体験型レクチャー

IBM量子コンピュータ実機に触れる

アカウント登録から、実機での回路実行、Groverのアルゴリズムまで

1 アカウント作成

2 環境構築

3 回路を書く

4 実機で実行

5 結果を読む

6 Grover

OVERVIEW

このレクチャーで体験すること

01

実機とつながる

IBM Quantum Platformでアカウントを作り、自分の手元のPythonから本物の量子コンピュータに接続する

02

回路を書いて送る

量子ビットを重ね合わせ・もつれさせる回路をQiskitで組み、実機のキューに送信する

03

ノイズを体感する

理論値とのズレから、現在の量子コンピュータが抱える誤り・ノイズの現実を数字で確認する

04

量子の武器を試す

Groverのアルゴリズムを実行し、量子探索がランダム探索より速いことを自分の結果で実感する

STEP 1 — 準備

IBM Quantum Platformのアカウントを作る

1. https://quantum.cloud.ibm.com/ にアクセス
2. メールアドレスと電話番号でアカウント作成
3. 初回ログインから30日間は無料で利用可能
4. 継続利用にはクレジットカード情報の登録が必要

無料プラン（Open Plan）

28日ごとに最大10分の量子ランタイムを実機で利用可能。

2026年3月の拡充で、直近12か月に20分利用したユーザーには以後12か月で180分が付与される。

最新機種 IBM Quantum Heron r2（ibm_kingston）も全ユーザーに開放。

STEP 1 — 準備

「鍵」を2つ取得する：APIキーとCRN

APIキー

＝あなた自身の認証キー

どこにある？

ダッシュボード上部の「API key」から生成

注意点

44文字のキーは一度しか表示されない。必ずコピーして保存する。

CRN

＝どのインスタンスを使うか

どこにある？

左メニュー「Instances」タブの一覧に表示

注意点

行にカーソルを合わせるとコピーアイコンが出る。Open Planでは自動で1つ用意されている。

STEP 2 — 環境構築

手元のパソコンをQiskitが動く環境にする

① Pythonをインストール

python.org からインストーラーを取得し実行。ターミナルで python3 --version を叩いて確認する。

② 作業環境とライブラリを用意

仮想環境を作り、その中にQiskit本体・IBM接続用ライブラリ・Jupyterをまとめてインストールする。

③ Jupyter Notebookを起動

ターミナルで jupyter notebook と打つとブラウザでノートブック画面が開く。

```
$ mkdir ~/quantum-work
$ cd ~/quantum-work
$ python3 -m venv venv
$ source venv/bin/activate

(venv) $ pip install \
    qiskit \
    qiskit-ibm-runtime \
    jupyter matplotlib pylatexenc

(venv) $ jupyter notebook
```

STEP 2 — 環境構築

取得した鍵をQiskitに登録する

```
from qiskit_ibm_runtime import QiskitRuntimeService

QiskitRuntimeService.save_account(
    token="＜APIキー＞",
    channel="ibm_quantum_platform",
    instance="＜CRN＞",
    name="my_account",
    overwrite=True
)
```

一度実行すれば、認証情報はローカルに保存される。以後は毎回この設定を書く必要はない。

2回目以降の接続はこれだけ

```
service = QiskitRuntimeService()
```

⚠ importを忘れずに

このコードを動かす前に、from qiskit_ibm_runtime import QiskitRuntimeService の実行が必要。カーネル再起動後は、この行も忘れずに再実行する。

⚠ セキュリティ上の注意

APIキーは他人に見せない秘密情報。誤って共有した場合は、ダッシュボードから該当キーを削除し再発行する。

TIPS — 環境構築

つまずいたときの3つのコツ

① コマンドはどこに打つ？

1. **ターミナル**
   venvの起動・pip install・jupyter notebookの起動はここ
2. **ブラウザ（Jupyter）**
   起動後に開く画面。ここでノートブックを作る
3. **セルの中**
   Pythonのコードはここだけに書く

② セルの実行はShift+Enter

```
Shift + Enter
```

- 上部の「▶ Run」ボタンでも実行できる
- 実行中はセル左が [*] と表示される
- 完了すると [1] のように番号が入る

③ 動かなくなったら「リセット」

貼り付けミスやエラーが続くときは、直そうとするより最初からやり直す方が早い。

- 「Kernel」→「Restart Kernel」
- リセットで変数はすべて消える
- 1番目のセルから順に全部実行し直す

量子論理回路とは|0>と|1>の重ね合わせ状態を信号に用いて、さまざまな論理演算を行うもの

重ね合わせ状態

- １量子ビット（２成分ベクトル）

|0>状態　　|1>状態

$$\psi = c_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + c_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

$$= c_0|0\rangle + c_1|1\rangle$$

- 2量子ビット（2つの２成分ベクトルの組み合わせ）

$$\psi = c_{00} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} + c_{01} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} + c_{10} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} + c_{11} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$= c_{00}|0\rangle|0\rangle + c_{01}|0\rangle|1\rangle + c_{10}|1\rangle|0\rangle + c_{11}|1\rangle|1\rangle$$

2量子ビットは４つの独立な状態があるので、４成分ベクトルで書くこともある。

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

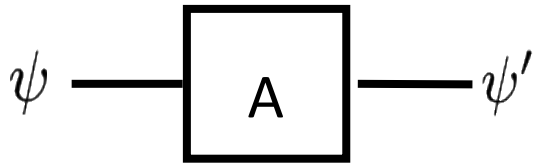
$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 1 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\psi = \begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix}$$

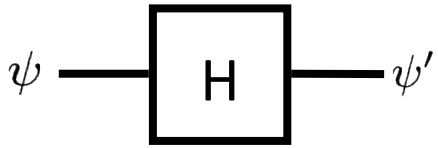
論理演算:状態ベクトルに行列をかける操作に対応

- 1量子ビットのとき　ψ：量子ゲートの入力
  ψ'：量子ゲートの出力

$$\psi = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \longrightarrow \psi' = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

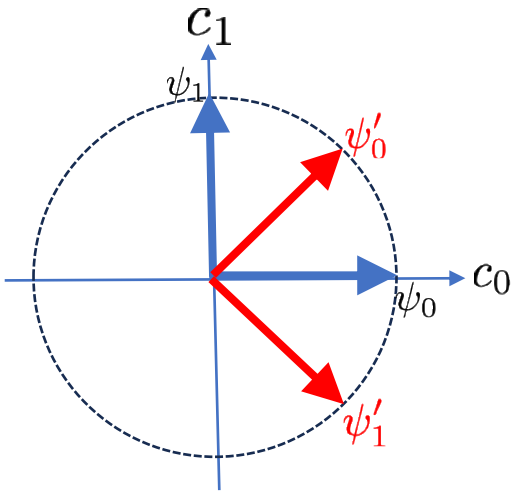


例）アダマールゲート



$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

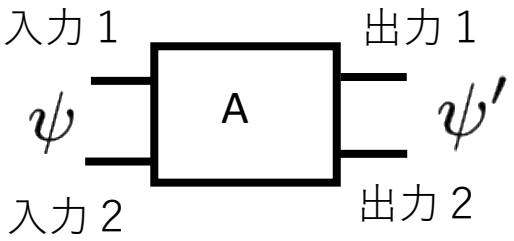
$$\begin{cases} \psi_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \longrightarrow \psi_0' = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ \psi_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \longrightarrow \psi_1' = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \end{cases}$$



演算の結果、長さ１のベクトルが方向の違う
長さ1のベクトルに変換されている。

論理演算:状態ベクトルに行列をかける操作に対応

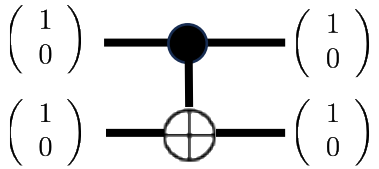
- 2量子ビットのとき $\psi = \begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix} \longrightarrow \psi' = \underbrace{\begin{pmatrix} A_{00,00} & A_{00,01} & A_{00,10} & A_{00,11} \\ A_{01,00} & A_{01,01} & A_{01,10} & A_{01,11} \\ A_{10,00} & A_{10,01} & A_{10,10} & A_{10,11} \\ A_{11,00} & A_{11,01} & A_{11,10} & A_{11,11} \end{pmatrix}}_{=A} \begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix}$

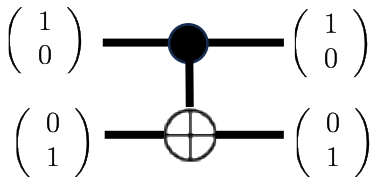


ψ ：量子ゲートの入力１,２
ψ'：量子ゲートの出力１,２

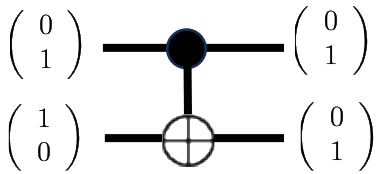
例）CNOTゲート(コントロール ノット ゲート)

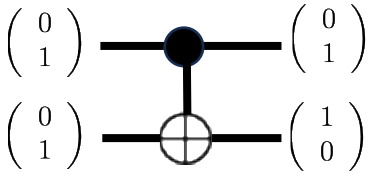
$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$





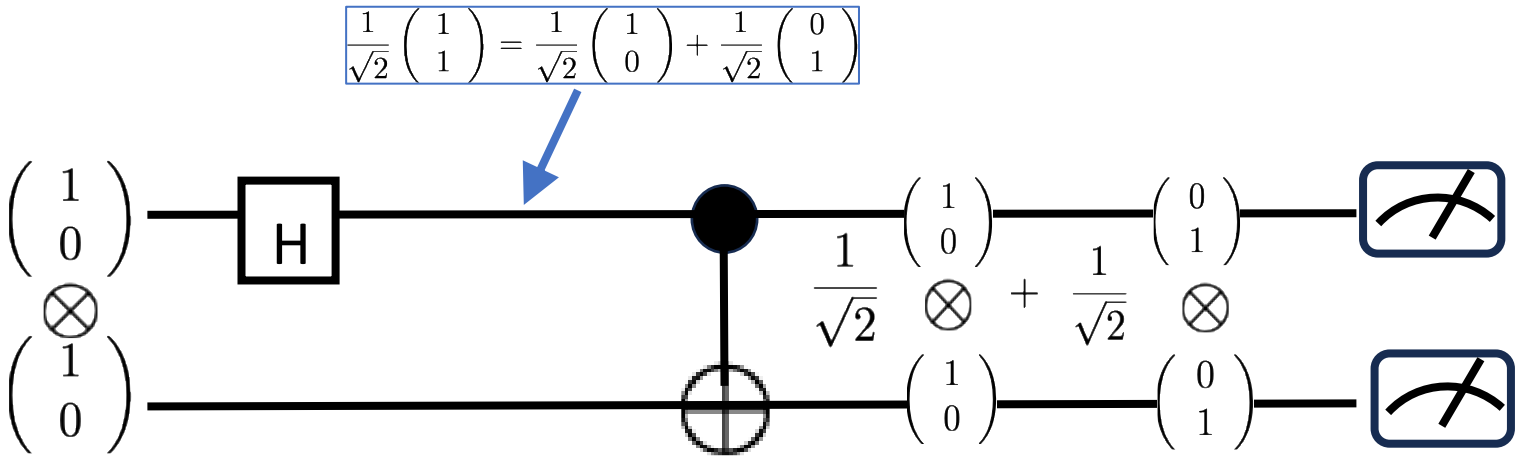
入力１が|0>なら
出力2 = 入力２





入力１が |1>なら
出力2 は 入力２を反転したもの

最初の回路：量子もつれ（ベル状態）を作る



測定を行うと、
それぞれ50%の確率で (出力１, 出力２）＝|0>|0>）または　＝|1>|1>
という結果が出る。

この量子回路は出力１と出力２が連動する状態（強く相関する状態）を生み出す。
このような状態を「量子的にもつれた状態」という。

STEP 3 — 回路

最初の回路：量子もつれ（ベル状態）を作る

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2)
qc.h(0)        # 量子ビット0を重ね合わせに
qc.cx(0, 1)    # 量子ビット0の状態が|1>なら量子ビット1を反転（もつれ生成）
qc.measure_all()

qc.draw('mpl')
```

回路図の表示には matplotlib と pylatexenc が必要：
pip install matplotlib pylatexenc
インストール後はカーネルを再起動してから実行する。

何をしている回路？

H ゲート

量子ビット0を「0でもあり1でもある」重ね合わせ状態にする

CXゲート

0と1を結びつけ、片方を測るともう片方の結果が決まる「もつれ」を作る

期待される結果

00 と 11 だけが、それぞれ約50%ずつ出るはず

STEP 4 — 実機実行

回路を実機のキューに送る、3ステップ

① バックエンドを選ぶ

```
backend = service.least_busy(
    operational=True,
    simulator=False)
```

② 実機用に変換

```
from
qiskit.transpiler.preset_passmanagers
import \
    generate_preset_pass_manager

pm = generate_preset_pass_manager(
    backend=backend,
    optimization_level=1)
isa = pm.run(qc)
```

③ 送信して結果取得

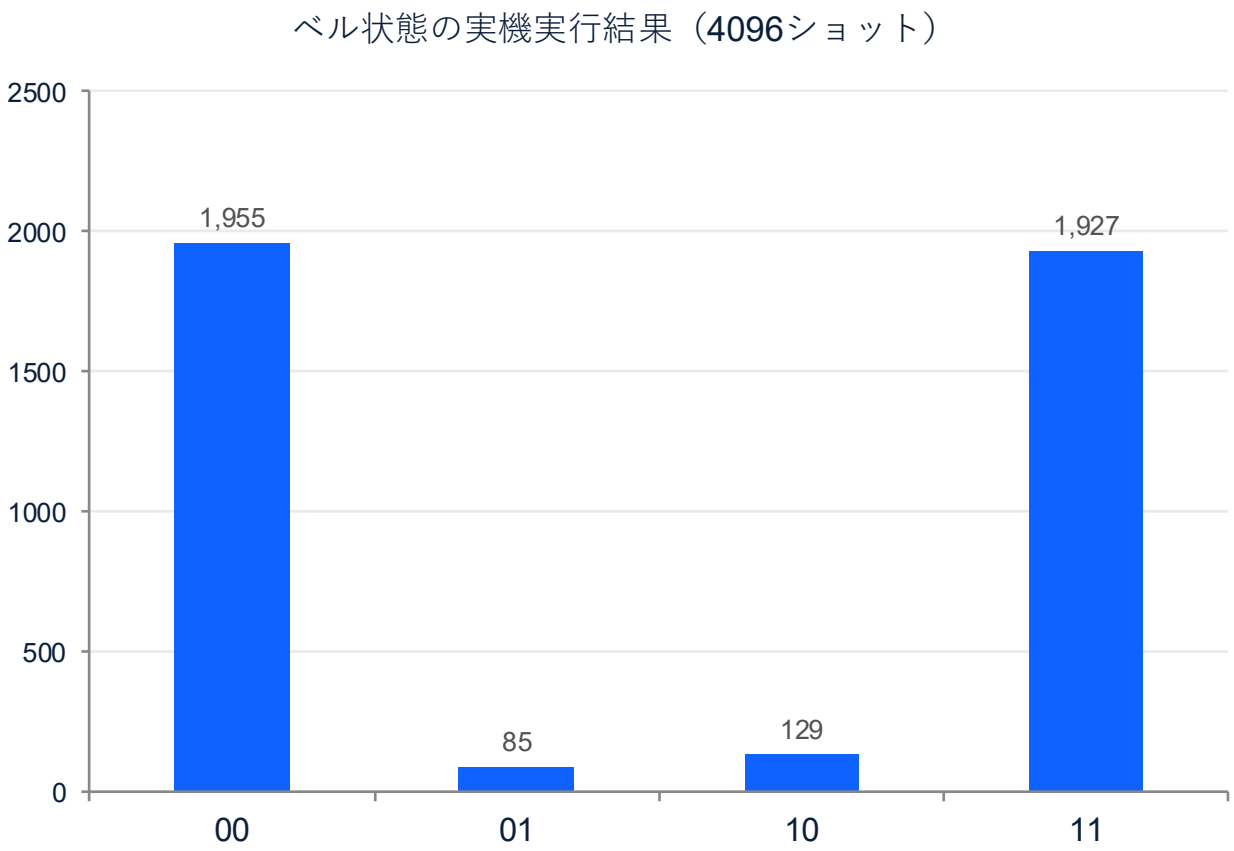
```
from qiskit_ibm_runtime import
SamplerV2 as Sampler

sampler = Sampler(mode=backend)
job = sampler.run([isa])
counts = job.result()[0]
    .data.meas.get_counts()
```

実機は混雑することがあり、キューで数分〜数十分待つ場合がある。job.status() で QUEUED / RUNNING / DONE を確認できる。

STEP 5 — 結果を読む

実機の結果には「ノイズ」が乗る



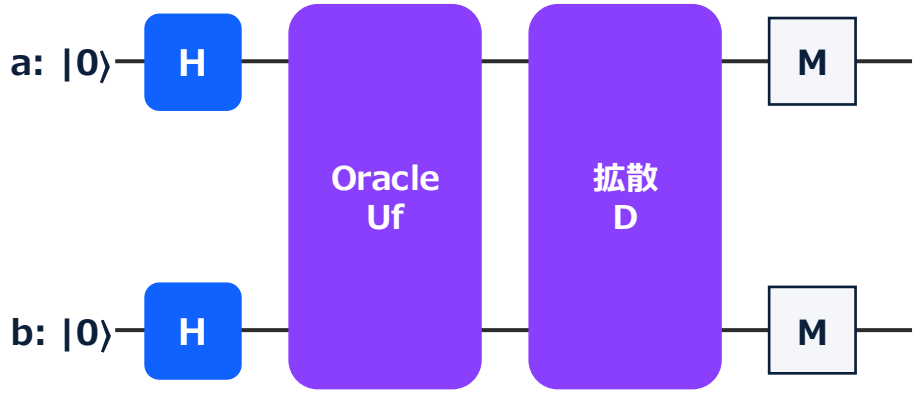
なぜ 01・10 が出るのか

- ゲートエラー：H・CXを物理的に実行する際の誤差
- 測定エラー：0と1を読み違える
- デコヒーレンス：熱・電磁ノイズで量子状態が壊れる

正答率は約95%。現在のNISQ（ノイズあり中規模量子）デバイスとしては良好な部類。

2つ目の回路：量子探索Groverのアルゴリズム

- 2個のビットに対して、0(False)または１(True)の値を対応させる写像fがあるとする。
- どれか一つだけが正解すなわち1(True)の値を返すものとする。
- 例)f(0,0)=f(0,1)=f(1,0)=0, f(1,1)=1
- どの組み合わせが正解か知りたい。
- 全部しらみつぶしに計算すれば、正解には到達できるが、量子コンピュータを使ってできないか？



H → オラクル → 拡散 を1回繰り返すだけで、
2量子ビットなら正解が確定する

1. アダマールゲートを使って、あらゆる可能な状態の重ね合わせを作り、
2. オラクルゲートを使って、写像fの結果を使って、正解の状態の符号を反転する。
3. 最後に拡散ゲートを使って、正解の状態だけが増幅されるようにする。

STEP 6 — 応用

Groverのアルゴリズム：量子で「探す」

4つの箱（00, 01, 10, 11）の中から、正解1つ（11）を見つけたい

1

重ね合わせ

4つの状態を均等な確率で同時に持たせる

2

オラクル

正解（11）だけに「印」をつける（位相反転）

3

拡散演算子

印がついた状態の振幅を増幅し、確率を引き上げる

2量子ビット（4通り）の場合、この3ステップの繰り返しは1回だけで最適になる

STEP 6 — 応用

Groverの回路を実装する（正解＝11）

```
qc = QuantumCircuit(2)

# ステップ1: 重ね合わせ
qc.h([0, 1])

# ステップ2: オラクル（|11>の位相反転）
qc.cz(0, 1)

# ステップ3: 拡散演算子（振幅増幅）
qc.h([0, 1])
qc.x([0, 1])
qc.cz(0, 1)
qc.x([0, 1])
qc.h([0, 1])

qc.measure_all()
```

実行の流れは共通

ベル状態のときと同じ手順で実機に送信する。

1. backend選択
2. トランスパイル
3. Samplerで送信
4. 結果を取得

STEP 6 — 応用

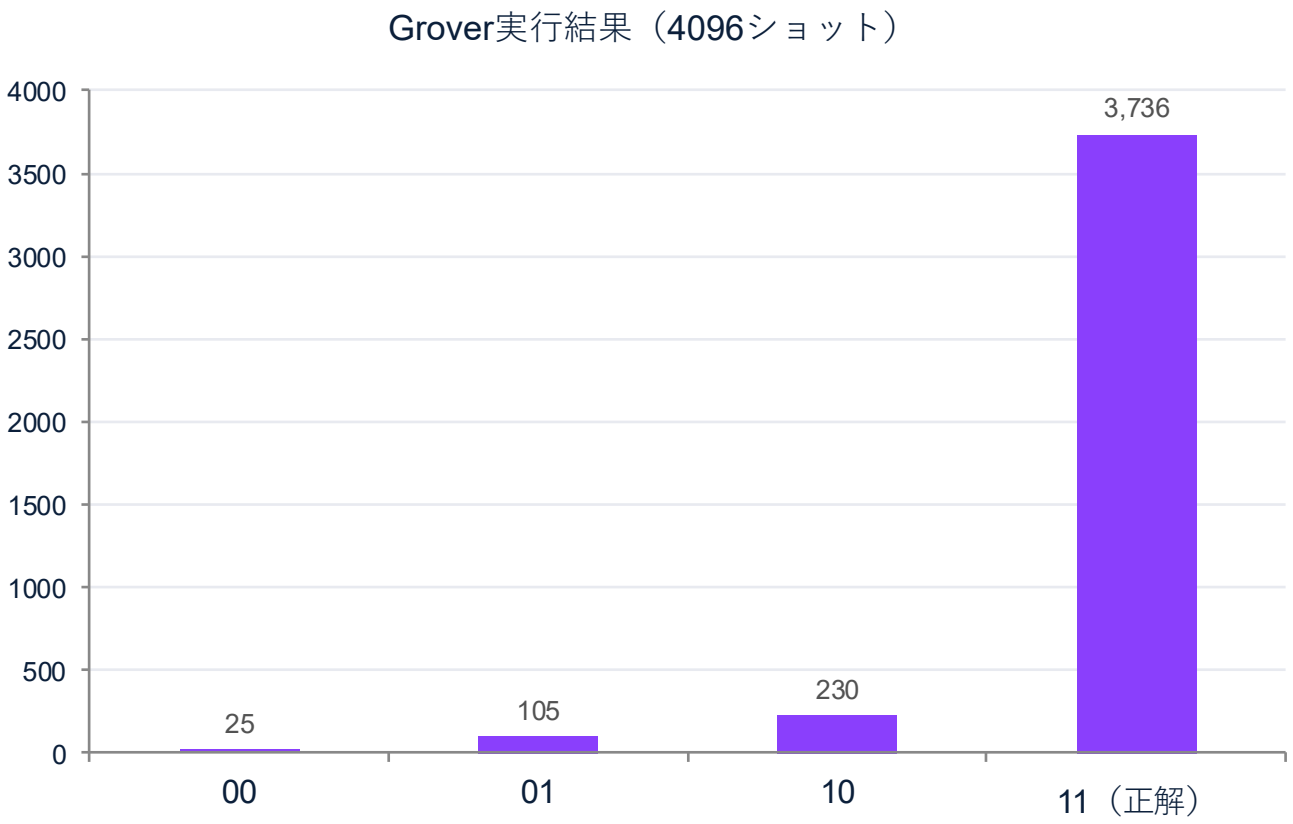
状態ベクトルで見るGroverの動き

	00	01	10	11
初期状態	1	0	0	0
① 重ね合わせ後	1/2	1/2	1/2	1/2
② オラクル後	1/2	1/2	**1/2**	-1/2
③ 拡散演算子後	0	0	**0**	-1

最終ベクトルは (0, 0, 0, -1) = -|11⟩。符号は測定確率に影響しないため、理論上は100%の確率で「11」が観測される。実機の結果が91%だったのは、ここにゲート誤差などのノイズが加わったため。

STEP 6 — 応用

結果：量子探索はランダムより速い



約91%

の確率で正解「11」が観測された

ランダム探索なら正解率は25%。量子探索により約3.6倍見つけやすくなった。理論上の上限は100%で、残りの誤差はゲートエラーなど実機ノイズによるもの。

SUMMARY

今日、体験したこと

- IBM Quantum Platformでアカウントを作り、APIキーとCRNで実機に接続した
- Pythonの仮想環境にQiskitを整え、認証情報を一度だけ保存した
- ベル状態の回路を実機に送信し、量子もつれと実機ノイズの両方を実測した
- Groverのアルゴリズムで、量子探索がランダムより速いことを自分の数字で確認した

NEXT STEPS

さらに試してみよう

シミュレータと比較

AerSimulatorでノイズなしの理想結果を見て、実機との差を体感する

量子ビット数を増やす

3つ以上の量子ビットでもつれや探索を試し、ノイズの積み重なりを見る

別のオラクルを作る

Groverの正解を11以外に変えて、探索対象を自由に変更してみる

エラー軽減を試す

動的デカップリングや測定誤り軽減オプションでノイズを抑える