Quantum Portfolio Optimization

Nigel K. Phillips

July 24, 2024

Abstract

This project leverages quantum computing to optimize financial portfolios using real-world data. The aim is to explore the application of quantum algorithms to enhance the efficiency and scalability of financial optimization tasks.

1 Introduction

The objective of this project is to leverage quantum computing for optimizing financial portfolios using real-world data. This project explores the application of quantum algorithms to enhance the efficiency and scalability of financial optimization tasks.

2 Methodology

2.1 Data Collection

- **Source:** Yahoo Finance API
- **Assets:** S&P 500, EuroStoxx 50, Nikkei 225, FTSE 100, Gold
- **Period:** January 2019 to May 2024

2.2 Data Preprocessing

- Filled missing values using forward fill.
- Calculated daily returns from adjusted closing prices.

2.3 Model Creation

The Binary Quadratic Model (BQM) was formulated using mean returns and the correlation matrix.

- **Variables:** Represent asset selection.
- **Objective:** Maximize returns with constraints modeled by the correlation matrix.

2.4 Quantum Optimization

- **Solver Used:** D-Wave Leap Hybrid Sampler
- **Optimization Process:**
  1. Define BQM for the assets.
  2. Use the quantum solver to find the optimal portfolio.
  3. Measure computation time.

3 Results

3.1 Optimal Portfolio

- **Selected Assets:** Nikkei 225 (Index 2)
- **Allocation:** {0: 0, 1: 0, 2: 1, 3: 0, 4: 0}

3.2 Quantum Computation Time

- **Total Time:** Approximately 3.54 seconds

3.3 Scalability Testing

- **Expanded Problem Sizes:** Tested with 10, 20, 30, 40, 50 assets.
- **Computation Times:**
  - 10 assets: 3.99 seconds
  - 20 assets: 3.60 seconds
  - 30 assets: 3.47 seconds
  - 40 assets: 3.67 seconds
  - 50 assets: 3.73 seconds

4 Analysis

4.1 Efficiency and Performance

The quantum solver demonstrated efficient computation times across different problem sizes.

4.2 Practical Implications

The model selected a practical portfolio allocation based on historical data, demonstrating its potential utility in real-world financial optimization.

4.3 Scalability

The scalability testing showed that the quantum approach could handle larger datasets efficiently. While the results are promising, further testing with even larger datasets and more complex models is needed to fully realize the quantum advantage.

4.4 Future Work

- Experiment with different solver parameters to optimize results.
- Incorporate additional financial indicators and constraints to refine the model.
- Collaborate with financial experts to validate and improve the practical applicability of the model.

5 Conclusion

This project successfully demonstrated the application of quantum computing in financial portfolio optimization, highlighting its potential for efficient and scalable solutions.

6 Appendix

6.1 Code Listings

Listing 1: fetch_data.py

```
import yfinance as yf
import pandas as pd

def fetch_data(tickers, start, end):
    data = yf.download(tickers, start=start, end=end)['Adj Close']
    data.ffill(inplace=True)
    return data

if __name__ == "__main__":
    tickers = ['^GSPC', '^STOXX50E', '^N225', '^FTSE', 'GLD']
    data = fetch_data(tickers, "2019-01-01", "2024-05-31")
    data.to_csv('data/historical_data.csv')
```

Listing 2: preprocess_data.py

```
import pandas as pd

def preprocess_data(file_path):
```

```
    data = pd.read_csv(file_path, index_col='Date')
    returns = data.pct_change().dropna()
    return returns

if __name__ == "__main__":
    returns = preprocess_data('data/historical_data.csv')
    returns.to_csv('data/returns.csv')
```

Listing 3: create_bqm.py

```
import dimod
import pandas as pd

def create_real_world_bqm(returns):
    num_assets = returns.shape[1]
    mean_returns = returns.mean().values
    correlation_matrix = returns.corr().values

    bqm = dimod.BinaryQuadraticModel('BINARY')
    for i in range(num_assets):
        for j in range(i + 1, num_assets):
            bqm.add_interaction(i, j, correlation_matrix[i, j])
    for i in range(num_assets):
        bqm.add_variable(i, -mean_returns[i])  # Negative for maximization in a

    return bqm

if __name__ == "__main__":
    returns = pd.read_csv('data/returns.csv', index_col='Date')
    bqm = create_real_world_bqm(returns)
```

Listing 4: solve_bqm.py

```
from dwave.system import LeapHybridSampler
import pandas as pd
import dimod

def solve_with_dwave(bqm):
    sampler = LeapHybridSampler()
    solution = sampler.sample(bqm)
    return solution

if __name__ == "__main__":
    returns = pd.read_csv('data/returns.csv', index_col='Date')
    bqm = create_real_world_bqm(returns)
    solution = solve_with_dwave(bqm)
    print("Optimal Portfolio:", solution.first.sample)
```

Listing 5: measure_time.py

```
from dwave.system import LeapHybridSampler
import pandas as pd
import time
import dimod

def measure_quantum_time(bqm):
    start_time = time.time()
    sampler = LeapHybridSampler()
    solution = sampler.sample(bqm)
    return time.time() - start_time, solution

if __name__ == "__main__":
    returns = pd.read_csv('data/returns.csv', index_col='Date')
    bqm = create_real_world_bqm(returns)
    q_time_real_world, solution_real_world = measure_quantum_time(bqm)
    print("Quantum Time with Real-World Data:", q_time_real_world)
    print("Solution with Real-World Data:", solution_real_world.first.sample)
```