the part of the curve in the first and second quadrants depending on the objective and investor ability/characteristics.

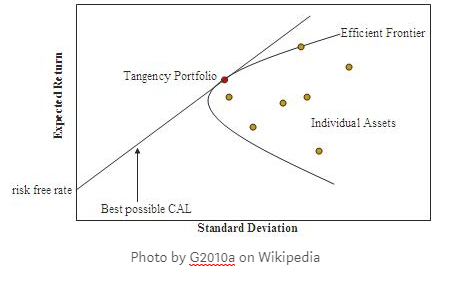


FIg. 1 - Representation of efficient frontier (Wikipedia)

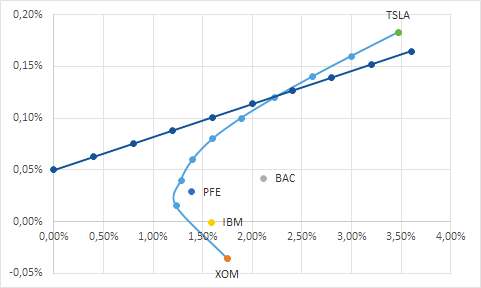
# **GRG Nonlinear Solver**

GRG stands for “Generalized Reduced Gradient”. In its most basic form, this solver method looks at the gradient or slope of the objective function as the input values (or decision variables) change and determines that it has reached an optimum solution when the partial [derivatives](https://engineerexcel.com/evaluating-derivatives-of-equations-in-excel-with-vba/) equal zero.

Of the two [nonlinear solving](https://engineerexcel.com/solving-systems-of-simultaneous-nonlinear-equations-in-excel/) methods, GRG Nonlinear is the fastest. That speed comes with a compromise though.

The downside is that the solution you obtain with this algorithm is highly dependent on the initial conditions and may not be the global optimum solution. The solver will most likely stop at the local optimum value nearest to the initial conditions, giving you a solution that may or may not be optimized globally.

Using the solver from the Excel tools in order to obtain the plot representing the efficient frontier applied to our choice of assets. We obtain the following results.



##### Fig. 2 - Portfolio optimization using Excel solver

We can see the same plot as the theory and the representation of the different stocks alone. Thus, we can see the stock with the most return is Tesla which is coherent with the evolution of the stock in recent years.

**Classical Machine Learning**

Principal Component Analysis

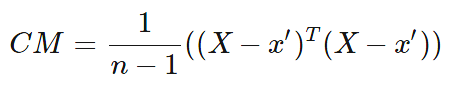
**Introduction**

Having a large number of dimensions in the feature space can dramatically impact the performance of machine learning algorithms, especially in the real-time environment. Therefore, many algorithms of dimensionality reduction and features selection are used on the original dataset to reduce the number of input features. This enables the machine learning algorithm to train faster, reduce the complexity of a model and make it easier to interpret.

In our research, we use Principal Component Analysis (PCA) algorithms to reduce the dataset dimensionality.

Large datasets are increasingly common and are often difficult to interpret. Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, by creating new uncorrelated variables that successively maximize variance. Using PCA, we can reduce the computational costs and the error of parameter estimation by reducing the number of dimensions of the feature space by extracting a subspace that describes the data best.

Technically, after standardizing the data, PCA extracts the eigenvectors and eigenvalues from the covariance matrix (CM):

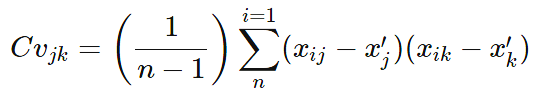


where

x′ is the mean vector:



and the covariance between two features:



The eigenvalues are then arranged in descending order, and k eigenvectors corresponding to k eigenvalues are chosen, where k is the number of dimensions of the new feature subspace (K<d). Next, PCA builds the projection matrix W from the selected k eigenvectors. And finally, it transforms the original dataset X via W to obtain a k-dimensional feature subspace Y=X∗W.

**Implementation**

In order to have a more efficient analysis of the data we use, we perform a PCA to retrieve the principal components that can keep most of the useful information of the data. This analysis allows us to have less complex data that will feed the machine learning model.

Thus we perform PCA on our portfolio of Exxon, Bank of America, IBM, Pfizer and Tesla between 01/01/2015 and 12/31/2020. We obtain the following results.

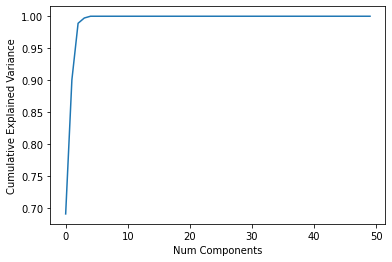


Fig. 1 - PCA on the portfolio

This graph tells us how much information is explained depending on the number of components we use. Thus, the more components we use, the more complex the data is and the more it reflects the information contained. However, as we can see, with a small number of components, we can still have a high explained variance.

This is what is powerful with the PCA: the dimension reduction performed on the data. In our case, with only 5 components, we have a cumulative explained variance score of 1 which means that adding more components will not explain any further the information contained in the data. Thus, we can perform a dimension reduction where we modify the data in order to reduce the components to have simple datasets to use for machine learning.

Once we have these new dataset of training and testing, we can move on to the next part which is the explanation of the machine learning model we use in order to analyze the data before calculating the optimal portfolio.

## Long Short-Term Memory model

Long Short-Term Memory (LSTM) is an extension of recurrent neural networks introduced by Hochreiter and Schmidhuber in 1997 that is capable of learning long term dependencies in data. This is achieved because the recurring module of the model has a combination of four layers interacting with each other.

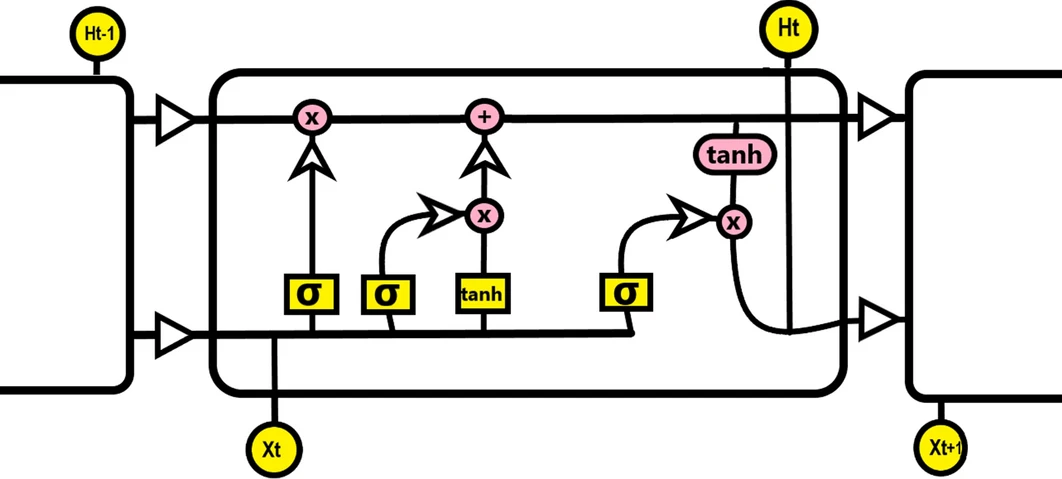
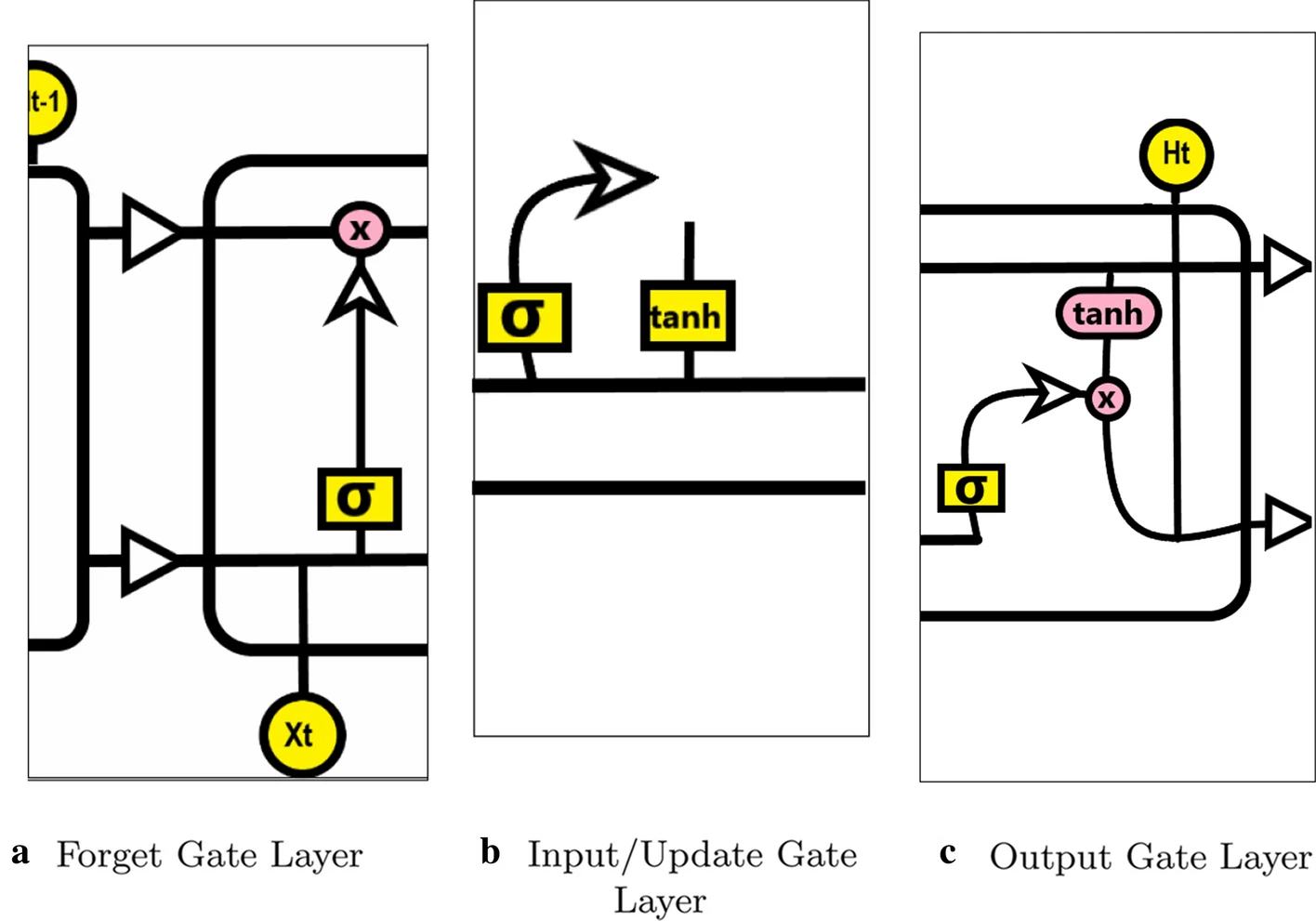


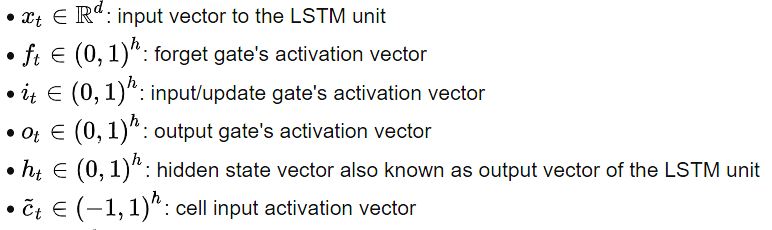
Fig. 1 - LSTM architecture

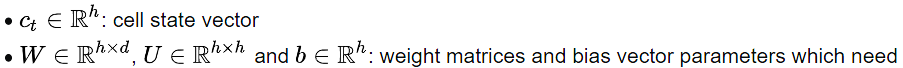


The principal component of LSTM is the cell state and three gates which provides them with the power to selectively learn, unlearn or retain information from each of the units.The cell state in LSTM helps the information to flow through the units without being altered by allowing only a few linear interactions. To add or remove information from the cell state, the gates are used to protect it, using the sigmoid function (one means allows the modification, while a value of zero means denies the modification.). We can identify three different gates :

* Forget gate layer : Look at the input data, and the data received from the previously hidden layer, then decide which information LSTM is going to delete from the cell state, using a sigmoid function (One means keep it, 0 means delete it). It is calculated with the following expression:
  + ft = σ(Wf.[ht−1,xt]+bf)
* Input/Update gate layer : Decides which information LSTM is going to store in the cell state. At first, the input gate layer decides which information will be updated using a sigmoid function, then a Tanh layer proposes a new vector to add to the cell state. Then the LSTM updates the cell state, by forgetting the information that we decided to forget, and updating it with the new vector values. It is calculated as:
  + it = σ(Wi.[ht−1,xt]+bi) and
  + =tanh(Wc.[ht−1,xt]+bC)
* Output Layer : decides what will be our output by executing a sigmoid function that decides which part of the cell LSTM is going to output, the result is passed through a Tanh layer (value between − 1 and 1) to output only the information we decide to pass to the next neuron. It is calculated as:
  + Ot=σ(Wo[ht−1,xt]+bo) and
  + ht=ot∗tanh(Ct)

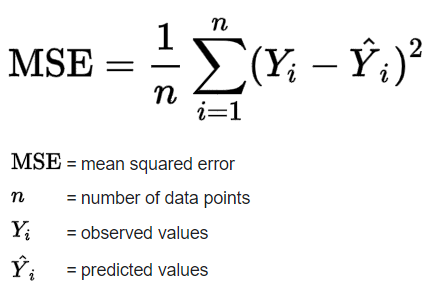
Variables:

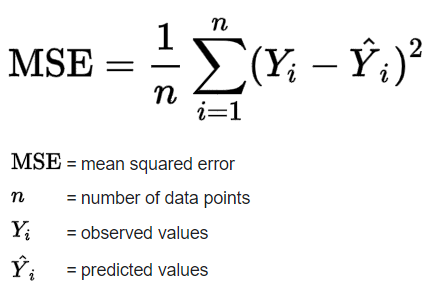




**Implementation**

We use a simple architecture with the LSTM with 2 layers of 800 neurons. Moreover, the error is calculated using the mean squared error defined as the following function.





This function enables the LSTM model to calculate the distance between the prediction made by calculation and the real value defined in the dataset.

Once the model is generated, it is possible to train it using data processed by PCA in order to simplify the prediction by not considering too complex data. After the training we obtain the following graph for the loss function.

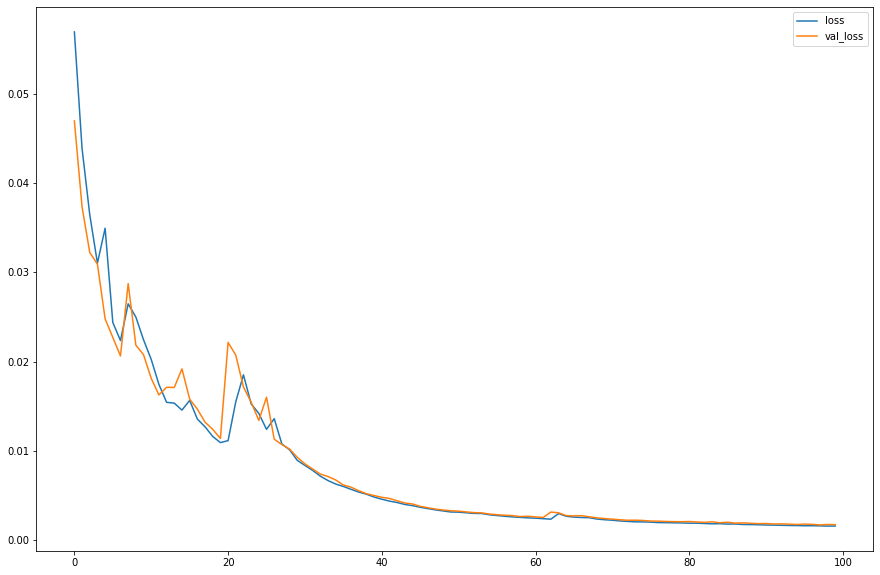


Fig. 2 - Plot of loss

This tells us the training has been successful because the loss almost goes to zero. We also obtain the accuracy of the model with the following graph. This accuracy is obtained after a lot of training sessions. That is why it is almost at the same value. However, we can see that we obtain around 20% of accuracy in the end. The problem is that we have to avoid overfitting but the series are so chaotic the problem is hard to solve.

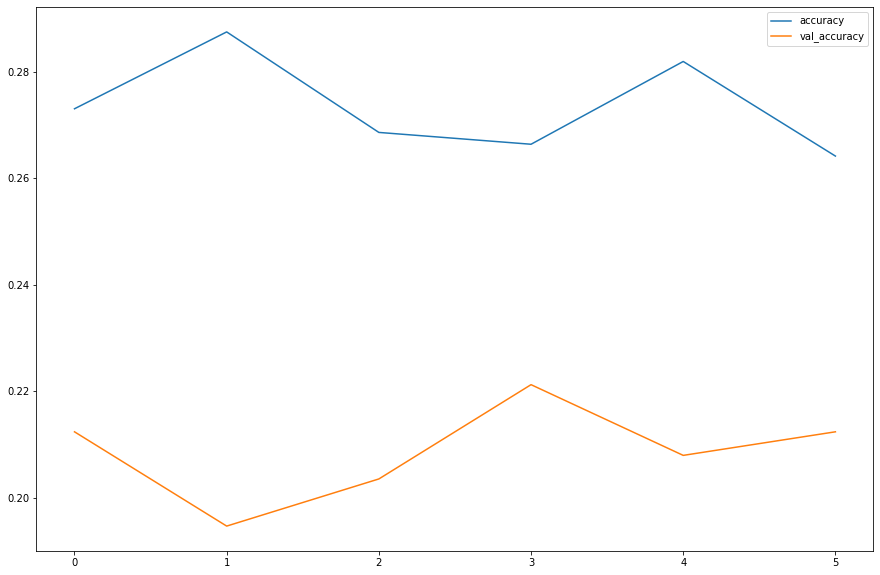


Fig. 3 - Plot of accuracy

After this training we obtain the final version of the data regarding the portfolio optimization in this new dataset.

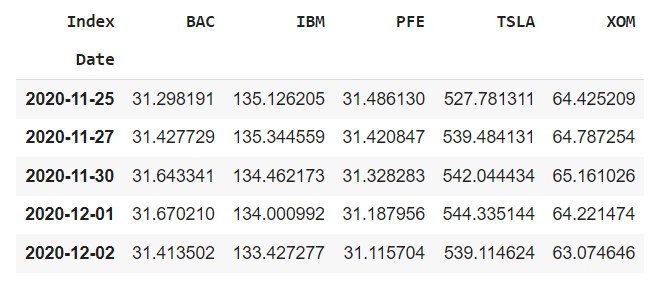


Fig. 4 - Final dataset to optimize

We can see in the following figure the evolution of the Exxon stock and the prediction made by LSTM according to the last training of the model.

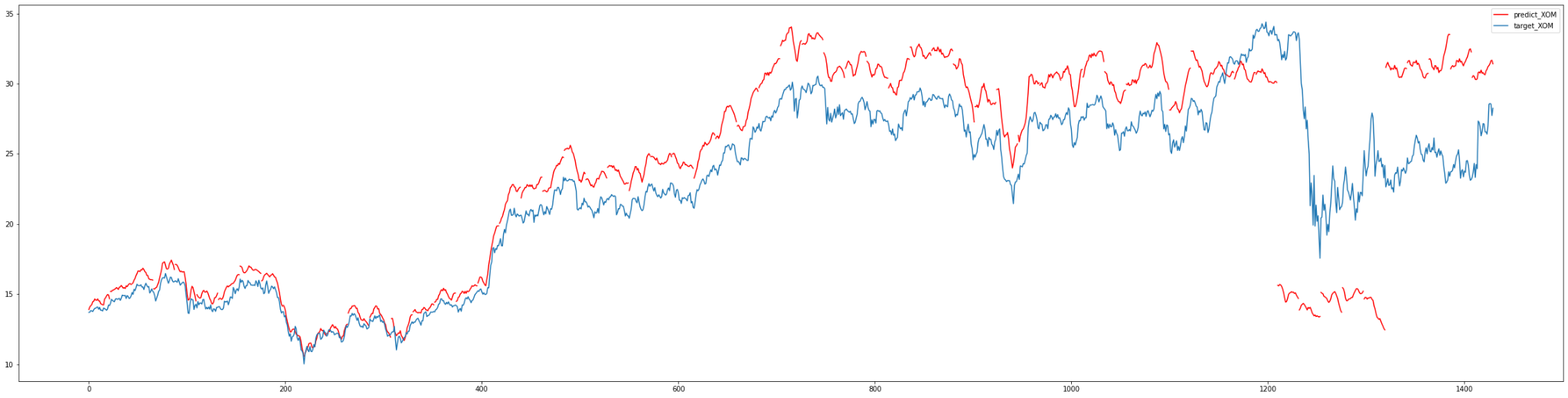


Fig. 5 - LSTM prediction for Exxon stock

Once we have this new dataset with a prediction of 22 days after the end of the first dataset, we are able to do the optimization. Indeed, the aim of the use of machine learning techniques such as PCA and LSTM is to obtain a new set of data that can tell new trends for the different assets. With this new information regarding the portfolio, we are able to have a better optimization based on the future behaviour of the market.

Now, we can go to the final part of this technique which is the optimization of the portfolio using the SciPy library. This method will allow us to obtain a final portfolio based on the analysis and prediction of machine learning models.

## SciPy optimization

**Introduction**

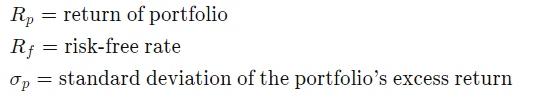
SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

It is perfect to test another optimization procedure using the tools from this Python library. Indeed, these optimizers are part of many techniques proposed by SciPy. The aim is to implement a code that will be based on one of the algorithms to optimize our portfolio based on the following assets: Exxon, Bank of America, IBM, Pfizer, Tesla.

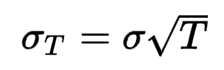
In the scipy.optimization library, we will use the minimize function. Indeed, the aim is to use different metrics that will be minimized in order to compare the different resulting portfolios.

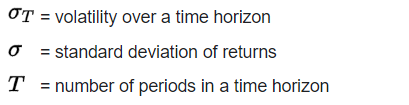
To do so, we first have the sharpe ratio. As a recap, the sharpe ratio measures the return of an investment in relation to the risk-free rate. However, because this value needs to be maximized, we use the negative of the sharpe ratio to minimize it.





Then, we will find the best portfolio based on the optimization of the volatility. This metric can be defined as a statistical measure of the dispersion of returns for a given asset. For example, when the stock market rises and falls more than one percent over a sustained period of time, it is called a "volatile" market. Minimizing the volatility in order to create the best portfolio is thus minimizing the risk of having high variations.





**Implementation**

To implement this method, we use a get\_metrics() function that allows us to define the weight, the sharpe ratio and the volatility regarding the data on which we made the previous analysis.

First, we need to define bounds which tell the minimize() function that each of our positions can only be between 0% and 100% of the allocation. Second, we need to define the constraint which will be a function that ensures in the end the sum of weights is equal to 100% allocated portfolio. In other words, we use all the "capital" we can. Third, we need to define our initial guess of the weights. The initial guess can be anything but in this case let's make it easy and start with an equally distributed portfolio. Thus we have 5 symbols so each symbol will be 20% of the portfolio.

Once we've defined these steps we can run the optimization by passing through the arguments defining the method as SLSQP which is short for Sequential Least Squares Programming. We can then run the minimize method and finally obtain the results.

When we run the minimize function of the SciPy library and return the results, we obtain the following:

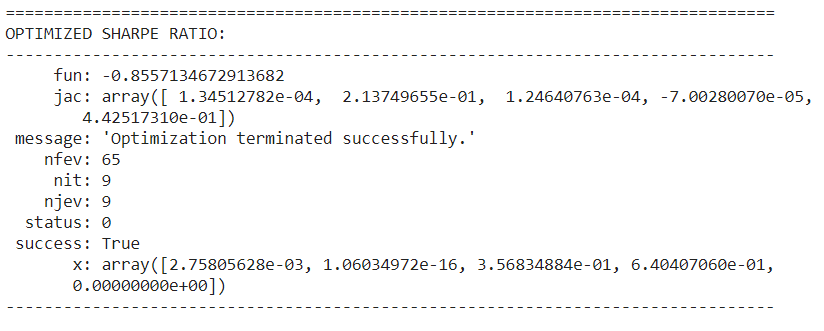


Fig. 1 - Arguments of the minimize function from SciPY

The optimized sharpe ratio is contained in the *fun* variable which is negative because of the nature of the sharpe ratio.

The weights of our portfolio are contained in the *x* variable. We have thus the following for the assets we chose using the sharpe ratio as the metric to minimize.

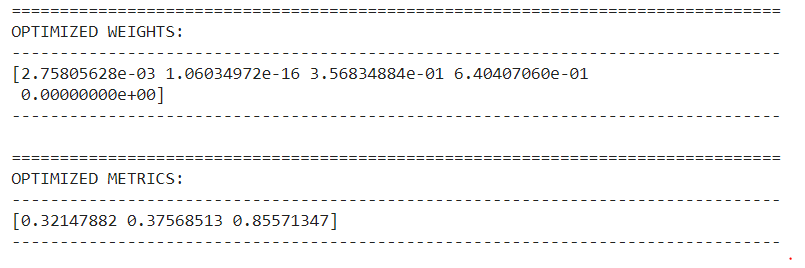


Fig. 2 - Optimized weights using sharpe ratio

As we can see, we obtain the following portfolio:

* Bank of America: 0.003
* IBM: 0
* Pfizer: 0.357
* Tesla: 0.640
* Exxon: 0

Indeed, we can see in the metrics the value of the sharpe ratio that is very high in this case with 0.86.

The aim is to compare the results obtained with the volatility as a metric. In that case, we put the risk at the center of the strategy to obtain a portfolio. The implementation is the same as for the sharpe ratio but now we use the minimize function on the calculated volatility of the stocks on the period of time we take which is 5 years. We obtain the following results.

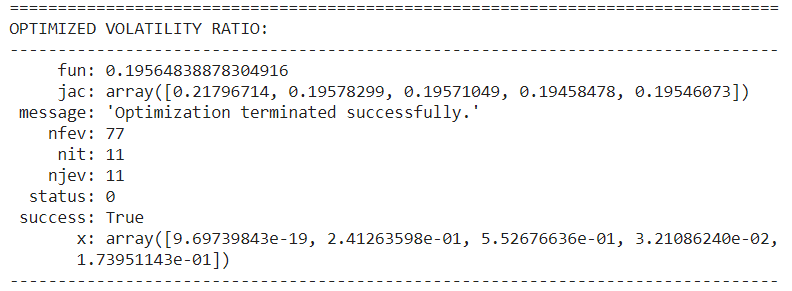


Fig. 3 - Arguments of the minimize function from SciPY

Same as before, we can see the minimized value for the volatility in the *fun* variable and the weights of the portfolio in the *x* function. The final portfolio is the following.

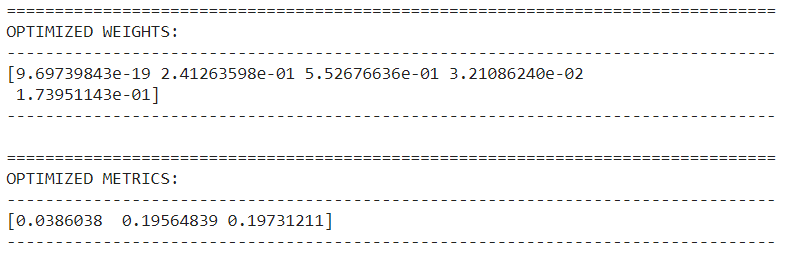


Fig. 4 - Optimized weights using volatility

In this version of the portfolio, we can see the weights are more diversified than previously. Indeed, we have:

* Bank of America: 0
* IBM: 0.241
* Pfizer: 0.553
* Tesla: 0.032
* Exxon: 0.174

This portfolio is more diversified over the different chosen assets which is coherent with the minimization of the volatility where we want to keep a smooth asset. In that case, Tesla has a lot less weight.

Finally, we can compare the two volatility and see that the previous portfolio has 0.38 compared to 0.20 in the second case. However, the second portfolio has a lot less important sharpe ratio with only 0.20.

**Quantum Machine Learning**

# Quantum Approximation Optimization Algorithm

**Introduction**

The problem of portfolio optimization can be represented as finding a way to minimize a given cost function that is related to the risk taken when choosing an asset. Thus, the weight associated with each asset needs to be updated in order to obtain a portfolio that has a low risk but with good benefits too.

By the nature of the problem we are dealing with, it is indeed possible to implement a portfolio optimization solution using quantum computing. Quantum computing tends to use the properties of particles in order to perform different kinds of computations applied to specific problems.

To do so, quantum computers use what we call qubits or quantum bits of information that obey the properties of quantum mechanics. This offers a new way of thinking about computation using superposition or entanglement. However, to obtain a result in the end, these qubits need to be measured so that we have a classical result in the end. This result, like classical computing, when considering a single bit of information, is either 0 or 1.

This way of calculations obviously needs a different theoretical frame in order to be put at its advantage. Thus, we adapt the equation accordingly.

In the formalism that we use to solve this problem, we are given the following function to compute.

Respect the following condition:

With the following notation:

* denoting the vector of binary decision variables which indicates the assets to pick or not
* the expected return for the assets
* the covariance between the assets
* controls the risk appetite of the decision maker
* describing the budget or the number of assets to be selected among the possibilities

We need to have the simplification that all assets have the same price, meaning a normalization factor between the assets and the full budget that has to be spent so all chosen assets will either be part of the portfolio or not as the result of the calculation. Thus, a constraint that follows is that the assets all have the same weight in the portfolio. The problem is to choose among the list that we have which assets are the best to pick in order to have the best profits.

It is finally easy to see how this formalism is an answer to the portfolio optimization problem using quantum computers to perform it. Now that we have the settings, we need a way to implement this problem and use quantum computers to solve it.

For that, IBM provides Qiskit, an open-source SDK in Python to code quantum algorithms. The presented formalism is made in order to match the tools offered by Qiskit. In fact, the problem can be mapped to a Hamiltonian whose ground state corresponds to the optimal solution. We will thus use the Quantum Approximation Optimization Algorithm in the form given by the Qiskit libraries.

Here, we have a representation of the architecture used to implement the QAOA algorithm.

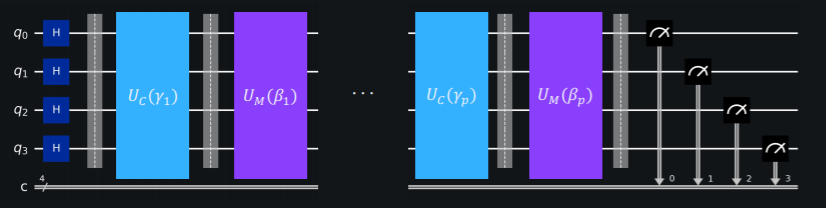
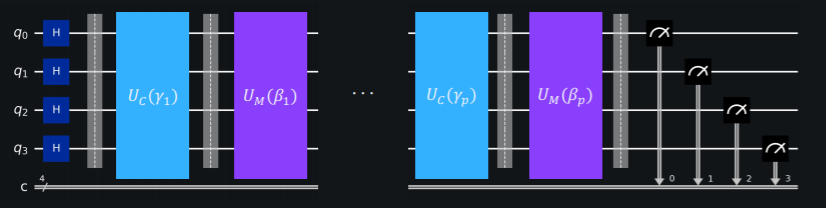
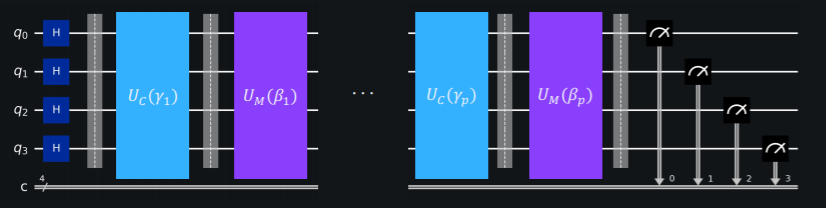


Fig. 1 - A possible implementation of the QAOA algorithm

The unitary operator refers to the cost layer and to the mixer layer. The first encodes the Hamiltonian of the problem with the cost function and the optimization problem. The second mixes the results in order to make the assets that are valuable appear more often and thus they will come out more often after the measurement.

Once we have the structure of the algorithm, we are able to implement it in order to solve our problem of portfolio optimization.

**Implementation**

The first part is the definition of the problem instance such as the number of considered assets and the encoding using the Hamiltonian. In our case, we choose the following assets: Exxon, Bank of America, IBM, Pfizer, and Tesla. This problem configuration allows us to have assets from a large range of big companies in very different industries. We use 5 qubits to represent each asset in the end. Then, we load the data from Yahoo Finance between 01/01/2015 until 12/31/2020 into a data frame and plot the covariance matrix.

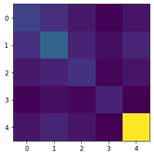


Fig. 2 - Covariance matrix

We can see this result confirms the portfolio we chose. Indeed, the covariance matrix shows no direct correlation between the different sets of data referring to the time evolution of each asset. This can offer a variety of portfolio optimization because the different assets do not influence one another.

Now that the assets and the number of qubits used for the algorithm are defined, we can set the parameters for QAOA. We set the following:

This penalty variable allows us to set a parameter to scale the budget penalty term as presented in the definition of the condition equation.

From that, we can use the get\_operator() function from the portfolio class of Qiskit finance that takes the covariance terms as implementation of the data to obtain qubitOp and offset in order to run the quantum algorithm. We obtain the following configuration.



Fig. 3 - Parameters setting

Moreover, to compare the results obtained by the Qiskit implementation of QAOA, we use the Numpy eigensolver as a classical reference. We obtain the following result for our case.

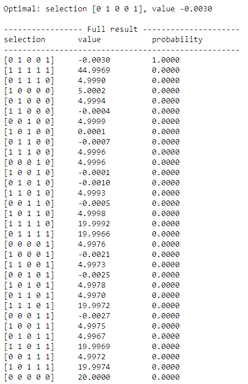


Fig. 4 - Numpy eigensolver results

We can see the optimal portfolio being [0,1,0,0,1] with a high probability of 1. As presented before, in this quantum approach, because we are dealing with qubits resulting either in 0 or 1, the portfolio is whether we take the asset or not. Thus, the classical eigensolver indicates the optimal portfolio is to take the full budget and split between Bank of America and Tesla.

Now, we can use the QAOA function of Qiskit algorithms library and set the following parameters: qubitOp as the operator and cobyla as the optimizer. We obtain the following results.

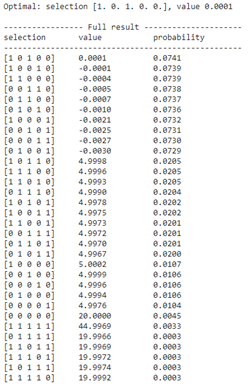


Fig. 4 - QAOA results

The optimal selection found by the QAOA algorithm is [1,0,1,0,0] with probability of 0.0741 which is very low and close to the 9 other configurations with 2 assets taken into account. This portfolio is thus different from the Numpy selection because it takes Exxon and Tesla as the best assets to allocate the budget.

However, we can see that the portfolio deduced from the Numpy eigensolver has a 0.0729 probability and is part of the ones with probability over 0.07. Thus, at these levels, the selections can be considered as good candidates for best performances portfolio. The problem is that the optimal selection cannot be fully distinguished from the other over 10-3 which is very low considering all the possibilities.

In conclusion, this approach using the QAOA algorithm defined in the Qiskit libraries obtains results that are difficult to interpret because of the low probabilities regarding the selections of portfolios. However, as seen in the study presented by Qiskit at this [link](https://qiskit.org/documentation/finance/tutorials/01_portfolio_optimization.html), they manage to obtain the same results as the Numpy eigensolver which means the quantum solution has potential in the future.

# Improving Variational Quantum Optimization using CVaR

**Introduction**

In quantum mechanics observables are defined as expected values ⟨ψ|H|ψ⟩. This leads to the natural choice of the sample mean as the objective function for the classical optimization problems embedded in VQE and QAOA. We argue that for problems with a diagonal Hamiltonian, such as Computational Optimization problems, the sample mean may be a poor choice in practice. This is because when H is diagonal, there exists a ground state which is a basis state. If determining the value Hj,j of a basis state |j⟩ is classically easy, the state with the minimum eigenvalue computed by an algorithm is simply the best measurement outcome among all measurements performed.

It is therefore reasonable to focus on improving the properties of the best measurement outcome, rather than the average. Consider two algorithms A1 and A2 applied to a problem with Hamiltonian H, minimum eigenvalue λmin and ground state |j⟩. Suppose A1 produces a state |ψ1⟩ and A2 produces a state |ψ2⟩, with the following properties:

⟨ψ1|H|ψ1⟩/λmin=1.1 and ⟨ψ2|H|ψ2⟩/λmin=2.0

We argue that from a practical point of view, algorithm A2 is likely to be more useful than A1. This is because even if A1 leads to samples with a better objective function value on average, it will never yield the ground state |j⟩ ; whereas A2, which is much worse on average, has a positive and sufficiently high probability of yielding the ground state, so that with enough repetitions we can be almost certain of determining |j⟩.

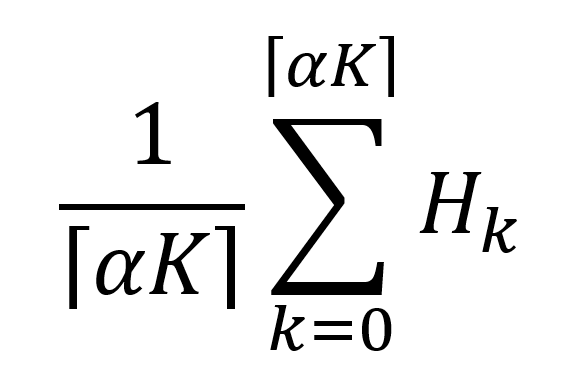
For a given set of shots with corresponding objective values of the considered optimization problem, the CVaR with confidence level α∈[0,1] is defined as the average of the α best shots. Thus, α=1 corresponds to the standard expected value, while α=0 corresponds to the minimum of the given shots, and α∈(0,1) is a tradeoff between focusing on better shots, but still applying some averaging to smoothen the optimization landscape.

In light of this discussion, one way to attain our goal would be to choose, as the objective function, the minimum observed outcome over a set of measurements: min {H1,...,HK}

However, for finite K this typically leads to a non-smooth, ill-behaved objective function that is difficult to handle for classical optimization algorithms.

To help smooth the objective function, while still focusing on improving the best measured outcomes rather than the average, we propose the Conditional Value at Risk (CVaR, also called Expected Shortfall) as the objective function. Formally, the CVaR of a random variable X for a confidence level α ∈ (0,1] is defined as the following:

where FX denotes the cumulative density function of X. In other words, CVaR is the expected value of the lower α-tail of the distribution of X. Without loss of generality, assume that the samples Hk are sorted in nondecreasing order, i.e. Hk+1≥Hk. Then, for a given set of samples {Hk}k=1,...,K and value of α, the CVaRα is defined as



Note that the limit α↘0 corresponds to the minimum, and α=1 corresponds to the expected value of X. In this sense, CVaR is a generalization of both the sample mean ([5](https://www.arxiv-vanity.com/papers/1907.04769/#S2.E5)), and the best observed sample: min{H1,..,HK}

For small, nonzero values of α, CVaR still puts emphasis on the best observed samples, but it leads to a smoother and easier to handle objective function. It is clear that this can be applied to both VQE and QAOA, simply by replacing the sample mean ([5](https://www.arxiv-vanity.com/papers/1907.04769/#S2.E5)) with CVaR α in the classical optimization algorithm. We call the resulting algorithms CVaR-VQE and CVaR-QAOA, respectively.

**Implementation**

Once we have the QAOA algorithm thanks to the Qiskit libraries, it is possible to improve the results using CVaR as previously presented. To do so, we use the same definition of the problem as the QAOA version and we implement the function get\_expectation().

This function creates the quantum circuit as a Quadratic Unconstrained Binary Optimization instance or QUBO which allows the use of the CVaR to speedup the optimization process. Then the function computes the CVaR using a fraction α of the best measured outcomes according to the equation presented previously.

Depending on the value of α, the result of the CVaR which considers only the best measurements could be different. Thus, we can print the surface plot of the data which shows what landscape has been optimized to find the best solution in every case. We always compare to the classical eigensolver which gives us the best portfolio being Bank of America and Tesla. We obtain the following results for different values of α.

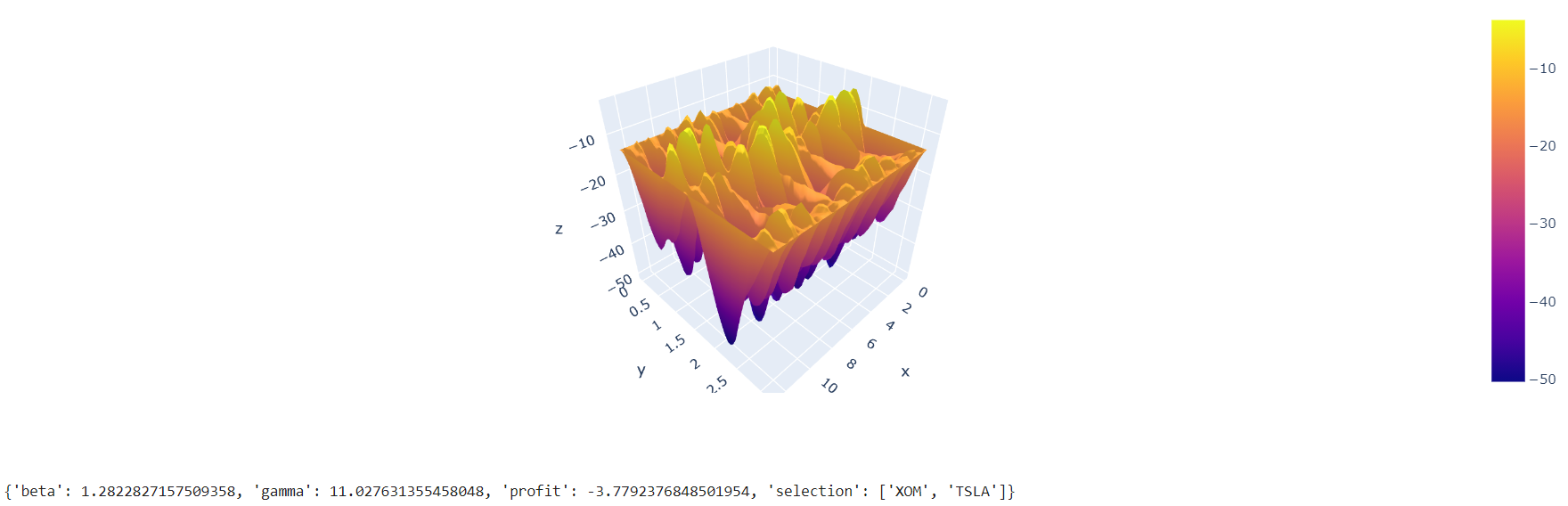
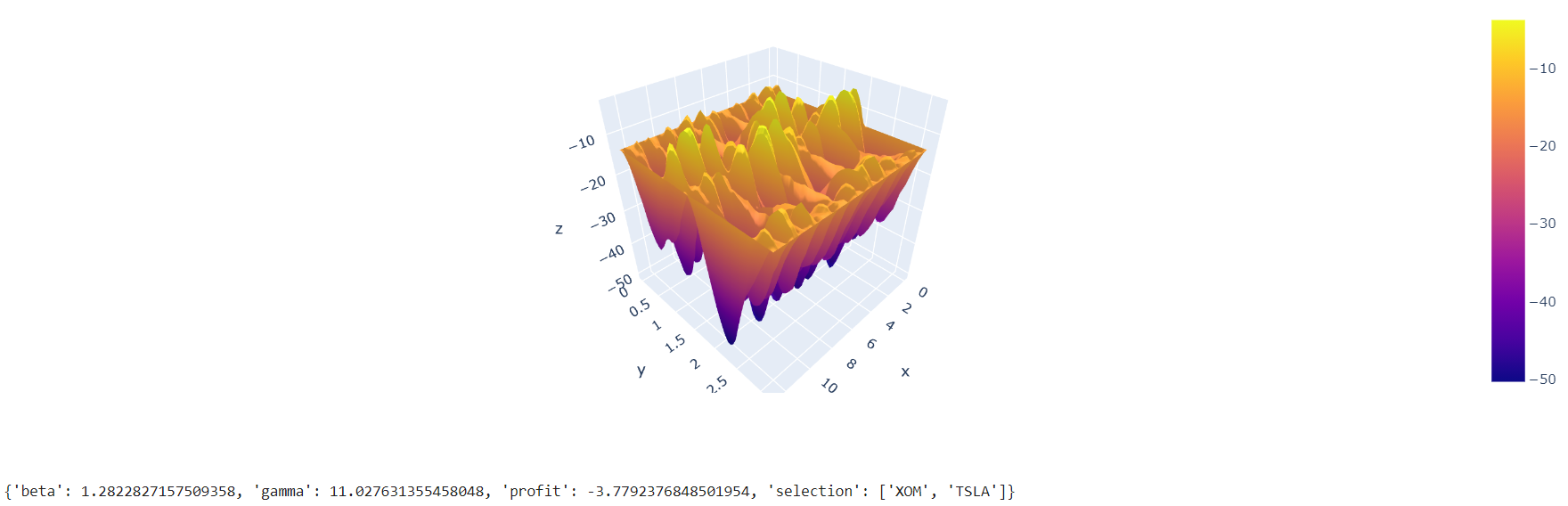




Fig. 1 - Results for α=1

In this case we take the entire set of best measurements to compute the CVaR. We obtain in this case the best selection of Exxon and Tesla. When we compare it to the classical portfolio we have the right pick for Tesla but not for Bank of America. Moreover, when we refer to the previous results with the cobyla optimizer we have a different result. Indeed, the first version returned Exxon and IBM as the best pick. Thus, we can see that even with α=1, we have some improvement of the QAOA optimization process using CVaR. Now, we will decrease the α parameter and see what happens.

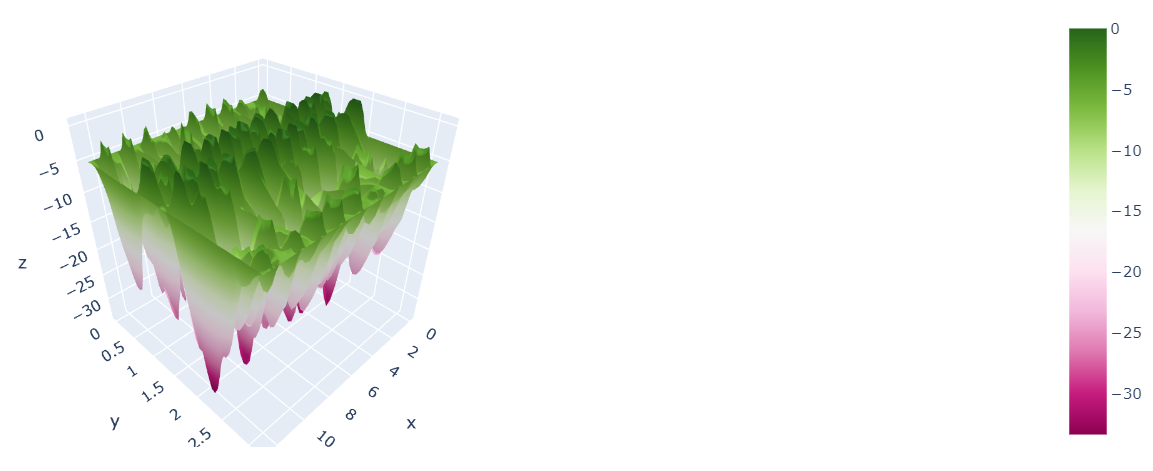
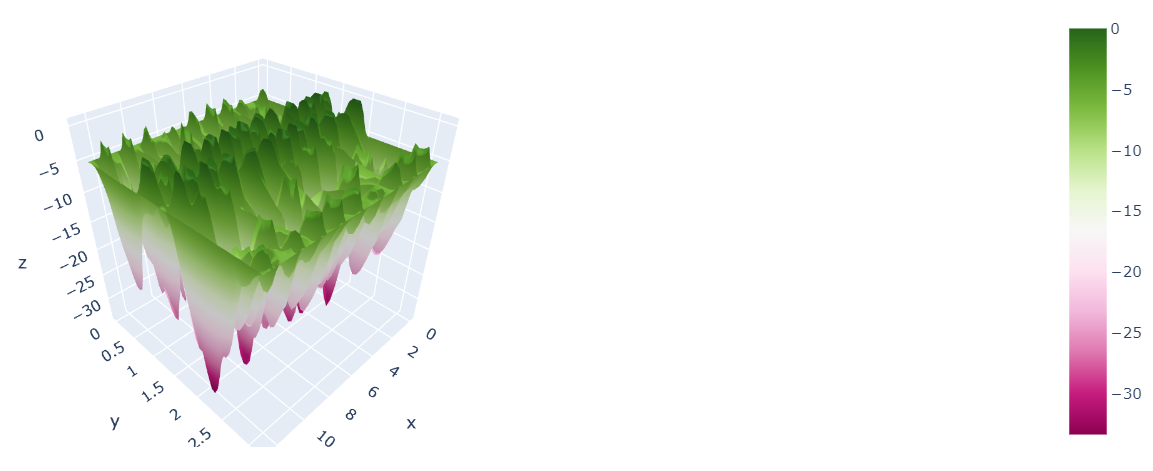




Fig. 2 - Results for α=0.7

Now, we choose to remove some measurements in order to speed up the optimization process. We can see that removing those measurements causes a loss of precision compared to the classical eigensolver. Indeed, we find the same result as the first implementation of QAOA with cobyla optimizer. We obtain similar results for α=0.5.

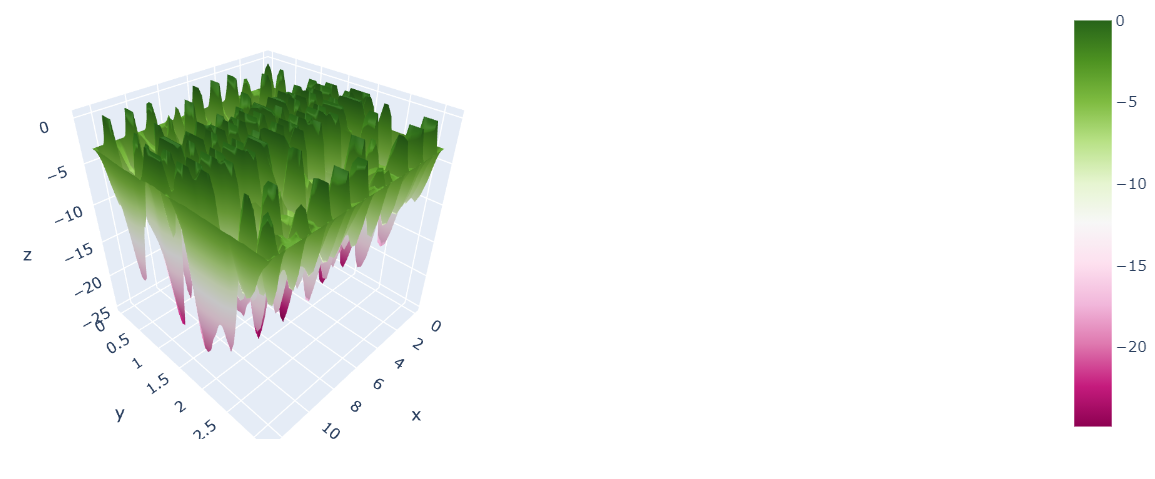
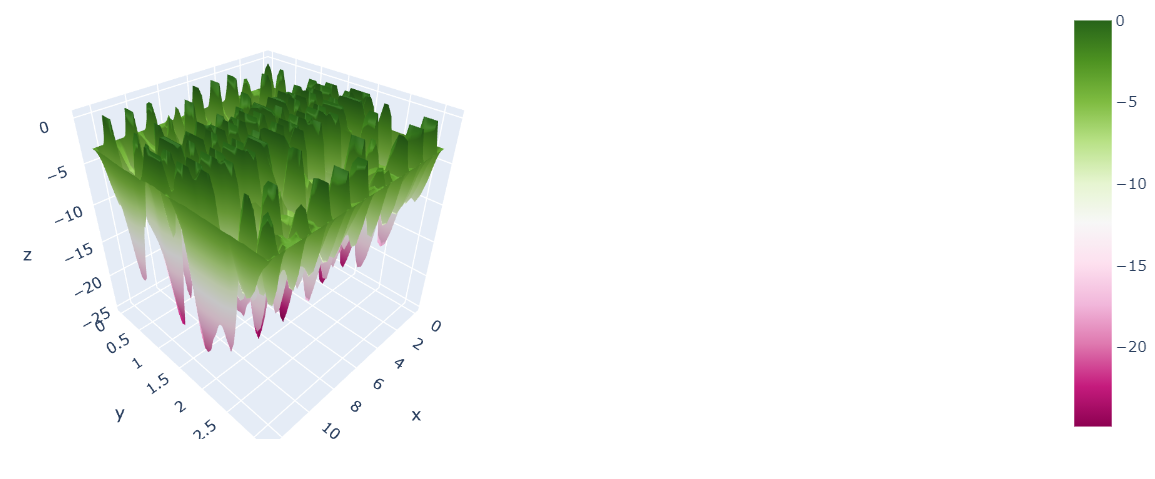




Fig. 3 - Results for α=0.5

When we refer to the table of probabilities obtained in the first implementation of QAOA, we can see that the best result was the portfolio Exxon and IBM. Indeed, if we remove some of the best measurements, this solution has a more important weight and thus the optimizer will converge to this solution. This is why even if we decrease α, we will reach a plateau of the same results. This assumption is confirmed by the results for α=0.2.

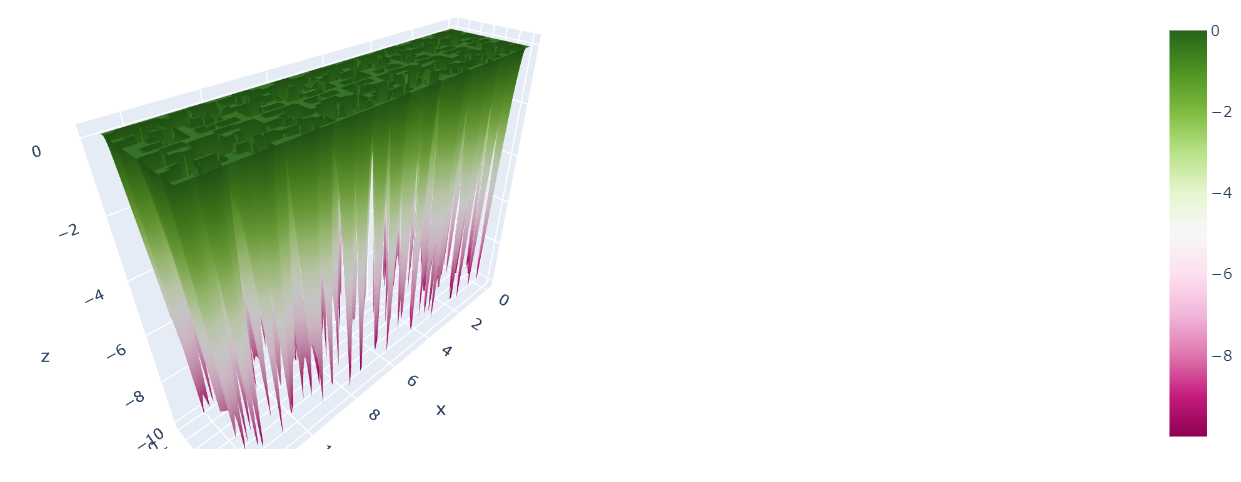
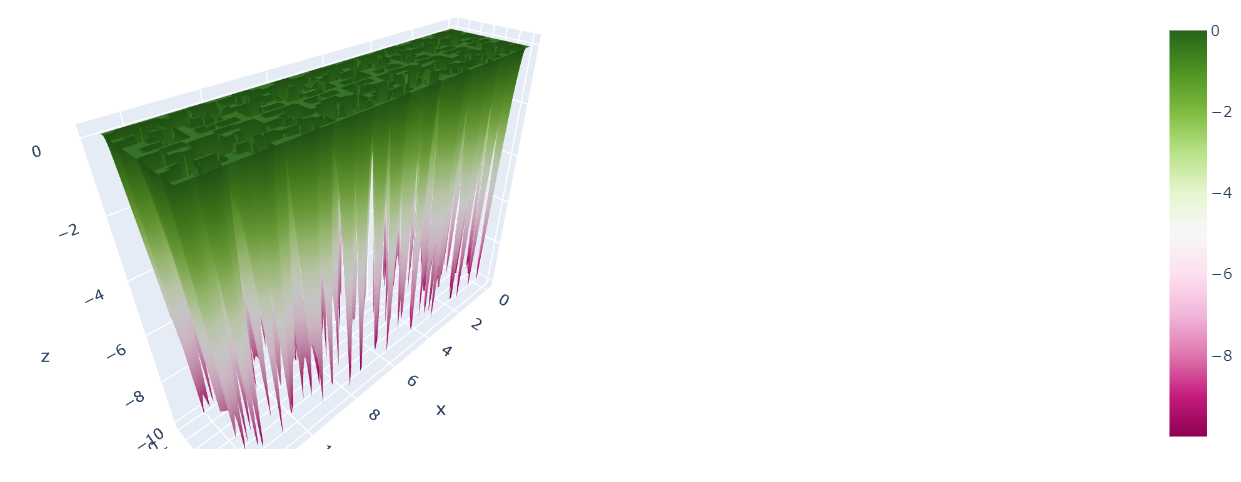




Fig. 4 - Results for α=0.2

We can see that the resulting portfolio is always the same. When we analyze the landscape, we can see only wells defined in pink which represent a lot of different solutions the optimizer can fall into. Thus by removing a lot of the measurements, the result becomes more uncertain. This leads to the result we obtain with α=0.05.





Fig. 5 - Results for α=0.05

When we remove 95% of the measurements, the solution can be either no assets or all of them. However, because we need to spend the budget, it is automatically all the qubits that are activated. Thus we obtain the portfolio by choosing all the assets. This gives apparently no advice on how to optimize it in the context of quantum solutions where either we choose the asset or not.

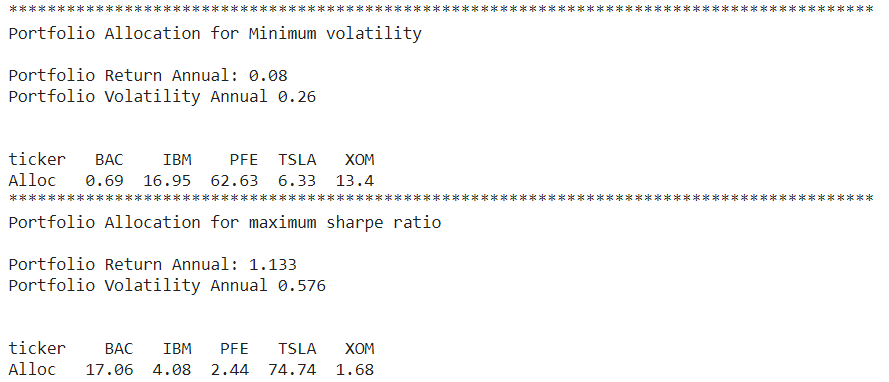
**Conclusion**

Comparison

All these different approaches allow us to obtain in the end a comparison in order to see which one is better suited as a solution of the portfolio optimization problem. To have the most relevant comparison, we analyze the annual return of the portfolio obtained for the duration of the study between 2015 and 2020. We obtain the following results.

**Efficient Frontier Theory**

With this approach, we use the default solver present in Google Sheets or Excel to obtain the result based on this portfolio theory. We obtain the following portfolio for the best return with the optimization of the sharpe ratio.



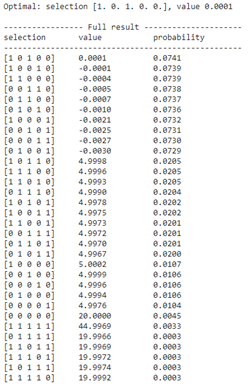
**Classical Machine-Learning**

In this approach, we keep the optimization of the portfolio using the sharpe ratio as a metric because it is the one that was considered for the other. Thus, the final portfolio after the analysis of LSTM is the following:

* Bank of America: 0.003
* IBM: 0
* Pfizer: 0.357
* Tesla: 0.640
* Exxon: 0

**Quantum Machine Learning**

The result of this method is a portfolio where we divide all the budget into the different chosen assets. The weights are thus equally shared among the assets that were found relevant at the end of the QAOA calculations. We obtain the final optimal selection for QAOA based on the scope of the study which corresponds to a portfolio with 50% on Exxon and 50% on Tesla.



Thanks to those different results, we can plot the return of the portfolio over the following year 2021 so far to see which one is the best. We obtain the following graph.

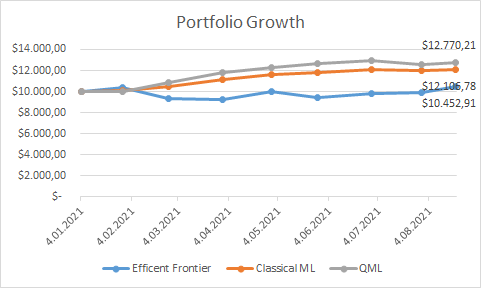


Fig. 1 - Comparison graph of different returns according to the approaches

Conclusion

We can see that for this situation, if we take the analysis of the Quantum Machine-Learning, we obtain better profits for the year 2021. Indeed, the analysis given by the three approaches are based on past data in order to build the best portfolio for the following year.

Moreover, we can see that the portfolio obtained with Classical Machine-Learning has close results to the Quantum version even though they share Tesla but the rest of the portfolio is totally different.

Finally, the Non ML solver gives the less interesting results. Indeed, this approach is based on the past data to obtain the best portfolio. Thus, it has a poor capacity of obtaining a good prediction result.

As a conclusion, Quantum Machine-Learning is a viable solution to take into account when talking about this optimization problem. Moreover, in the particular situation we have now, this analysis is the best we have.