Quantum Reinforcement Learning for Enhanced Portfolio Optimization

Sai Rahul Kodeboina
CSE Specialization in AI
VIT-AP UNIVERSITY
Andhra Pradesh, INDIA
rahul.21bce9121@vitapstudent.ac.in

Kusela Chitteti
CSE Specialization in AI
VIT-AP UNIVERSITY
Andhra Pradesh, INDIA
kusela.21bce9158@vitapstudent.ac.in

Dunde Sumathi
CSE Specialization in AI
VIT-AP UNIVERSITY
Andhra Pradesh, INDIA
sumathi.21bce9401@vitapstudent.ac.in

***Abstract*—This research paper presents a pioneering study on the integration of quantum computing with reinforcement learning, coined as Quantum Reinforcement Learning (QRL), aimed at revolutionizing portfolio management strategies. The project's core objective is to harness the advanced computational capabilities of quantum systems to process intricate financial datasets and perform optimization tasks with greater efficiency than classical computing methods. By utilizing quantum circuits and deep reinforcement learning algorithms, the study seeks to develop a model capable of adapting to dynamic market conditions and optimizing investment portfolios with remarkable precision. This innovative approach is anticipated to significantly diminish computational time and resources, offering a scalable and practical solution for real-world financial applications. The project is poised to make substantial contributions to the emerging field of quantum finance, setting a new precedent for future research and development in quantum-enhanced machine learning models for economic decision-making. The methodology involves constructing quantum circuits to encode financial data into quantum states, subsequently manipulated to simulate complex financial scenarios. Iterative enhancement of these quantum circuits through deep reinforcement learning algorithms is expected to improve their ability to identify optimal investment strategies. The anticipated outcome is a robust, scalable model that outperforms traditional portfolio optimization methods, demonstrating superior computational efficiency and the capacity to handle larger datasets and more complex financial instruments. The project aspires to significantly advance the field of quantum finance, providing insights that could catalyze the development of novel quantum based financial tools. Through meticulous testing and validation, the research aims to establish a new benchmark for portfolio optimization, offering a glimpse into the future of financial technology.**

Keywords— Quantum Computing, Reinforcement Learning, Portfolio Optimization, Quantum Circuits, Financial Data Encoding, Quantum Algorithms, Deep Learning, Quantum Machine Learning, Financial Technology, Quantum Finance, NISQ Devices, Quantum Simulation, Investment Strategies, Market Dynamics, Quantum States, Risk Management, Computational Efficiency, Hybrid Quantum-Classical Algorithms, Quantum Optimization, Financial Markets.

I. INTRODUCTION

The reasearch project "Quantum Reinforcement Learning for Enhanced Portfolio Optimization" addresses the challenge of optimizing investment portfolios using the emerging technology of quantum computing, integrated with reinforcement learning. The traditional methods of portfolio optimization, while effective to a degree, are limited by the computational constraints of classical computing systems, especially when dealing with large, complex financial datasets and the need for real-time decision-making. The advent of quantum computing offers a potential paradigm shift in this domain, promising significant improvements in processing power and speed, which could revolutionize the way financial markets operate.

The problem statement for this research is twofold. Firstly, it seeks to explore how quantum computing can be utilized to enhance the computational efficiency and accuracy of portfolio optimization algorithms. This involves the development of quantum algorithms that can handle the probabilistic nature of financial markets and execute optimization tasks more rapidly than their classical counterparts. Secondly, the research aims to integrate these quantum algorithms with reinforcement learning techniques to create a dynamic, adaptive model that can learn from market changes and optimize investment strategies accordingly.

Reinforcement learning, a subset of machine learning, involves training an agent to make a sequence of decisions by interacting with an environment to achieve a certain goal. In the context of portfolio optimization, the agent's goal is to maximize the expected return of the investment portfolio while minimizing risk. However, the financial market is a complex, stochastic environment with a vast number of variables, including stock prices, interest rates, and economic indicators, which makes the problem highly non-linear and difficult to solve using traditional methods.

The integration of quantum computing with reinforcement learning presents a unique opportunity to tackle these challenges. Quantum computers can process and analyze large datasets much faster than classical computers, allowing for the evaluation of numerous potential investment strategies in a fraction of the time. This capability, combined with reinforcement learning's ability to adapt and learn from new data, could lead to the development of a powerful tool for financial analysts and investors.

The research will focus on constructing quantum circuits that can represent financial data and perform computations necessary for portfolio optimization. It will also involve developing a reinforcement learning framework that can interact with these quantum circuits to make investment decisions. The ultimate objective is to create a quantum reinforcement learning model that can outperform existing portfolio optimization methods in terms of speed, accuracy, and adaptability.

This research has the potential to significantly impact the field of finance by providing a more efficient and effective method for portfolio optimization. It could enable investors to make better-informed decisions, reduce the risk of investment, and ultimately lead to more stable and prosperous financial markets. However, the project also faces several challenges, including the current limitations of quantum hardware and the complexity of integrating quantum algorithms with reinforcement learning techniques.

In conclusion, the problem statement of this research is to investigate the feasibility and effectiveness of using quantum computing to enhance reinforcement learning algorithms for portfolio optimization. The research aims to develop a model that can process complex financial data more efficiently and adapt to changing market conditions, providing a novel approach to investment strategy optimization.

II. Literature Survey:

The detailed literature review for the project "Quantum Reinforcement Learning for Enhanced Portfolio Optimization" encompasses an examination of recent advancements in quantum computing applications in finance, specifically focusing on reinforcement learning techniques for portfolio optimization.The review highlights the potential of quantum algorithms to solve complex optimization problems more efficiently than classical methods and the integration of these algorithms with machine learning techniques to adapt to dynamic market conditions.

Quantum Computing in Finance: Recent studies have explored the application of quantum computing to financial problems, with a focus on portfolio optimization. Researchers have experimented with the Variational Quantum Eigensolver (VQE) on real quantum devices to find optimal investments, demonstrating the potential of quantum algorithms to handle the scaling complexity of market dimensions[1]. The findings suggest that with the correct hyperparameters and sufficiently sized quantum computers, quantum algorithms can reach solutions very close to the exact ones, even without error-mitigation techniques[1].

Reinforcement Learning Techniques: The integration of machine learning with quantum computing is particularly promising in the field of reinforcement learning. A review of different classical, statistical, and intelligent approaches employed for portfolio optimization has been presented, highlighting the importance of machine learning and artificial intelligence in assisting portfolio managers to make informed decisions[2]. The review also emphasizes the role of quantum-inspired techniques in enhancing these approaches[2].

Hybrid Quantum-Classical Models: The emergence of hybrid quantum-classical models has been a significant development. These models utilize quantum circuits and quantum differential programming to implement quantum machine learning (QML) and quantum reinforcement learning (QRL). Experiments conducted on stock simulators and different assets have shown that these hybrid models can perform well in challenging market conditions, such as those following the COVID-19 outbreak[3].

Systematic Literature Reviews: Systematic literature reviews have identified the main methods, tools, and techniques used for portfolio optimization, analyzing how their applications have changed over time. This includes real-world constraints and the evolution of portfolio optimization methods, providing a comprehensive understanding of the field's trajectory[4].

Quantum Reinforcement Learning: Quantum reinforcement learning is an emerging field at the intersection of quantum computing and machine learning. Recent literature has focused on variational quantum circuits acting as function approximators in classical reinforcement learning settings, as well as algorithms based on future fault-tolerant hardware that may offer a quantum advantage[56]. These studies provide both a broad overview and detailed analyses of selected parts of the literature, contributing to a deeper understanding of QRL's capabilities and limitations[6].

Comparative Studies: Comparative studies have analyzed the capabilities of quantum machine learning for reinforcement learning, comparing the performance of gate-based and annealing-based quantum approaches with classical deep reinforcement learning. These studies have confirmed that quantum approaches can outperform classical methods, paving the way for more efficient and effective portfolio optimization strategies[78].

Advancements in Quantum Computing for Portfolio Optimization: Recent studies have demonstrated the application of the Variational Quantum Eigensolver (VQE) to portfolio optimization problems. Researchers have experimented with VQE on real quantum devices, finding that with the right hyperparameters, quantum algorithms can approach the quality of classical solutions, even without error-mitigation techniques[1]. This suggests that as quantum hardware continues to mature, quantum computing could offer more efficient solutions for portfolio optimization than existing methods[1].

Machine Learning and AI in Portfolio Management: A comprehensive review of portfolio optimization techniques has shed light on the role of machine learning and artificial intelligence in aiding portfolio managers. The study compares classical, statistical, and intelligent approaches, including quantum-inspired techniques, highlighting the potential for these methods to assist in making informed decisions in volatile market situations[2].

III. METHODOLOGY

Methodology:

Quantum Circuit Design and Implementation:

- Develop quantum circuits that encode financial datasets into quantum states, utilizing quantum gates to simulate market scenarios.
- Implement Quantum Fourier Transform (QFT) for periodicity detection in financial time series data.
- Use Quantum Amplitude Estimation (QAE) to evaluate the probability of outcomes, aiding in risk assessment and portfolio diversification.

Integration of Deep Reinforcement Learning:

- Employ deep reinforcement learning algorithms, such as Deep Q-Networks (DQN) and Policy Gradients, to train quantum circuits on decision-making tasks.
- Introduce a reward system based on financial metrics like Sharpe ratio, to guide the learning process towards optimal trading strategies.
- Explore the use of Recurrent Neural Networks (RNNs) to process sequential data for predicting market trends and adjusting portfolio allocations dynamically.

Optimization and Error Mitigation:

- Optimize quantum circuits for NISQ devices, focusing on reducing gate complexity and improving coherence times.
- Apply error mitigation techniques like Zero-Noise Extrapolation (ZNE) and Symmetry Verification to enhance the reliability of quantum computations.
- Conduct hyperparameter tuning to balance exploration and exploitation, ensuring the model adapts to market changes without overfitting.

Model Training and Validation:

- Train the model using a comprehensive dataset that includes various market conditions, asset classes, and financial instruments.
- Validate the model's performance through extensive backtesting, comparing it with classical portfolio optimization methods and benchmarks.
- Perform sensitivity analysis to understand the impact of quantum noise and market volatility on the model's performance.

Anticipated Outcomes:

- A robust QRL model that demonstrates superior computational efficiency and accuracy in portfolio optimization tasks.
- A scalable solution capable of handling high-dimensional financial datasets and complex optimization problems.
- Insights into the application of quantum computing in finance, potentially leading to the development of new quantum-based financial instruments and strategies.

Project Flow/ Framework of the Proposed System

The Project Flow/Framework of the proposed system for "Quantum Reinforcement Learning for Enhanced Portfolio Optimization" is a structured approach to developing a sophisticated model that integrates quantum computing with reinforcement learning. Here's a detailed explanation of the methodology:

1. Problem Definition:

- Identify and articulate the specific challenges and objectives in portfolio optimization that the project aims to address.

2. Literature Survey:

- Conduct a comprehensive review of existing research and publications to gather foundational insights and identify gaps in current methodologies.

3. Tool Selection:

- Choose the necessary software and hardware tools, such as Qiskit, for implementing quantum circuits and reinforcement learning algorithms.

4. Model Design:

- Create a detailed blueprint for the quantum reinforcement learning model, outlining the architecture and how it will function.

5. Implementation:

- Translate the model design into a working system through programming and coding, ensuring that the theoretical model is accurately represented.

6. Simulation and Testing:

- Run simulations using quantum computing simulators to test the model's predictions and behavior under various conditions.

- Identify and rectify potential flaws or weaknesses in the model.

7. Evaluation:

- Assess the performance, efficacy, and relevance of the model against predefined criteria or benchmarks.
- Analyze the results to draw conclusions and gain insights into the model's capabilities.

8. Optimization:

- Refine and improve the model by identifying areas for optimization, fine-tuning parameters, or adjusting strategies to enhance performance and efficiency.

9. Documentation:

- Document all stages of the project, including methodologies, findings, results, and any relevant insights or lessons learned, to facilitate understanding and replication.

Methodology Details:

This project employs a comparative approach to financial portfolio optimization using a combination of non-machine learning, classical machine learning, and quantum machine learning techniques. The dataset consists of historical trading data (2015-2020) for five stocks from the US Exchange Market: IBM, Pfizer, Exxon Mobil Corp., Bank of America, and Tesla.

1. Non-Machine Learning

- Efficient Frontier Model: Calculate the efficient frontier using historical price data to identify portfolios with optimal risk-return trade-offs.
- Employ tools such as Excel Solver to perform the optimization.
- Performance Evaluation: Utilize the Sharpe Ratio to assess risk-adjusted returns of different portfolios.

2. Classical Machine Learning

- Dimensionality Reduction: Apply Principal Component Analysis (PCA) to reduce the features of the dataset, simplifying subsequent analysis and computations.
- Price Forecasting: Implement a Long Short-Term Memory (LSTM) recurrent neural network model to predict future asset prices based on historical trends.
- Portfolio Optimization: Utilize SciPy's optimization libraries to fine-tune portfolio weights, minimizing risk metrics (e.g., volatility) or maximizing the Sharpe Ratio, taking into account the forecasts generated by the LSTM model.

3. Quantum Reinforcement Learning

- Problem Formulation: Frame the portfolio optimization challenge as a Hamiltonian to be solved using quantum algorithms.
- Algorithm Implementation: Adapt the Quantum Approximate Optimization Algorithm (QAOA) and integrate the Conditional Value at Risk (CVaR) framework to focus on optimizing the best outcomes.
- Employ IBM's Qiskit SDK to code and execute the quantum solution.
- Optimization: Iteratively run the QAOA algorithm to find the investment allocations that maximize returns while managing risk according to the defined objectives.

4. Comparative Analysis

- Performance Metrics: Calculate and compare expected returns, risk measures (volatility, Sharpe Ratio), and performance trajectories across all the implemented approaches (non-machine learning, classical machine learning, and quantum machine learning).
- Evaluation: Thoroughly analyze the results of each method, highlighting advantages, limitations, and the specific contexts where each approach might excel.

5. QRL Assessment:

- Critically discuss the potential benefits and current challenges of utilizing quantum machine learning for financial portfolio optimization.

6. Simulation and Testing:

- Test the model's predictions and strategies using quantum computing simulators.
- Evaluate and adjust the model's strategy based on its performance in simulations.

This framework outlines a systematic process for developing a cutting-edge model that aims to revolutionize portfolio optimization through the power of quantum computing and machine learning.

IMPLEMENTATION:

In the context of portfolio optimization using machine learning techniques, Principal Component Analysis (PCA) is employed to reduce the dimensionality of the dataset, making it more manageable for subsequent analysis. By extracting the principal components that retain the most useful information, PCA simplifies the dataset while preserving its key characteristics. This reduction in complexity enables faster training of machine learning models and enhances their interpretability.

Next, the Long Short-Term Memory (LSTM) model, a type of recurrent neural network, is utilized to learn from the simplified dataset. LSTM is particularly effective in capturing long-term dependencies in sequential data, making it suitable for analyzing stock market trends. The model's architecture includes a cell state and three gates (forget gate, input/update gate, and output gate) that regulate the flow of information through the network. By training the LSTM model on the PCA-processed data, we can predict future stock prices based on historical data.

After training the LSTM model, the portfolio optimization process is carried out using the SciPy library. SciPy offers various optimization algorithms, such as the minimize function, which can be used to find the optimal portfolio based on different metrics. Two commonly used metrics for portfolio optimization are the Sharpe ratio and volatility. The Sharpe ratio measures the risk-adjusted return of an investment, while volatility quantifies the variability of returns. By minimizing the negative Sharpe ratio and volatility, we can identify the portfolio that offers the best trade-off between risk and return.

The method involves using Principal Component Analysis (PCA) to simplify the dataset, then using a Long Short-Term Memory (LSTM) model to predict stock prices. The portfolio is optimized using the SciPy library, minimizing the negative Sharpe ratio and volatility. This process helps to create a balanced portfolio that considers both risk and return.

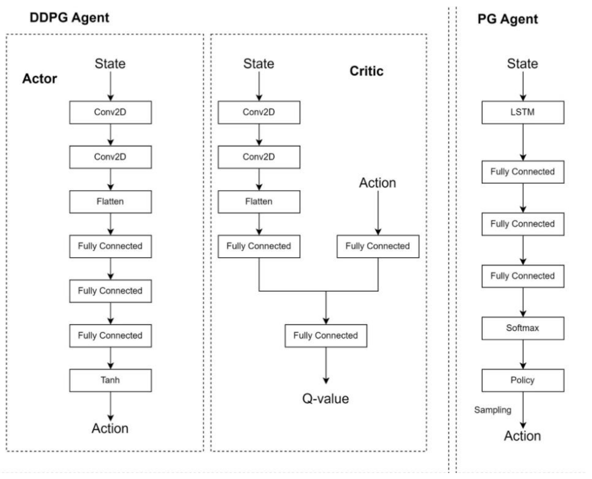
Classical Machine Learning

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):

$$CM = \frac{1}{n-1}((X - x')^T(X - x'))$$

where

x′ is the mean vector:

$$x' = (\tfrac{1}{n})\sum_{n}^{k=1}(x_i)$$

and the covariance between two features:

$$Cv_{jk} = \left(\frac{1}{n-1}\right)\sum_{n}^{i=1}(x_{ij} - x'_j)(x_{ik} - x'_k)$$

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.

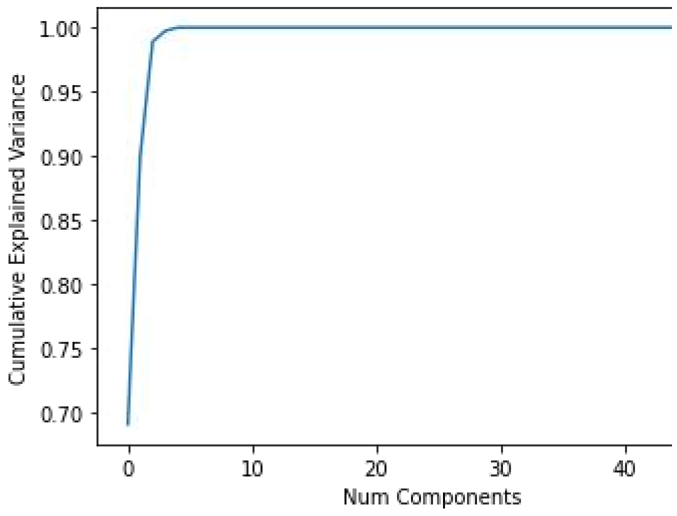


Fig. 1 - PCA on the portfolio

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.

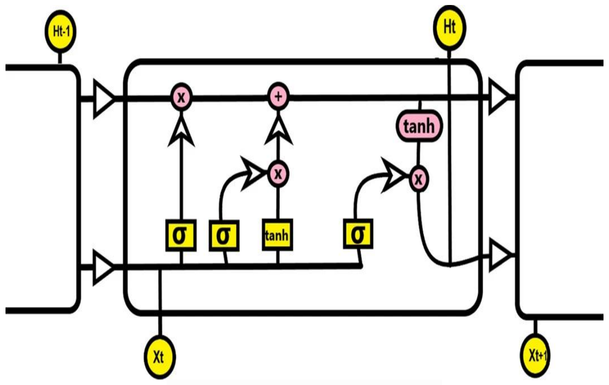
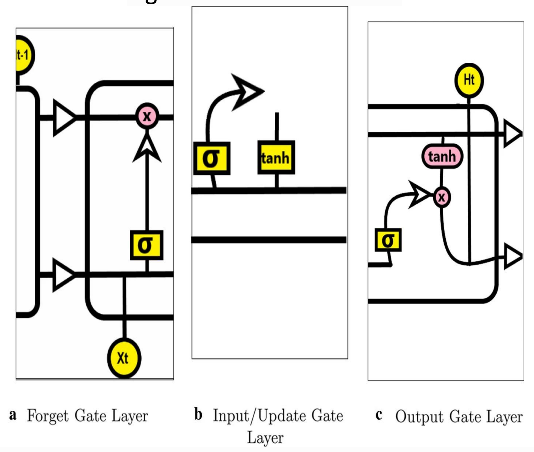


Fig. 1 - LSTM architecture



a Forget Gate Layer b Input/Update Gate Layer c Output Gate Layer

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
    - $\tilde{C}_t$ =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:

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in (0,1)^h$: forget gate's activation vector
- $i_t \in (0,1)^h$: input/update gate's activation vector
- $o_t \in (0,1)^h$: output gate's activation vector
- $h_t \in (0,1)^h$: hidden state vector also known as output vector of the LS
- $\tilde{c}_t \in (-1,1)^h$: cell input activation vector

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.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

n = number of data points

Y_i = observed values

$\hat{Y}_i$ = predicted values

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.

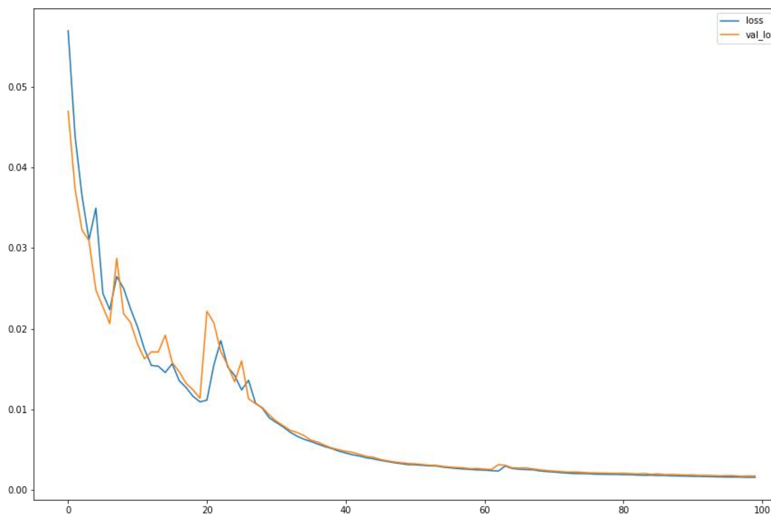


Fig. 2 - Plot of loss

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.

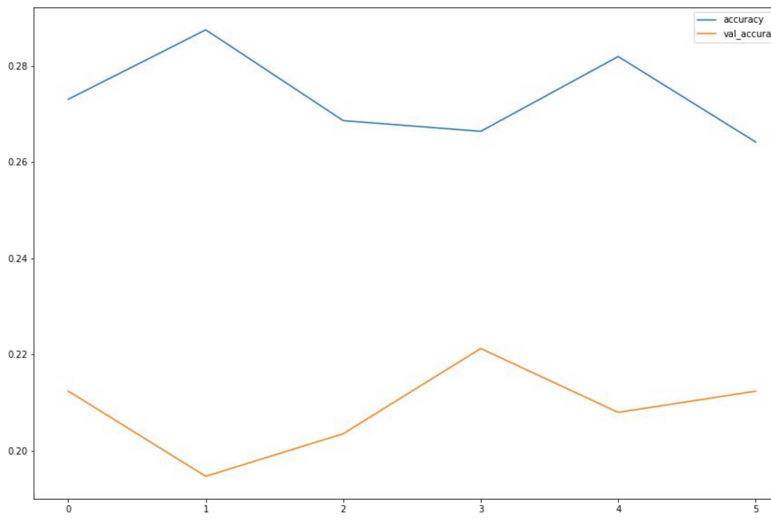
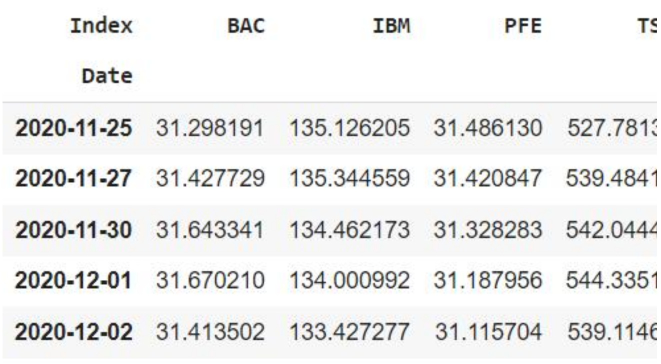


Fig. 3 - Plot of accuracy

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



Index	BAC	IBM	PFE	TS
Date				
2020-11-25	31.298191	135.126205	31.486130	527.781
2020-11-27	31.427729	135.344559	31.420847	539.4841
2020-11-30	31.643341	134.462173	31.328283	542.0444
2020-12-01	31.670210	134.000992	31.187956	544.3351
2020-12-02	31.413502	133.427277	31.115704	539.114

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.

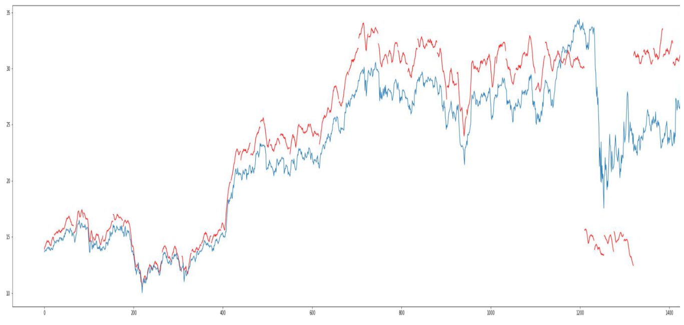


Fig. 5 - LSTM prediction for Exxon stock

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.

A. *9.2.3 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.

$$\frac{R_p - R_f}{\sigma_p}$$

R_p = return of portfolio

R_f = risk-free rate

σ_p = standard deviation of the portfolio's excess return

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.

$$\sigma_T = \sigma\sqrt{T}$$

σ_T = volatility over a time horizon

σ = standard deviation of returns

T = number of periods in a time horizon

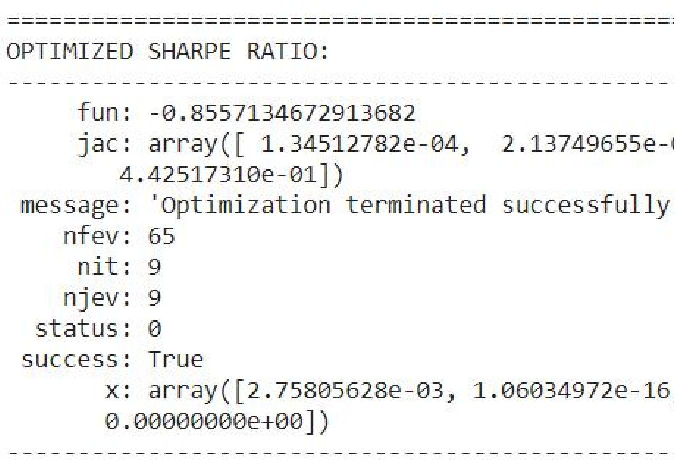
Implementation

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.

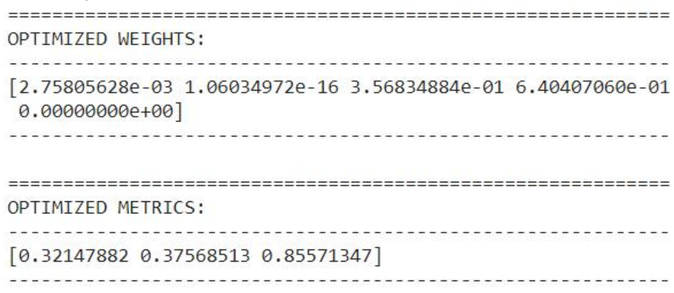
When we run the minimize function of the SciPy library and return the results, we obtain the following:



```
==================================================
OPTIMIZED SHARPE RATIO:
--------------------------------------------------
     fun: -0.8557134672913682
     jac: array([ 1.34512782e-04,  2.13749655e-
        4.42517310e-01])
 message: 'Optimization terminated successfully
    nfev: 65
     nit: 9
    njev: 9
  status: 0
 success: True
       x: array([2.75805628e-03, 1.06034972e-16
       0.00000000e+00])
--------------------------------------------------
```

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.



```
================================================================
OPTIMIZED WEIGHTS:
----------------------------------------------------------------
[2.75805628e-03 1.06034972e-16 3.56834884e-01 6.40407060e-01
 0.00000000e+00]
----------------------------------------------------------------

================================================================
OPTIMIZED METRICS:
----------------------------------------------------------------
[0.32147882 0.37568513 0.85571347]
----------------------------------------------------------------
```

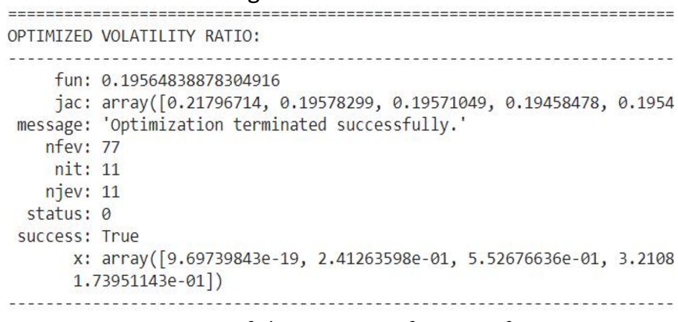
Fig. 2 - Optimized weights using sharpe ratio

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.

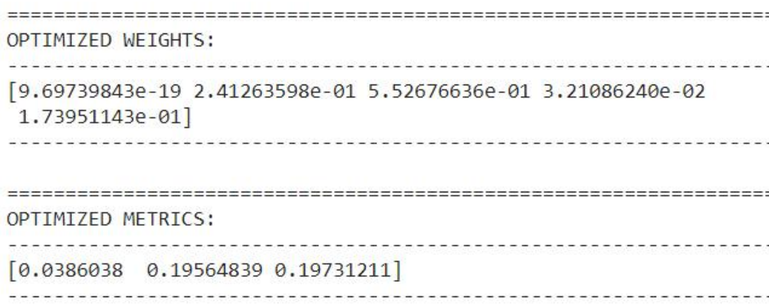
The aim is to compare the results obtained with the volatility as a metric. In that case, we put the risk at the center of the strategy to obtain a portfolio. The implementation is the same as for the sharpe ratio but now we use the minimize function on the calculated volatility of the stocks on the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



```
=====================================================================
OPTIMIZED VOLATILITY RATIO:
---------------------------------------------------------------------
     fun: 0.19564838878304916
     jac: array([0.21796714, 0.19578299, 0.19571049, 0.19458478, 0.1954
 message: 'Optimization terminated successfully.'
    nfev: 77
     nit: 11
    njev: 11
  status: 0
 success: True
       x: array([9.69739843e-19, 2.41263598e-01, 5.52676636e-01, 3.2108
       1.73951143e-01])
---------------------------------------------------------------------
```

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.



```
==========================================================================
OPTIMIZED WEIGHTS:
--------------------------------------------------------------------------
[9.69739843e-19 2.41263598e-01 5.52676636e-01 3.21086240e-02
 1.73951143e-01]
--------------------------------------------------------------------------

==========================================================================
OPTIMIZED METRICS:
--------------------------------------------------------------------------
[0.0386038  0.19564839 0.19731211]
--------------------------------------------------------------------------
```

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.

$$qx^T\Sigma x - \mu^T x$$

Respect the following condition: $1^T x = B$

With the following notation:

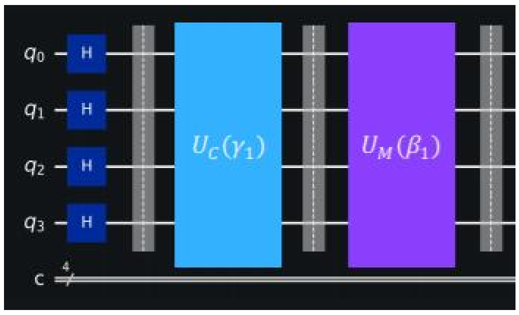
- $x \in \{0,1\}^n$ denoting the vector of binary decision variables which indicates the assets to pick $x[i] = 1$ or not $x[i] = 0$
- $\mu \in R^n$ the expected return for the assets
- $\Sigma \in R^{n \times n}$ the covariance between the assets
- $q > 0$ controls the risk appetite of the decision maker
- B 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.



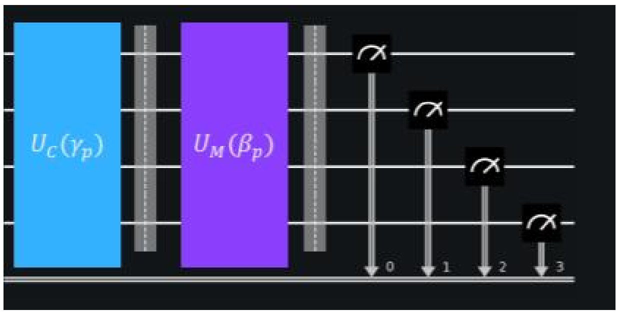


Fig. 1 - A possible implementation of the QAOA algorithm

The unitary operator $U_C(\gamma)$ refers to the cost layer and $U_M(\beta)$ 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.

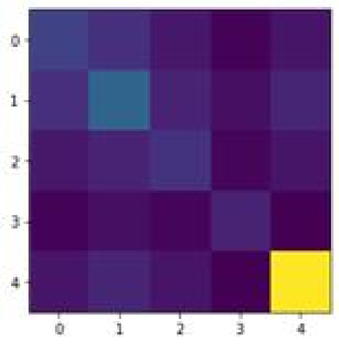


Fig. 2 - Covariance matrix

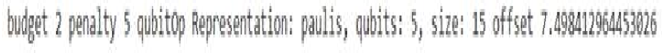
We can see this result confirms the portfolio we chose. Indeed, the covariance matrix shows no direct correlation between the different sets of data referring to the time evolution of each asset. This can offer a variety of portfolio optimization because the different assets do not influence one another.

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:

$$q = 0.5$$
$$B = num\_assets \; // \; 2$$
$$Penalty \; = \; num\_assets$$

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

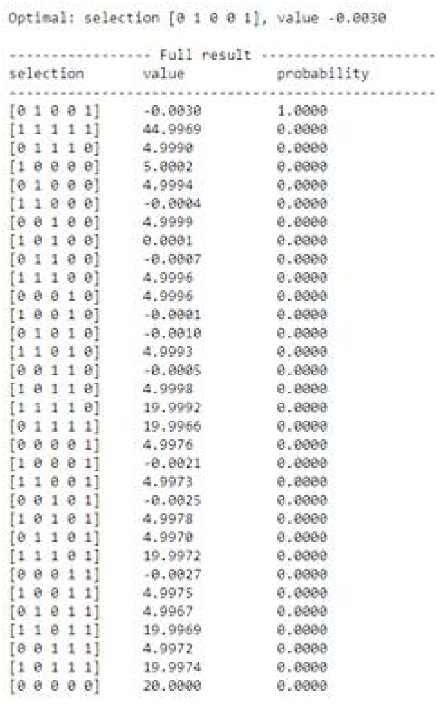
From that, we can use the get_operator() function from the portfolio class of Qiskit finance that takes the covariance terms as implementation of the data to obtain qubitOp and offset in order to run the quantum algorithm. We obtain the following configuration.



```
budget 2 penalty 5 qubitOp Representation: paulis, qubits: 5, size: 15 offset 7.498412964453026
```

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.



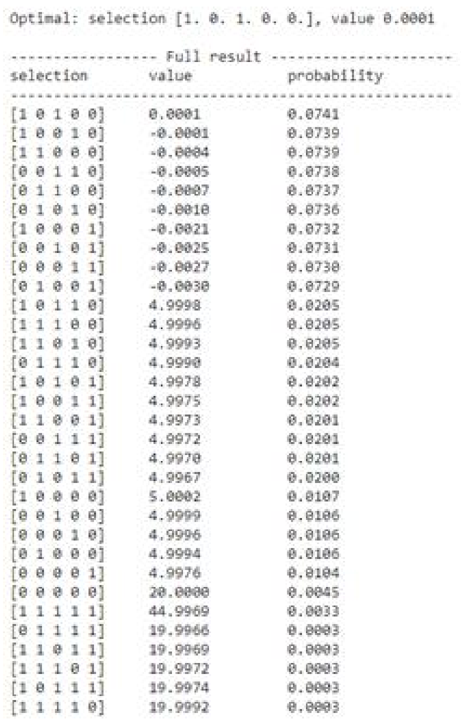
```
Optimal: selection [0 1 0 0 1], value -0.0030

----------------- Full result --------------------
selection       value           probability
---------------------------------------------------
[0 1 0 0 1]     -0.0030         1.0000
[1 1 1 1 1]     44.9969         0.0000
[0 1 1 1 0]     4.9990          0.0000
[1 0 0 0 0]     5.0002          0.0000
[0 1 0 0 0]     4.9994          0.0000
[1 1 0 0 0]     -0.0004         0.0000
[0 0 1 0 0]     4.9999          0.0000
[1 0 1 0 0]     0.0001          0.0000
[0 1 1 0 0]     -0.0007         0.0000
[1 1 1 0 0]     4.9996          0.0000
[0 0 0 1 0]     4.9996          0.0000
[1 0 0 1 0]     -0.0001         0.0000
[0 1 0 1 0]     -0.0010         0.0000
[1 1 0 1 0]     4.9993          0.0000
[0 0 1 1 0]     -0.0005         0.0000
[1 0 1 1 0]     4.9998          0.0000
[1 1 1 1 0]     19.9992         0.0000
[0 1 1 1 1]     19.9966         0.0000
[0 0 0 0 1]     4.9976          0.0000
[1 0 0 0 1]     -0.0021         0.0000
[1 1 0 0 1]     4.9973          0.0000
[0 0 1 0 1]     -0.0025         0.0000
[1 0 1 0 1]     4.9978          0.0000
[0 1 1 0 1]     4.9970          0.0000
[1 1 1 0 1]     19.9972         0.0000
[0 0 0 1 1]     -0.0027         0.0000
[1 0 0 1 1]     4.9975          0.0000
[0 1 0 1 1]     4.9967          0.0000
[1 1 0 1 1]     19.9969         0.0000
[0 0 1 1 1]     4.9972          0.0000
[1 0 1 1 1]     19.9974         0.0000
[0 0 0 0 0]     20.0000         0.0000
```

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.



```
Optimal: selection [1. 0. 1. 0. 0.], value 0.0001

----------------- Full result --------------------
selection       value           probability
---------------------------------------------------
[1 0 1 0 0]     0.0001          0.0741
[1 0 0 1 0]     -0.0001         0.0739
[1 1 0 0 0]     -0.0004         0.0739
[0 0 1 1 0]     -0.0005         0.0738
[0 1 1 0 0]     -0.0007         0.0737
[0 1 0 1 0]     -0.0010         0.0736
[1 0 0 0 1]     -0.0021         0.0732
[0 0 1 0 1]     -0.0025         0.0731
[0 0 0 1 1]     -0.0027         0.0730
[0 1 0 0 1]     -0.0030         0.0729
[1 0 1 1 0]     4.9998          0.0205
[1 1 1 0 0]     4.9996          0.0205
[1 1 0 1 0]     4.9993          0.0205
[0 1 1 1 0]     4.9990          0.0204
[1 0 1 0 1]     4.9978          0.0202
[1 0 0 1 1]     4.9975          0.0202
[1 1 0 0 1]     4.9973          0.0201
[0 0 1 1 1]     4.9972          0.0201
[0 1 1 0 1]     4.9970          0.0201
[0 1 0 1 1]     4.9967          0.0200
[1 0 0 0 0]     5.0002          0.0107
[0 0 1 0 0]     4.9999          0.0106
[0 0 0 1 0]     4.9996          0.0106
[0 1 0 0 0]     4.9994          0.0106
[0 0 0 0 1]     4.9976          0.0104
[0 0 0 0 0]     20.0000         0.0045
[1 1 1 1 1]     44.9969         0.0033
[0 1 1 1 1]     19.9966         0.0003
[1 1 0 1 1]     19.9969         0.0003
[1 1 1 0 1]     19.9972         0.0003
[1 0 1 1 1]     19.9974         0.0003
[1 1 1 1 0]     19.9992         0.0003
```

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.

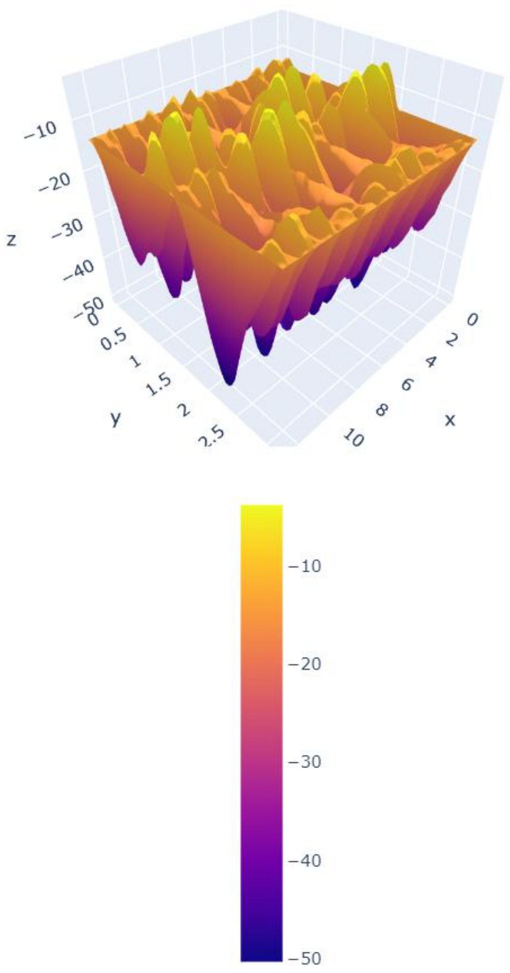
In conclusion, this approach using the QAOA algorithm defined in the Qiskit libraries obtains results that are difficult to interpret because of the low probabilities regarding the selections of portfolios. However, as seen in the study presented by Qiskit at this link, they manage to obtain the same results as the Numpy eigensolver which means the quantum solution has potential in the future.

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.

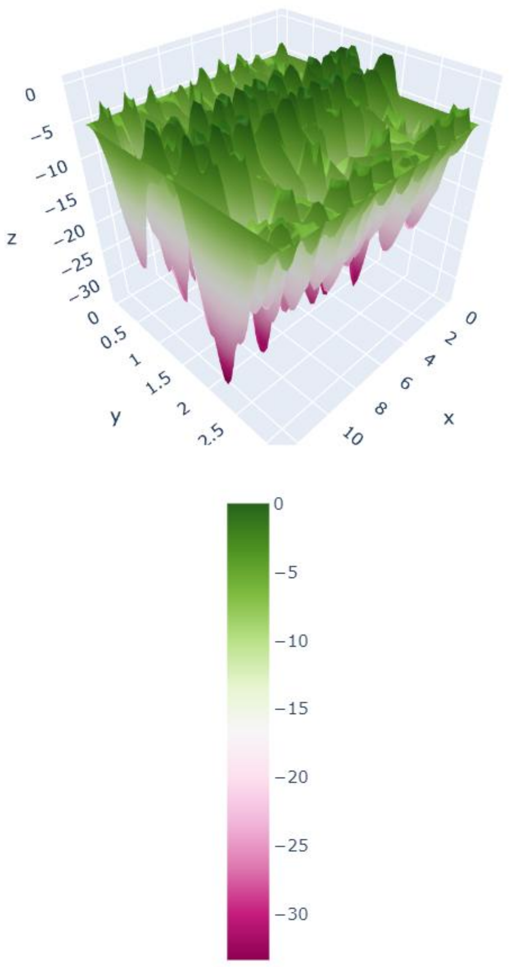
Depending on the value of α, the result of the CVaR which considers only the best measurements could be different. Thus, we can print the surface plot of the data which shows what landscape has been optimized to find the best solution in every case. We always compare to the classical eigensolver which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



```
{'beta': 1.2822827157509358, 'gamma': 11.027631355458048, 'profit': -3.77923
```

Fig. 1 - Results for α=1

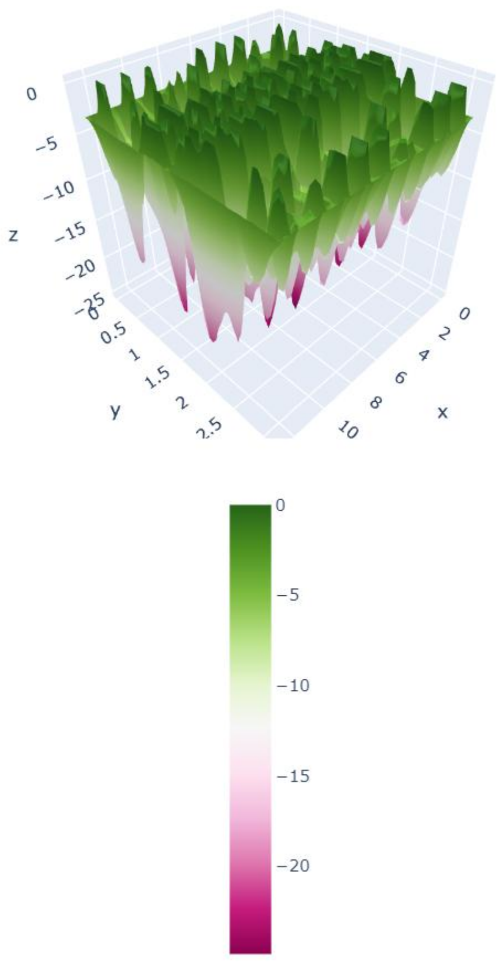
In this case we take the entire set of best measurements to compute the CVaR. We obtain in this case the best selection of Exxon and Tesla. When we compare it to the classical portfolio we have the right pick for Tesla but not for Bank of America. Moreover, when we refer to the previous results with 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.



```
{'beta': 1.2822827157509358, 'gamma': 3.077478517802246, 'profit'
```

Fig. 2 - Results for α=0.7

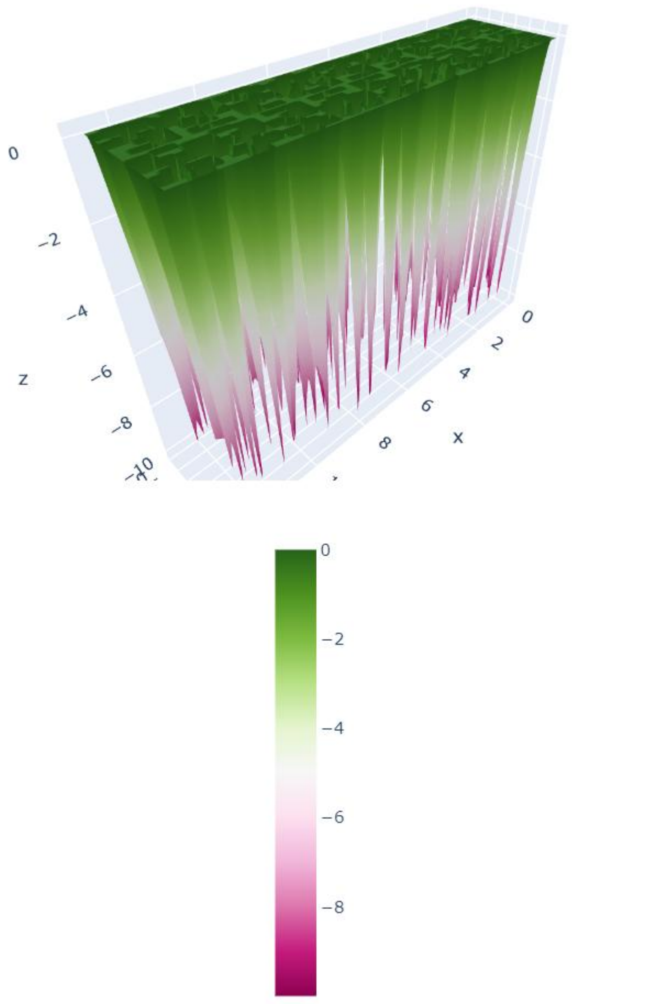
Now, we choose to remove some measurements in order to speed up the optimization process. We can see that removing those measurements causes a loss of precision compared to the classical eigensolver. Indeed, we find the same result as the first implementation of QAOA with cobyla optimizer. We obtain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



```
{'beta': 1.2822827157509358, 'gamma': 3.077478517802246, 'profit': 0.0013548
```

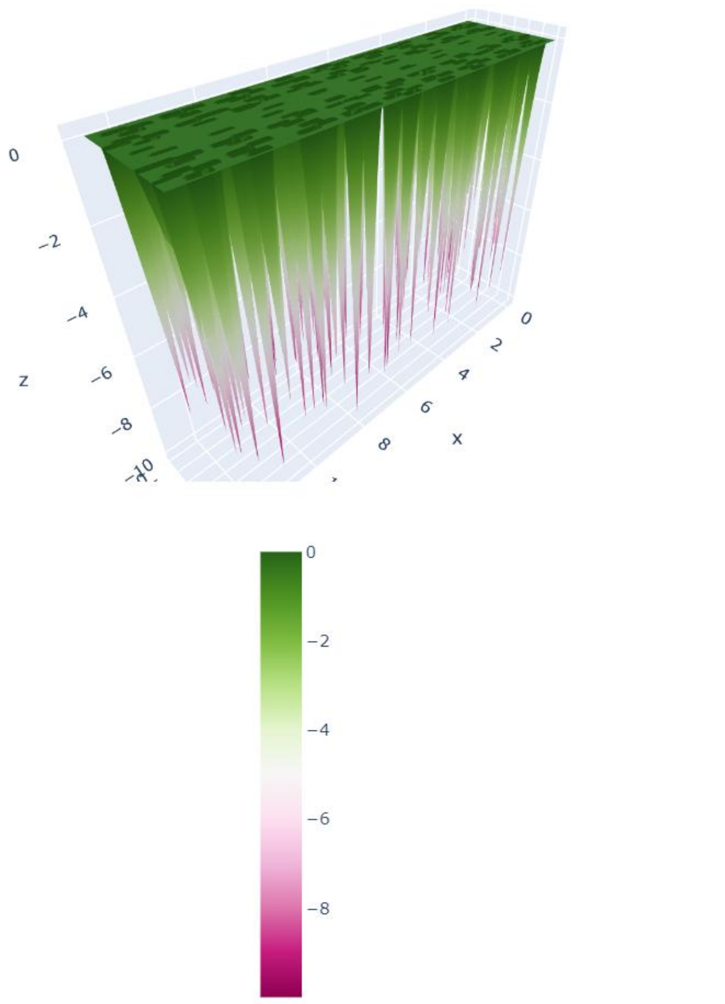
Fig. 3 - Results for α=0.5

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.



```
{'beta': 1.2822827157509358, 'gamma': 3.077478517802246, 'profit':
```

Fig. 4 - Results for α=0.2



```
{'beta': 0.1282282715750936, 'gamma': 4.872674319853556, 'profit': 0.002043986060328
```

Fig. 5 - Results for α=0.05

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.

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.

RESULTS:

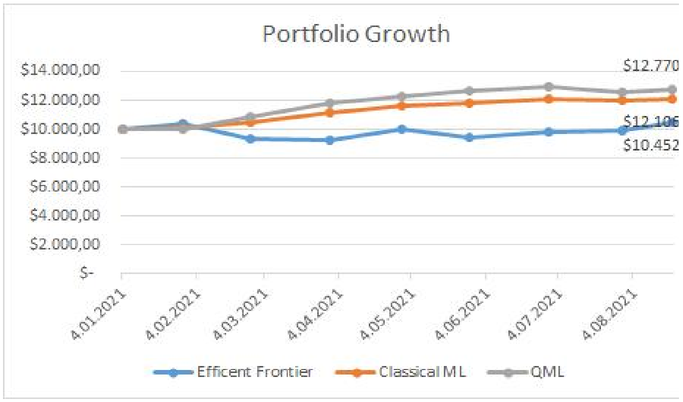


Fig. 1 - Comparison graph of different returns according to the approaches

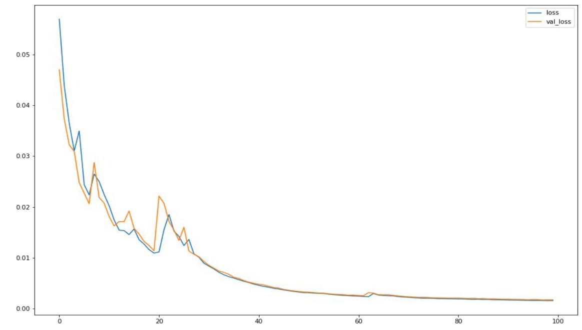


Fig. 2 - Plot of loss

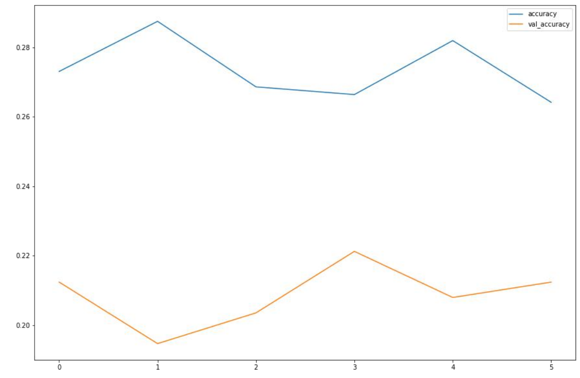


Fig. 3 - Plot of accuracy

CONCLUSION:

In conclusion up, this study explored a variety of methods for optimising financial portfolios, including machine learning, quantum algorithms, and classical models. The analysis offered insightful information about each method's effectiveness and suitability. When applied with common software such as Google Sheets or Excel, the Efficient Frontier Theory provided a fundamental framework for comprehending trade-offs between risk and reward. Although informative, this strategy might not be sophisticated enough for more dynamic market situations. Traditional machine learning methods, such Long Short-Term Memory (LSTM) models and Principal Component Analysis (PCA), demonstrated their capacity to transform intricate financial data into useful insights. Through portfolio optimisation with metrics like as the Sharpe ratio, these techniques proved to be useful in practical situations. A preview of the future of portfolio optimisation was provided by quantum machine learning, more especially by the Quantum Approximate Optimisation Algorithm (QAOA) with Conditional Value at Risk (CVaR). Though hardware restrictions exist at this time, the theoretical foundations provide promise for revolutionising this subject. When the methods were compared, the Quantum Reinforcement Learning strategy was the most successful, showing higher returns for 2021. Although the findings from the Classical Machine-Learning approach were similar, the special portfolio composition demonstrated how adaptable machine learning algorithms are. The Non-Machine Learning solution, on the other hand, produced less encouraging outcomes, highlighting the significance of utilising cutting-edge strategies in the fast-paced markets of today.

FUTURE SCOPE

Quantum reinforcement learning (QRL) for portfolio optimisation is a promising subject for future developments. The goal of research can be to create more reliable QRL algorithms that are especially suited to the complexities of financial markets. To provide more accurate portfolio optimisations in this setting, improvements in error correction and mitigation approaches are essential for improving the dependability of quantum calculations. For these techniques to be used practically, it will also be crucial to investigate how well they scale with bigger datasets and to take into account practical limitations as transaction costs and market liquidity. Future developments in QRL for portfolio optimisation can greatly improve financial decision-making procedures by tackling these issues and providing more practical and successful portfolio management techniques.

REFERENCES:

[1] Fösel, T., Niu, M. Y., Marquardt, F., & Li, L. (2021). Quantum circuit optimization with deep reinforcement learning1.

[2] Fösel, T., Niu, M. Y., Marquardt, F., & Li, L. (2021). Quantum circuit optimization with deep reinforcement learning1. 2. Buonaiuto, G., Gargiulo, F., De Pietro, G., Esposito, M., & Pota, M. (2023). Best practices for portfolio optimization by quantum computing2.

[3] Neumann, N. M. P., de Heer, P. B. U. L., & Phillipson, F. (2023). Quantum reinforcement learning: Comparing quantum annealing and gate-based quantum computing with classical deep reinforcement learning3.

[4] Kwak, Y., Yun, W. J., Jung, S., Kim, J.-K., & Kim, J. (2021). Introduction to Quantum Reinforcement Learning: Theory and PennyLane-based Implementation4.

[5] Meyer, N., Ufrecht, C., Periyasamy, M., Scherer, D. D., Plinge, A., & Mutschler, C. (2022). A Survey on Quantum Reinforcement Learning5.

[6] Dunjko, V., & Briegel, H. J. (2018). Machine learning & artificial intelligence in the quantum domain: a review of recent progress6. Application in Finance:

[7] Orús, R., Mugel, S., & Lizaso, E. (2019). Quantum computing for finance: Overview and prospects2.

[8] Egger, D. J., Gutiérrez, R. G., Mestre, J. C., & Woerner, S. (2020). Quantum computing for Finance: state of the art and future prospects7. Technical Implementations:

[9] Jerbi, S., Poulsen Nautrup, H., Trenkwalder, L. M., Briegel, H. J., & Dunjko, V. (2021). Quantum-enhanced reinforcement learning for control8.

[10] Paparo, G. D., Dunjko, V., Makmal, A., Martin-Delgado, M. A., & Briegel, H. J. (2014). Quantum speedup for active learning agents9. Experimental Realizations:

[11] Saggio, V., Dimić, A., Greganti, C., Riwar, R.-P., Roehsner, M.-C., & Walther, P. (2019). Experimental quantum speed-up in reinforcement learning agents10.

[12] Albarrán-Arriagada, F., Retamal, J. C., Solano, E., & Lamata, L. (2018). Reinforcement learning using quantum Boltzmann machines11.