Qubits
qubits.top

量子计算
——量子金融

Quantum Finance

网址：www.qubits.top

作者：Calvin Tang

邮箱：179209347@qq.com

介绍

本教程基于 IBM 的 **Qiskit，Qiskit[finance]** 编写。

本教程包含：

1. 分散投资数学模型
2. 量子算法 - 通过VQE求解问题
3. 代码实例

 * TODO：完善算法的详细解读

Qiskit：
https://qiskit.org/documentation/getting_started.html
Qiskit finance：
https://qiskit.org/documentation/finance/tutorials/index.html

Github & Gitee 代码地址：
https://github.com/mymagicpower/qubits/tree/main/quantum_qiskit_finance/02_portfolio_diversification.py
https://gitee.com/mymagicpower/qubits/tree/main/quantum_qiskit_finance/02_portfolio_diversification.py

Calvin，QQ: 179209347 Mail: 179209347@qq.com

Qiskit，Qiskit[finance] 配置和安装

Qubits
qubits.top

虚拟环境

```
# 创建虚拟环境
conda create -n ENV_NAME python=3.8.0
# 切换虚拟环境
conda activate ENV_NAME
# 退出虚拟环境
conda deactivate ENV_NAME
# 查看现有虚拟环境
conda env list
# 删除现有虚拟环境
conda remove -n ENV_NAME --all
```

安装 Qiskit

```
pip install qiskit
```

```
# install extra visualization support
# For zsh user (newer versions of macOS)
# pip install 'qiskit[visualization]'

pip install qiskit[visualization]
```

安装 Qiskit[finance]

```
# For zsh user (newer versions of macOS)
# pip install 'qiskit[finance]’

pip install qiskit[finance]
```

Calvin，QQ: 179209347 Mail: 179209347@qq.com

分散投资数学模型

数学模型将资产根据相似度聚簇，每个簇选择一个代表作为指数基金投资组合一部分。

$$\rho_{ij} = \text{similarity between stock } i \text{ and stock } j.$$

$$\rho_{ii} = 1, \rho_{ij} \leq 1 \text{ for } i \neq j$$

我们感兴趣的问题是f最大化：

$$f = \max_{x_{ij}, y_j} \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$$

约束条件：

$$\sum_{i=1}^{n} y_j = q,$$

- y_j= 1 如果选中，否则为0
- q 为基金中股票数量

$$\sum_{j=1}^{n} x_{ij} = 1, \text{ for } i = 1, \dots, n, \quad x_{ij} \leq y_j, \text{ for } i = 1, \dots, n; j = 1, \dots, n, \quad x_{jj} = y_j, \text{ for } j = 1, \dots, n,$$

$$x_{ij}, y_j \in \{0, 1\}, \text{ for } i = 1, \dots, n; j = 1, \dots, n.$$

x_{ij}代表股票 j 跟指数基金中的 i 最相似，如果 j 最相似则为1，否则为0.

https://qiskit.org/documentation/finance/tutorials/02_portfolio_diversification.html

Calvin，QQ: 179209347 Mail: 179209347@qq.com

A Hybrid Approach

Construct a binary polynomial optimization

From (M) one can construct a binary polynomial optimization with equality constraints only, by substituting the $x_{ij} \leq y_j$ inequality constraints with the equivalent equality constraints $x_{ij}(1 - y_j) = 0$. Then the problem becomes:

$$(BPO) \quad f = \max_{x_{ij}, y_j} \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$$

subject to the clustering constraint, the integral constraints, and the following modified consistency constraints:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \text{for } i = 1, \dots, n,$$
$$x_{ij}(1 - y_j) = 0, \quad \text{for } i = 1, \dots, n; \; j = 1, \dots, n,$$
$$x_{jj} = y_j, \quad \text{for } j = 1, \dots, n.$$

https://qiskit.org/documentation/finance/tutorials/02_portfolio_diversification.html

Calvin, QQ: 179209347 Mail: 179209347@qq.com

Construct the Ising Hamiltonian

We can now construct the Ising Hamiltonian (QUBO) by penalty methods (introducing a penalty coefficient A for each equality constraint) as

$$(IH) \quad H = \sum_{i=1}^{n}\sum_{j=1}^{n} \rho_{ij} x_{ij} + A\Big(\sum_{j=1}^{n} y_j - q\Big)^2 + \sum_{i=1}^{n} A\Big(\sum_{j=1}^{n} x_{ij} - 1\Big)^2 + \sum_{j=1}^{n} A(x_{jj} - y_j)^2 +$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} A\left(x_{ij}(1 - y_j)\right).$$

From Hamiltonian to Quadratic Programming (QP) formulation

Let us concatenate the decision variables in one vector

$$\mathbf{z} = [x_{11}, x_{12}, \ldots, x_{1n}, x_{22}, \ldots, x_{nn}, y_1, \ldots, y_n],$$

whose dimension is $\mathbf{z} \in \{0,1\}^N$, with $N = n(n+1)$ and denote the optimal solution with $\mathbf{z}^*$, and the optimal cost f^*.

In the vector $\mathbf{z}$, the Ising Hamiltonian elements can be rewritten as follows,

First term:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \rho_{ij} x_{ij} = [\rho_{11}, \rho_{12}, \ldots, \rho_{1n}, \rho_{22}, \ldots, \rho_{nn} | \mathbf{0}_n]\mathbf{z} =: \mathbf{c}_0^T \mathbf{z}$$

Second term:

$$A\Big(\sum_{j=1}^{n} y_j - q\Big)^2 = A\Big(\sum_{j=1}^{n} y_j\Big)^2 - 2Aq\sum_{j=1}^{n} y_j + Aq^2 = A\mathbf{z}^T \left[\frac{\mathbf{0}_{n^2}}{\mathbf{1}_n}\right] [\mathbf{0}_{n^2} | \mathbf{1}_n]\,\mathbf{z}$$
$$- 2Aq[\mathbf{0}_{n^2} | \mathbf{1}_n]\mathbf{z} + Aq^2 =: \mathbf{z}^T \mathbf{Q}_0 \mathbf{z} + \mathbf{c}_1^T \mathbf{z} + r_0$$

From Hamiltonian to Quadratic Programming (QP) formulation

Third term:

$$\sum_{i=1}^{n} A\Big(\sum_{j=1}^{n} x_{ij} - 1\Big)^2 = A\sum_{i=1}^{n}\Big(\sum_{j=1}^{n} x_{ij}\Big)^2 - 2A\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} + nA =$$

which is equivalent to:

$$= A\mathbf{z}^T\left(\sum_{i=1}^{n}\begin{bmatrix}\mathbf{0}_{n(i-1)}\\ \mathbf{1}_n\\ \mathbf{0}_{n(n-i)}\\ \hline \mathbf{0}_n\end{bmatrix}\begin{bmatrix}\mathbf{0}_{n(i-1)} & \mathbf{1}_n & \mathbf{0}_{n(n-i)} & |\mathbf{0}_n\end{bmatrix}\right)\mathbf{z} - 2A[\mathbf{1}_{n^2}|\mathbf{0}_n]\mathbf{z} + nA$$

$$=: \mathbf{z}^T\mathbf{Q}_1\mathbf{z} + \mathbf{c}_2^T\mathbf{z} + r_1$$

From Hamiltonian to Quadratic Programming (QP) formulation

Fourth term:

$$A\sum_{j=1}^{n}(x_{jj}-y_j)^2$$

$$= A\mathbf{z}^T\left(\sum_{j=0}^{n-1}\begin{bmatrix}\mathbf{0}_{nj+j}\\ 1\\ \mathbf{0}_{n^2-(nj+j+1)}\\ \hline \mathbf{0}_j\\ -1\\ \mathbf{0}_{n-j-1}\end{bmatrix}\begin{bmatrix}\mathbf{0}_{nj+j} & 1 & \mathbf{0}_{n^2-(nj+j+1)} & |\mathbf{0}_j & -1 & \mathbf{0}_{n-j-1}\end{bmatrix}\right)\mathbf{z} = A\mathbf{z}^T\mathbf{Q}_2\mathbf{z}$$

From Hamiltonian to Quadratic Programming (QP) formulation

Fifth term:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} A\left(x_{ij}(1-y_j)\right) = A[\mathbf{1}_{n^2}|\mathbf{0}_n]\mathbf{z} + A\mathbf{z}^T\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\left[\begin{array}{c|c} \mathbf{0}_{n^2\times n^2} & -1/2_{(ij,j)} \\ \hline -1/2_{(j,ij)} & \mathbf{0}_n \end{array}\right]\right)\mathbf{z}$$

$$=: \mathbf{z}^T\mathbf{Q}_3\mathbf{z} + \mathbf{c}_3^T\mathbf{z}$$

Therefore, the formulation becomes,

$$(IH-QP) \qquad \max_{\mathbf{z}\in\{0,1\}^{n(n+1)}} \mathbf{z}^T(\mathbf{Q}_0+\mathbf{Q}_1+\mathbf{Q}_2+\mathbf{Q}_3)\mathbf{z} + (\mathbf{c}_0+\mathbf{c}_1+\mathbf{c}_2+\mathbf{c}_3)^T\mathbf{z} + r_0 + r_1 + r_2$$

which can be passed to the variational quantum eigensolver.

Quantum Computing with IBM Q

For the quantum solution, we use Qiskit. We first define a class QuantumOptimizer that encodes the quantum approach to solve the problem and then we instantiate it and solve it. We define the following methods inside the class:

- `exact_solution` : to make sure that the Ising Hamiltonian is correctly encoded in the Z basis, we can compute its eigendecomposition classically, i.e., considering a symmetric matrix of dimension $2^N \times 2^N$. For the problem at hand $n = 3$, that is $N = 12$, seems to be the limit for many laptops;
- `vqe_solution` : solves the problem (M) via the variational quantum eigensolver (VQE);
- `qaoa_solution` : solves the problem (M) via a Quantum Approximate Optimization Algorithm (QAOA).

步骤1 & 步骤2

步骤1： 初始化参数

```
quantum_optimizer = QuantumOptimizer(rho, n, q)
```

- the similarity matrix rho;
- the number of assets and clusters n and q;

步骤2： 二值化编码

```
try:
    import cplex

    # warnings.filterwarnings('ignore')
    quantum_solution, quantum_cost = quantum_optimizer.exact_solution()
    print(quantum_solution, quantum_cost)
    classical_solution, classical_cost = classical_optimizer.cplex_solution()
    print(classical_solution, classical_cost)
    if np.abs(quantum_cost - classical_cost) < 0.01:
        print("Binary formulation is correct")
    else:
        print("Error in the formulation of the Hamiltonian")
except Exception as ex:
    print(ex)
```

步骤3： 在Z轴上，将问题编码为伊辛哈密尔顿量

```
ground_state, ground_level = quantum_optimizer.exact_solution()
print(ground_state)

try:
    if np.abs(ground_level - classical_cost) < 0.01:
        print("Ising Hamiltonian in Z basis is correct")
    else:
        print("Error in the Ising Hamiltonian formulation")
except Exception as ex:
    print(ex)
```

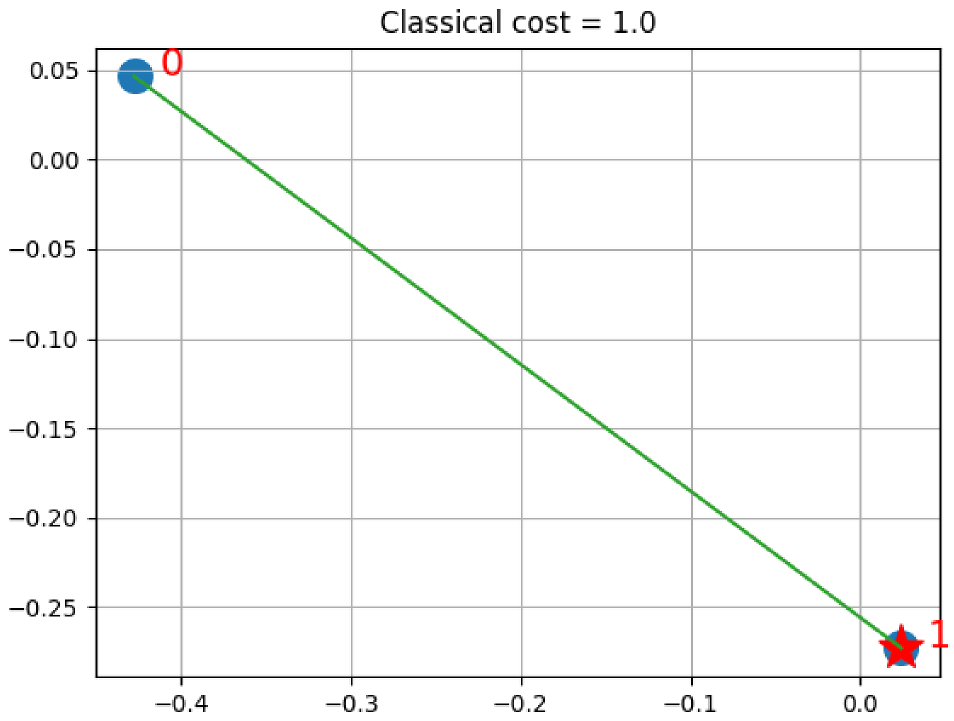
步骤4： 通过VQE求解问题

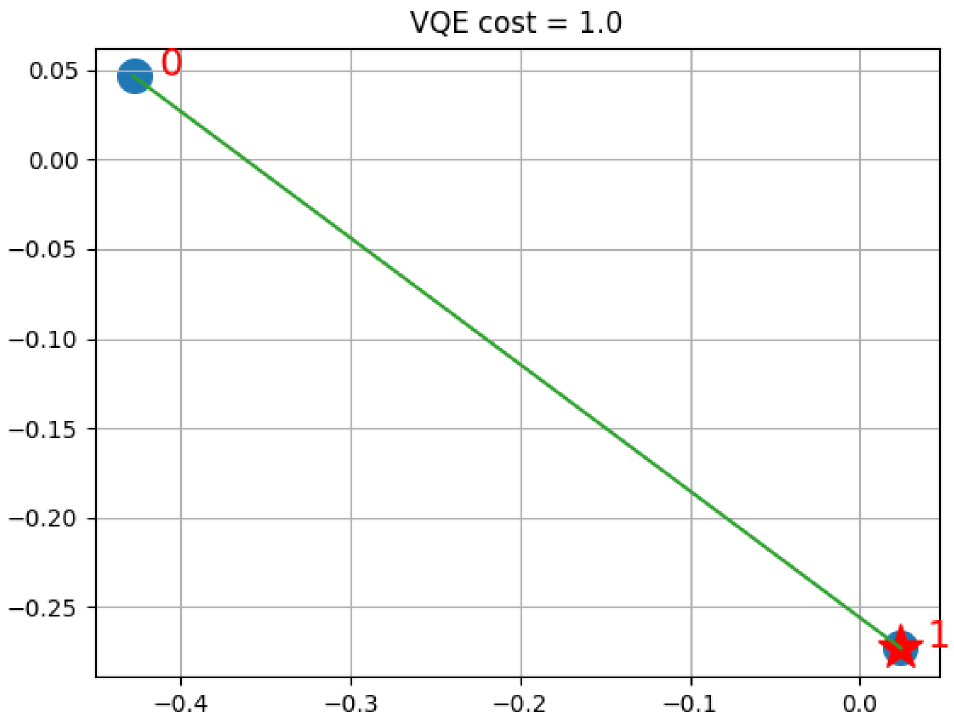
```
vqe_state, vqe_level = quantum_optimizer.vqe_solution()
print(vqe_state, vqe_level)

try:
    if np.linalg.norm(ground_state - vqe_state) < 0.01:
        print("VQE produces the same solution as the exact eigensolver.")
    else:
        print(
            "VQE does not produce the same solution as the exact eigensolver, but that
is to be expected."
        )
except Exception as ex:
    print(ex)
```

Solve the problem via VQE. Notice that depending on the number of qubits, this can take a while: for 6 qubits it takes 15 minutes on a 2015 Macbook Pro, for 12 qubits it takes more than 12 hours. For longer runs, logging may be useful to observe the workings; otherwise, you just have to wait until the solution is printed.

步骤5： 可视化


Classical cost = 1.0
0
1
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.4
−0.3
−0.2
−0.1
0.0


VQE cost = 1.0
0
1
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.4
−0.3
−0.2
−0.1
0.0

参考

[1] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser, Location of bank accounts to optimize float: an analytical study of exact and approximate algorithms, Management Science, vol. 23(8), 1997

[2] E. Farhi, J. Goldstone, S. Gutmann e-print arXiv 1411.4028, 2014
https://arxiv.org/abs/1411.4028

[3] G. Cornuejols and R. Tutuncu, Optimization methods in finance, 2006
http://web.math.ku.dk/~rolf/CT_FinOpt.pdf

[4] DJ. Berndt and J. Clifford, Using dynamic time warping to find patterns in time series. In KDD workshop 1994 (Vol. 10, No. 16, pp. 359-370).

[5] Max-Cut and Traveling Salesman Problem
https://github.com/Qiskit/qiskit-optimization/blob/main/docs/tutorials/06_examples_max_cut_and_tsp.ipynb

Qubits
qubits.top

Thank

You