Qiskit Document Tutorials 勉強会
Finance

Daiki Murata

本日はこちらの２＋１トピックを取り上げます。

1. Portfolio Optimization

2. Portfolio Diversification

✓ Loading and Processing Stock-Market Time-Series Data

Portfolio Optimizationとは

複数のリスク資産へ投資をする際に、リスクを抑えつつリターンを得るための最適な資産配分(ポートフォリオ)を求めることです。

理想

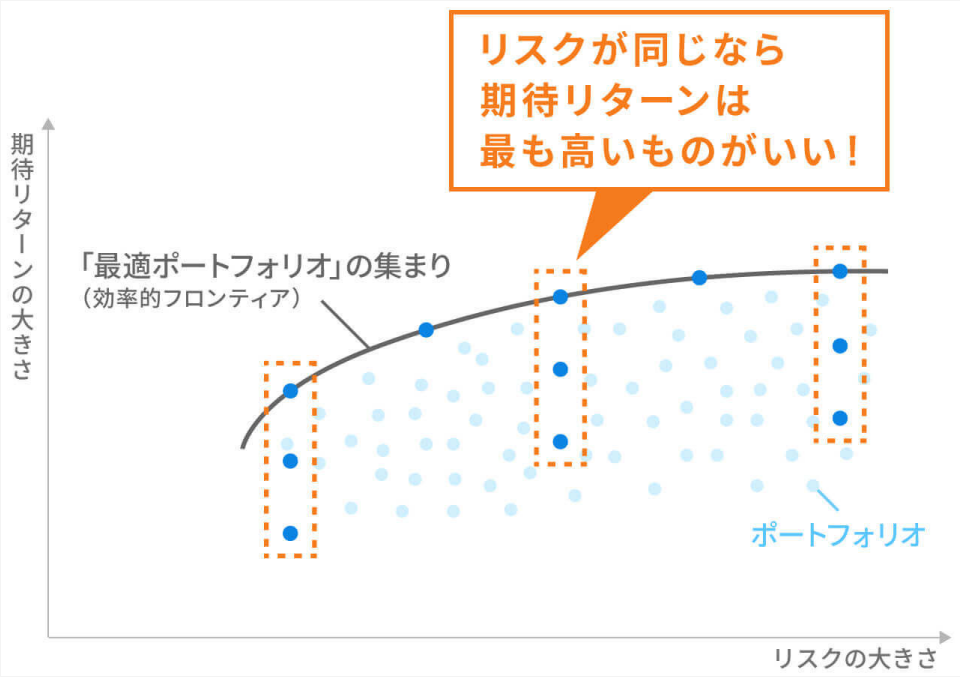
リスク最小・リターン最大

このような解は得られない
(ハイリスク・ハイリターン)

現実

リスクとリターンのトレードオフを考慮

理論的な枠組みとして「現代ポートフォリオ理論」が提唱されていますが、古典コンピューターでは効率的な解放は存在せず、量子コンピューターによる試みが期待されます。



出典：https://www.wealthnavi.com/contents/column/41/

最適なポートフォリオの選択は、リスク(資産の収益分散)とリターンのトレードオフを最適化する問題として定式化できます。

$$\text{minimize} \quad q\boldsymbol{x}^T\Sigma\boldsymbol{x} - \boldsymbol{\mu}^T\boldsymbol{x}$$
$$\text{subject to } \mathbf{1}^{\mathbf{T}}\boldsymbol{x} = B$$

$\boldsymbol{x} \in \{0,1\}^n$	ポートフォリオの配分
$\Sigma \in \mathbb{R}^{n\times n}$	アセットの共分散行列
$\boldsymbol{\mu} \in \mathbb{R}^n$	アセット毎の期待リターン
$q > 0$	リスク許容度
B	ポートフォリオの銘柄数

（補足）

期待リターン(利回り)は日次の変化率y_tの期間Tでの平均から求めます。

$$\mu = \frac{1}{T}\sum_{t=1}^{T} y_t = \frac{1}{T}\sum_{t=1}^{T} \frac{P_t - P_{t-1}}{P_{t-1}}$$

共分散行列

$$\Sigma = \frac{1}{T}\sum_{t=1}^{T} (\boldsymbol{y_t} - \boldsymbol{\mu})(\boldsymbol{y_t} - \boldsymbol{\mu})^T$$

それではで$4(=n)$銘柄から$2(=B)$銘柄の選択を実装をします。
まずRandomDataProviderを利用して疑似データを生成します。

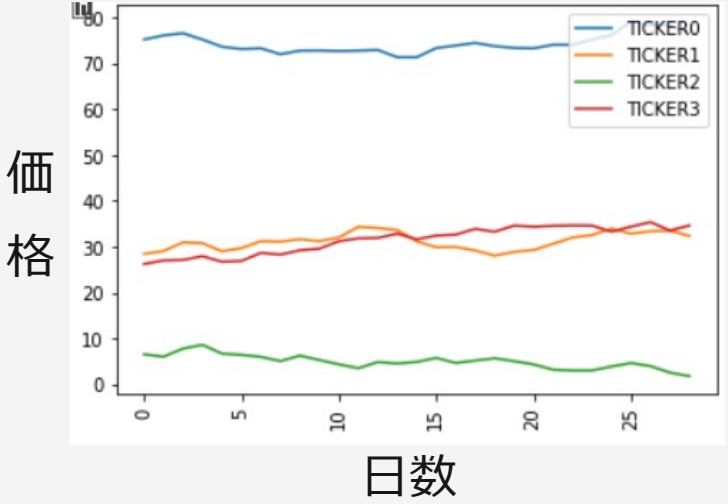
minimize $q\boldsymbol{x}^T\Sigma\boldsymbol{x} - \boldsymbol{\mu}^T\boldsymbol{x}$
subject to $\mathbf{1}^T\boldsymbol{x} = B$

```
# set number of assets (= number of qubits)
num_assets = 4

# Generate expected return and covariance matrix from (random) time-series
stocks = [("TICKER%s" % i) for i in range(num_assets)]
data = RandomDataProvider(tickers=stocks,
                start=datetime.datetime(2016,1,1),
                end=datetime.datetime(2016,1,30))
data.run()
mu = data.get_period_return_mean_vector()
sigma = data.get_period_return_covariance_matrix()
```

←μ:期待リターン

←Σ:共分散行列

2016/1/1 ~ 2016/1/30



(補足)
VQEを使って最小値を求めることを念頭に、問題を量子コンピューターに渡せるように変換していきます。

$$\text{minimize} \quad q\boldsymbol{x}^T\Sigma\boldsymbol{x} - \boldsymbol{\mu}^T\boldsymbol{x}$$
$$\text{subject to } \mathbf{1}^{\mathbf{T}}\boldsymbol{x} = B$$

ペナルティ項を追加して制約を取り除く

$$\text{minimize} \quad q\boldsymbol{x}^T\Sigma\boldsymbol{x} - \boldsymbol{\mu}^T\boldsymbol{x} + A\left(\mathbf{1}^{\mathbf{T}}\boldsymbol{x} - B\right)^2$$

値を演算子(量子ゲート)に対応させる
$x_i = (I - Z)/2$

ハミルトニアン(Pauli演算子で構成)

ただし、この変換はqiskit.financeのメソッドで一発で求まります。

VQEとQAOAを使って最適化してみます。
VQEは少ない測定回数で最適解の近似値を、QAOAは一定の測定回数は必要なものの最適解を求めることができそうです。

銘柄iを組み入れるかどうか

ハイパーパラメーター$q = 0.5, A = n$

Classical

```
Optimal: selection [0 1 0 1], value -0.0064

----------------- Full result ---------------------
selection       value           probability
-----------------------------------------------------
 [0 1 0 1]      -0.0064         1.0000
 [1 1 1 1]      15.9973         0.0000
 [0 1 1 1]      3.9970          0.0000
 [1 0 1 1]      3.9998          0.0000
 [0 0 1 1]      -0.0006         0.0000
 [1 1 0 1]      3.9939          0.0000
 [1 0 0 1]      -0.0037         0.0000
 [0 0 0 1]      3.9960          0.0000
 [1 1 1 0]      4.0011          0.0000
 [0 1 1 0]      0.0007          0.0000
 [1 0 1 0]      0.0037          0.0000
 [0 0 1 0]      4.0033          0.0000
 [1 1 0 0]      -0.0022         0.0000
 [0 1 0 0]      3.9975          0.0000
 [1 0 0 0]      4.0004          0.0000
 [0 0 0 0]      16.0000         0.0000
```

VQE

```
Optimal: selection [1. 0. 0. 1.], value -0.0037

----------------- Full result ---------------------
selection       value           probability
-----------------------------------------------------
 [1 0 0 1]      -0.0037         0.9683
 [1 1 0 0]      -0.0022         0.0167
 [0 1 0 1]      -0.0064         0.0150
 [0 0 1 1]      -0.0006         0.0000
 [0 1 1 1]      3.9970          0.0000
 [0 1 1 0]      0.0007          0.0000
 [1 0 1 1]      3.9998          0.0000
 [1 1 1 1]      15.9973         0.0000
 [0 0 1 0]      4.0033          0.0000
 [0 1 0 0]      3.9975          0.0000
 [1 1 1 0]      4.0011          0.0000
 [0 0 0 0]      16.0000         0.0000
 [1 0 1 0]      0.0037          0.0000
 [1 1 0 1]      3.9939          0.0000
 [0 0 0 1]      3.9960          0.0000
 [1 0 0 0]      4.0004          0.0000
```

QAOA

```
Optimal: selection [0. 1. 0. 1.], value -0.0064

----------------- Full result ---------------------
selection       value           probability
-----------------------------------------------------
 [0 1 0 1]      -0.0064         0.1671
 [1 0 0 1]      -0.0037         0.1669
 [1 1 0 0]      -0.0022         0.1667
 [0 0 1 1]      -0.0006         0.1666
 [0 1 1 0]      0.0007          0.1665
 [1 0 1 0]      0.0037          0.1662
 [0 0 1 0]      4.0033          0.0000
 [1 0 0 0]      4.0004          0.0000
 [0 0 0 0]      16.0000         0.0000
 [1 1 1 0]      4.0011          0.0000
 [1 0 1 1]      3.9998          0.0000
 [0 1 0 0]      3.9975          0.0000
 [0 1 1 1]      3.9970          0.0000
 [0 0 0 1]      3.9960          0.0000
 [1 1 0 1]      3.9939          0.0000
 [1 1 1 1]      15.9973         0.0000
```

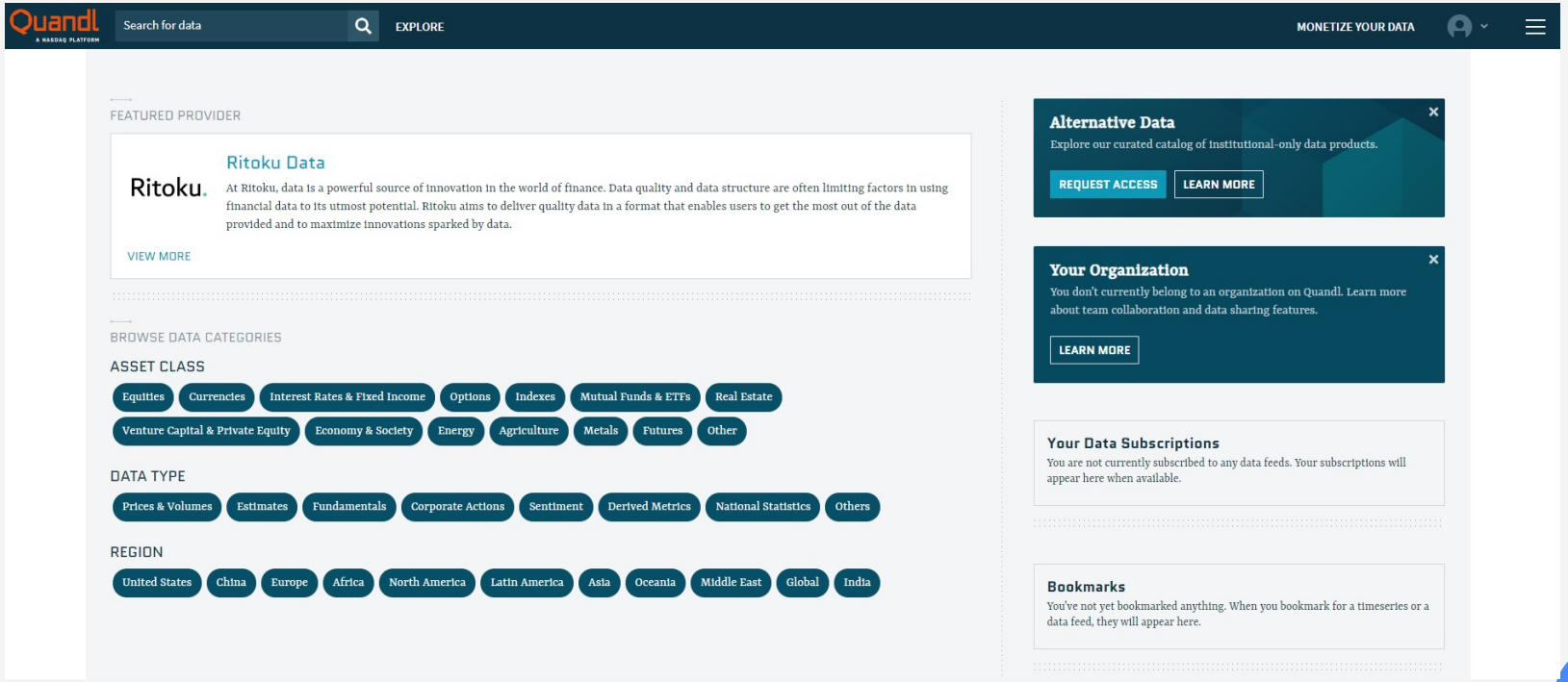
使い方は分かったので実際のデータでやってみます。

1. Portfolio Optimization

2. Portfolio Diversification

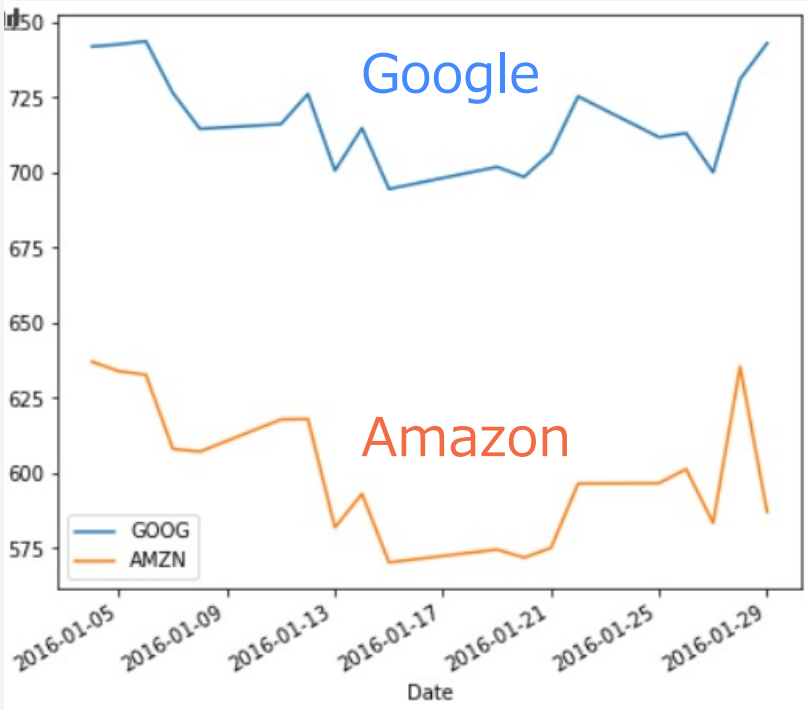
✓ **Loading and Processing Stock-Market Time-Series Data**

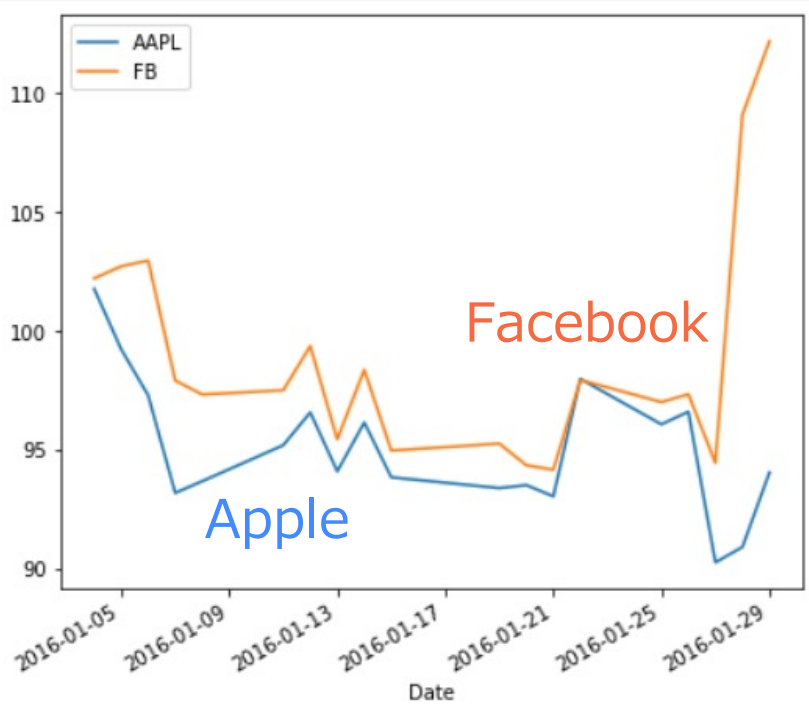
これまでは疑似データを利用してきましたが、QuandlのAPIから GAFAの株価終値を取得して先ほどの計算をやってみましょう。



https://www.quandl.com/

QuandlのAPI tokenをセットして
2016/1/1~2016/1/30のGAFAの株価終値を取得します。
1カ月分の時系列データを使ってポートフォリオを最適化します。





VQEは最適解が求まっていますが、QAOAはかなり怪しい結果が返ってきています。アルゴリズムの選択が重要そうです。

Classical

```
Optimal: selection [0 1 0 0], value 7.935526345268836

----------------- Full result ----------------------
selection       value           probability
------------------------------------------------------
 [0 1 0 0]      7.9355          1.0000
 [1 1 1 1]      917.3854                0.0000
 [0 1 1 1]      378.4472                0.0000
 [1 0 1 1]      841.3461                0.0000
 [0 0 1 1]      335.8303                0.0000
 [1 1 0 1]      761.1137                0.0000
 [0 1 0 1]      296.5424                0.0000
 [1 0 0 1]      695.3195                0.0000
 [0 0 0 1]      264.1707                0.0000
 [1 1 1 0]      248.9854                0.0000
 [0 1 1 0]      18.3955         0.0000
 [1 0 1 0]      213.3822                0.0000
 [0 0 1 0]      16.2149         0.0000
 [1 1 0 0]      164.1584                0.0000
 [1 0 0 0]      138.8003                0.0000
 [0 0 0 0]      16.0000         0.0000
```

Appleのみ

VQE

```
Optimal: selection [0.0000 1.0000 0.0000 0.0000], v

----------------- Full result ----------------------
selection       value           probability
------------------------------------------------------
 [0 1 0 0]      7.9355          0.9205
 [0 0 1 0]      16.2149         0.0371
 [0 1 1 0]      18.3955         0.0227
 [0 0 0 0]      16.0000         0.0196
 [0 0 1 1]      335.8303                0.0001
 [0 1 1 1]      378.4472                0.0000
 [0 1 0 1]      296.5424                0.0000
 [1 1 0 0]      164.1584                0.0000
 [0 0 0 1]      264.1707                0.0000
 [1 0 0 0]      138.8003                0.0000
 [1 0 1 0]      213.3822                0.0000
 [1 1 1 0]      248.9854                0.0000
 [1 0 0 1]      695.3195                0.0000
 [1 1 0 1]      761.1137                0.0000
 [1 1 1 1]      917.3854                0.0000
 [1 0 1 1]      841.3461                0.0000
```

Appleのみ

QAOA

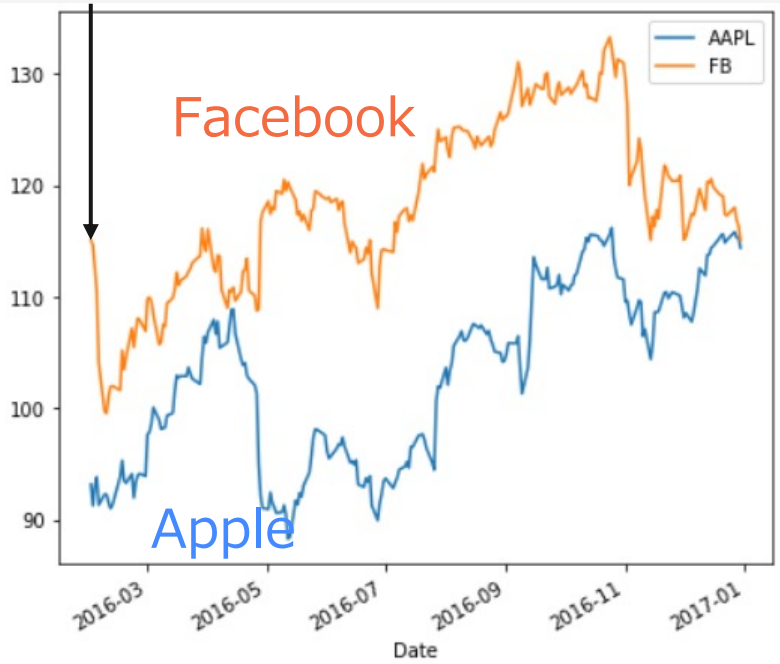
```
Optimal: selection [1.0000 1.0000 1.0000 0.0000], v

----------------- Full result ----------------------
selection       value           probability
------------------------------------------------------
 [1 1 1 0]      248.9854                0.1658
 [0 1 0 1]      296.5424                0.1551
 [0 0 0 1]      264.1707                0.1286
 [1 0 0 0]      138.8003                0.1055
 [0 1 0 0]      7.9355          0.0996
 [0 0 0 0]      16.0000         0.0638
 [1 1 0 0]      164.1584                0.0613
 [0 0 1 0]      16.2149         0.0479
 [0 1 1 1]      378.4472                0.0462
 [0 1 1 0]      18.3955         0.0382
 [0 0 1 1]      335.8303                0.0331
 [1 0 1 0]      213.3822                0.0319
 [1 0 1 1]      841.3461                0.0155
 [1 0 0 1]      695.3195                0.0033
 [1 1 0 1]      761.1137                0.0024
 [1 1 1 1]      917.3854                0.0017
```

Google
Apple
Facebook

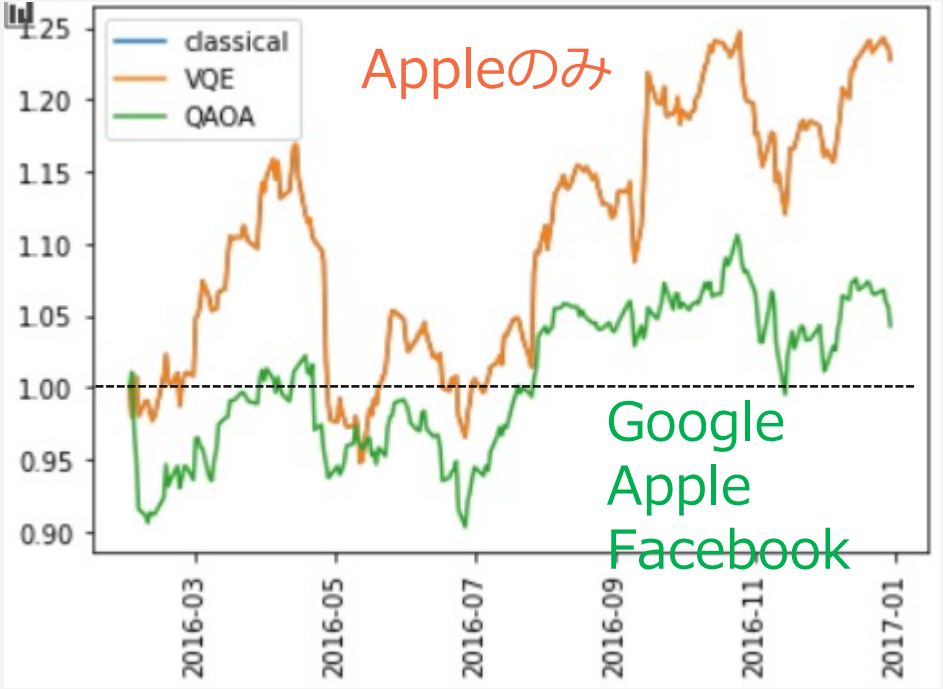
手に入れた結果を使ってバックテストをしてみます。
2016/1/31に結果に基づいて株式を購入し、2016/12/31までの
利回りを計算してみます。

ここで購入

ここで購入

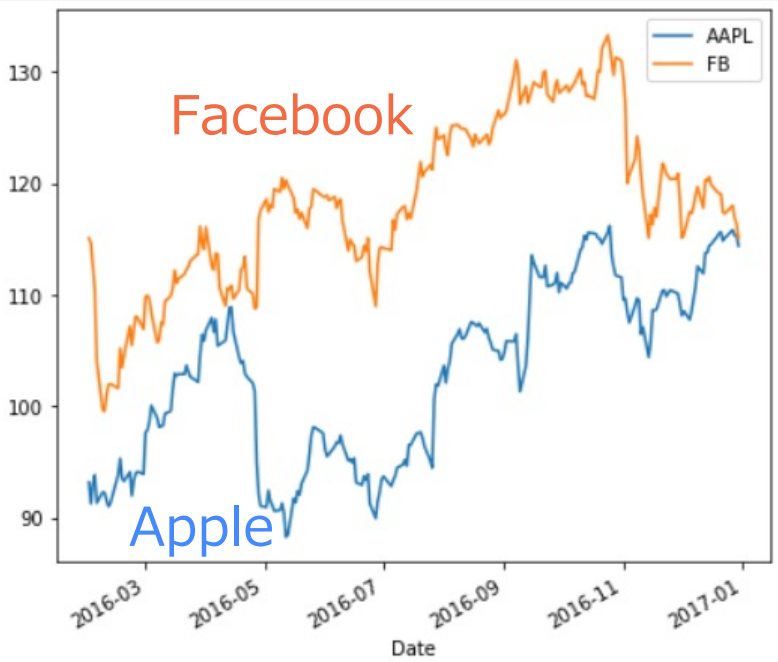


購入金額に対して日次の利回りを計算します。
VQE:ほぼ通年プラスの利回りで年末には20%のパフォーマンス
QAOA:半年以上7%程度の損失、年末にはフラットまで復調

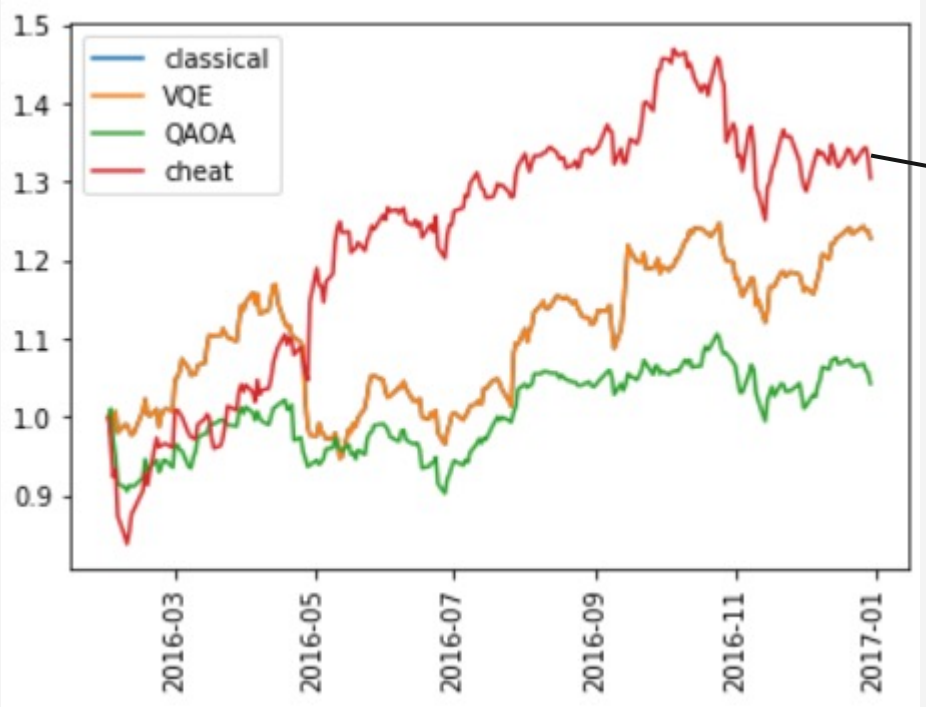


余談ですが、、、
結果を知ってから考えるとAmazonだけを購入したほうがパフォーマンスは良さそうな気がします。

2016/1/31~2016/12/31



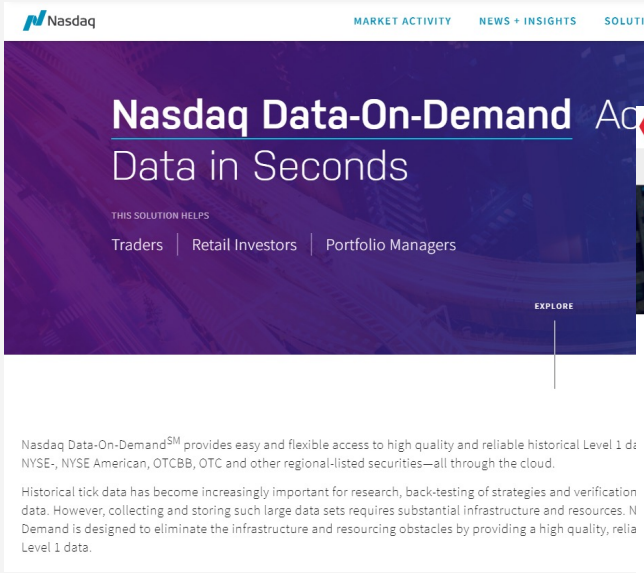
やっぱりそうでした（笑）30%近い利回りを挙げています。
1カ月のデータから1年先は判断できないですね、、、



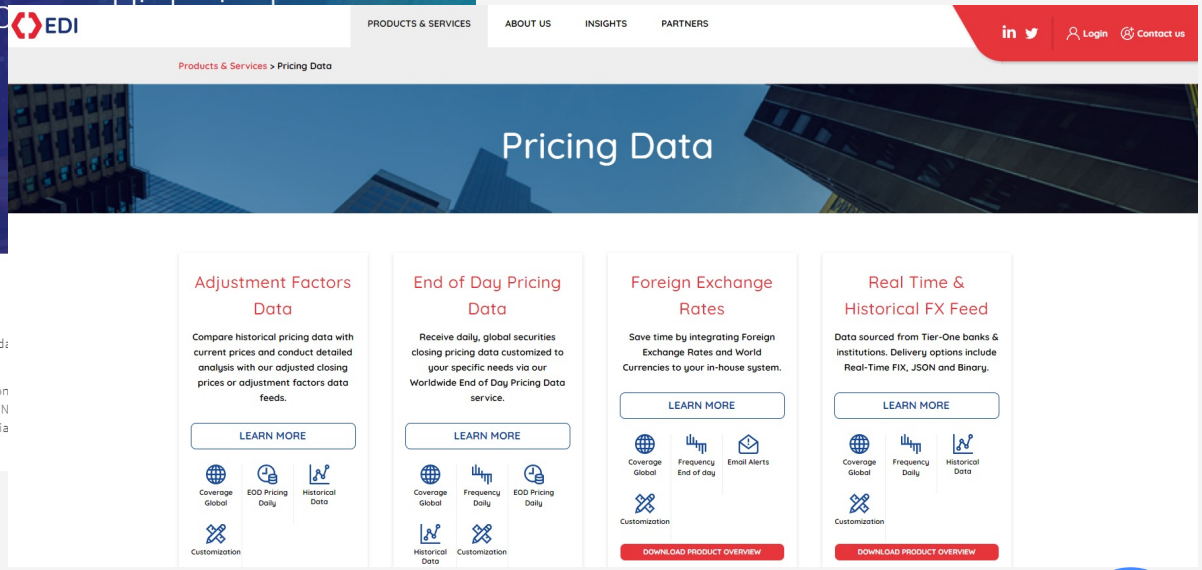
Amaozonのみ

結論　やっぱり投資は難しい。。。

よりリアルタイムできめ細かいデータが必要な場合はNASDAQやExchange Data International(EDI)提供のAPIから取得することができます。(有料・トライアルあり)



https://www.nasdaq.com/solutions/nasdaq-data-on-demand



https://www.exchange-data.com/

1. Portfolio Optimization

2. Portfolio Diversification

✓ Loading and Processing
Stock-Market Time-Series Data

Portfolio Diversificationとは

インデックスファンドを組む際に、少数の銘柄でインデックスと同等のパフォーマンスを挙げるポートフォリオを選択する手法です。

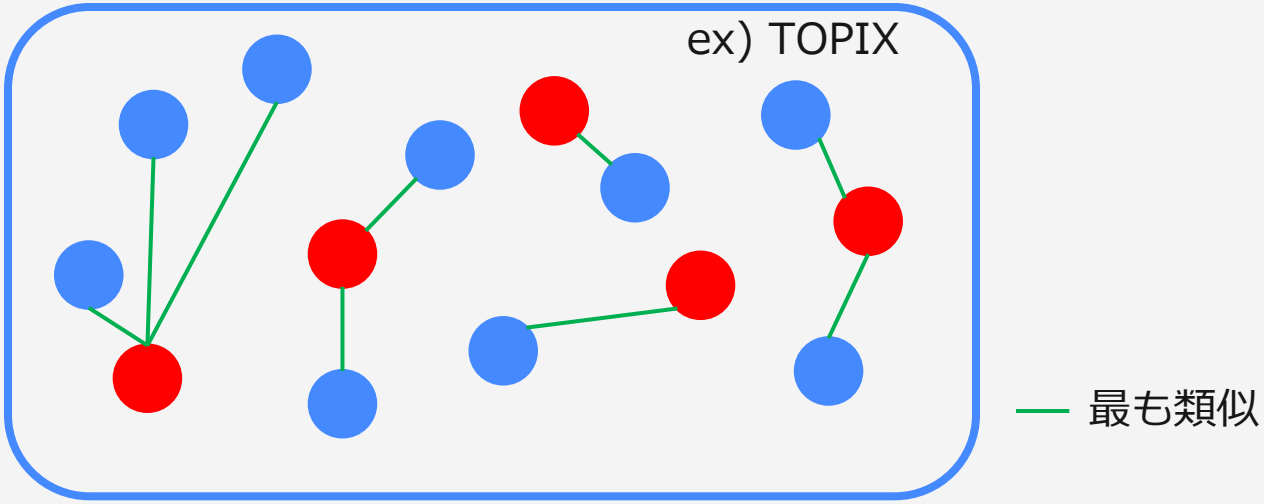
最も安易な方法はインデックと同じ銘柄を同じ比率でファンドを組むことですが、次の２点から現実的ではありません。

- 構成銘柄数が多い(例えばTOPIXは2000銘柄以上)
- 頻繁なリバランスで取引コストがかさむ

東証1部上場の全銘柄
（2020年１月21日現在、2,159社）

そこで、この問題のゴールは次のように設定できます。

インデックス構成銘柄から一定数の銘柄を選択してポートフォリオを組む。
ただし、選択されない銘柄は構成銘柄の中で選択された銘柄のいずれかに
最も類似するように選ぶ。



問題を定式化してみましょう。

maximize $\sum_{i=1}^{n}\sum_{j=1}^{n}\rho_{ij}\,x_{ij}$

subject to $\sum_{j=1}^{n} y_j = q$

ファンドに組み込むのはq銘柄

$\sum_{j=1}^{n} x_{ij} = 1$

最も類似する銘柄は一つ

$x_{ij} \le y_j, x_{jj} = y_j$

ρ_{ij}	銘柄iとjの類似度 (ex:相関)
$x_{ij} \in \{0,1\}$	インデックスファンドの銘柄jは銘柄iに最も類似しているか
$y_j \in \{0,1\}$	銘柄jがインデックスファンドに組み込まれているかどうか
q	選択する銘柄数

この問題も次のようなzベクトルを導入する工夫をすると
前テーマと同様な手法でパウリ演算子から成るハミルトニアンを
得ることが出来ます。

$$z = [x_{11}, x_{12}, \cdots, x_{nn}, y_1, \cdots y_n]$$

$N = n^2 + n$ビット

ペナルティ項を追加して制約を取り除く

値を演算子(量子ゲート)に対応させる

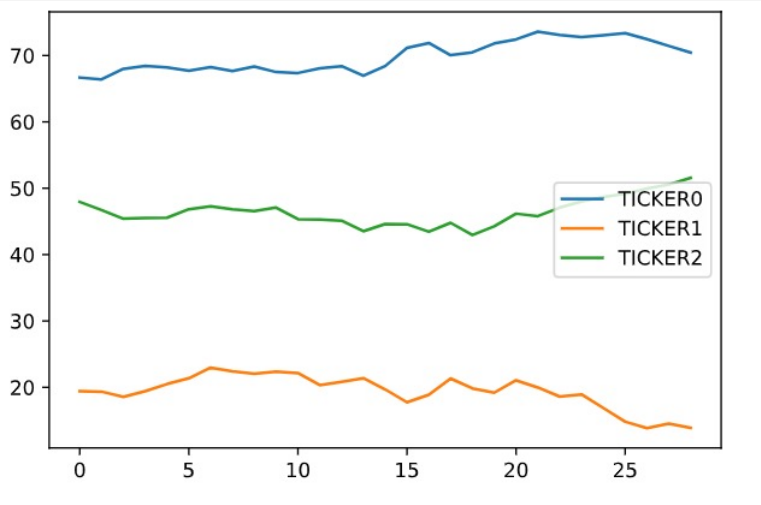
$x_i = (I - Z)/2$

$$\max_{z \in \{0,1\}^N} \boldsymbol{zQz} + \boldsymbol{g^T z} + c$$

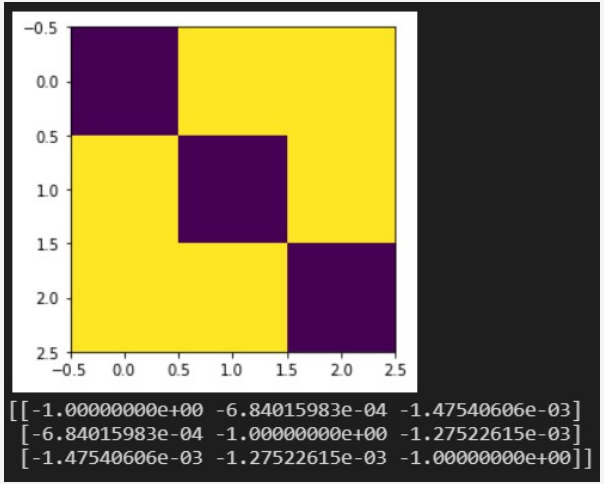
```
qubitOp = portfolio_diversification.get_operator(self.rho, self.n, self.q)
```

qiskitで実装してみましょう。
3銘柄からなる疑似データを生成して実験してみます。

時系列データ



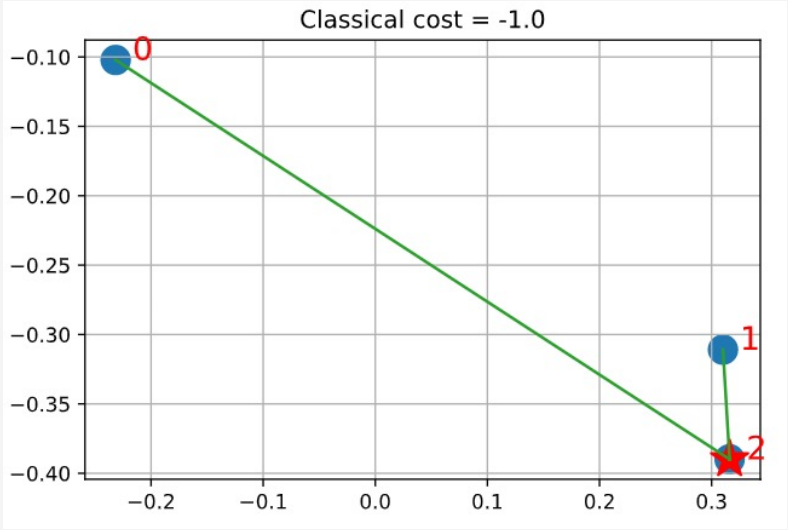
similarity matrix

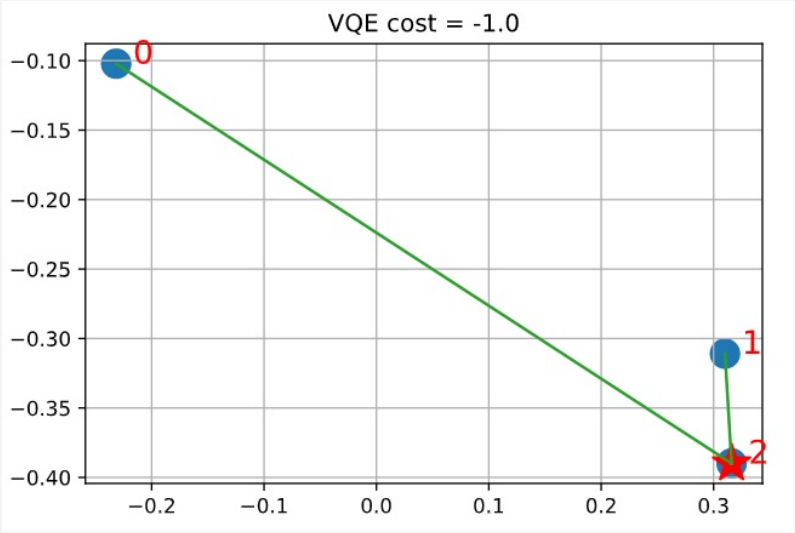



```
data = RandomDataProvider(tickers = stocks,
                          start = datetime.datetime(2016,1,1),
                          end = datetime.datetime(2016,1,30))
data.run()
rho = data.get_similarity_matrix()
```


Qt

VQEを使って最適化を行います($N = 3^2 + 3 = 12qubit$)。
TICKER2が3銘柄を代表する動きであり、インデックスファンドに組み込むべき銘柄であることがわかりました。





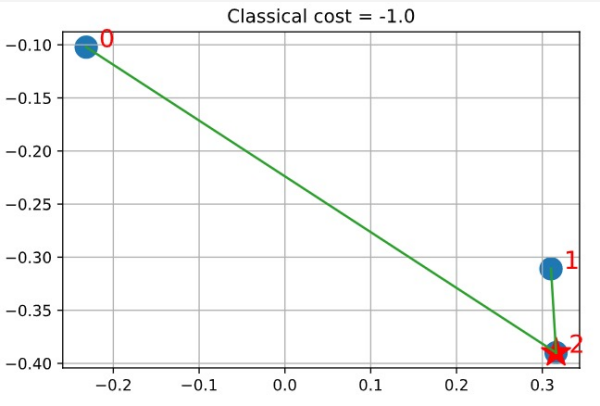
各銘柄(座標に意味はない)

最も類似

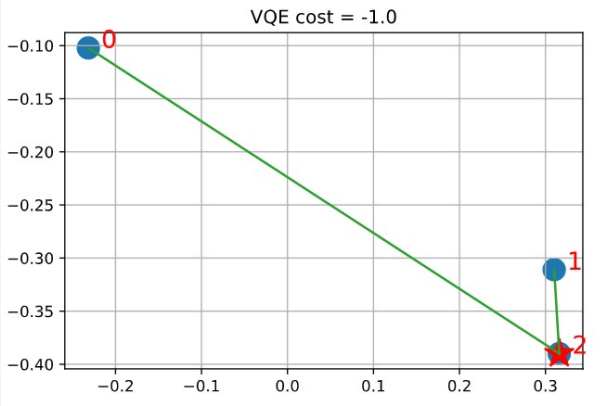
ファンドに組み込む銘柄

Qt

ただしVQEを複数回試行すると異なる結果が得られます。
ヒューリスティックアルゴリズムの注意すべき点と言えます。



試行１



各銘柄(座標に意味はない)

ファンドに組み込む銘柄

最も類似

試行２

Qt

Thank you !