『Qiskit Tutorial』勉強会
アルゴリズム編 –
Grover のアルゴリズムと振幅増幅

2021年6月1日

東京ラボ / IBM Garage
天野 武彦 (Amano, Takehiko)
Qiskit Advocate (2020)
Twitter: @ibmamnt

Qt

Quantum Amplitude Amplification(量子振幅増幅)アルゴリズムについて

- オリジナルGroverアルゴリズムの発展系とみなされています。
  - オリジナルのGrover のアルゴリズムは N 件のデータベースから 1 件を検索するものでした(ただしオリジナル論文の最後のセクションに複数の検索への拡張の言及があります）。
  - またGrover のアルゴリズムの初期状態は Walsh-Hadamard 変換になっていました。

$$|\psi\rangle = H^{\otimes n} |0\rangle$$

- Quantum Amplitude Amplification では複数の解を探索し、さらに初期状態をあるオペレータ A を用いて状態を初期化するのが特徴です。

$$|\psi\rangle = A |0\rangle$$

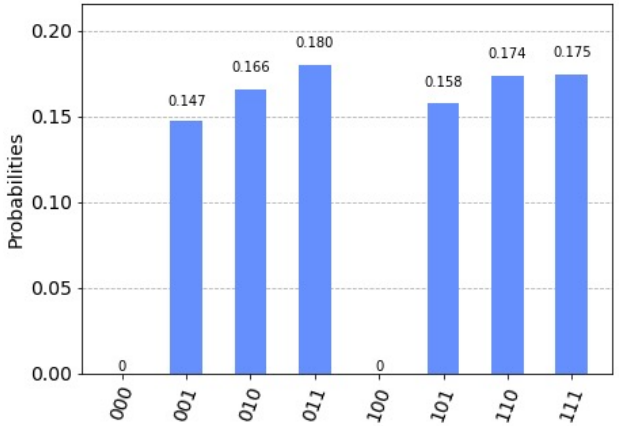
量子振幅増幅および振幅推定(Amplitude Estimation)の問題定義

- ある状態 $|\psi\rangle$ が存在するとする。
  - 状態 $|\psi\rangle$ は 計算基底 $|x\rangle$ $(x = 0,1 \ \ldots 2^{n-1})$で展開できる。 $|\psi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle$
- ある関数 $f(x)$ があり、展開された基底の中で $f(x) = 1$ のものを ‘good’ 状態、$f(x) = 0$ のものを ‘bad’ 状態と呼ぶことにする。
  - 1のものの部分状態 (sub state) を$|\psi_{good}\rangle$ とし
  - 0のものの部分状態 (sub state) を $|\psi_{bad}\rangle$とする
- 状態 $|\psi\rangle$の測定結果のうち good state にある確率 a を求めなさい。
  - おそらく複数の x が good 状態になっている。
  - good 状態を見出す確率の総和が a になる。

要するに $\langle\psi_{good}|\psi_{good}\rangle$ を求めるという問題です。

ここでよくある誤解を解く

- Oracle が分かっていれば増幅される対象がわかるので、そもそも検索しなくてもよいのでは？
  - Walsh-Hadamard 変換で初期状態を用意した場合はそう。
  - 量子振幅増幅 では任意の状態から始まるので、振幅対象が含まれているかどうかはわからない (状態は測定しないと確定しないことを思い出しましょう)
  - $A|0\rangle$ に希望する状態が含まれているかどうかは測定しないとわからない（最低限 N 回 、実際にはこれ以上のサンプリングが必要)
  - 量子振幅増幅 (及び 量子振幅推定) は計算量が $O(\sqrt{N})$ であるというのがポイント。



例えば上の図にあるような状態では |000> |100> を検索しても確率はほぼゼロなので何回測定しても結果は出てこない。

アルゴリズムのあらまし(1/2)

1. 計算基底 $|x\rangle$は $\langle x|x'\rangle = \delta_{xx'}$なので 、$|\psi_{good}\rangle$ と $|\psi_{bad}\rangle$ は直行する。

2. $|\varphi\rangle$ は以下の線形結合で表せるとします。

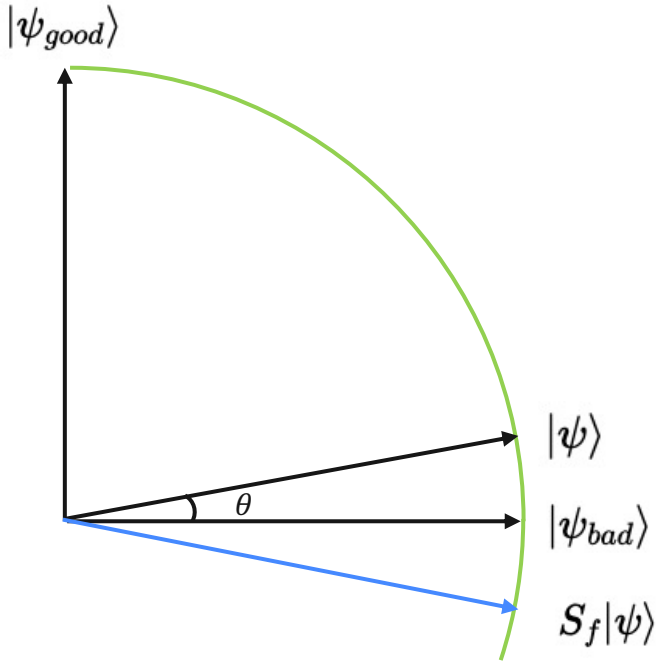
$$|\psi\rangle = \cos\theta|\psi_{bad}\rangle + \sin\theta|\psi_{good}\rangle$$

3. Oracle S_f は次の動作をします。

$$S_f|x\rangle = |x\rangle \quad (\text{for } x \text{ s.t. } f(x)=0)$$
$$S_f|x\rangle = -|x\rangle \quad (\text{for } x \text{ s.t. } f(x)=1)$$

これは次と同じです。

$$S_f\,|\psi_{bad}\rangle = |\psi_{bad}\rangle$$
$$S_f\,|\psi_{good}\rangle = -\,|\psi_{good}\rangle$$



結果として以下の状態が得られます。

$$S_f|\psi\rangle = \cos\theta|\psi_{bad}\rangle - \sin\theta|\psi_{good}\rangle$$

Grover の検索アルゴリズム と Quantum Amplitude Amplification/Estimation
https://whyitsso.net/physics/quantum_mechanics/AA_AE_Grover.htmlから引用

アルゴリズムのあらまし(2/2)

4. 得られた状態を $|\psi\rangle$ を軸として反転させます (S_ψ を適用)。 S_ψ は以下の演算子。

$$S_\psi = 2|\psi\rangle\langle\psi| - I$$

S_φは次の形に変形できる。

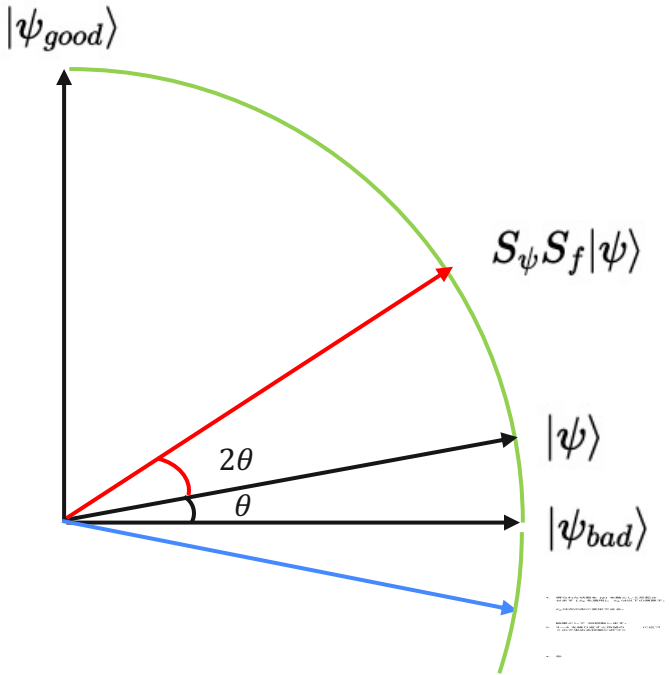
$$AS_0A^\dagger$$

$$S_0 = 2|0\rangle|\langle 0| - I$$

$$(※ |\psi\rangle = A|0\rangle)$$

結果として 2θ回転します。

5. 3～4 を繰り返すと所望の$|\psi_{good}\rangle$ に近づくので求める状態に近づく



註：何回ぐらい回転すれば良いかわからない場合はとりあえずぐるぐる回して周期を求めることで期待値が出ます。これが**量子位相推定**です。

Grover の検索アルゴリズム と Quantum Amplitude Amplification/Estimation
https://whyitsso.net/physics/quantum_mechanics/AA_AE_Grover.htmlから引用

Qiskit での実装

– Qiskit では GroverOperator クラスで実装。演算子は次のように定義されています。

$$Q = AS_0A^\dagger S_f$$

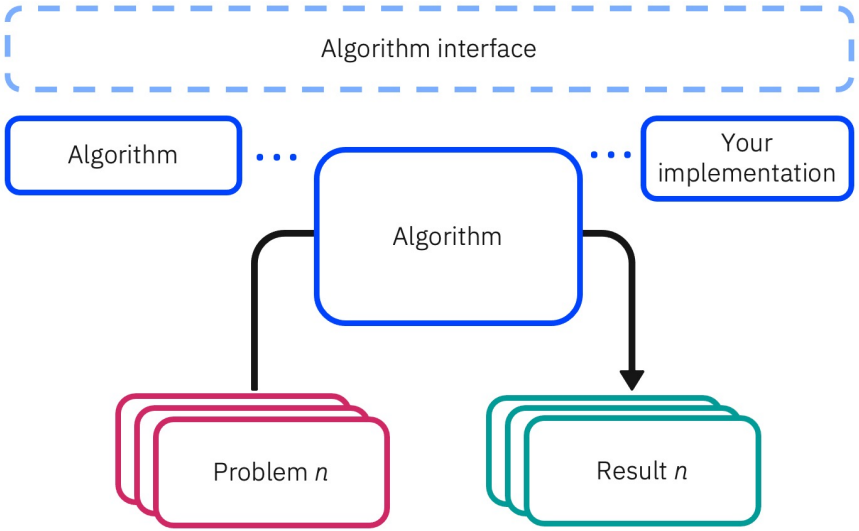
$$S_f : |x\rangle \mapsto (-1)^{f(x)}|x\rangle$$

– Quantum Amplitude Amplification を以下のクラスで実装しています。

クラス	場所	説明
GroverOperator	qiskit.circuit.library	上記の Q 演算子を作成
AmplificationProblem	qiskit.algorithms	Grover のような振幅増幅アルゴリズムへの入力
Grover	qiskit.algorithms	Groverの探索アルゴリズム本体
GroverResult	qiskit.algorithms	Grover結果

GroverOperatorはGroverクラスで利用しているのであまり知らなくても良いかも知れません。

Qiskit v0.25 以降からのアルゴリズム利用の手引き



Qiskit v0.25 以降からは次のようにアルゴリズムを利用するように変更されました。

1. 問題 (Problem) を設定する
2. アルゴリズムを適用する
3. 結果を取得し調べる

Grover アルゴリズムの使用方法概要

```
good_state = ['11']
oracle = QuantumCircuit(2)
oracle.cz(0, 1)

qasm = QuantumInstance(BasicAer.get_backend('qasm_simulator'))

problem = AmplificationProblem(oracle, is_good_state=good_state)

grover = Grover(quantum_instance=qasm)

result = grover.amplify(problem)

result.top_measurement, result.circuit_results
```

動作させるバックエンドの設定

問題の設定。ここではオラクルとgood 状態を指定

Grover クラスのインタンスを作成。引数にはQuantumInstance を指定

Grover 実行 (amplify)

わずか数行で Grover アルゴリズムが実行できます

Qiskit textbook の Grover アルゴリズム説明との比較

```
n = 2
grover_circuit = QuantumCircuit(n)
def initialize_s(qc, qubits):
    """qcの 'qubits' にH-gate を適用"""
    for q in qubits:
        qc.h(q)
    return qc
grover_circuit = initialize_s(grover_circuit, [0,1])
grover_circuit.cz(0,1) # オラクル

# Diffusion operator (U_s)
grover_circuit.h([0,1])
grover_circuit.z([0,1])
grover_circuit.cz(0,1)
grover_circuit.h([0,1])
sv_sim = Aer.get_backend('statevector_simulator')
job_sim = execute(grover_circuit, sv_sim)
statevec = job_sim.result().get_statevector()
grover_circuit.measure_all()

qasm_simulator = Aer.get_backend('qasm_simulator')
shots = 1024
results = execute(grover_circuit, backend=qasm_simulator,
shots=shots).result()
answer = results.get_counts()
plot_histogram(answer)
```

```
good_state = ['11']
oracle = QuantumCircuit(2)
oracle.cz(0, 1)

qasm = QuantumInstance(BasicAer.get_backend('qasm_simulator'))
problem = AmplificationProblem(oracle, is_good_state=good_state)
grover = Grover(quantum_instance=qasm)
result = grover.amplify(problem)
result.top_measurement, result.circuit_results
```

大幅な工数削減効果！！

AmplificationProblem クラス

– 振幅増幅問題の定義を行うクラスです。

CLASS AmplificationProblem(oracle, state_preparation=None, grover_operator=None, post_processing=None, objective_qubits=None, is_good_state=None)

パラメータ	説明
oracle	Bad 状態で反転させる QuantumCircuit もしくは Statevector。Statevector の場合には指定した状態に対して -1 を与える対角行列が oracle として生成される。
state_preparation	初期状態を生成する行列(オペレーター) A 。指定しないとウォルシュ・アダマールで初期化
grover_oprator	Grover オペレーターとして $Q = AS_0 A^{\dagger} S_f$ 以外のものを利用する場合に指定
post_processing	結果の後続処理をしたい場合に指定
objective_qubits	指定した量子ビットを測定。指定しない場合には全ての量子ビットを測定
is_good_state	$f(x) = 1$ かどうかを判定する関数、もしくは good 状態を指定(複数)。

Oracle の指定

– Oracle は複数の方法で指定できます。

指定方法	説明	例
Statevector	状態ベクトルで指定。 反転させる状態を指定する。	problem = AmplificationProblem(Statevector.from_label('111'), is_good_state=['111'])
QuantumCircuit	量子回路を指定	oracle = QuantumCircuit(2) oracle.cz(0, 1) problem = AmplificationProblem(oracle, is_good_state=['11'])
PhaseOracle	論理記号式から Oracle を構成	oracle = PhaseOracle('x1 & x2 & (not x3)')

探索の成功の判定

- grover.amplify() のリターン値において result.oracle_evaluation 変数が False の場合に探索失敗、True は成功。

```
oracle = Statevector.from_label('011')
good_state = ['111']
problem = AmplificationProblem(oracle, is_good_state=good_state)
grover = Grover(quantum_instance=qasm)
result = grover.amplify(problem)
print('Success!' if result.oracle_evaluation else 'Failure!')
print('Top measurement:', result.top_measurement)
```

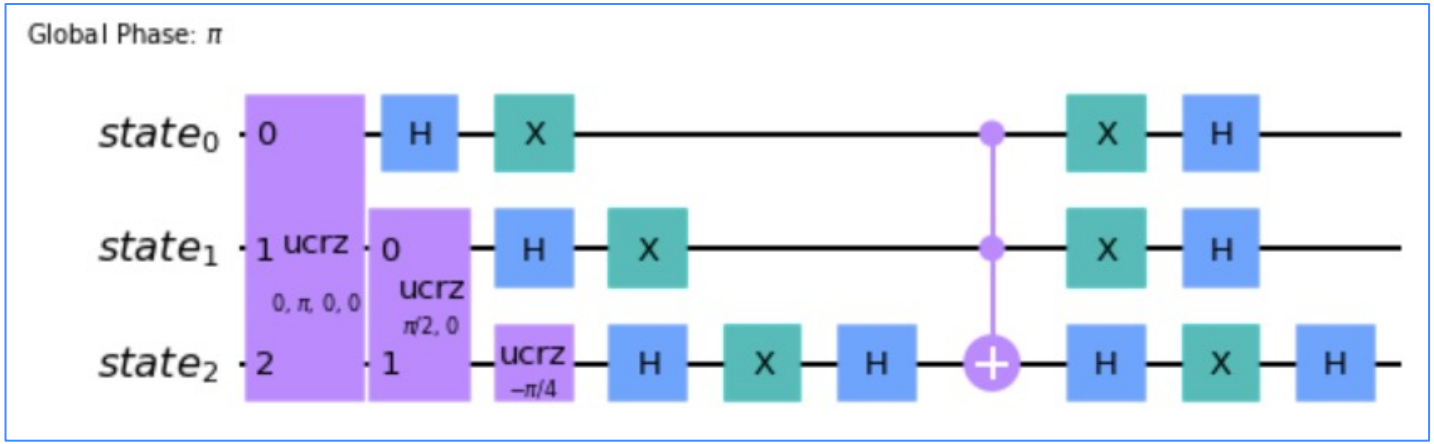
```
Failure!
Top measurement: 011
```

good_state ‘111’ を探したが見つからなかった！

Oracle が Statevector から作成される場合の Grover オペレーター

– 先ほどの 実験では oracle を Statevector から作成しました。この場合、Grover オペレータは次のような回路になります。中身は対応する状態を -1 (Z演算子) するもの。

```
problem.grover_operator.draw(output='mpl')
```



PhaseOracle の使い方

- Oracle の作成方法として、PhaseOracle を利用することができます。PhaseOracle では bool 演算や、DIMACS CNFフォーマットを指定できます。
- bool 演算例

```
# |11⟩を探す場合には、’x1 & x2’ などように指定
oracle = PhaseOracle(‘x1 & x2‘)
```

- DIMACS CNF フォーマット読み取り令

```
oracle = PhaseOracle.from_dimacs_file(<ファイル名>)
```

```
c DIMACS CNF file with 3 satisfying
assignments: 1 -2 3, -1 -2 -3, 1 2 -3.
p cnf 3 5
-1 -2 -3 0
1 -2 3 0
1 2 -3 0
1 -2 -3 0
-1 2 3 0
```


DIMACS-CNF：https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html

PhaseOracle 例

次の例ではブール演算を指定しています。’x1 & x2 & (not x3)’ の場合は、”110”(古典ビット列) が該当します。

```
oracle = PhaseOracle('x1 & x2 & (not x3)')
oracle.draw('mpl')
good_state = ['011']
problem = AmplificationProblem(oracle, is_good_state=good_state)
grover = Grover(quantum_instance=qasm)
result = grover.amplify(problem)
print('Success!' if result.oracle_evaluation else 'Failure!')
print('Top measurement:', result.top_measurement)
```

```
Success!
Top measurement: 011
```

is_good_state で関数の指定

– AmplificationProblem の is_good_state は Good 状態の文字リストの他、関数を指定できます。

```
def callable_good_state(bitstr):
    if bitstr == "11":
        return True
    return False
oracle = Statevector.from_label('11')
problem = AmplificationProblem(oracle, is_good_state=callable_good_state)
grover = Grover(quantum_instance=qasm)
result = grover.amplify(problem)
print('Success!' if result.oracle_evaluation else 'Failure!')
print('Top measurement:', result.top_measurement)
```

```
Success!
Top measurement: 11
```

より一般的な Amplitude Amplification

- 冒頭で説明したように、Amplitude Amplification は 特定の状態 $|\psi\rangle = A|0\rangle$ から始まり、OracleOperator にも利用されます。

$$Q = AS_0A^\dagger S_f$$
$$S_f : |x\rangle \mapsto (-1)^{f(x)}|x\rangle$$

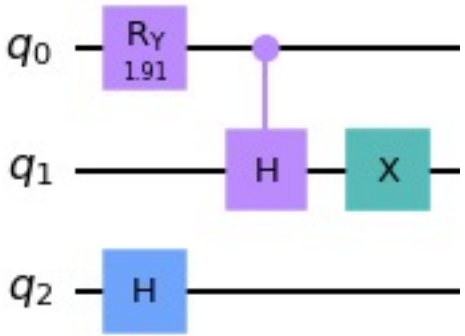
State_preparation に 状態 $|\psi\rangle$ を指定する

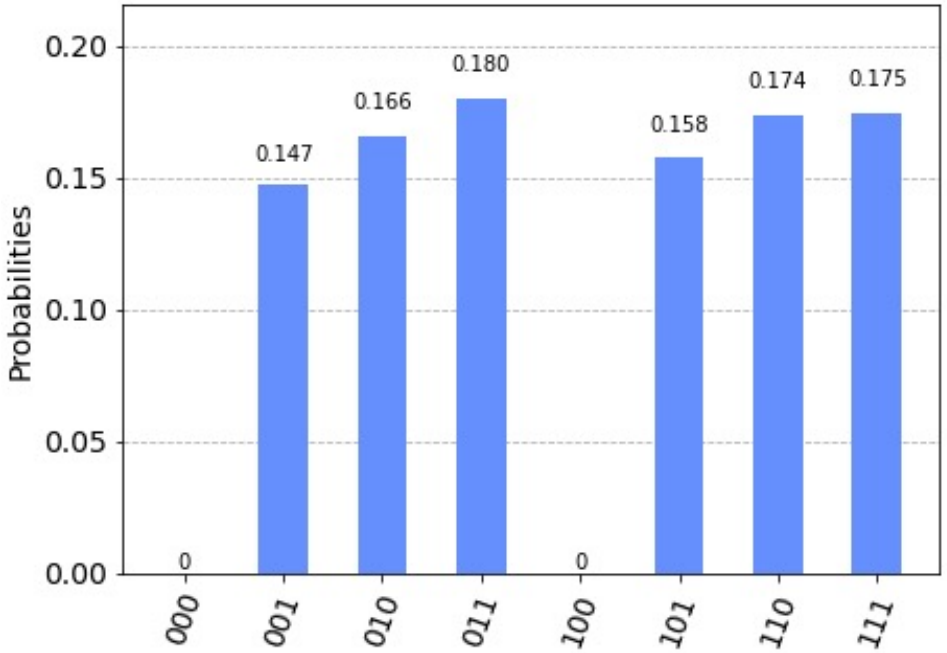
CLASS AmplificationProblem(oracle, state_preparation=None, grover_operator=None, post_processing=None, objective_qubits=None, is_good_state=None)

state_preperation

チュートリアルでは次の状態を利用しています。

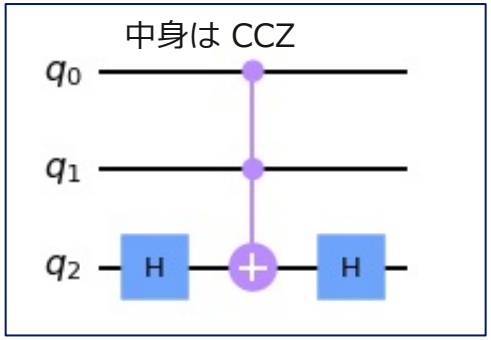
```
theta = 2 * np.arccos(1 / np.sqrt(3))
state_preparation = QuantumCircuit(3)
state_preparation.ry(theta, 0)
state_preparation.ch(0,1)
state_preparation.x(1)
state_preparation.h(2)
```





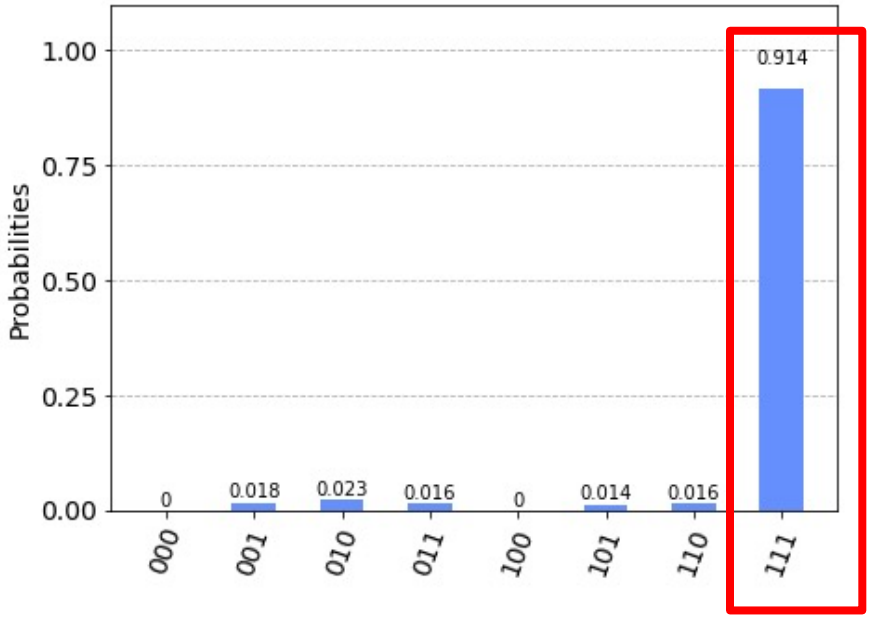
Oracle の指定と振幅増幅結果

– 0ビット目、1ビット目が 1 の場合に、2ビット目を反転させる回路です。 |111> が増幅対象になります。確かに |111> が増幅されました。



$$\begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.0 \end{bmatrix}$$

|111> を反転



反復(iterations)の数について

- 反復(iterations)数は明示的に(リストとして)指定できます。指定しない場合は次の計算式で与えられるようです(最大反復数)。見つかった場合は反復が停止します。

  max(10, 2 ** amplification_problem.oracle.num_qubits)

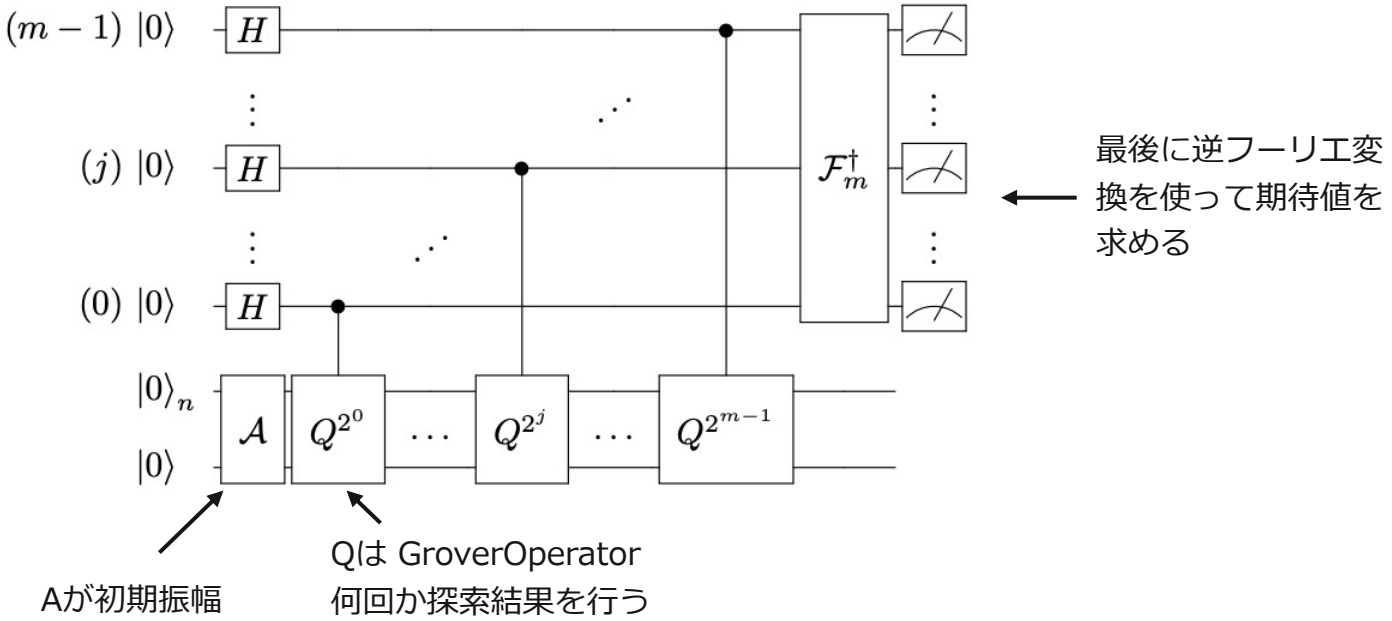
- 反復の数がわからない場合には sample_from_iterations を指定すると0 から iterations の間のランダムな値を使います（わからない場合に有効)。

```
# using sample_from_iterations
oracle = QuantumCircuit(2)
oracle.cz(0, 1)
grover = Grover(oracle=oracle, good_state=['11'], iterations=[1, 2, 3],
sample_from_iterations=True)
```

- わかっている場合には optimal_num_iterations() メソッドで指定できます。

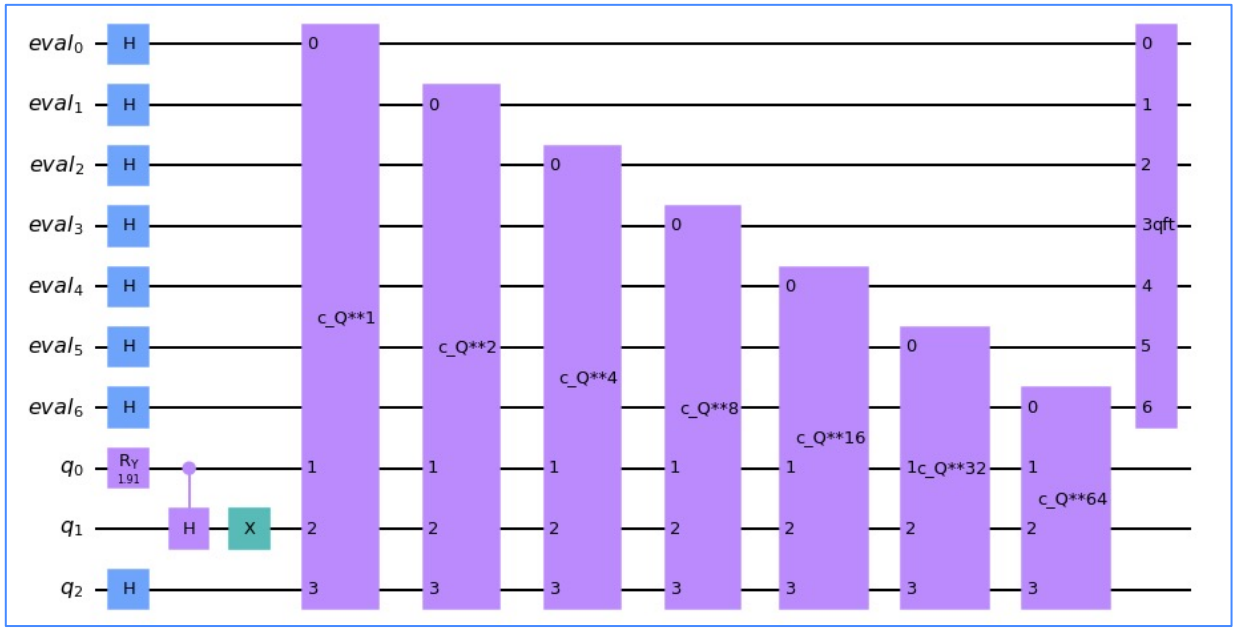
[ご参考] – QAE (Quantum Amplitude Estimation - 量子振幅推定)

– 今回は説明していませんが、問題は $\langle\psi_{good}|\psi_{good}\rangle$ を求めるものでした。この値は QAE (Quantum Amplitude Estimation) で Qiskit では qiskit.algorithms.AmplitudeEstimation で実装されています(2021年5月時点でこのAPIを解説したチュートリアルはない)。



せっかくなので試してみました

```
from qiskit.algorithms import AmplitudeEstimation, EstimationProblem
problem2 = EstimationProblem(state_preparation=state_preparation, objective_qubits=[0,1,2],
grover_operator=problem.grover_operator)
ae = AmplitudeEstimation(num_eval_qubits=7, quantum_instance=qasm)
ae.construct_circuit(problem2).draw("mpl")
```

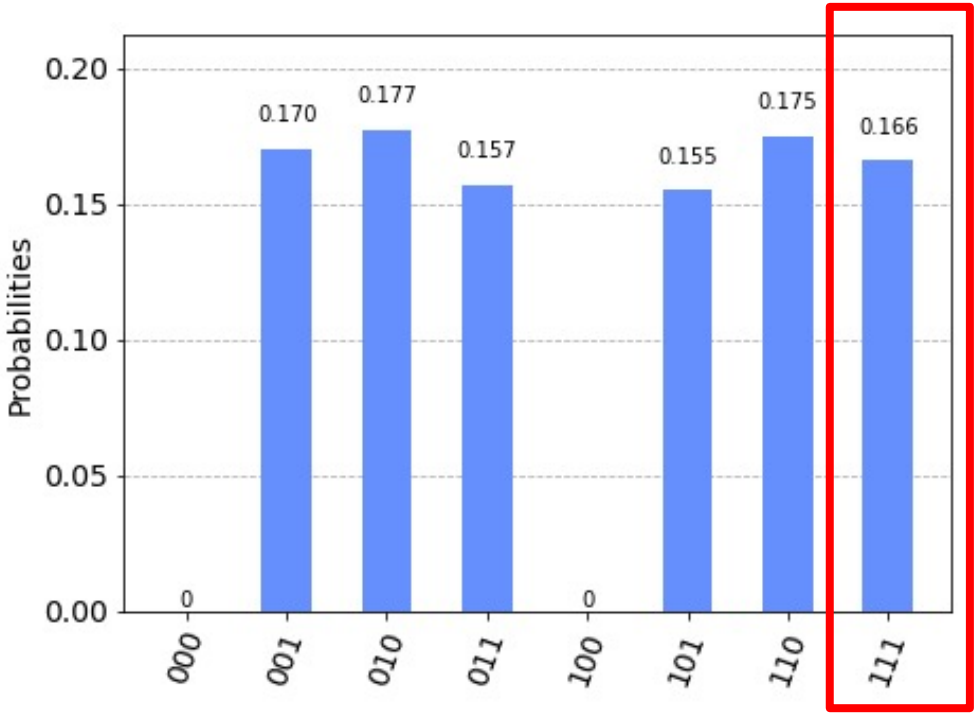


結果

|111> の確率振幅の情報が出て来ました
（result.estimationは確率振幅の推定値です）

```
result = ae.estimate(problem2)
print('Grid-based estimate:', result.estimation)
print('Improved continuous estimate:', result.mle)
```

```
Grid-based estimate: 0.1642205
Improved continuous estimate: 0.16656527548736885
```



まとめ

- 量子振幅増幅 およびそれを組み合わせた 量子振幅推定 などは 金融分野での VaR (Value at Risk) の導出に適用例があります。
  - VaR は予想最大損失額で、保有している資産が最大どのくらい損失をうけるかを予想する指標です。
- そのほかの例としては、IBM 田中氏による「Qiskitチュートリアル勉強会 金融編-3、ヨーロピアン・コール・オプションの価格推定」をぜひご参照ください。
- モンテカルロシミュレーションやベイズ推定を実施している分野で活躍が期待できると思います（個人的な感想）

ご静聴ありがとうございました！