古典プログラマ向け
量子プログラミング入門

- 量子ゲート編 -

OsSAL.org サル量子部

https://www.ossal.org/qc/

宮地直人（miyachi@langedge.jp）

Ver1.0 2019年9月11日

属性： @le_miyachi

技術：古典PKIプログラマ

仕事：ぼっち有限会社（電子署名系）

量子：独自に勉強（書籍・勉強会）

趣味：勉強会の開催、OSS開発

活動：OsSAL.org （オッサル）他

オープンソース署名＆認証ラボ

Part 0：イントロダクション（プロローグ）

Richard Feynman（リチャード ファイマン）

“If you think you understand quantum mechanics, you don’t understand quantum mechanics.”

「もしあなたが量子力学を理解できたと思うならば、それは量子力学を理解できていないと言うことだ。」

※ と言うことできっと私も理解できていないので間違い等ご指摘を！

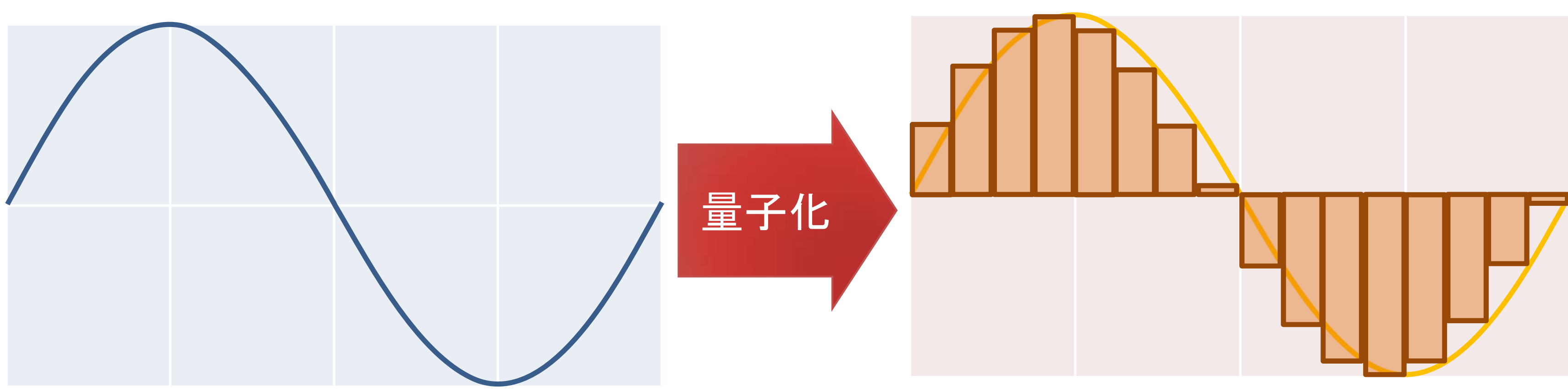
量子とは？

一般的な定義：

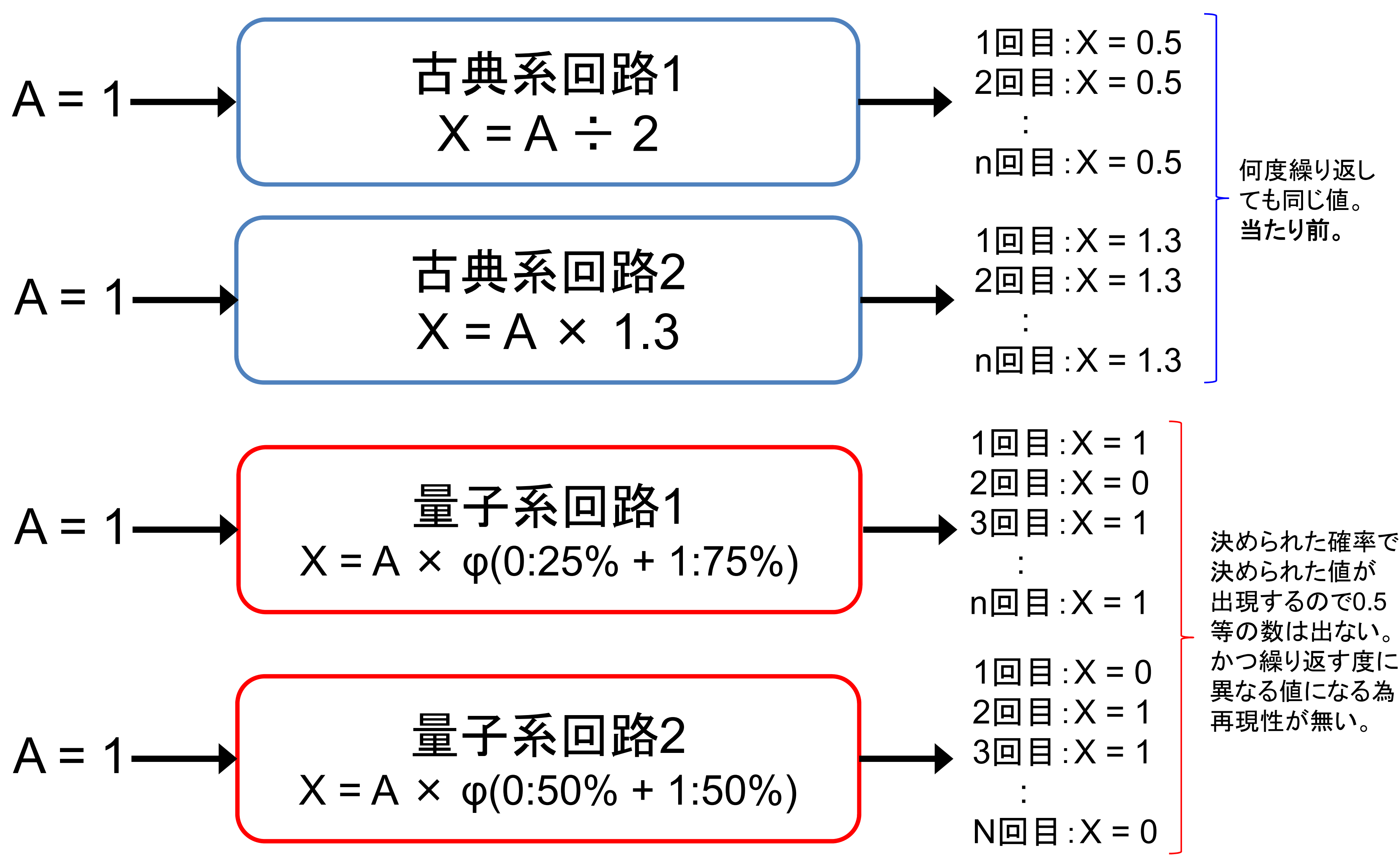
物理量を実数では無く単位量の整数倍で表す場合にその単位量を**「量子」**と呼ぶ。

※ 量子論/力学以外でも使われる場合がある。

例：アナログ波形のデジタル化を量子化と呼ぶ。



古典系と量子系の測定



量子系（量子論の世界）

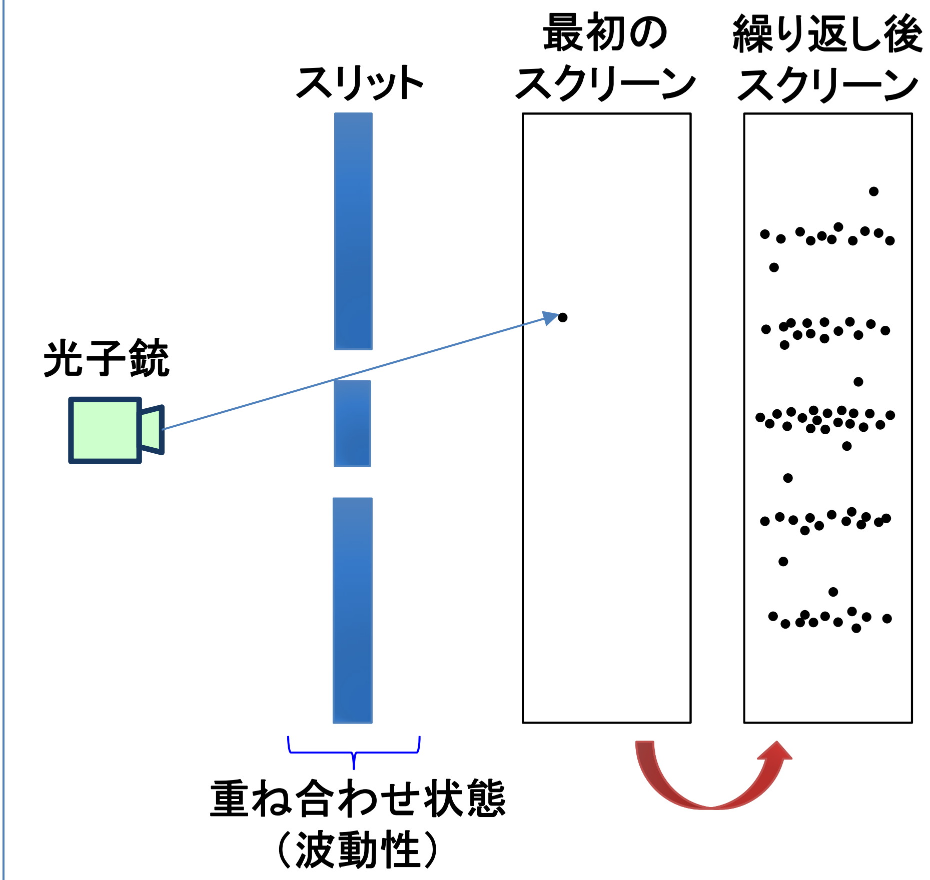
- 量子論で記述された（量子）系が取る状態。
- 古典論では全く同じ系で測定された物理量は毎回同じ測定値（実数）を示す。 量子の対語は古典
- 量子論では全く同じ系で測定された物理量は毎回異なった測定値を示す。
- 量子状態の測定値は実数では無く固有値によるとびとびの値を取る（だから量子）。
- どの測定値となるかは波動関数により決まる確率分布にて示される（コペンハーゲン解釈）。
- 一般に原子以下のミクロな世界は量子状態。

二重スリット実験（光の波動性）

二重スリット通過後に干渉縞が表示される。
と言うことは光に波動性があることを示す。
干渉縞は同じ位相を持つ光源が2つあり、
距離により位相が強めあう場所と弱めあう
場所がある為に生じる現象である。

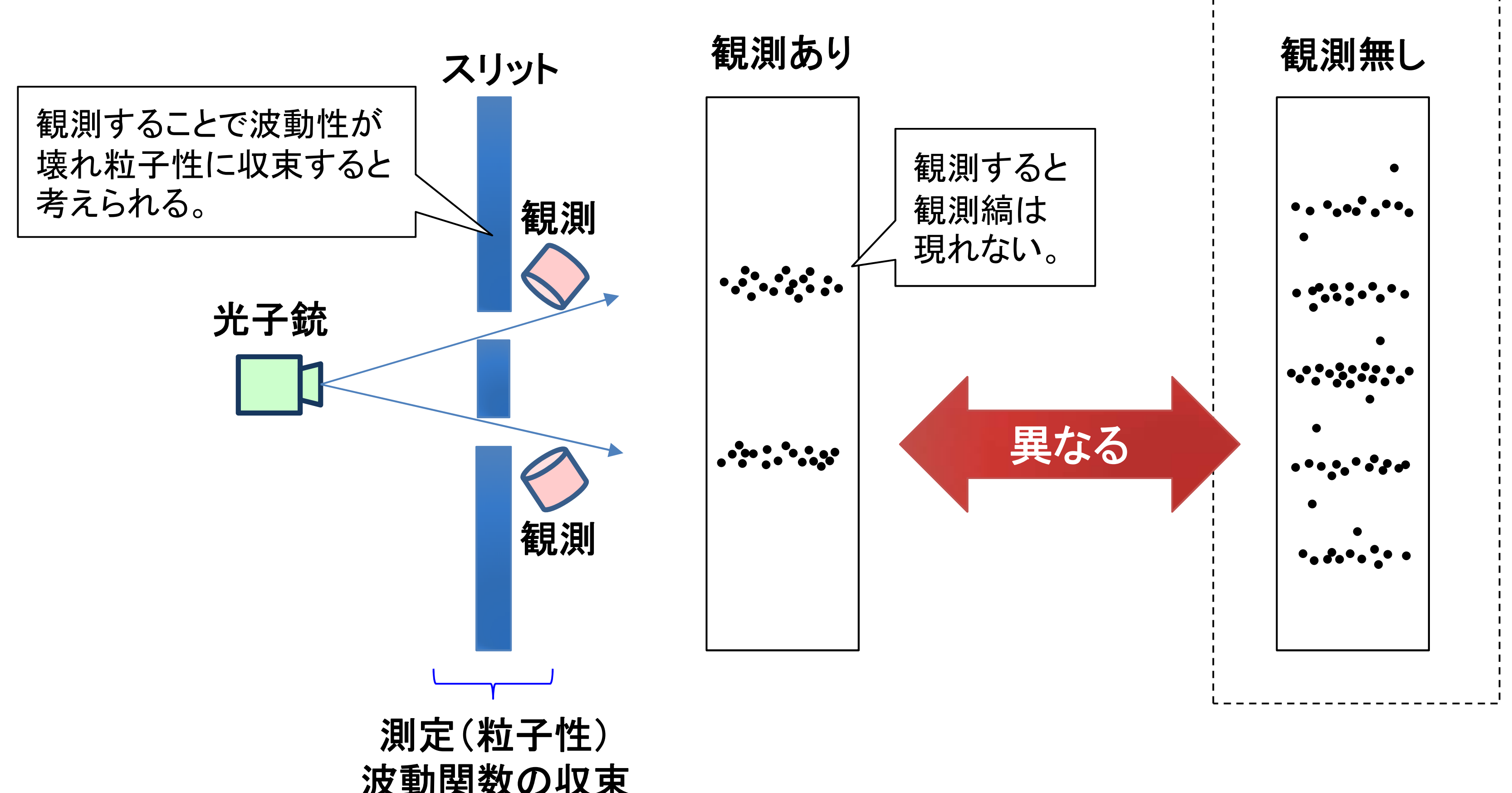
スリット
スクリーン
光源
干渉縞

光子1個ずつ発射可能な光子銃と光子1個を
認識できるスクリーンを使う。光子1個ずつを
打ち出しても繰り返すうちに干渉縞が出てくる。
つまり光子1個でも波動性があることになる。



二重スリット実験の観測問題

観測問題：どちらのスリットを通過したかを観測することにより干渉縞が出なくなる。
コペンハーゲン解釈では観測することで量子干渉が壊れて粒子として収束していると考える。
しかし他にも解釈は存在しており（平行宇宙論等）、正確なことは分かっていない。



波動と粒子の二重性（量子重ね合わせ）

アナログ的（複数状態の並列計算）

観測しなくても
時間が経つと崩壊する

デジタル的（計算結果：0 or 1）

観測

重ね合わせ状態（波動性）

固有値状態（粒子性）

収束

量子デコヒーレンス

複数の状態が同時に存在して
量子干渉を生じている
（コペンハーゲン解釈）

波動関数（列ベクトル）を
|ψ> と表現すると

$$|\psi> = c_0|\psi_0> + c_1|\psi_1>$$
$$|c_0|^2 + |c_1|^2 = 1.0$$

c_0 と c_1 は複素数

固有状態のうち1つの状態

$|\psi_0\rangle$ または $|\psi_1\rangle$ のいずれか

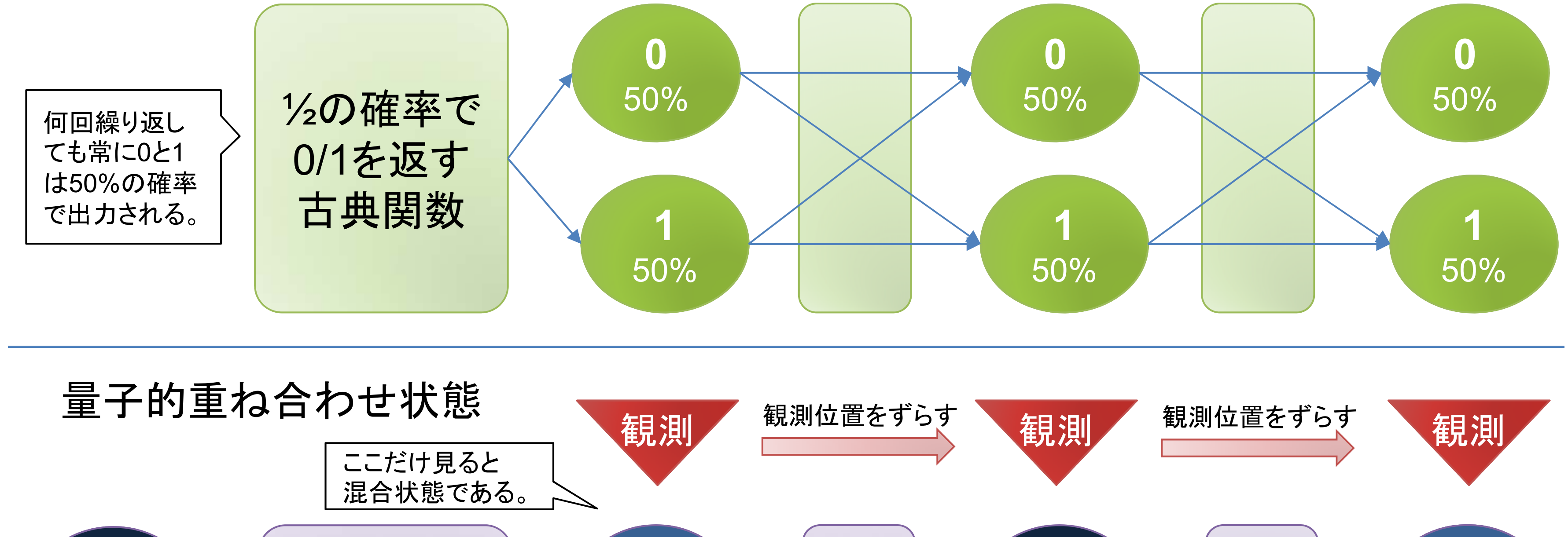
波動性は失われる
複素数虚数部は意味を持たない

コペンハーゲン解釈では、粒子状態
は波動がパルス状になったと考える。
位置が決まると波長は定まらなくなる。

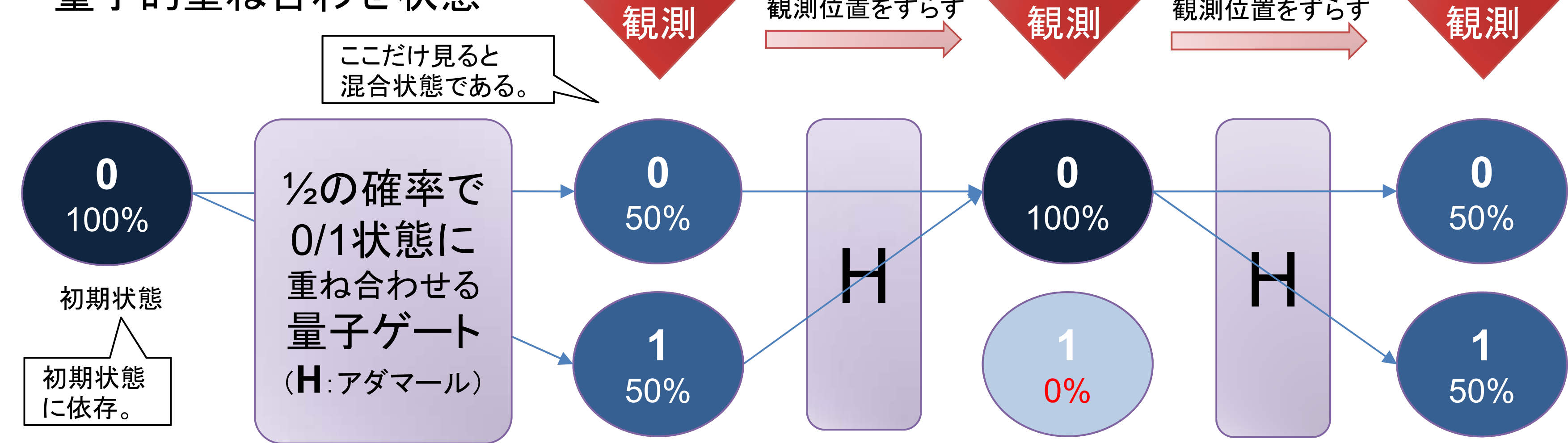
※ コヒーレンス時間（状態の量子干渉が失われるまでの時間）は通常短い。

量子的重ね合わせと古典的混合状態

古典的混合状態



量子的重ね合わせ状態



※ 量子には波動性がある為に確率以外の要素（波動の位相）がある。

量子ビット（重ね合わせの実現）

- 0と1の量子重ね合わせの状態が可能な単位。
- 観測により2つの固有値（0か1）に収束する。
- 物理的に実現するには幾つかの方法がある。

方式	概要	開発
超伝導 量子ビット	現在主流となっている超伝導状態のシリコン回路で量子ビットを実現する方式（極低温）	IBM, Google, D-Wave
イオントラップ	捕獲したイオンをレーザーで冷却して利用（室温） 理論的には量子ビット間の全結合が可能	IonQ
量子ドット	原子10～50個で構成した微小半導体を利用（極低温?）	Intel
トポロジカル	超電導体とトポロジカル絶縁体による量子ビット（極低温?）	Microsoft
NVセンター ダイヤモンド窒素-空孔中心	ダイヤモンドの炭素を窒素に置き換えて生じる欠損部に電子を捕獲して量子ビットに利用（室温）	（研究レベル）
QNN 量子ニューラルネットワーク	光パルスを量子ビットとして利用（室温） 全結合によるアニーリング型の計算が可能（らしい）	NTT/NII/東大 （ImPACT）

可逆コンピュータ

可逆コンピュータは非量子な場合には低電力になっても低性能になる為にほぼ実用化されていない。

情報が失われる時にエネルギー(熱)が消費される。

可逆コンピュータが実現すると低電力化が可能となる。

IBMから1960年代に出た理論で今も研究が続いている。

※ 量子コンピュータは可逆コンピュータの一種である(詳細は次ページ)。

NOTゲート	ANDゲート	CNOTゲート

NOTゲート

A	X
1	0
0	1

出力から入力が再現できるので可逆

可逆

ANDゲート

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

出力から入力が再現できない

非可逆(量子計算では使えない)

CNOTゲート

A	B	X	Y
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Yだけ見るとXOR

可逆

なぜ量子コンピュータは可逆なのか？

量子計算：

1. 量子力学の計算にはシュレディンガー方程式を利用。
2. シュレディンガー方程式は時間に可逆な性質を持つ。

従って：

シュレディンガー方程式を使う量子計算も可逆となる必要がある。

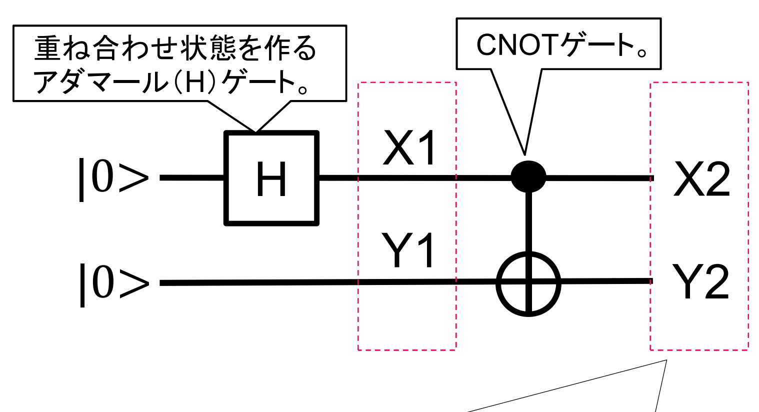
つまり：

（ゲート型）量子コンピュータは可逆コンピュータ。

※ 後ほどもう少し詳しくシュレディンガー方程式を見ます。

量子もつれ（量子エンタングルメント）

- 量子もつれは、2つの粒子（量子ビット）が互いに影響をおよぼし合い、一方を測定すると、もう一方の値が確定する現象である。
- 複数の量子ビット間を、量子もつれにより関連付けることで量子回路を構築する。以下HとCNOTによる例。



$X1 = |0\rangle : 50\% + |1\rangle : 50\%$
$Y1 = |0\rangle$

$X2 = X1 = |0\rangle : 50\% + |1\rangle : 50\%$
X1が $|0\rangle$ ならY2も $|0\rangle$ （Y1そのまま）
X1が $|1\rangle$ ならY2も $|1\rangle$ （Y1を反転する）

X2が0ならY2も0に、Y2が1ならX2も1と、観測

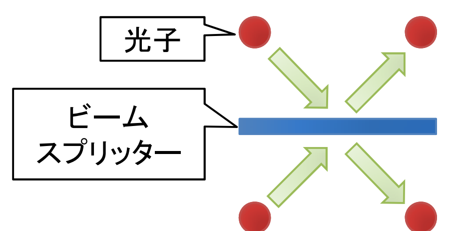
$X2Y2 = |00\rangle : 50\% + |11\rangle : 50\%$
※ $|01\rangle$ や $|10\rangle$ は0%

光の量子コンピュータ

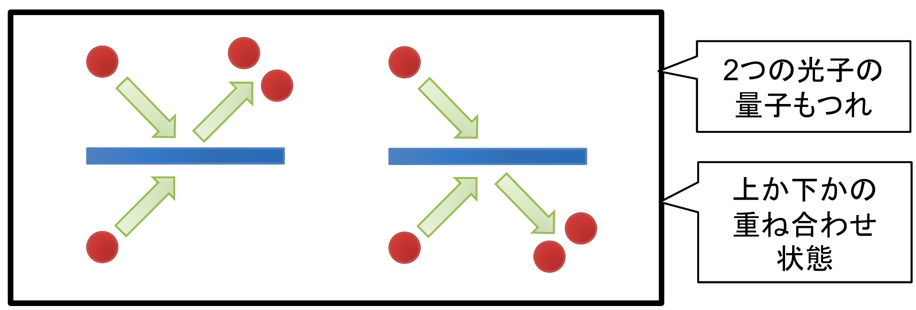
上下から光子対を入射する。

- ビームスプリッターとはハーフミラーのこと。

それぞれ反射
または透過



片方が反射し
もう片方が透過



※ 測定するとこちらのいずれかになる

ループ型光量子コンピュータ

1つの可変式ビームスプリッターを何度も使うことで量子計算を行っていく。まだ完成していない。

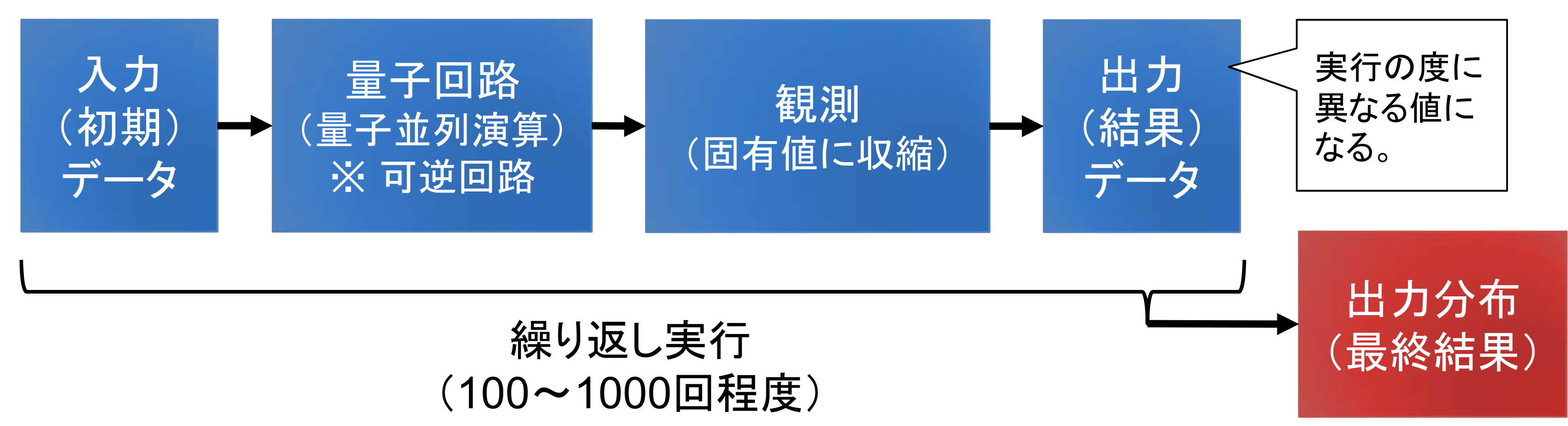
複数のビームスプリッターを使うことで量子計算を行っていくタイプもある。

外ループ
内ループ
測定結果で
光操作
可変式
スプリッター
出力
ホモダイン測定器
入力:
光パルス発生器
光スイッチ

量子コンピュータの概要

1. 0と1の2つの基底を持つ複数の量子ビットを利用。
2. 重ね合わせた状態のままで量子並列演算を行う。
3. 量子ビット同士を絡み合わせて量子回路を構築。
4. 重ね合わせ状態の出力を観測して固有値に収縮。
5. 繰り返し実行し確率的な固有値の出力分布を得る。

→ 二重スリット実験時に光子の分布により干渉縞を確認するようなもの。



量子コンピュータの種類

量子ゲート型（狭義の量子コンピュータ）

方式：量子ゲートの量子回路による量子計算

対象問題：汎用（ただし量子アルゴリズムの範囲内）

開発企業：IBM/Google/Intel/Alibaba/Microsoft等

量子アニーリング型（正確には量子シミュレータ）

方式：イジングモデルを使った量子シミュレーション

対象問題：最適化問題特化（深層学習等への応用）

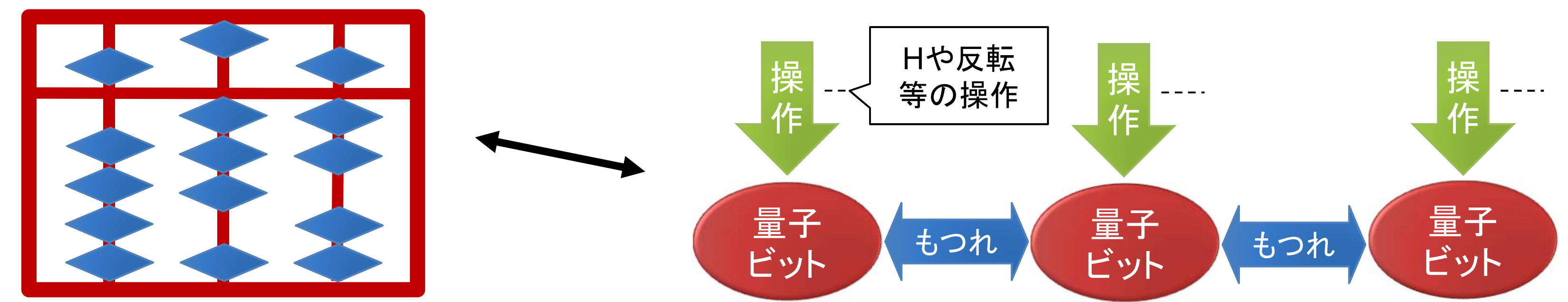
開発企業：D-Wave（非量子型では富士通と日立）

※ 非量子：富士通「デジタルアニーラ」、日立「CMOSアニーリングマシン」。

※ 他に光を使ったCIM（コヒーレントイジングマシン）もあるがここでは省略。

量子ゲート型とソロバン

ソロバンは珠（たま）を配置したハードウェアを、指で弾いて行くことで計算を進めて行く。各珠の間には桁上がり等の関連性がある。



量子ゲート型は量子ビットを配置したハードウェアを、レーザーや電界で弾いて行くことで計算を進めて行く。各量子ビット間には量子もつれによる関連性がある。

ソロバンは可逆回路でもある。ただしソロバンは同じ操作をすれば毎回同じ値になるが、量子では異なる。

量子アニーリング型の計算

どちらも問題をイジングモデルとして定式化する必要があり数学が必要。

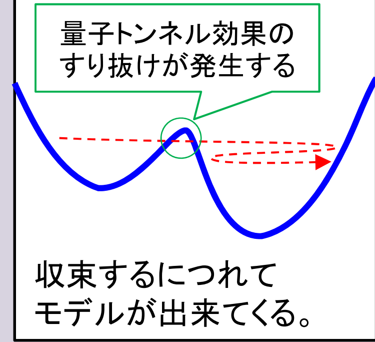
量子アニーリング

エネルギー小

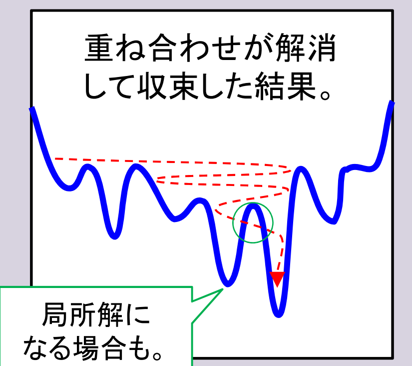
初期状態

量子ビット間をもつれ合わせて重ね合わせ状態を作るので最初はフラットなモデル。

中間状態



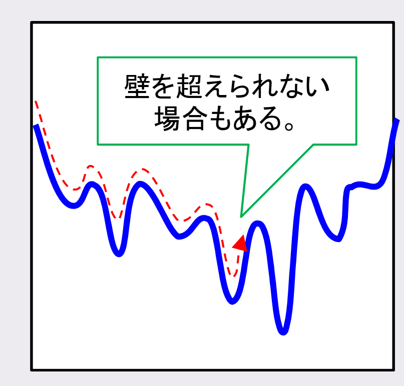
最終状態



時間と共に量子重ね合わせ状態が解消されていく（横磁場を弱めて行く）

量子アニーリングでは量子ビット間の全結合（もつれ）が理想だが…

シミュレーテッドアニーリング（SA）



シミュレーテッドアニーリング（SA）は先にモデルをセットして最適解を求めて行く。

シミュレーテッドアニーリングは古くから使われており、現在でも富士通や日立が専用ハードウェアを使ったアニーリングを実現（例：デジタルアニーラ）している。

NISQの時代（今後5～10年程度）

Q2B: QUANTUM FOR BUSINESS 2017

物理学者 John Preskill による基調講演の論文

「*Quantum Computing in the NISQ era and beyond*」

https://arxiv.org/abs/1801.00862

NISQ: Noisy Intermediate-Scale Quantum

ノイズエラーがあり中規模量子ビット数の時代

50～数百量子ビット程度

現在は標準ハードウェアと標準ソフトウェア（アルゴリズム）を確立する時期で、特にソフトウェアは量子シミュレータを使って勉強しておくことで量子プログラミング時代に備える。

量子コンピュータの未来予想

どこかでブレークスルーを生じる可能性はゼロではないが…

量子超越
（古典コンピュータでは解けない問題）

RSA暗号解読

2015　2019　2023～2030　2040～2060くらい（予測困難）

	NISQ *ノイズあり中規模*	*量子加速* *（Quantum Speedup）*	*万能（Universal）量子コンピュータ*
実用性 汎用性	ほぼない（一部を除き古典機の方が早い）	一部あり（古典機よりも早い分野がある）	あり（ただし得意不得意はある）
信頼性	ノイズ（エラー）あり	（部分的な）エラー訂正	エラー訂正あり
量子数	$10～10^2$	（エラー訂正あり）$10^2～10^6$	10^6（数百万）以上
利用形態	専用機	専用機・外部接続装置	単独・古典機と融合

参考：（ノイマン型）古典コンピュータの歴史

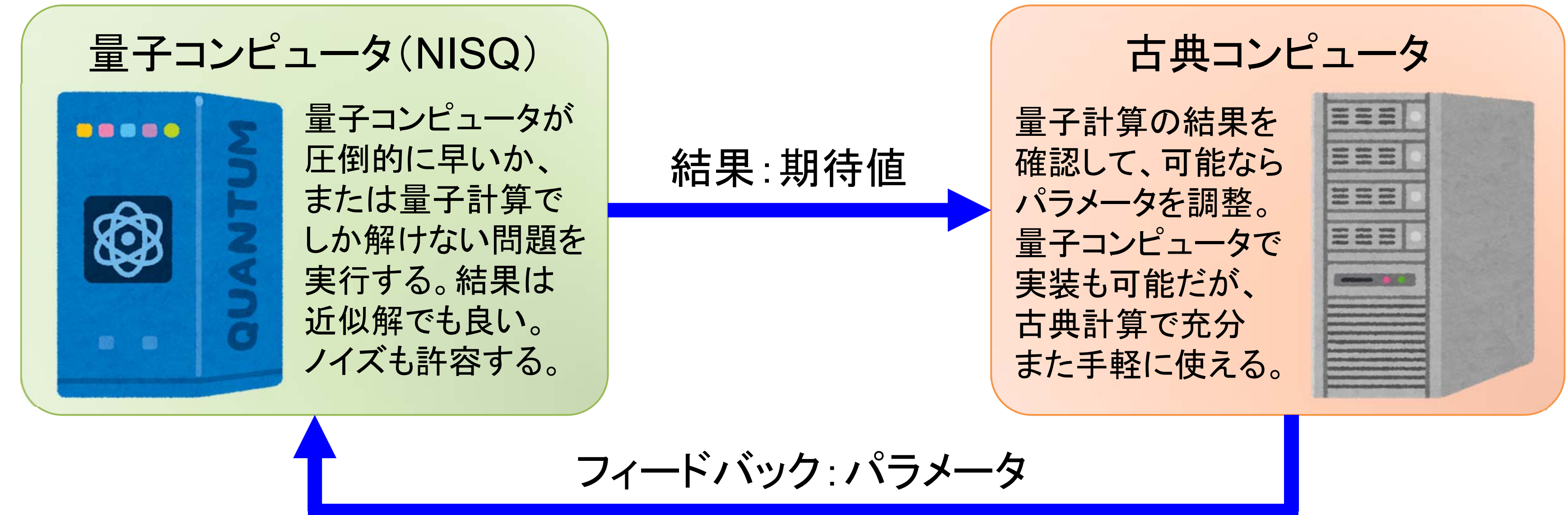
同じような経緯か？

1950　1955　1970

第1世代	第2世代	第3世代
真空管・試用や研究	トランジスタ・商用化	集積回路・パーソナル化

古典量子ハイブリッドアルゴリズム

現実的な利用方法として古典コンピュータとの組合せ利用が進んでいる。
第2部で説明するショアのアルゴリズムもある意味ハイブリッド計算である。



主な用途:

- 近似最適化:QAOA (Quantum Approximate Optimization Algo)
- 基底状態探索:VQE (Variational Quantum Eigensolver)
- 機械学習:QCL (Quantum Circuit Learning)

量子計算フレームワーク（量子ゲート型）

IBMやGoogleは自社量子コンピュータを使う為の量子計算フレームワークを公開している。実機だけではなくシミュレーション機能を持っているので、量子プログラミングの勉強用として最適だが小規模の量子回路のみとなる。

項目	IBM Qiskit	Google Cirq
ロゴ	Quantum Information Science Kit	Cirq
構成	Terra: 量子計算の基盤部（Python） Aqua: 量子アルゴリズムのライブラリ OpenQASM: 量子低レベル中間言語	Cirq:量子計算基盤Pythonライブラリ OpenFermion: 量子化学ライブラリ
提供	オープンソース（GitHub）	オープンソース（GitHub）
取得	https://qiskit.org/ https://github.com/Qiskit	https://github.com/quantumlib/Cirq
情報	https://qiskit.org/documentation/ja/	https://cirq.readthedocs.io/en/latest/
その他	IBM Q Experience: GUI利用 ※ GUIからOpenQASMに展開し実行 https://quantumexperience.ng.bluemix.net/	2018年夏に正式公開されたライブラリ 量子コンピュータ実機はまだ使えない

量子計算フレームワーク（量子アニーリング型）

日本のベンチャーが開発した量子計算フレームワークも公開されている。
D-Wave社のOceanもQiskit/Cirqと同じく基本的には自社ハード用SDKである。

項目	Bluegat	D-Wave Ocean SDK
ロゴ	Blueqat	D:Wave The Quantum Computing Company™
構成	量子ゲート型の計算（本来ゲート型） 量子アニーリング計算も可能	量子アニーリング型の計算
提供	オープンソース（GitHub）	オープンソース（GitHub）
取得	https://github.com/Blueqat	https://github.com/dwavesystems/dwave-ocean-sdk
開発	株式会社 MDR https://mdrft.com/?hl=ja	D-Wave Systems, Inc.（カナダ） https://www.dwavesys.com/
その他	Qiskit/Cirqよりも回路記述が簡単。 前身は量子アニーリング用のWildqat。 日本語のSlackも参加可能。 勉強会・ブログ等で積極的に日本語で情報発信をしている。	D-Waveマシン用のSDK。 シミュレーテッドアニーリング計算も可能。 計算に実機を使う為には登録が必要。 D-Wave Leap https://www.dwavesys.com/take-leap

量子計算シミュレータ

富士通のデジタルアニーラや、
日立のCMOSアニーリングも、
ある意味ではシミュレータである。

今、我々に必要なもの：

エラーなし高速の量子計算シミュレータ

目的：量子アルゴリズムの学習の為

量子コンピュータの実機とシミュレータの比較：

実機（NISQ）：エラーがあり小規模（各種制限あり）のみ

シミュレータ：エラーなしで中規模（制限少）、ただし限界あり

※ シミュレータもNISQ計算のQiskit/Cirqはその意味では向いていない。

- Qulacs（QunaSys社）：C++実装で高速化
- QGATE（個人の趣味）：GPUを使って高速化

https://numba.pydata.org/ （Blueqat 0.3.9 の numba バックエンドで利用）

古典プログラマ向け量子プログラミング入門 [量子ゲート編] 目次:

Part 1: 関連数学と1量子ビット操作

1-1: 線形代数学の基本知識

1-2: ブラケット記法と量子計算

1-3: ブロッホ球と1量子ビット操作

1-4: IBM Q

Part 2: 量子ゲート型のプログラミング

2-1: 複数量子ビット操作

2-2: 量子アルゴリズムの基本

2-3: ドイチェ アルゴリズム

2-4: グローバー検索（量子検索）

2-5: 量子フーリエ変換

2-6: ショアのアルゴリズム

2-7: エラー訂正問題

2-8: Cirq（Google）・Blueqat（MDR）

2-9: 量子ゲート編 付録

Part 1：関連数学と1量子ビット操作

参考: シュレディンガー方程式と波動関数

※ 理解できなくても量子プログラミングでの大きな問題にはなりません

(時間依存する)シュレディンガー方程式

$\hbar$: プランク定数h / 2π
m : 質量

$$i\hbar\frac{d}{dt}\psi(x,t) = H\psi(x,t) = \left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right)\psi(x,t)$$

H:ハミルトニアン
時間から見た全エネルギー
Ψ:波動関数
運動エネルギー
ポテンシャルエネルギー
プサイ

シュレディンガー方程式は、波動関数の値を求める方程式ではなく、
波動関数の式そのものを求める方程式となっている。

古典力学では粒子のエネルギーは保存される(時間依存が無い)ので、
時間発展しないシュレディンガー方程式を導くことができる。

$$E\phi(x) = H\phi(x) = \left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right)\phi(x)$$

エネルギー固有値
φ:波動関数
(時間依存無し)
ファイ

http://scienceinoh.jp/schrodinger/

量子計算は複素数計算が必要

あなたとわたしの
シュレディンガー

シュレディンガー方程式

$$i\hbar\frac{d}{dt}\psi(x,t) = H\psi(x,t)$$

左辺に虚数単位 i が出てくる。
この方程式を成立させる為には右辺の H が実数であるので、波動関数 ψ が複素数で表現されなければならい。

波動力学と行列力学

時間発展しないシュレディンガー方程式（波動力学）

$$H\varphi(x) = E\varphi(x)$$

Hはハミルトニアン式、φ(x)は波動関数（固有関数）、Eはエネルギー固有値

行列における固有値問題の式

$$Ax = \lambda x$$

Aは正方行列、xは列ベクトル（固有ベクトル）、λはスカラー値（固有値）

量子計算は行列の固有値問題として解くことができる。
これをハイゼンベルク方程式（行列力学）と呼ばれる。
この場合に波動関数は固有ベクトルとして求められる。
※ 波動力学と行列力学が数学的に同じであることはシュレディンガーにより確認されている。

量子の状態はベクトルで表現できる

量子系の状態ベクトル

n次元量子の状態 = $\begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{n-1} \end{pmatrix}$ $= C_0|0\rangle + C_1|1\rangle + \dots + C_{n-1}|n\text{-}1\rangle$

基底: $|0\rangle, |1\rangle, \dots, |n\text{-}1\rangle$

ただし $|C_0|^2+|C_1|^2+|C_{n-1}|^2 = 1$

ただし C_0〜C_{n-1}は複素数。

量子ビットの状態 = $\begin{pmatrix} C_0 \\ C_1 \end{pmatrix}$ $= C_0|0\rangle + C_1|1\rangle$

基底: $|0\rangle$と$|1\rangle$

ただし $|C_0|^2+|C_1|^2 = 1$

それぞれ確率50%なら: $C_0 = C_1 = \frac{1}{\sqrt{2}}$

$|0\rangle$を得る確率は$|C_0|^2$
$|1\rangle$を得る確率は$|C_1|^2$

1-1: 線形代数学の基本知識

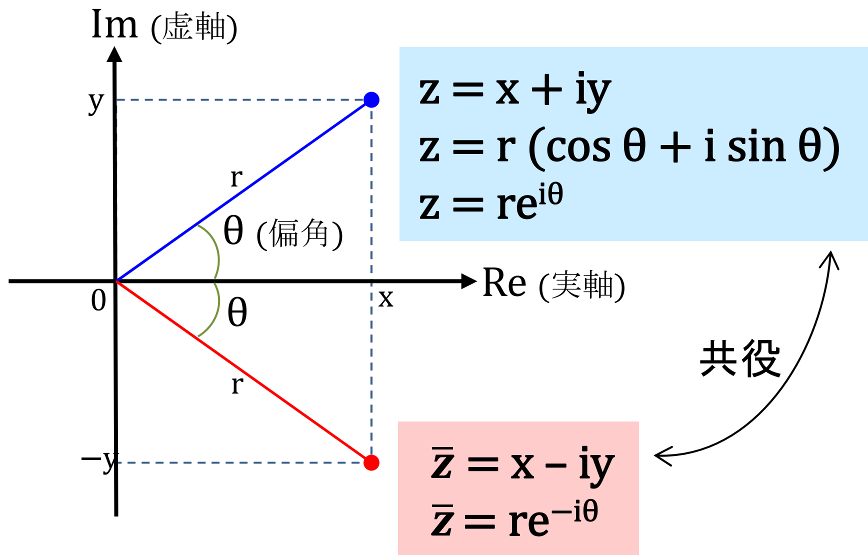
量子計算が行列演算で解けるということは、
行列演算の線形代数学の知識が必要です。

このパートは線形代数学をマスターしている人は
読み飛ばして頂いて構いません。

数学: 複素数と共役(複素共役)

名称	定義	例
実数	実在する数	$0, 2, 0.4, 1/3, \sqrt{2}$
虚数	2乗してマイナスになる数 i をつけて表す	$i = \sqrt{-1}, i3 = \sqrt{-3}, -i4 = -\sqrt{-4}$ $i^0 = 1, i^1 = i, i^2 = -1, i^3 = -i$
複素数	実数と虚数で示される数 実数部/虚数部が0も含む	$5 + i3, 2 - i4, -0.2 + i0.2,$ $-2 + i0 = -2, 0 - i3 = -3i$

複素平面における共役関係



複素共役(ふくそきょうやく)

虚数部がマイナスになっているペア値。

複素数 z に対して $\bar{z}$ と書く。

1. z が実数なら、$z = \bar{z}$
2. $\overline{(z_1 + z_2)} = \bar{z}_1 + \bar{z}_2$
3. $\overline{(z_1 \times z_2)} = \bar{z}_1 \times \bar{z}_2$
4. $\overline{(z_1 \div z_2)} = \bar{z}_1 \div \bar{z}_2$
5. $\bar{z}\, z = |z|^2$

数学: オイラーの公式

複素数の場合には
指数関数と三角関数
の相互変換が可能。

指数関数を三角関数で表す。

$$e^{i\theta} = \cos\theta + i\sin\theta$$

量子ビットの位相
を扱う為に重要

オイラーの公式の複素共役。

$$e^{-i\theta} = \cos\theta - i\sin\theta$$

三角関数を指数関数で表す。

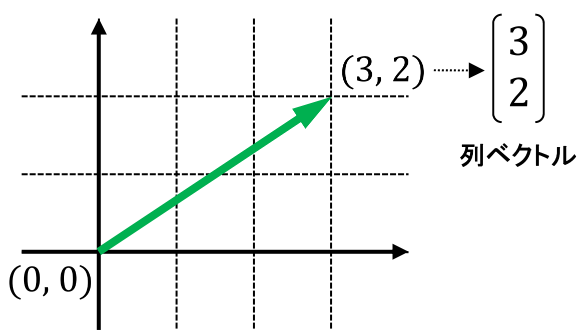
$$\sin\theta = \frac{e^{i\theta} - e^{-i\theta}}{2i}, \quad \cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

数学: ベクトルと基底ベクトル

ベクトル:始点/終点と向き/長さを持つ

- n次元の1行のみの行ベクトルや1列のみの列ベクトルとして扱う。
- 量子計算では原点を始点としたベクトルのみを扱う(線形変換)。
- 量子計算では座標を示す数は複素数。

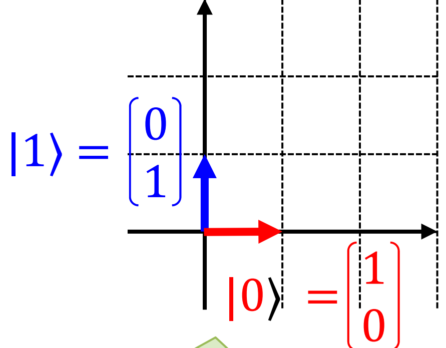
ベクトルと列ベクトル表記



基底ベクトル:基本単位となるベクトル

- 量子計算は $|0\rangle$ と $|1\rangle$ を規定ベクトルとする。「$|\psi\rangle$ (ケット)」に関しては後述。
- 規定ベクトルを基準にすることで、移動や回転が表現できる(線形変換)。
- 量子計算では回転のみを利用する。

基底ベクトル



注:値は複素数
例:$1 = 1 + i0$

数学: ベクトルの演算

スカラー倍:

$$m\,(a_1\ a_2\ \dots\ a_n) = (ma_1\ ma_2\ \dots\ ma_n)\quad , \quad m\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} mb_1 \\ mb_2 \\ \vdots \\ mb_n \end{pmatrix}$$

ベクトル同士の和と差（同じ次元数同士のみ）:

$$(a_1\ a_2\ \dots\ a_n) + (b_1\ b_2\ \dots\ b_n) = (a_1+b_1\ a_2+b_2\ \dots\ a_n+b_n)$$

$$(a_1\ a_2\ \dots\ a_n) - (b_1\ b_2\ \dots\ b_n) = (a_1-b_1\ a_2-b_2\ \dots\ a_n-b_n)$$

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1+b_1 \\ a_2+b_2 \\ \vdots \\ a_n+b_n \end{pmatrix}\quad , \quad \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1-b_1 \\ a_2-b_2 \\ \vdots \\ a_n-b_n \end{pmatrix}$$

数学: ベクトルの内積

$$\text{行ベクトル・列ベクトル} = (a_1\ a_2\ \ldots\ a_n) \cdot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

同じ次元間のみ
内積計算が可能

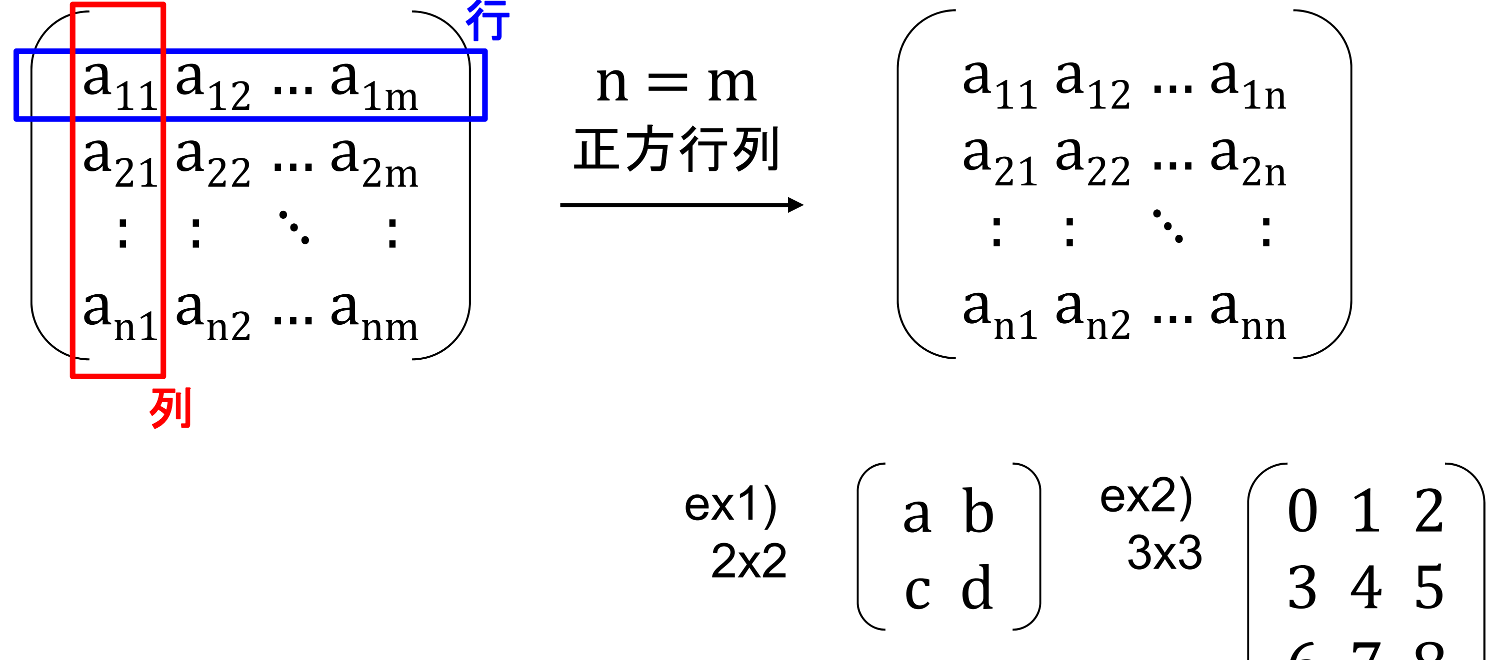
$$= a_1b_1 + a_2b_2 + \ldots + a_nb_n$$

$$= \sum_{j=0}^{n} a_jb_j$$

内積の結果はベクトル
ではなくスカラー値になる
ただし複素数なので注意

数学: 行列と正方行列

行列とは数・記号・式などを縦(列)と横(行)に並べたもの。
列数と行数が同じ場合には正方行列と呼び、量子計算においては正方行列のみを理解していれば良い。



数学: 行列の演算

スカラー倍:

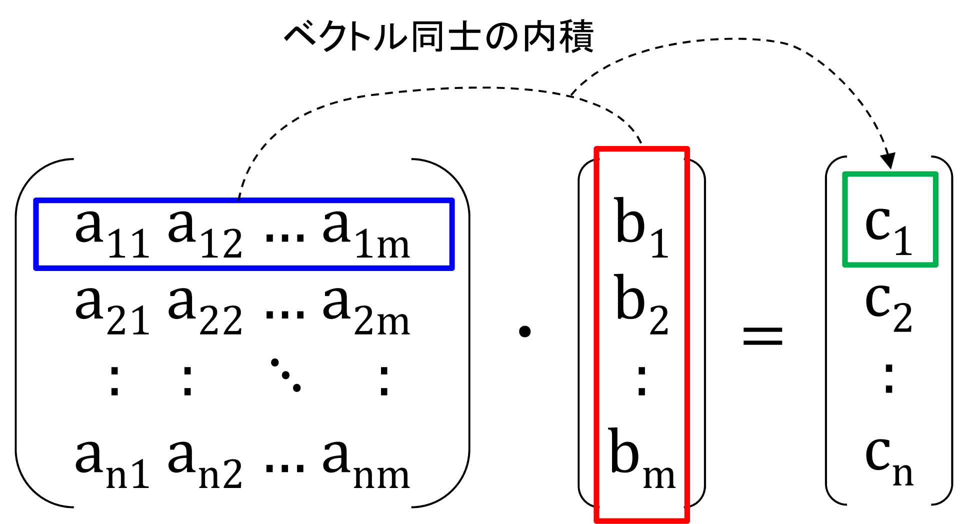
$$\lambda \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} \lambda a_{11} & \lambda a_{12} & \dots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \dots & \lambda a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda a_{n1} & \lambda a_{n2} & \dots & \lambda a_{nn} \end{pmatrix}$$

行列同士の和（同じ次元数同士のみ）:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nn}+b_{nn} \end{pmatrix}$$

数学: 行列と列ベクトルの内積

n行m列とm次の列ベクトルの内積はn次の列ベクトルになる:

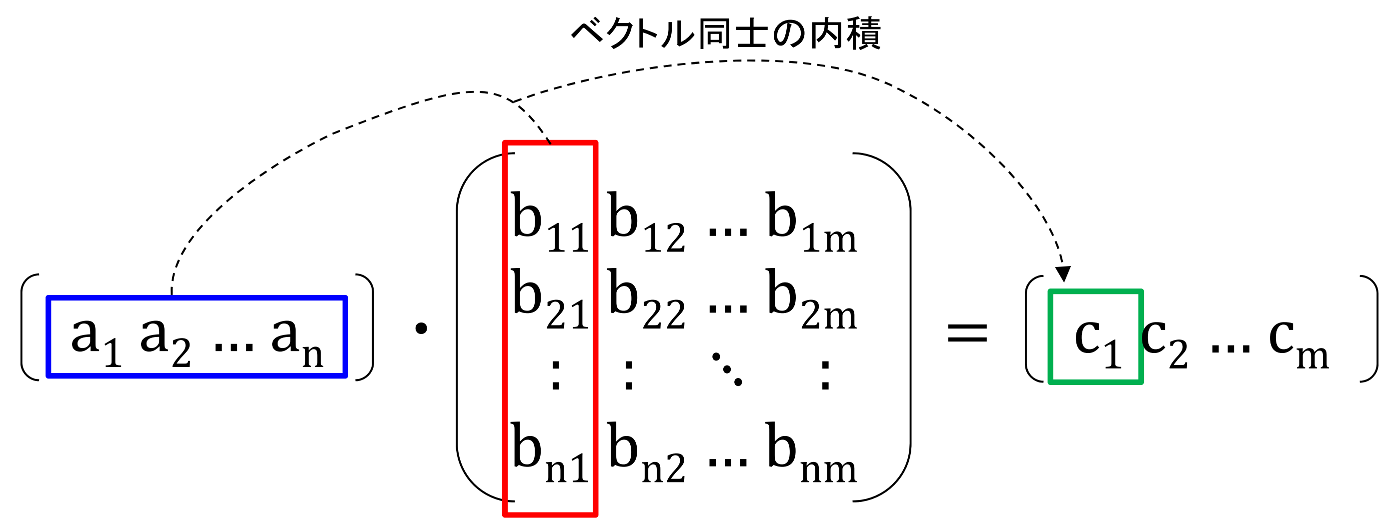


$$c_1 = a_{11}b_1 + a_{12}b_2 + \dots + a_{1m}b_m$$
$$c_2 = a_{21}b_1 + a_{22}b_2 + \dots + a_{2m}b_m$$
$$\vdots$$
$$c_n = a_{n1}b_1 + a_{n2}b_2 + \dots + a_{nm}b_m$$

$$C_1 = \sum_{j=1}^{m} a_{1jbj}$$

数学: 行ベクトルと行列の内積

n次の行ベクトルとn行m列の内積はm次の行ベクトルになる:

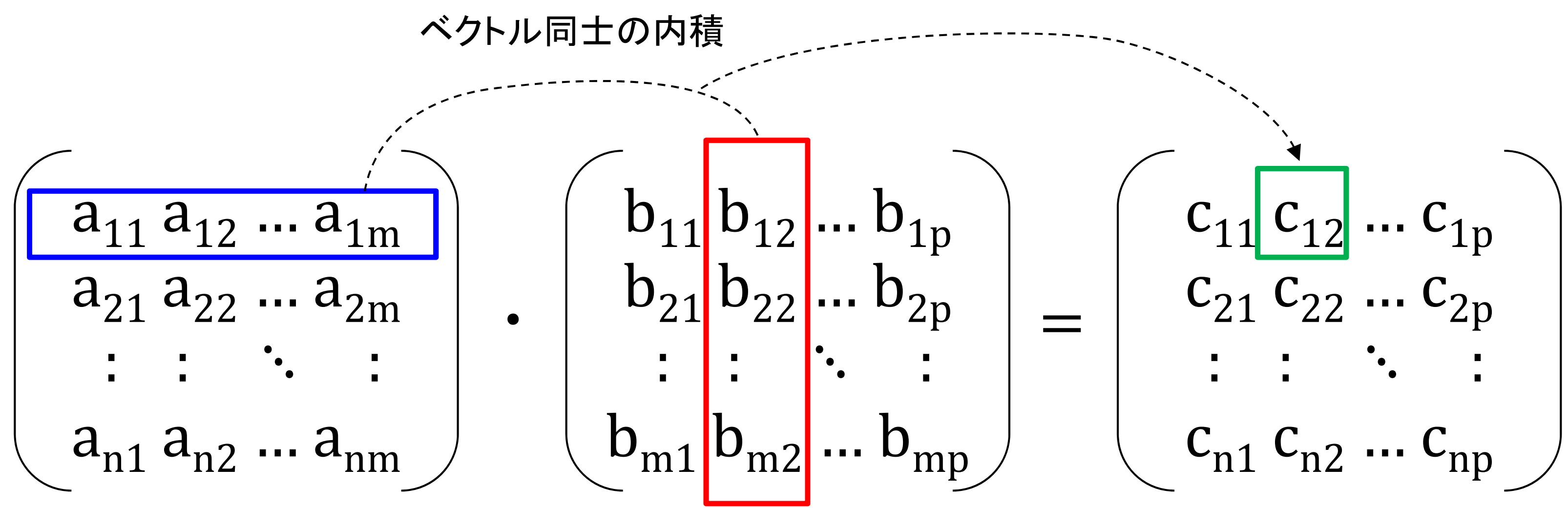


$$c_1 = a_1b_{11} + a_2b_{21} + \ldots + a_nb_{n1}$$
$$c_2 = a_1b_{12} + a_2b_{22} + \ldots + a_nb_{n2}$$
$$\vdots$$
$$c_m = a_1b_{1m} + a_2b_{2m} + \ldots + a_nb_{nm}$$

$$C_1 = \sum_{j=1}^{n} a_jb_{j1}$$

数学: 行列同士の内積

n行m列とn行m列との内積はn行m列の行列になる:



$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + \ldots + a_{1m}b_{m1}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + \ldots + a_{1m}b_{m2}$$
$$\vdots$$
$$c_{np} = a_{n1}b_{1p} + a_{n2}b_{2p} + \ldots + a_{nm}b_{mp}$$

数学: 単位行列

単位行列とは対角の値が1でそれ以外の値が0の正方行列:

Eはシュレディンガー方程式のエネルギー固有値Eとは異なるので注意すること。

$$\text{単位行列}E = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

ex1) 2x2

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

ex2) 4x4

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

単位行列Eの性質:

正方行列Aを単位行列Eにかけると元の正方行列Aのまま

$$AE = EA = A$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

数学: 逆行列

逆行列とは元の正方行列をかけると単位行列になる正方行列:

行列Aの逆行列を A^{-1}(インバース)と書く

逆行列A^{-1}の性質:

正方行列Aに逆行列A^{-1}をかけると単位行列Eになる

$$A\,A^{-1} = A^{-1}A = E$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

ex)
3x3

$$\begin{pmatrix} -1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

数学: 行列式と逆行列の計算

行列式は正方行列を式としてスカラー値を得る計算ができる:

行列A

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

行列式A

$$|A| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

3x3

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{21}a_{32}a_{13} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{23}a_{32}a_{11}$$

逆行列の計算:

$$A^{-1} = \frac{1}{|A|}\hat{A}$$

余因子行列: $\hat{A}$(エーハット)

$$\hat{A} = \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}$$

ここではこれ以上詳しく説明しない

数学: 固有値問題式変形 (固有値問題1)

$$\boldsymbol{Ax = \lambda x} \rightarrow \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$$

$\boldsymbol{A}$ は正方行列、$\boldsymbol{\lambda}$ は固有値(スカラー値)、$\boldsymbol{x}$ は列ベクトル

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

単位行列を使って展開

右辺を左辺に移動

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

係数をまとめる

$$\begin{pmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$x \neq 0$ かつ $y \neq 0$

この行列がゼロである必要がある。

$$\begin{pmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{pmatrix}$$

数学: 固有値の計算（固有値問題2）

$$\begin{pmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

この行列がゼロである必要がある

$x \neq 0$
$y \neq 0$

右行列式が成り立つ

$$\begin{vmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{vmatrix} = 0$$

固有値 λ_1 λ_2 を求める計算例:

※ 2次元の場合の固有値は2つ（重解なら1つ）。

$$A = \begin{pmatrix} 1 & 2 \\ -1 & 4 \end{pmatrix}$$

とした場合、

$$\begin{pmatrix} 1-\lambda & 2 \\ -1 & 4-\lambda \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

右行列式が成り立つ

$$\begin{vmatrix} 1-\lambda & 2 \\ -1 & 4-\lambda \end{vmatrix} = 0$$

$$(1-\lambda)(4-\lambda) + 2 = 0$$

$$\lambda^2 - 5\lambda + 6 = 0$$

$$(\lambda-2)(\lambda-3) = 0$$

λ が 2 か 3 の時に成り立つ。

答:固有値

$$\lambda_1 = 2$$

$$\lambda_2 = 3$$

数学: 固有ベクトルの計算（固有値問題3）

前頁の結果から:

$$\begin{pmatrix} 1-\lambda & 2 \\ -1 & 4-\lambda \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

の固有値は $\lambda_1 = 2$ と $\lambda_2 = 3$ である。
この時のそれぞれの固有ベクトルを計算。

$\lambda_1 = 2$ の時:

$$\begin{pmatrix} 1-2 & 2 \\ -1 & 4-2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -1 & 2 \\ -1 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow -x + 2y = 0$$

$-x + 2y = 0 \rightarrow$ λ_1の固有ベクトル解: $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ （の定数倍）

$\lambda_2 = 3$ の時:

$$\begin{pmatrix} 1-3 & 2 \\ -1 & 4-3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -2 & 2 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow -x + y = 0$$

$-x + y = 0 \rightarrow$ λ_2の固有ベクトル解: $\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ （の定数倍）

数学: 行列の対角化（固有値問題4）

$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ と、$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ を結合した行列Pを、$P = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$ とすると、

固有値λを対角化した行列を使って、 $AP = P\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ が成り立つ。

確認: $\underbrace{\begin{pmatrix} 1 & 2 \\ -1 & 4 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}}_{P} = \underbrace{\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}}_{P} \underbrace{\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}}_{\text{固有値行列}} = \begin{pmatrix} 4 & 3 \\ 2 & 3 \end{pmatrix}$

$AP = P\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ の両辺にP^{-1}を掛けて、

$$P^{-1}AP = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

が成り立つ。

固有ベクトルを結合した逆行列P^{-1}と行列Pで挟むことで対角化された固有値行列となる

数学: 固有値問題まとめ（固有値問題5）

$$\boldsymbol{Ax} = \boldsymbol{\lambda x} \rightarrow \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$$

$\boldsymbol{A}$ は正方行列、$\boldsymbol{\lambda}$ は固有値（スカラー値）、$\boldsymbol{x}$ は列ベクトル

- 2次元行列の固有値は2つある（量子ビットにも関連）。
  ※ ただし重解（同じ値）のケースでは1つだけ。
- 以下の行列式を解くことで固有値λの計算が可能。

$$\begin{vmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{vmatrix} = 0$$

固有値の1つをセットすることで
1つの固有ベクトルの式ができる。

- 固有値λが計算出来たら固有ベクトルも計算できる。
- 対角化により固有値と固有ベクトルの式が成立する。

$$P^{-1}AP = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \quad P = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$$

Pは2つの固有ベクトル
を繋げた行列

数学: 転置行列と複素共役行列

行列の転置とは行と列を入れ替える操作:

行列Aの転置行列を A^T (トランスポーズ)と書く

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \qquad A^T = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}$$

行列の複素共役は虚数部を反転する操作:

行列Bの複素共役行列を $\overline{B}$ (バー)と書く

ex)
3x3

$$B = \begin{pmatrix} 1 & 2i & 3 \\ 4 & 5-3i & -6i \\ 7i & 8+2i & 9 \end{pmatrix} \qquad \overline{B} = \begin{pmatrix} 1 & -2i & 3 \\ 4 & 5+3i & 6i \\ -7i & 8-2i & 9 \end{pmatrix}$$

数学: 共役転置行列（または随伴行列）

共役転置行列は、行列の複素共役と転置を行う:

行列Aの共役転置行列を A^*（スター）と書く

$$A^* = \overline{A}^T$$

物理学の定義: $A^\dagger = [\,A^T\,]^*$

ex) 3x3

$$B = \begin{pmatrix} 1 & 2i & 3 \\ 4 & 5-3i & -6i \\ 7i & 8+2i & 9 \end{pmatrix} \quad B^T = \begin{pmatrix} 1 & 4 & 7i \\ 2i & 5-3i & 8+2i \\ 3 & -6i & 9 \end{pmatrix}$$

$$B^* = \overline{B}^T = \begin{pmatrix} 1 & 4 & -7i \\ -2i & 5+3i & 8-2i \\ 3 & 6i & 9 \end{pmatrix}$$

数学: 正規行列

元行列と共役転置行列を掛けて、交換法則(可換)が成り立つ場合に、その行列を正規行列と呼ぶ:

$$A^* A = A A^*$$

物理学の定義: $A^\dagger A = A A^\dagger$

ex) 3x3

$$B = \begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 2 & 0 & 0 \end{pmatrix} \qquad B^* = \begin{pmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix}$$

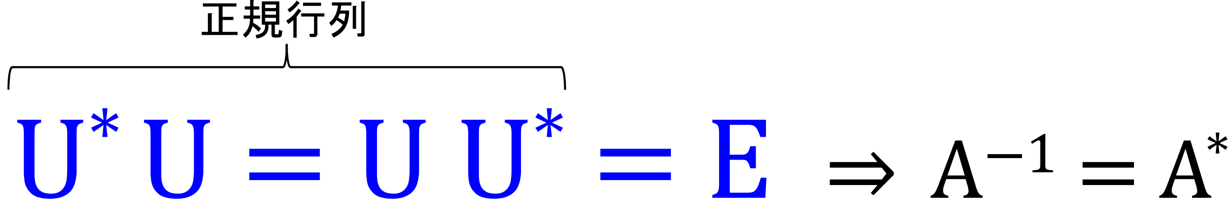
$$B^* B = \begin{pmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix} \begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 2 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

$$B B^* = \begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

数学: ユニタリ行列

元行列と共役転置行列を掛けると単位行列Eになる行列:
※ 逆行列と共役転置行列が等しくなる。
※ 内積の値(ベクトルの長さ)は変わらず複素空間の回転が可能。

量子ビットの
操作に利用
(後述)



物理学の定義: $U^\dagger U = U U^\dagger$ (ダガー) $= I$

ex)

$$A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix}$$

$$A^* A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$A A^* = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

数学: エルミート行列

エルミート行列は元行列と共役転置行列が同じ行列:

正規行列

$$A = A^* \Rightarrow \underbrace{A^* A = A A^*}_{\text{正規行列}} = A A = A^* A^*$$

物理学の定義:行列 A のエルミート行列は $A^\dagger$ (ダガー)

ex) 2x2 $\begin{pmatrix} 1 & 2+i \\ 2-i & 3 \end{pmatrix}$

ex) 3x3 $\begin{pmatrix} 1 & 2-i & 3+i \\ 2+I & 5 & 4+i \\ 3-i & 4-I & 6 \end{pmatrix}$

エルミート行列の対角は実数になる。
エルミート行列の固有値は実数になる。

量子ビットの
観測に利用
(後述)

数学: 正規行列・エルミート行列・ユニタリ行列

$$A^{*}A = AA^{*}$$

量子ビットの操作は全てユニタリ行列を使ったユニタリ変換

エルミート行列

$$A = A^{*}$$

$$A^{*}A = AA^{*} = AA$$

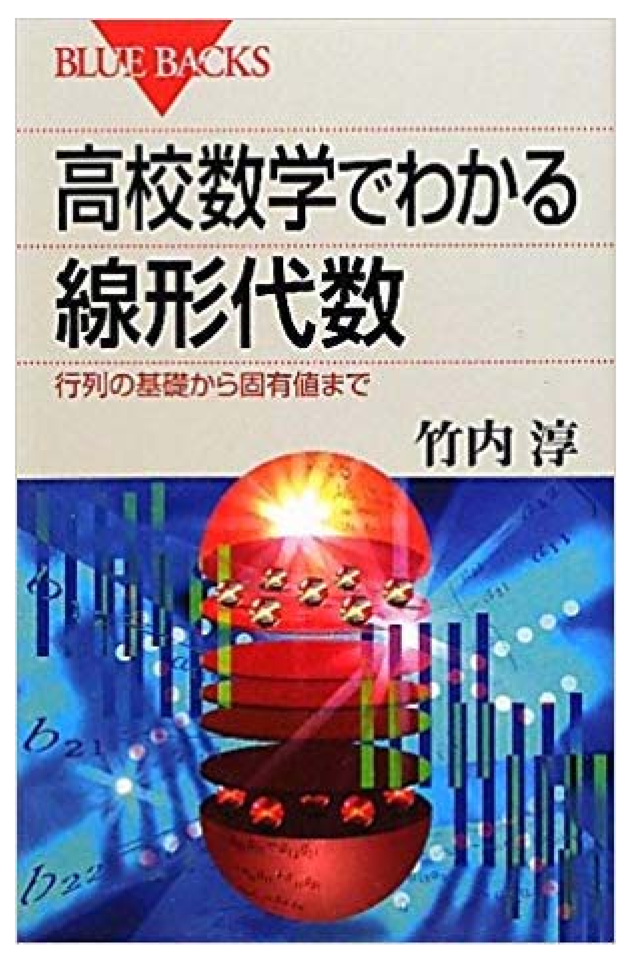
ユニタリ行列

$$A^{-1} = A^{*}$$

$$A^{*}A = AA^{*} = E$$

エルミート行列かつユニタリ行列と言うケースもある。

数学: 推薦図書



高校数学でわかる線形代数 行列の基礎から固有値まで

(ブルーバックス)

竹内 淳 (著) - 231 ページ

出版社: 講談社 (2010/11/20)

Kindle版:950円

https://www.amazon.co.jp/gp/product/B00J9YQF3E/

1-1で説明した内容の多くが本図書で説明されています。量子力学と固有値問題に関しても説明あります。値段も安いし良書だと思います。

1-2: ブラケット記法と量子計算

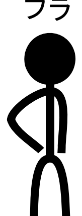
このパートからいよいよ量子計算関連する話になります。

と同時に数学から物理学へ話が変わります。

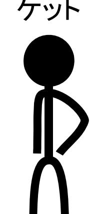
物理学と数学の違い

紛らわしい…特にA*… このページ以降は物理学の 作法で記述して行きます。	物理学 （量子力学）	数学 （線形代数学）
複素共役 （虚数部の±反転）	A^* （スター）	$\bar{A}$ （バー）
複素共役転置	$A^\dagger$ （ダガー）	A^* （スター）
エルミート （自己随伴）	$A = A^\dagger$	$A = A^*$
ユニタリ	$U^{-1} = U^\dagger$	$U^{-1} = U^*$
単位行列	I （id:単位ゲート）	E

ブラケット(BraKet)記法



$\langle \mathrm{Bra} |$ ：ブラ(行)ベクトル
$| \mathrm{Ket} \rangle$ ：ケット(列)ベクトル



n次元ブラ:
(行)ベクトル $\langle \varphi | = \begin{pmatrix} d_0 & d_1 & \dots & d_n \end{pmatrix}$,

n次元ケット:
(列)ベクトル $| \psi \rangle = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}$

量子計算では
基底ベクトルとして
1 と 0 の2次元の
ブラケットを利用。

$\langle 0 | = \begin{pmatrix} 1 & 0 \end{pmatrix}$

$\langle 1 | = \begin{pmatrix} 0 & 1 \end{pmatrix}$

$| 0 \rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$| 1 \rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

量子計算結果の
$|0\rangle$ と $|1\rangle$ が重要

ブラケットベクトルの内積と外積

n次φブラ:（行ベクトル） $\langle\varphi| = \begin{pmatrix} d_0 & d_1 & \dots & d_n \end{pmatrix}$,　n次ψケット:（列ベクトル） $|\psi\rangle = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}$

がある時に、

内積: $\langle\varphi|\psi\rangle = \begin{pmatrix} d_0 & d_1 & \dots & d_n \end{pmatrix}\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = d_0c_0 + d_1c_1 + \dots + d_nc_n = \sum_{j=0}^{n} d_jc_j$

結果はスカラー

外積: $|\psi\rangle\langle\varphi| = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}\begin{pmatrix} d_0 & d_1 & \dots & d_n \end{pmatrix} = \begin{pmatrix} c_0d_0 & c_0d_1 & \dots & c_0d_n \\ c_1d_0 & c_1d_1 & \dots & c_1d_n \\ \vdots & \vdots & \ddots & \vdots \\ c_nd_0 & c_nd_1 & \dots & c_nd_n \end{pmatrix}$

結果は行列

同じベクトルの内積

ブラベクトル: $\langle\varphi| = \begin{pmatrix} a_0^* & a_1^* & \dots & a_n^* \end{pmatrix}$, ケットベクトル: $|\varphi\rangle = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$

がある時に内積計算は、

$$\langle\varphi|\varphi\rangle = \begin{pmatrix} a_0^* & a_1^* & \dots & a_n^* \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$$

$$= a_0^* a_0 + a_1^* a_1 + \dots + a_n^* a_n$$

量子計算では、内積の結果が1となるように規格化されている。

$$\langle\varphi|\varphi\rangle = 1$$

※ これはベクトルの長さが1と言うことと同じ。

テンソル積（1階のテンソル積）

2つの2次元ベクトル $|\psi_1\rangle$ と $|\psi_2\rangle$ がある時、

$$|\psi_1\rangle = \begin{pmatrix} a \\ b \end{pmatrix} = a|0\rangle + b|1\rangle \qquad |\psi_2\rangle = \begin{pmatrix} c \\ d \end{pmatrix} = c|0\rangle + d|1\rangle$$

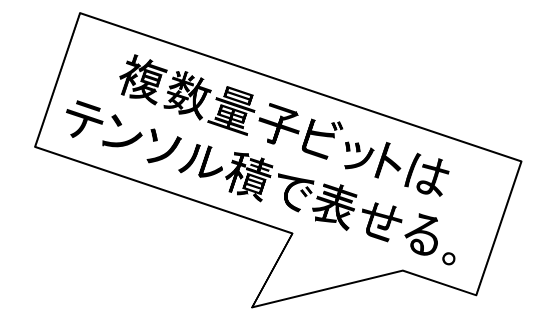
$|\psi_1\rangle$ と $|\psi_2\rangle$ のテンソル積⊗は4次元ベクトルとなる。

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{pmatrix} a\begin{pmatrix} c \\ d \end{pmatrix} \\ b\begin{pmatrix} c \\ d \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix} = \begin{matrix} ac|00\rangle + ad|01\rangle \\ + bc|10\rangle + bd|11\rangle \end{matrix}$$

テンソル積の記号⊗は省略されることが多い。

$$|\psi_1\rangle \otimes |\psi_2\rangle \;\Rightarrow\; |\psi_1\rangle|\psi_2\rangle \;\Rightarrow\; |\psi_1\psi_2\rangle$$

複数量子ビットとテンソル積



$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ であるので、

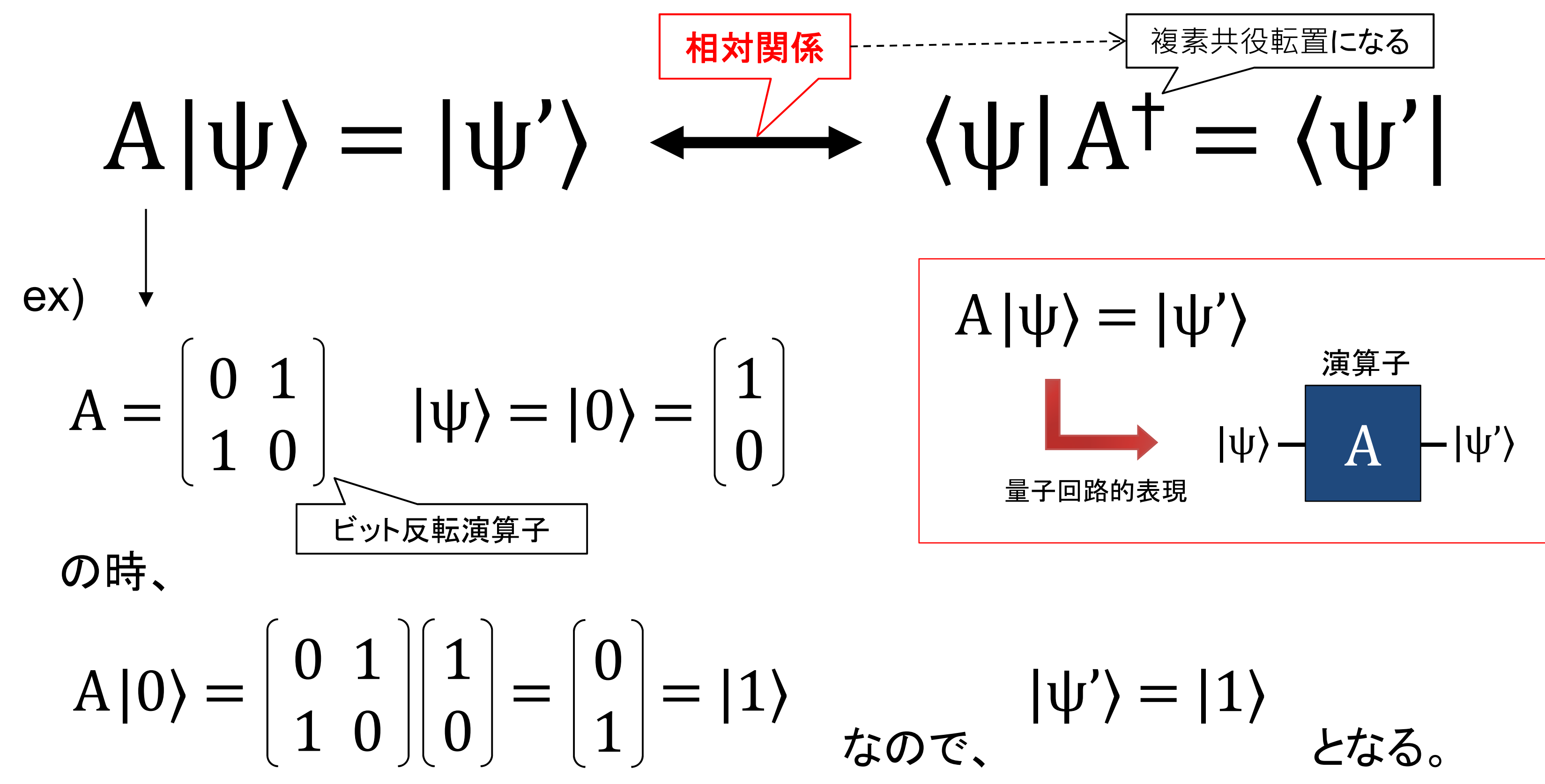
$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$|000\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad |001\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \ldots(\text{略})\ldots \quad |111\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$|111\rangle = |7\rangle$
と表示する場合もある

演算子（正方行列）

ブラケットを利用した計算の演算子は正方行列で表す。
演算子をA（正方行列）とすると、ケットは左から、ブラは右から演算子を適用して、新しいケットやブラに変換ができる。



ユニタリ(Unitary)演算

量子計算の演算子は全てユニタリ演算子 U となる。

$$U^{\dagger} = U^{-1} \quad (U^{\dagger}U = UU^{\dagger} = I)$$ の時、

ユニタリ演算により、$U|\psi\rangle = |\psi'\rangle$ となる。

ユニタリ演算子は**回転操作**(長さは変化しない)演算である。
この為に複数のユニタリ演算子を時間的に順番に利用できる。

ex)

$$U_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad U_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

位相反転演算子

時間

$U_2U_1|0\rangle$ ➡ $|0\rangle$ —[U_1]—[U_2]—

時間

$$U_2U_1|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -|1\rangle$$

マイナスは位相反転

エルミート(Hermitian)演算

エルミート行列Hは、

$H^{\dagger} = H \quad (H^{\dagger}H = HH^{\dagger} = HH)$ である。

2つの固有ベクトル φ_i と φ_j と、その固有値 λ_i と λ_j がある時、

式1: $H|\varphi_i\rangle = \lambda_i|\varphi_i\rangle$ と、式2: $H|\varphi_j\rangle = \lambda_j|\varphi_j\rangle$ となる。

式1の左辺から $\langle\varphi_j|$ を適用し $\langle\varphi_j|H|\varphi_i\rangle = \lambda_i\langle\varphi_j|\varphi_i\rangle$ となる。

式2を複素共役転置して式3: $\langle\varphi_j|H^{\dagger} = \langle\varphi_j|H = \lambda_j^{*}\langle\varphi_j|$ とし、

式3の右辺から $|\varphi_i\rangle$ を適用し $\langle\varphi_j|H|\varphi_i\rangle = \lambda_j^{*}\langle\varphi_j|\varphi_i\rangle$ となる。

結果: $\langle\varphi_j|H|\varphi_i\rangle = \lambda_i\langle\varphi_j|\varphi_i\rangle = \lambda_j^{*}\langle\varphi_j|\varphi_i\rangle$ より、

$(\lambda_i - \lambda_j^{*})\langle\varphi_j|\varphi_i\rangle = 0$ を得る。

$i = j$ なら $\lambda_i = \lambda_i^{*}$ で λ_i は実数、規格化より $\langle\varphi_i|\varphi_i\rangle = 1$ となる。

$i \neq j$ なら $\lambda_i - \lambda_j^{*}$ は0ではなく、$\langle\varphi_j|\varphi_i\rangle = 0$ (直交)となる。

演算子の行列表現（エルミート演算による対角化）

エルミート演算を波動ベクトルで囲った行列を演算子の行列表現と呼ぶ。ここでは演算を $\langle\varphi_n|H|\varphi_n\rangle$ で（n=0,1,2,...,n）とすると、

$$\begin{pmatrix} H_{00} & H_{01} & \cdots & H_{0n} \\ H_{10} & H_{11} & \cdots & H_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ H_{n0} & H_{n1} & \cdots & H_{nn} \end{pmatrix} = \begin{pmatrix} \langle\varphi_0|H|\varphi_0\rangle & \langle\varphi_0|H|\varphi_1\rangle & \cdots & \langle\varphi_0|H|\varphi_n\rangle \\ \langle\varphi_1|H|\varphi_0\rangle & \langle\varphi_1|H|\varphi_1\rangle & \cdots & \langle\varphi_1|H|\varphi_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle\varphi_n|H|\varphi_0\rangle & \langle\varphi_n|H|\varphi_1\rangle & \cdots & \langle\varphi_n|H|\varphi_n\rangle \end{pmatrix}$$

$i=j$ なら $\lambda_i=\lambda_i^*$ で λ_i は実数、規格化より $\langle\varphi_i|\varphi_i\rangle=1$ であり、
$i\neq j$ なら $\lambda_i-\lambda_j^*$ は0ではなく、$\langle\varphi_j|\varphi_i\rangle=0$（直交）となるので、

$$\langle\varphi_0|H|\varphi_0\rangle=\lambda_0\langle\varphi_0|\varphi_0\rangle=\lambda_0$$
$$\langle\varphi_1|H|\varphi_1\rangle=\lambda_1\langle\varphi_1|\varphi_1\rangle=\lambda_1$$
$$\vdots$$
$$\langle\varphi_n|H|\varphi_n\rangle=\lambda_n\langle\varphi_n|\varphi_n\rangle=\lambda_n$$

$$H=\begin{pmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

固有値の対角化

エルミート演算による期待値

観測前ベクトル $|\psi\rangle$ と、観測後固有ベクトル $|\varphi_n\rangle$ と、固有値 λ_n があり、どちらのベクトルも規格化されている(長さが1)とする。
この時の期待値(観測される物理量)は $P = \langle\psi|H|\psi\rangle$ で計算可能。

$$\langle\psi|H|\psi\rangle = \begin{pmatrix} \langle\varphi_0|\psi\rangle^* & \langle\varphi_1|\psi\rangle^* & \dots & \langle\varphi_n|\psi\rangle^* \end{pmatrix} \begin{pmatrix} \lambda_0 & 0 & \dots & 0 \\ 0 & \lambda_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \begin{pmatrix} \langle\varphi_0|\psi\rangle \\ \langle\varphi_1|\psi\rangle \\ \vdots \\ \langle\varphi_n|\psi\rangle \end{pmatrix}$$

$$= \begin{pmatrix} \langle\varphi_0|\psi\rangle^* & \langle\varphi_1|\psi\rangle^* & \dots & \langle\varphi_n|\psi\rangle^* \end{pmatrix} \begin{pmatrix} \lambda_0\langle\varphi_0|\psi\rangle \\ \lambda_1\langle\varphi_1|\psi\rangle \\ \vdots \\ \lambda_n\langle\varphi_n|\psi\rangle \end{pmatrix}$$

$$= \lambda_0\underbrace{|\langle\varphi_0|\psi\rangle|^2}_{\lambda_0\text{の期待値}} + \lambda_1\underbrace{|\langle\varphi_1|\psi\rangle|^2}_{\lambda_1\text{の期待値}} + \dots + \lambda_n\underbrace{|\langle\varphi_n|\psi\rangle|^2}_{\lambda_n\text{の期待値}}$$

ボルンの規則

量子系の任意ベクトル ψ の物理量(オブザーバブル)の観測時に、各測定値(固有値λ_n)が取る期待値は固有値のどれかとなり、また測定値(固有値λ_n・固有ベクトルφ_n)を得る確率は $|\langle\varphi_n|\psi\rangle|^2$ となる。

確率振幅を $c_n = \langle\varphi_n|\psi\rangle$ とした時に、全固有ベクトルは完全系をなすので、以下の式が成り立つ。なお c_n は複素数である。

$$|\psi\rangle = \sum_n c_n |\varphi_n\rangle = c_0|\varphi_0\rangle + c_1|\varphi_1\rangle + \dots + c_n|\varphi_n\rangle$$

$$|c_0|^2 + |c_1|^2 + \dots + |c_n|^2 = 1$$

全確率の合計は1(100%)となる

よって任意のベクトルは取りえる全ての固有ベクトルで表せる。
※ 期待値が確率振幅の二乗となることをボルンの規則と呼ぶ。

- ボルンの規則は他の方程式から導けないが実験結果と一致するので現在主流となっている計算方法である。

有限準位量子力学系（量子ビット＝2次元）

量子ビットは2つの固有ベクトル・固有値を持つ。
2つの固有（基底）ベクトルを $|0\rangle$ と $|1\rangle$ とする。

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

任意の量子ベクトル ψ に関して確率振幅 c_n を
使ってボルンの規則により以下の式が成り立つ、

確率振幅
c_0 と c_1 は
複素数

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

$$|c_0|^2 + |c_1|^2 = 1$$

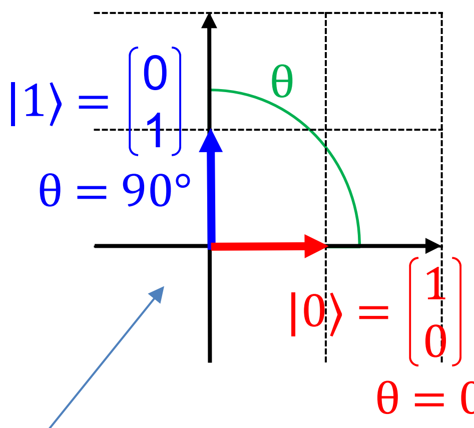
$$c_0 = \langle 0|\psi\rangle \qquad c_1 = \langle 1|\psi\rangle$$

1-3: ブロッホ球と1量子ビット操作

お待たせしました！
このパートからやっと量子計算の話になります。

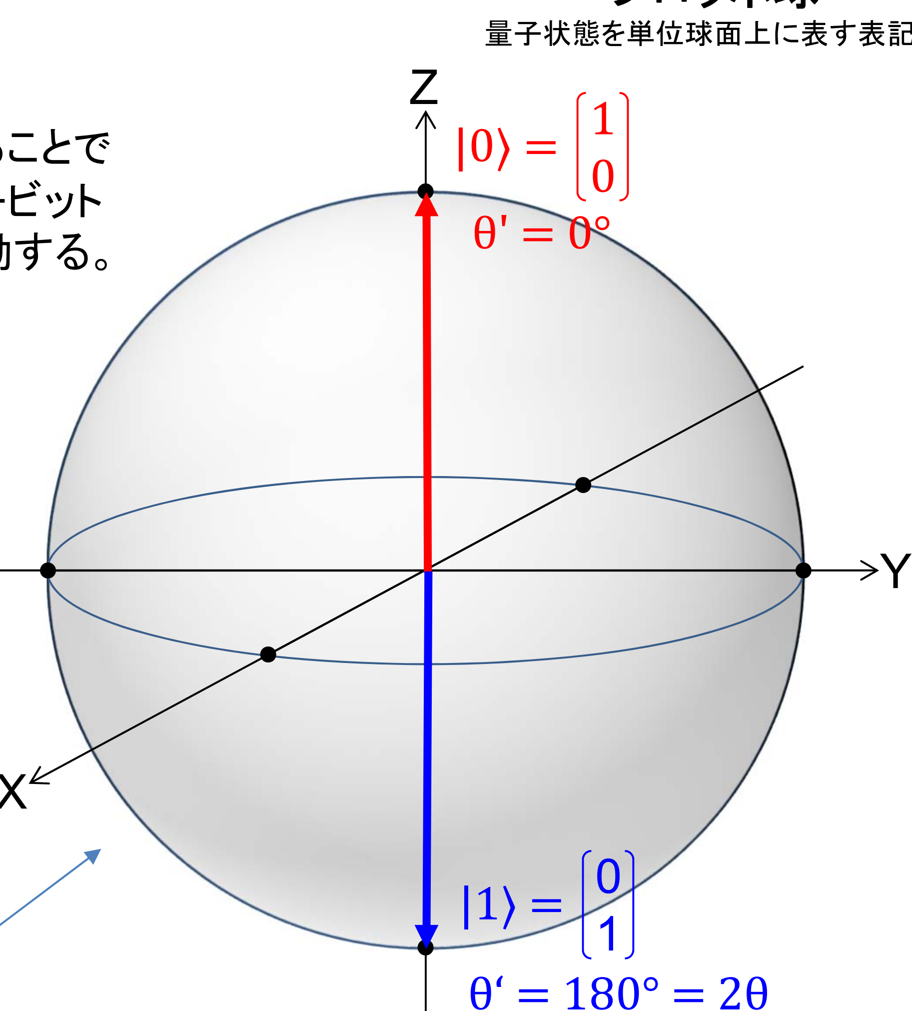
ここからが本題です！

量子ビットとブロッホ球



θ' = 2θとすることで
球面上を量子ビット
の状態が移動する。

θ' = 2θ

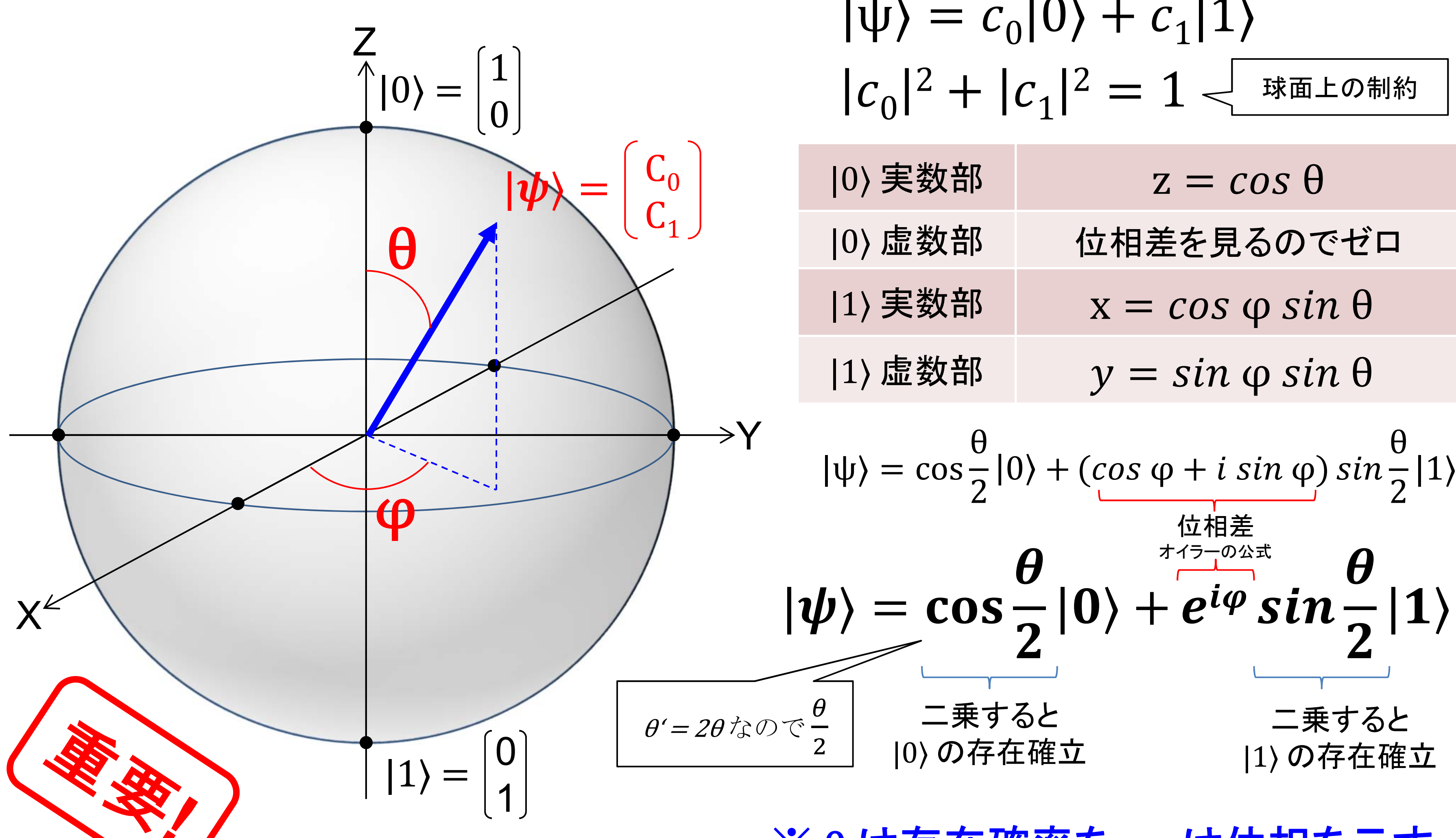


これでは上半球の部分しか
使えない…また虚数部がある
ので4次元の空間となる。
量子状態のイメージが掴めない…

虚数部（位相）は絶対値ではなく
干渉に関する相対値とする。
なので|0⟩の位置をゼロとして、
|1⟩の虚数部（Z軸の回転θ）に
位相差を示し3次元空間にする。

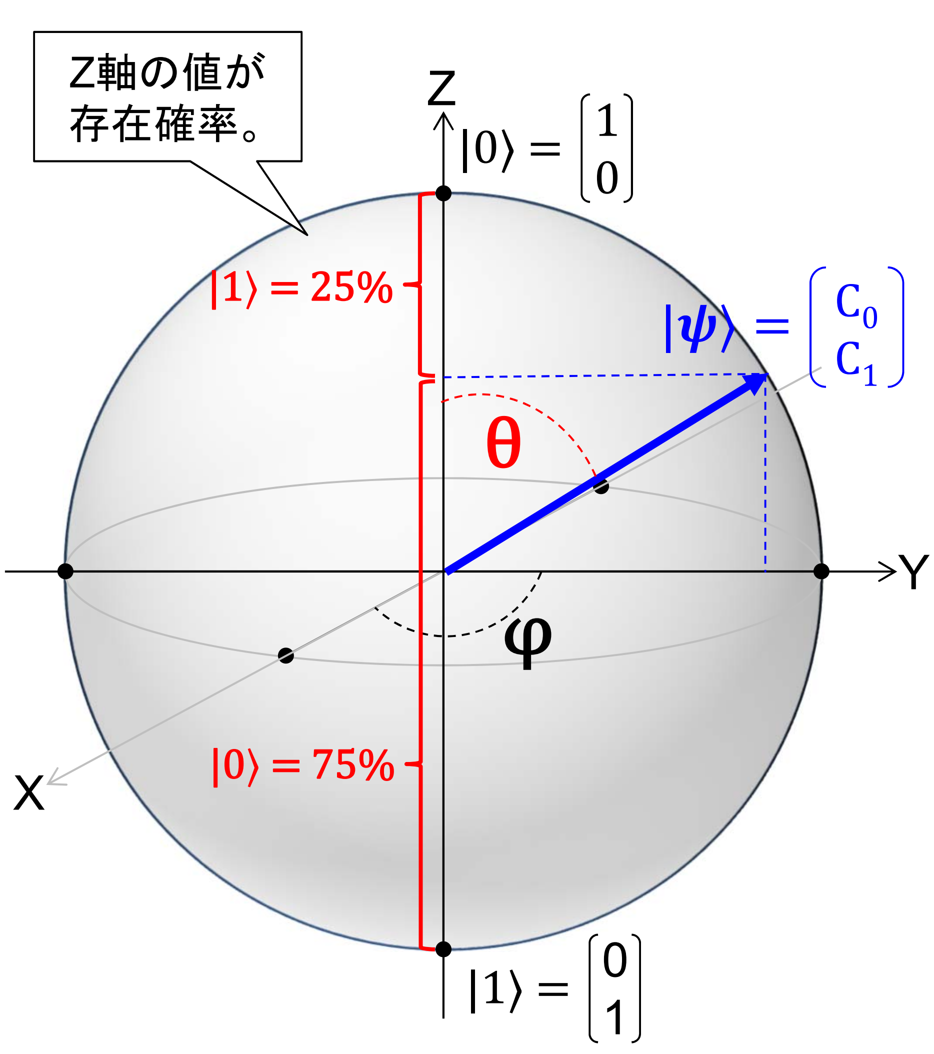
ブロッホ球は複素ヒルベルト空間を示す。

量子ビットの存在確率と位相



※ θ は存在確率を、φ は位相を示す。

存在確率の計算例（角度θのみに依存）



$\theta = \pi/3 = 60°$ の場合

$$
\begin{aligned}
|0\rangle &= |\cos(\pi/3\ /2)|^2 \\
&= |\cos(\pi/6)|^2 \\
&= |0.86602\ldots|^2 \\
&= 0.75 = \mathbf{75\%}
\end{aligned}
$$

$$
\begin{aligned}
|1\rangle &= |\sin(\pi/3\ /2)|^2 \\
&= |\sin(\pi/6)|^2 \\
&= |0.5|^2 \\
&= 0.25 = \mathbf{25\%}
\end{aligned}
$$

※ 位相 $e^{i\varphi}$ は虚数部なので1つの量子ビット計算には影響しない。

量子ビットに対するユニタリ演算

ユニタリ行列による座標変換: $U|\psi\rangle = |\psi'\rangle$

id 恒等演算 iden(q) = $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

単位行列 I なので何もしない演算となる。

X ビット反転演算 x(q) = $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

Y 位相ビット反転演算 y(q) = $\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$

Z 位相反転演算 z(q) = $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

位相のみを変換する場合は位相シフトゲートとも呼ぶ。

パウリ (Pauli) ゲート

IBMのQiskitを使う（ローカル環境）

環境：Anaconda3（Python3.5）

後述する IBM Q Experience の notebook 画面を使っても良い。

以下より環境に合わせてダウンロードとインストール

https://www.anaconda.com/distribution/

ライブラリ：Qiskit（キスキット）

Windows版：Anaconda Prompt
MacOS版：ターミナル

インストール

```
pip install qiskit
```

バージョン指定インストール

```
pip install qiskit=0.11.1
```

アンインストール

```
pip uninstall qiskit
```

※ Qiskitのバージョン確認：

In:
```
import qiskit
qiskit.__qiskit_version__
```

Out:
```
{'qiskit-terra': '0.9.0',
 'qiskit-ignis': '0.2.0',
 'qiskit-aqua': '0.6.0',
 'qiskit': '0.12.0',
 'qiskit-aer': '0.3.0',
 'qiskit-ibmq-provider': '0.3.2'}
```

本資料のソースは 0.12.0 と表示される環境にて確認しています。

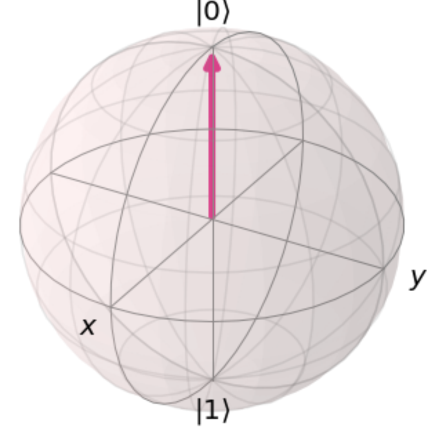
Qiskit はバージョンアップの頻度が多く、ソースがそのまま使えないケースもあるので注意が必要。

Qiskitでブロッホ球を表示する（恒等演算）

Jupiter Notebook から以下を実行（コピペで大丈夫です）。

```
from qiskit import *                              # 量子計算用
from qiskit.tools.visualization import *          # 結果表示用
backend = Aer.get_backend('statevector_simulator')  # シミュレータ指定
q = QuantumRegister(1)                   # 量子ビットを1つ用意
qc = QuantumCircuit(q)                   # 量子回路に量子ビットをセット
qc.iden(q[0])                            # 恒等演算（初期値 |0⟩ を出力）
r = execute(qc, backend).result()        # 回路実行して結果取得
psi = r.get_statevector(qc)              # ステータス取得
print(psi)                               # ステータス（ベクトル）表示
plot_bloch_multivector(psi)              # ブロッホ球表示（※ Qiskit 0.7以降）
```

※ Qiskit 0.6 以前は plot_state(psi, "bloch") を使う。

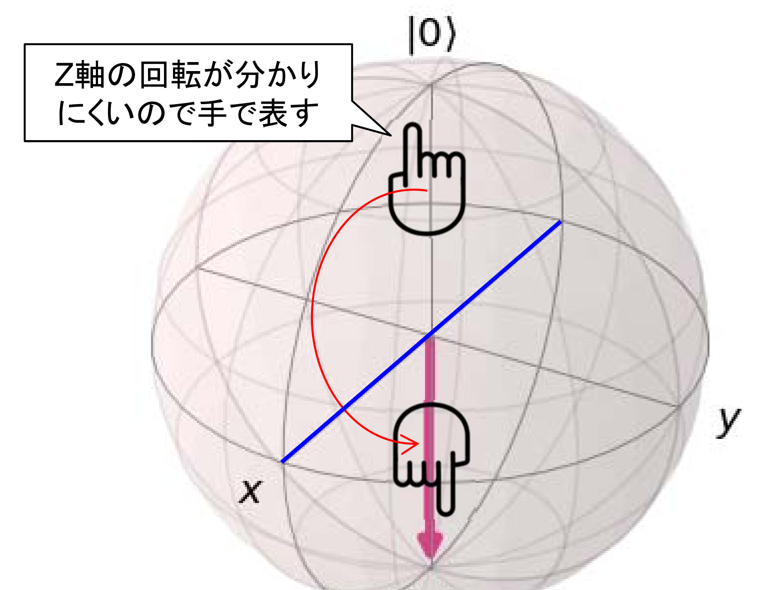


各量子ビットの初期状態は $|0\rangle$ なので、左のように $|0\rangle$ のブロッホ球が表示されるはず。

次ページからはプログラムの6行目の演算を変更することでユニタリ演算を確認して行く。

ビット反転演算 X ゲート

qc. x(q[0])



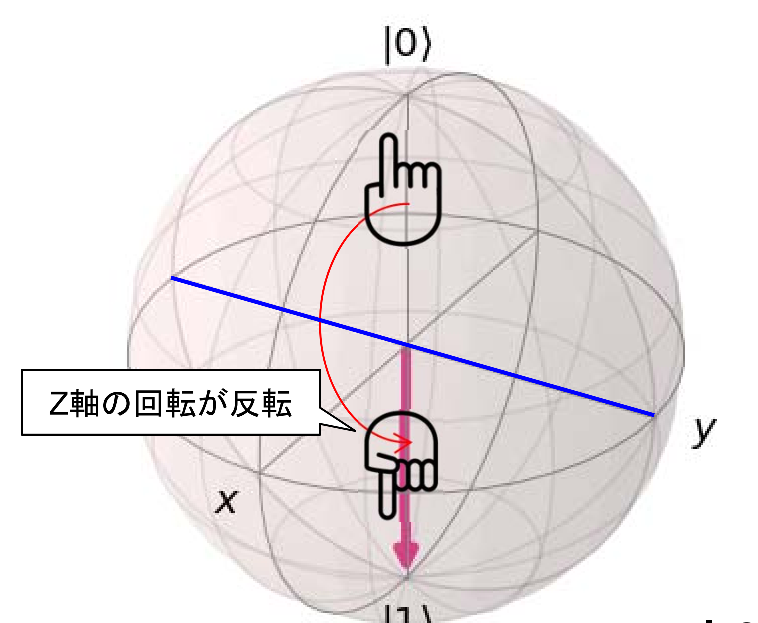
Xゲートの演算は、
ブロッホ球のX軸回りに、
180度(π)回転する。
ビットは反転するが、
位相φは変わらない。

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

位相ビット反転演算 Y ゲート

qc. y (q [0])



Yゲートの演算は、
ブロッホ球のY軸回りに、
180度(π)回転する。
ビットは反転し、
位相φも反転する。

虚数部は結果に影響しない

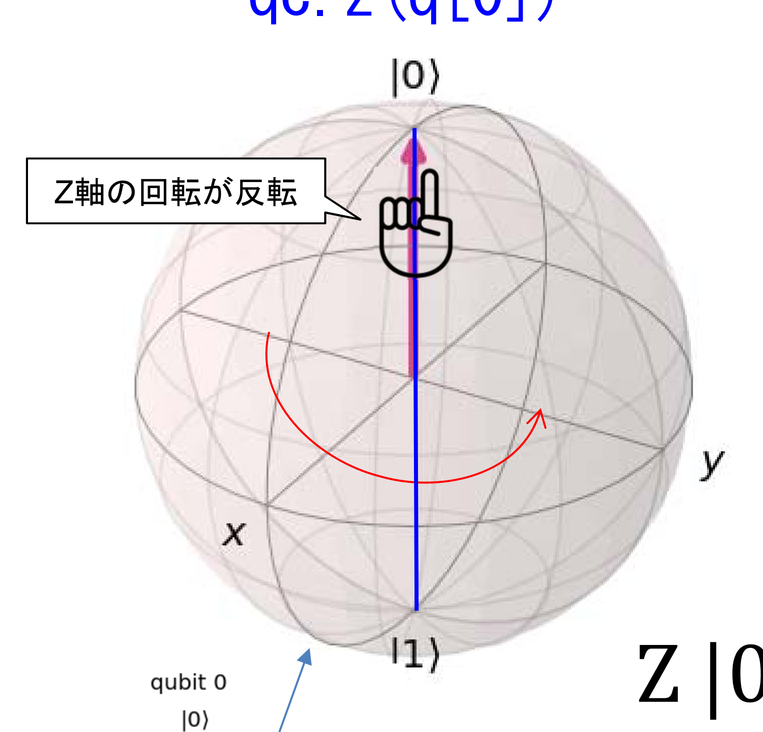
$$Y|0\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ i \end{pmatrix} = i|1\rangle$$

$$Y|1\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -i \\ 0 \end{pmatrix} = -i|0\rangle$$

マイナスが位相反転を示す

位相反転演算 Z ゲート

qc. z(q[0])



Zゲートの演算は、
ブロッホ球のZ軸回りに、
180度(π)回転する。
ビットは反転しないが、
位相φは反転する。

|0⟩の虚数部はゼロ固定

$$Z|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

$$Z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -|1\rangle$$

|1⟩のマイナスが位相反転を示す

アダマール H ゲート（重ね合わせ状態の生成）

アダマールによる座標変換: $H|\psi\rangle = |\psi'\rangle$

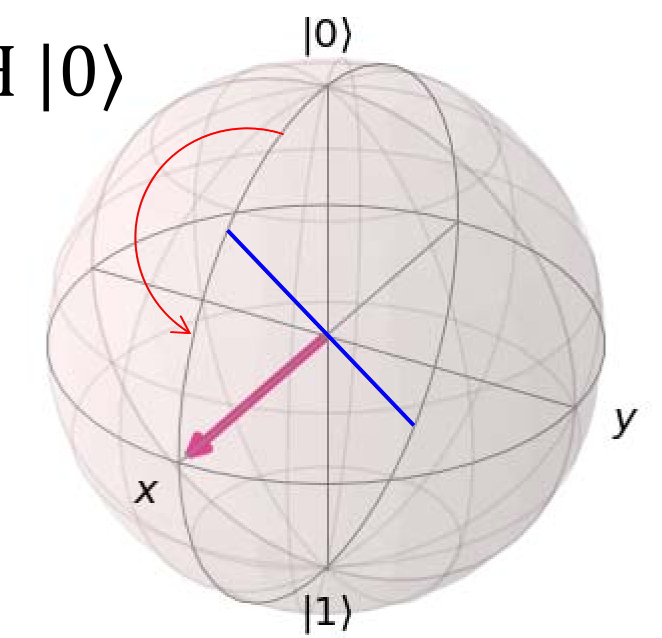
H　アダマール演算 h(q) $= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

Hadamard

$= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$

qc. h(q[0])

$H\,|0\rangle$



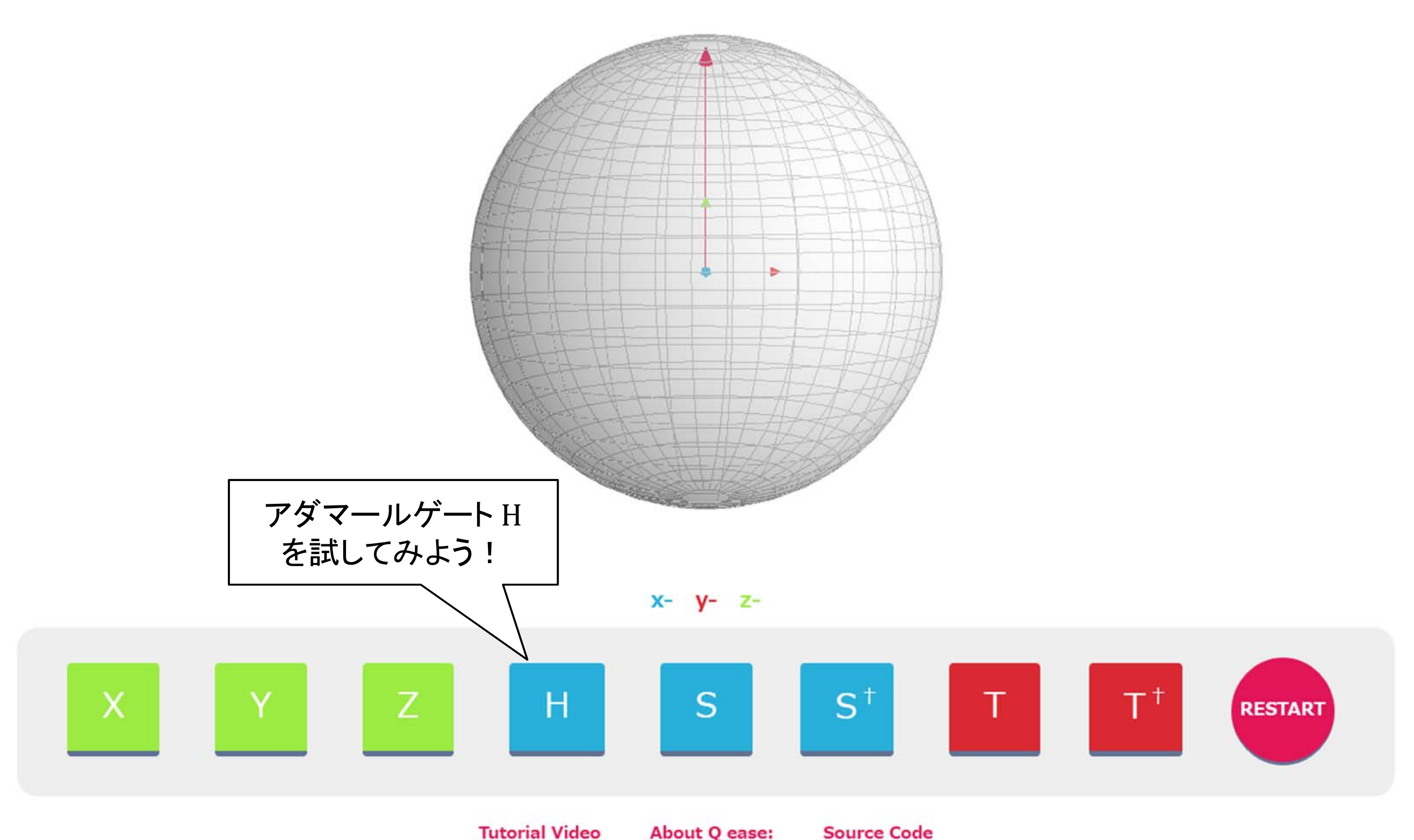
重要!

Hゲートの演算は、
ブロッホ球のXZ平面に、
45度傾いた軸回りに、
180度(π)回転する。

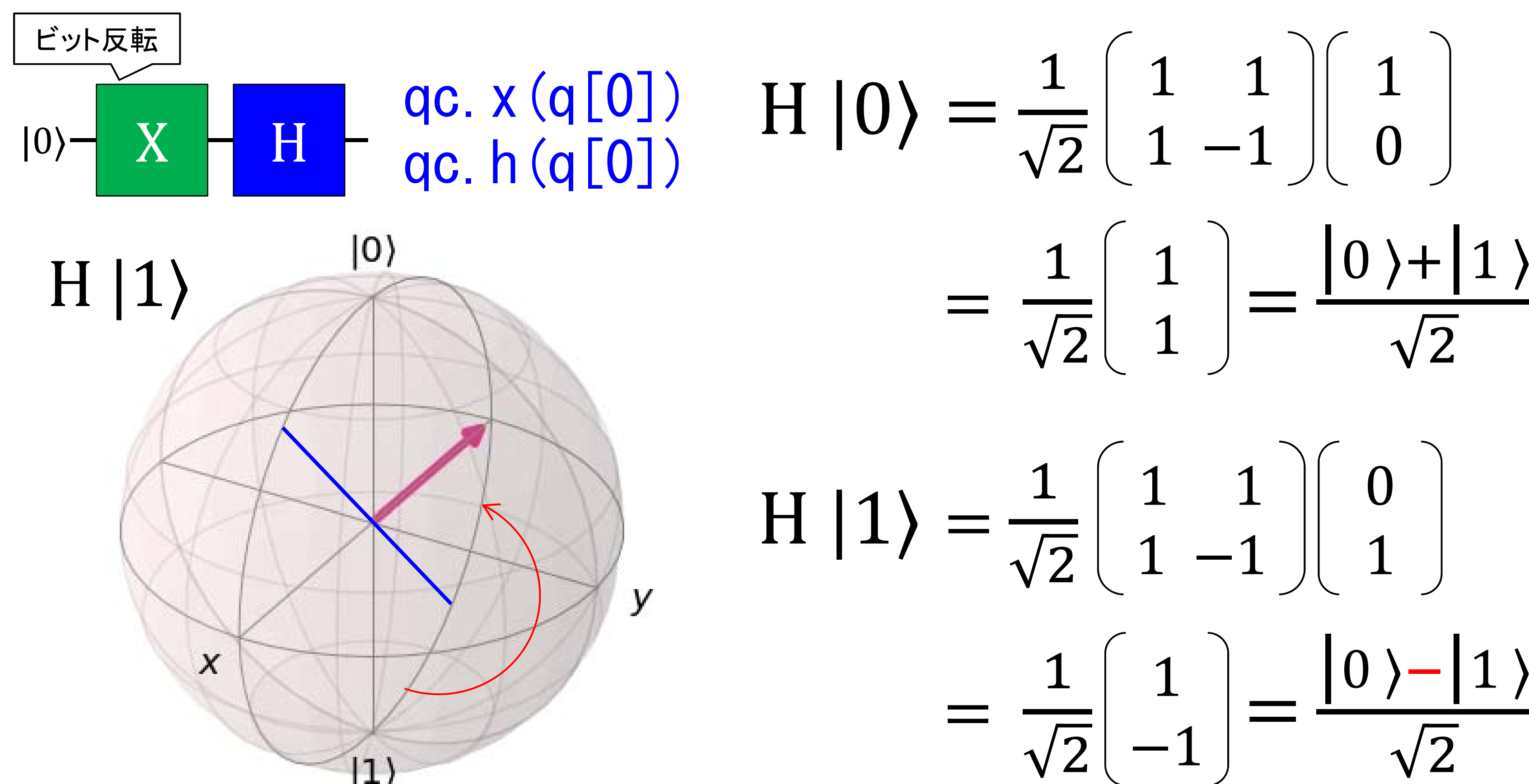
※ Y軸に90度回転する訳ではない。

Webでブロッホ球を表示する

Try Bloch!
https://qease.herokuapp.com/bloch/try/

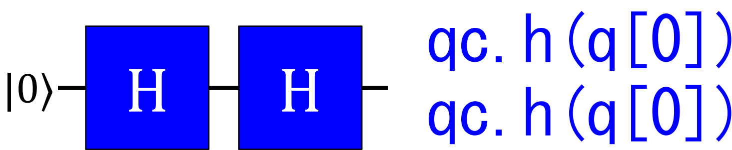


$|1\rangle$ へのアダマール H ゲート適用



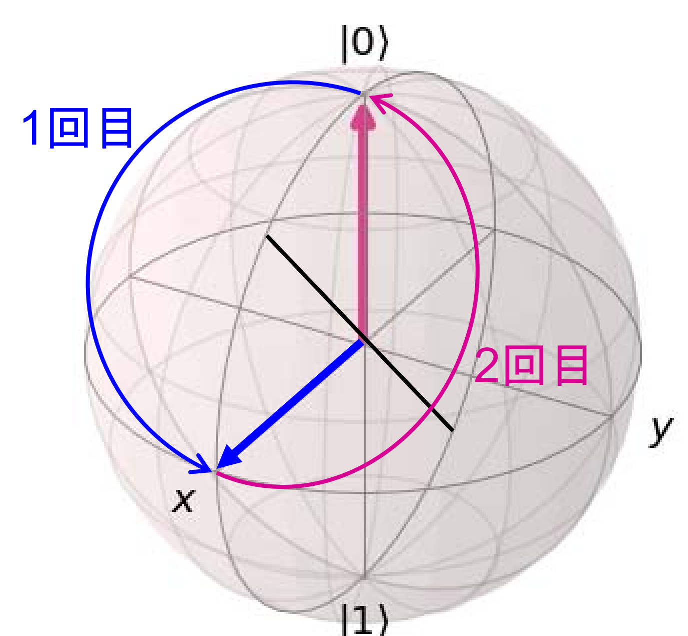
$H|0\rangle$ と $H|1\rangle$ はどちらも $|0\rangle$: 50% と $|1\rangle$: 50% となり同じ確率分布になる。
ただし位相が逆になっている。

アダマール H ゲートを2回適用



$$HH = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$ （単位行列）



$$HH\,|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

結論：2回アダマール演算を適用すると元のベクトルに戻る。

量子ゲートの組み合わせ

- アダマールゲートで挟むと変換が可能となる。

$\mathrm{HXH} = \mathrm{Z}$ ：ビット反転Xゲートを挟むと位相反転Zゲートに。

$\mathrm{HZH} = \mathrm{X}$ ：位相反転Zゲートを挟むとビット反転Xゲートに。

$\mathrm{HYH} = -\mathrm{Y}$ ：位相ビット反転Yゲートを挟むと位相のみ反転。

ex) HXHの計算

$$\mathrm{XH} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

↓

$$\mathrm{H(XH)} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \mathrm{Z}$$

位相シフト S/S†/T/T† ゲート（あまり使わない）

Z 位相反転演算 $z(q) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

反転なので逆位相も同じ

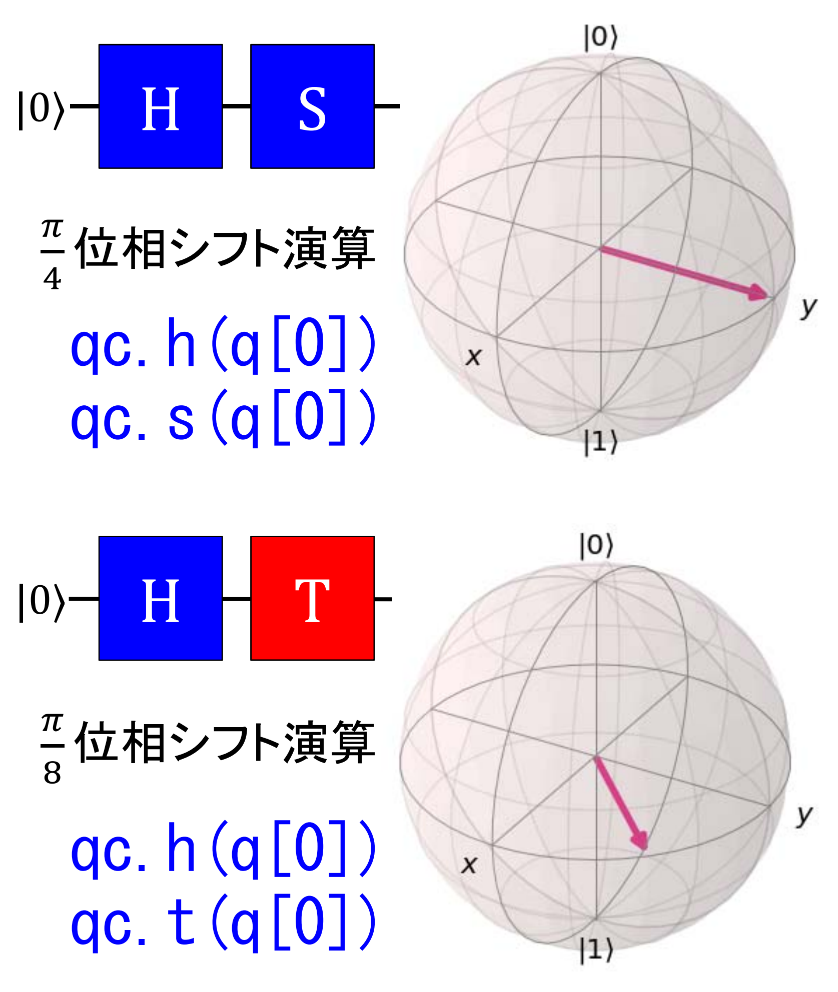
S $\frac{\pi}{4}$位相シフト演算 $s(q) = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$

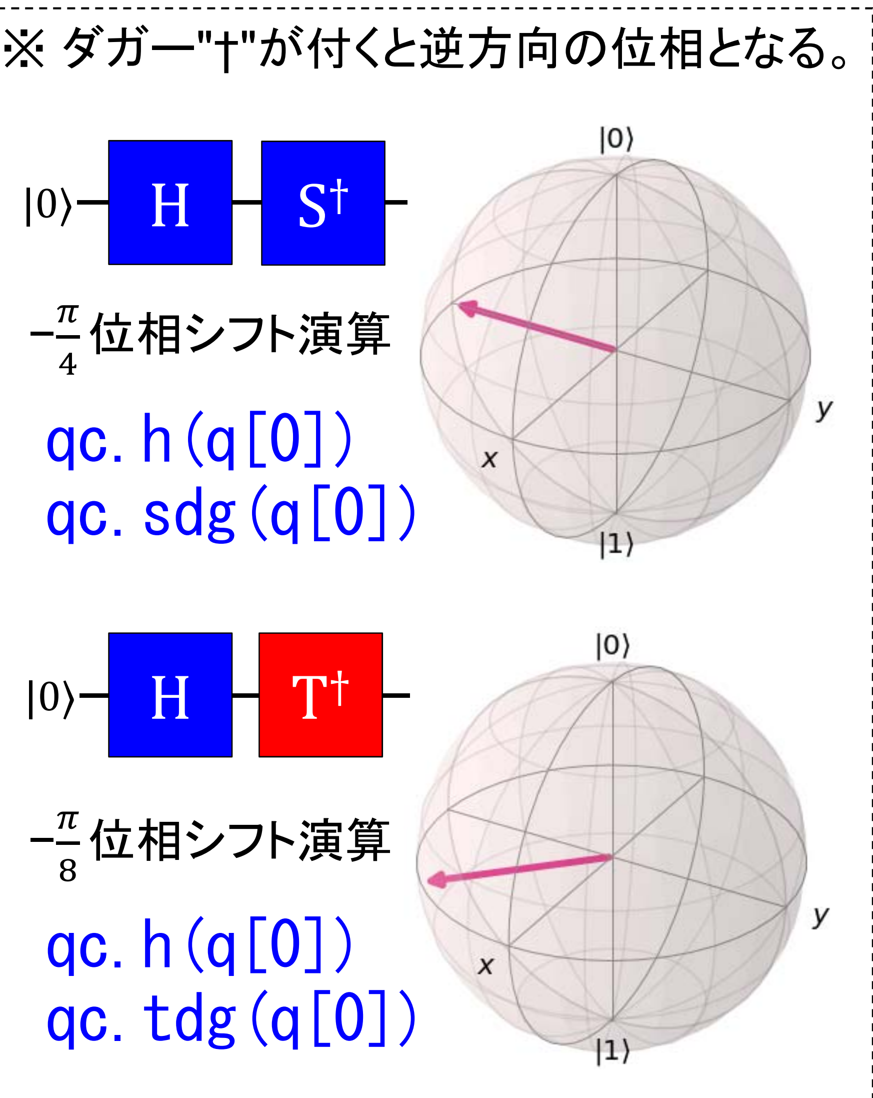
S† $-\frac{\pi}{4}$位相シフト演算 $s^{\dagger}(q) = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$

T $\frac{\pi}{8}$位相シフト演算 $t(q) = \begin{pmatrix} 1 & 0 \\ 0 & \frac{(1+i)}{\sqrt{2}} \end{pmatrix}$

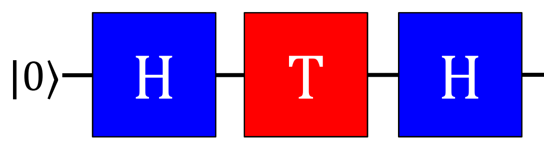
T† $-\frac{\pi}{8}$位相シフト演算 $t^{\dagger}(q) = \begin{pmatrix} 1 & 0 \\ 0 & \frac{(1-i)}{\sqrt{2}} \end{pmatrix}$

重ね合わせ状態での位相シフト

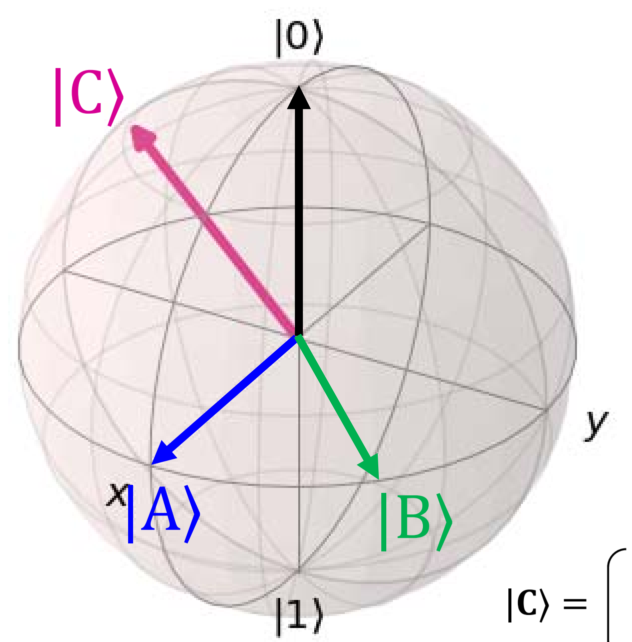




π/2 以外の位相シフトの意味




```
qc.h(q[0])  #|A⟩
qc.t(q[0])  #|B⟩
qc.h(q[0])  #|C⟩
```



$$|C\rangle = \begin{pmatrix} 0.85355339+0.35355339j \\ 0.14644661-0.35355339j \end{pmatrix}$$

$$|c_0|^2 = 0.85355339$$

$$|c_1|^2 = 0.14644661$$

課題:

量子演算を行う為にはブロッホ球上の任意の場所にベクトルを移動する必要がある。

解決方法:

アマダール演算と位相シフト演算を組み合わせることで任意の位置にベクトル移動することが可能となる。ただしπ/8のシフト演算だけだと取れる方向はある程度限定されてしまうが現在の量子計算では十分な精度らしい。

軸回転Rx/Ry/Rzゲート

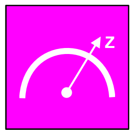
実際の量子プログラミングでは、任意角度での軸回転(シフト)ゲートが必要になる。

Rx X軸回転演算 $Rx(\theta, q) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$

Ry Y軸回転演算 $Ry(\theta, q) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$

Rz Z軸回転演算 $Rz(\theta, q) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$

測定（量子ビットを収束させて古典ビットとして取り出す）



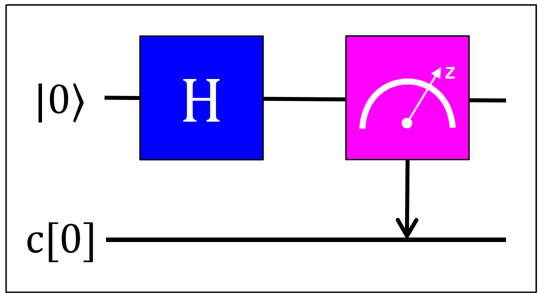
標準基底測定 measure(q, c)

Jupiter Notebook から以下を実行（コピペで大丈夫です）。

```
from qiskit import *                          # 量子計算用
from qiskit.tools.visualization import *      # 結果表示用
backend = Aer.get_backend('qasm_simulator')   # 量子シミュレータ指定
q = QuantumRegister(1)        # 量子ビットを1つ用意
c = ClassicalRegister(1)      # 古典ビットを1つ用意
qc = QuantumCircuit(q, c)     # 量子回路に量子ビットと古典ビットをセット
qc.h(q[0])                    # 量子重ね合わせ（アダマール演算）
qc.measure(q[0], c[0])        # 量子ビットq[0]を観測し古典ビットc[0]へ
r = execute(qc, backend, shots=1000).result() # 回路を1000回実行する
rc = r.get_counts()           # 結果取得
print(rc)                     # 結果表示
plot_histogram(rc)            # ヒストグラム表示
```

測定結果の例1：Hゲート（アダマール演算）

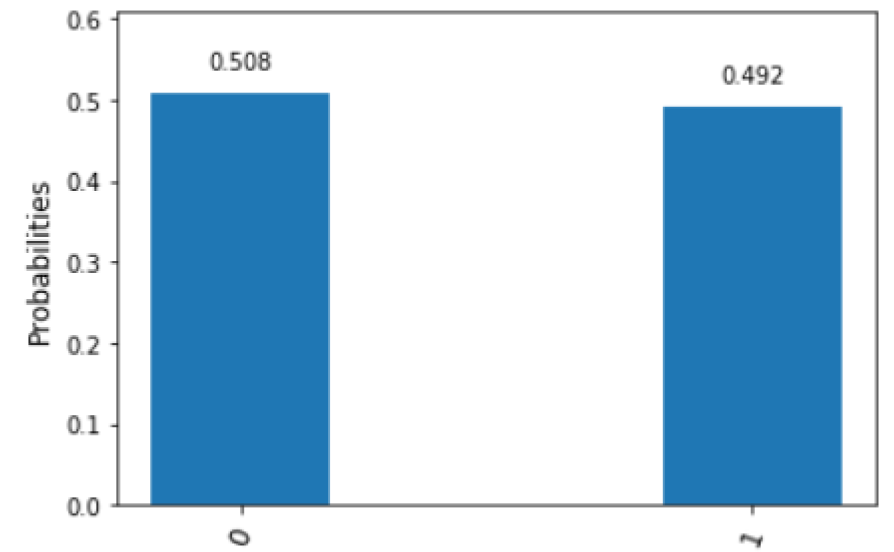
```
qc.h(q[0])
qc.measure(q[0],c[0])
```

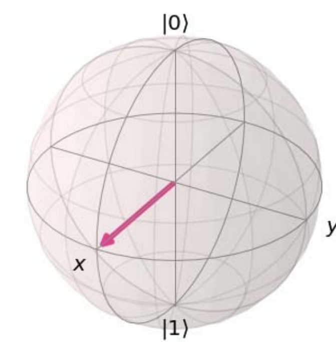



```
print(rc)                # 結果表示
plot_histogram(rc)       # ヒストグラム表示
```

```
{'0': 508, '1': 492}
```

'0' : 500, '1': 500 となるはずだが…NISQシミュレータなのでノイズを考慮している。

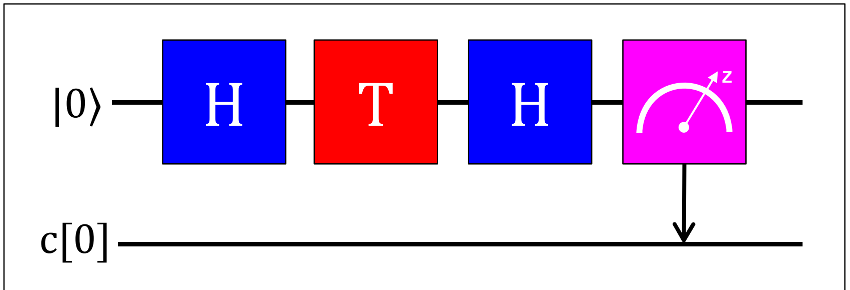




参考：ブロッホ球表示

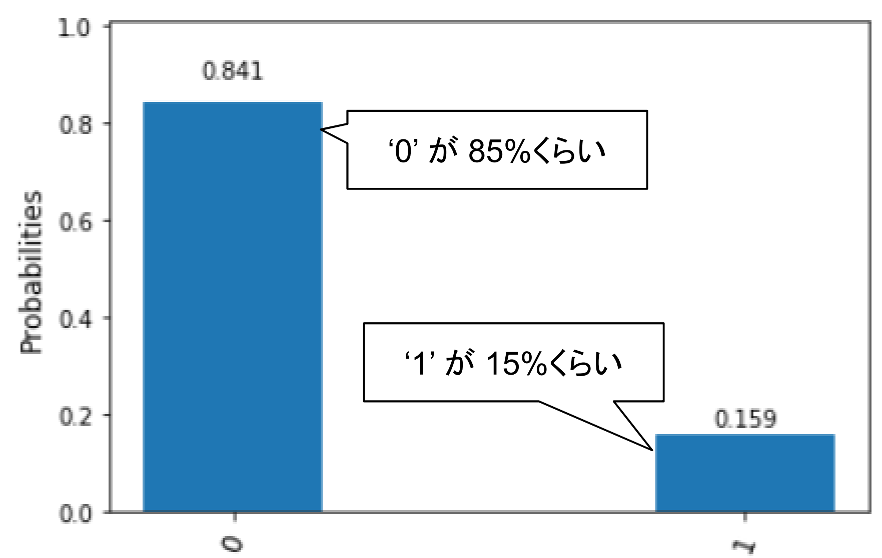
測定結果の例2：HゲートとTゲート

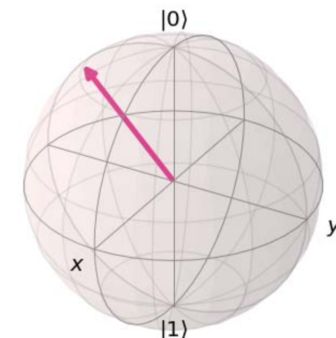
```
qc.h(q[0])
qc.t(q[0])
qc.h(q[0])
qc.measure(q[0],c[0])
```




```
print(rc)                    # 結果表示
plot_histogram(rc)           # ヒストグラム表示
```

{'0': 841, '1': 159}





参考：ブロッホ球表示

$$|\psi\rangle = \begin{pmatrix} 0.85355339+0.35355339j \\ 0.14644661-0.35355339j \end{pmatrix}$$

$$|c_0|^2 = (0.85355339)^2+(+0.35355339)^2 = 0.85355339 \ [85.36\%]$$

$$|c_1|^2 = (0.14644661)^2+(-0.35355339)^2 = 0.14644661 \ [14.64\%]$$

1-4：IBM Q と Qiskit

https://www.research.ibm.com/ibm-q/

Qiskit のバックエンド

- ’statevector_simulator’
  - 重ね合わせ状態のまま値を取得できるローカル量子シミュレータ。
  - ノイズ無しの理論値（確率振幅）を取得できる。
  - ブロッホ球（重ね合わせ状態）の表示時に使う。
- ’qasm_simulator’
  - ノイズありのローカル量子シミュレータ。
  - 通常の量子計算に使う標準のシミュレータ。

IBM Q Experience のクラウドサービス

- ’ibmq_qasm_simulator’
  - IBM Q の量子シミュレータ（アカウントが必要）。
- ’ibmq_ourense’ / ’ibmqx4’ / ’ibmqx2’
  - IBM Q の5量子ビット実機（アカウント・トークンが必要）。
- ’ibmq_16_melbourne’
  - IBM Q の16量子ビット実機（アカウント・トークンが必要）。

クラウド量子計算：IBM Q Experience

IBM Q を使う為のクラウドサービス

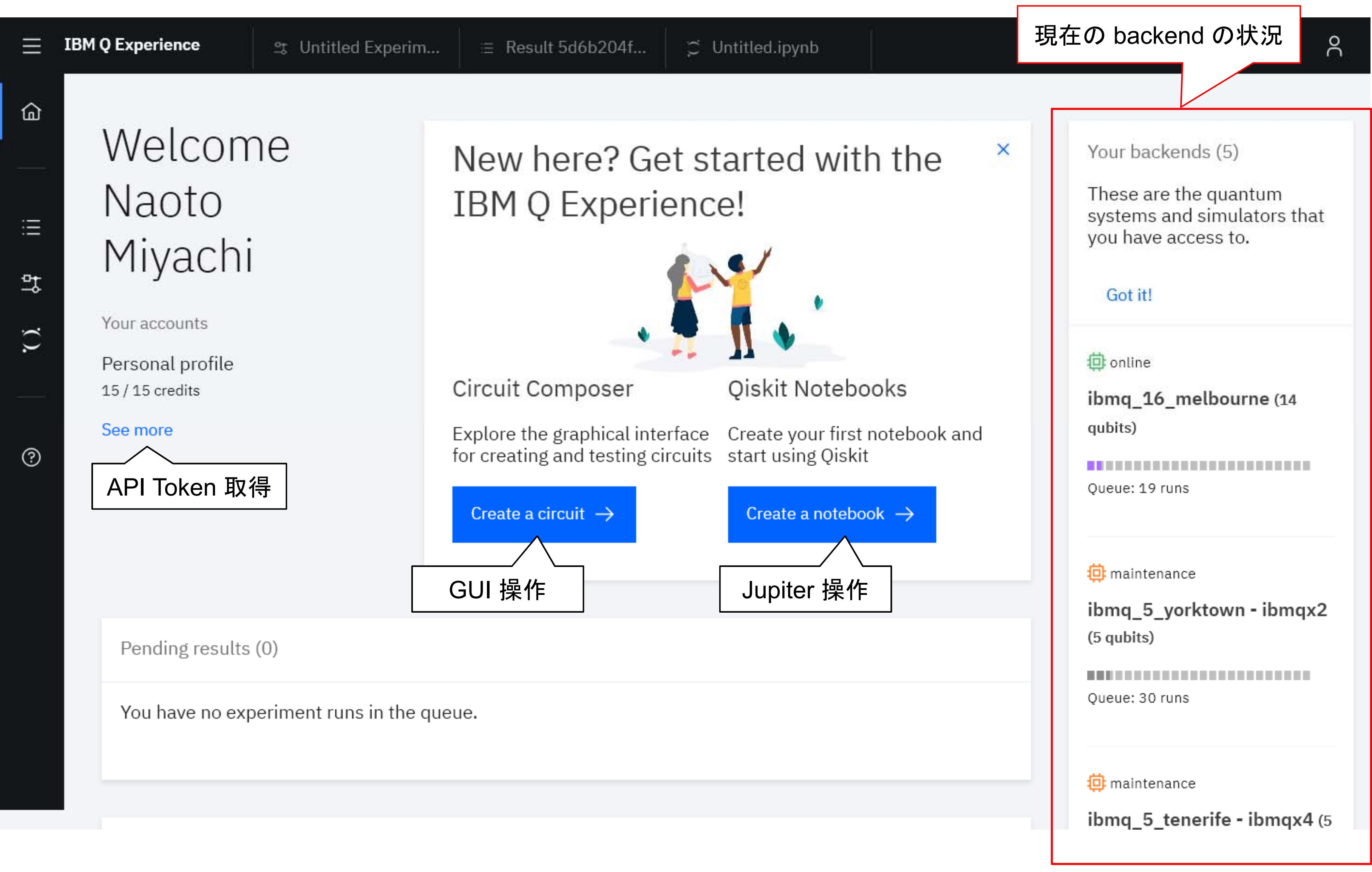
https://www.research.ibm.com/ibm-q/technology/experience/

サインインの為には登録（無料）が必要。
Linkedin/GitHub/Google/Twitterアカウントも利用可能

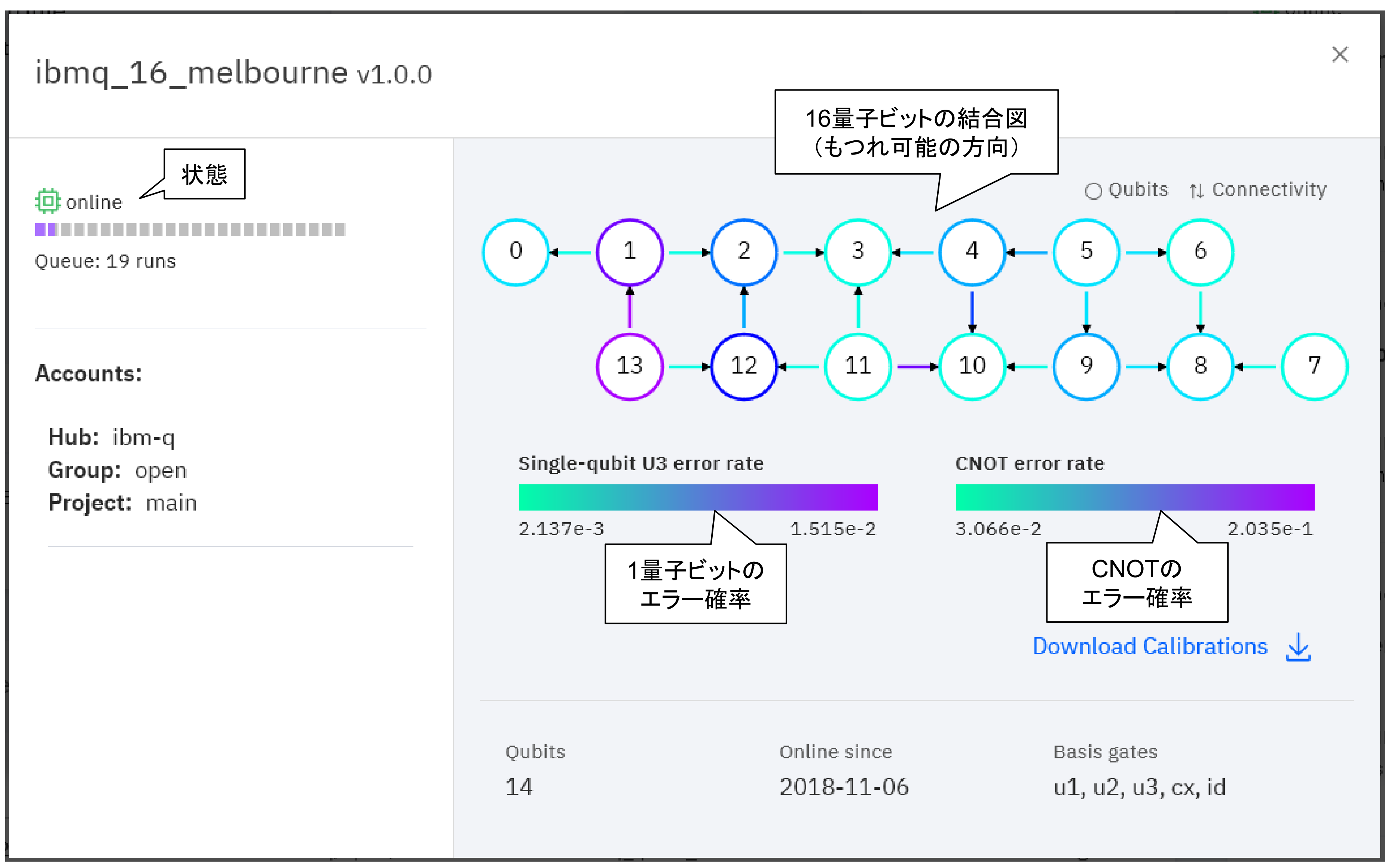
機能：

1. GUIを使って簡単な量子回路の編集が可能。
   - 実行時に以下の選択が可能：
     A) 実機を使わない量子シミュレーション（Qiskitと同じ）
     B) 5/16量子ビットの実機を使った量子計算（バッチ実行）
     C) 過去に同じ量子回路の実機を使った結果の利用
2. Qiskitを使った実行（Jupiter環境） ※ 新機能

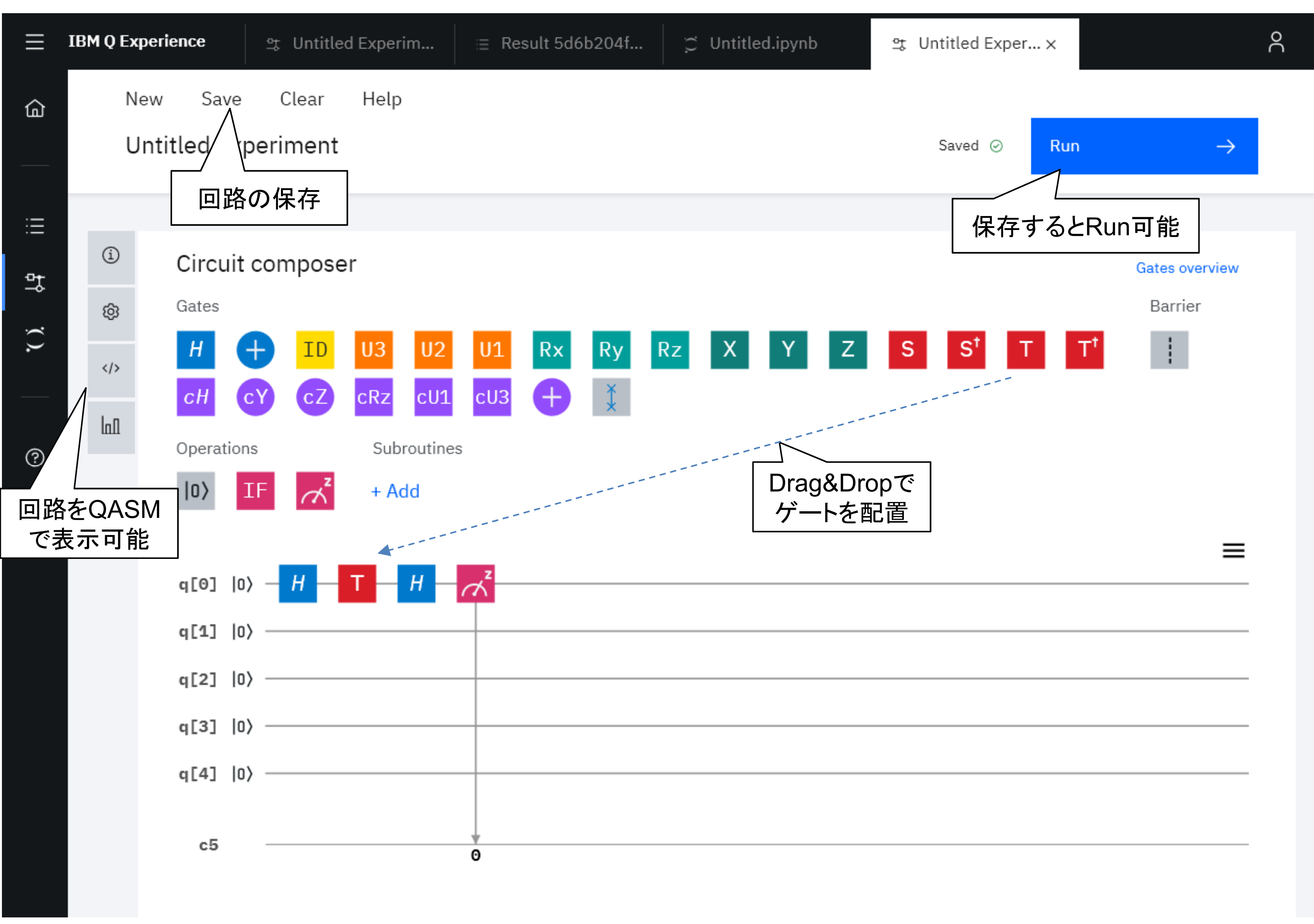
IBM Q Experience の Dashboard



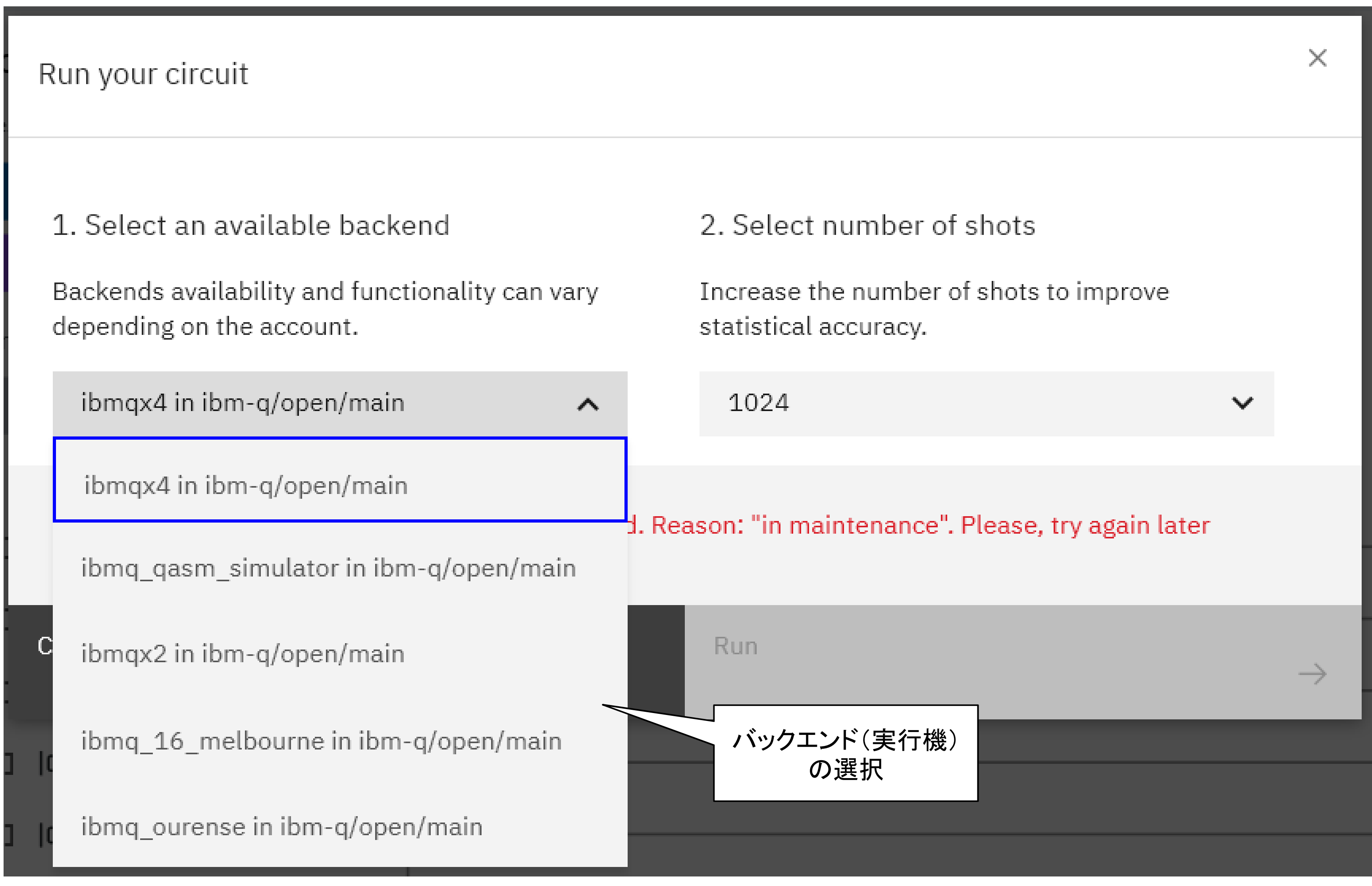
IBM Q 公開中16量子ビットシステム



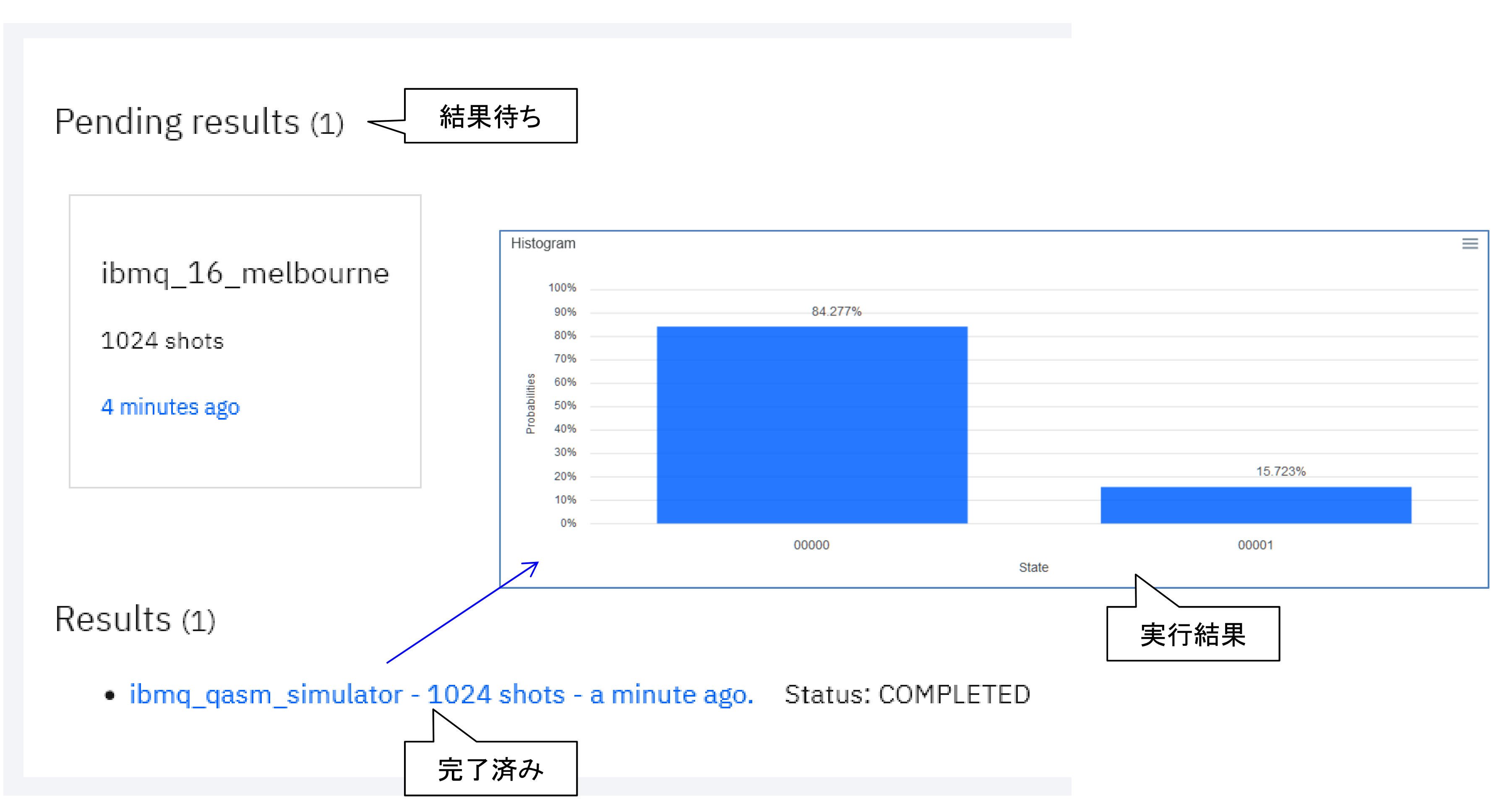
IBM Q Experience の GUI 編集



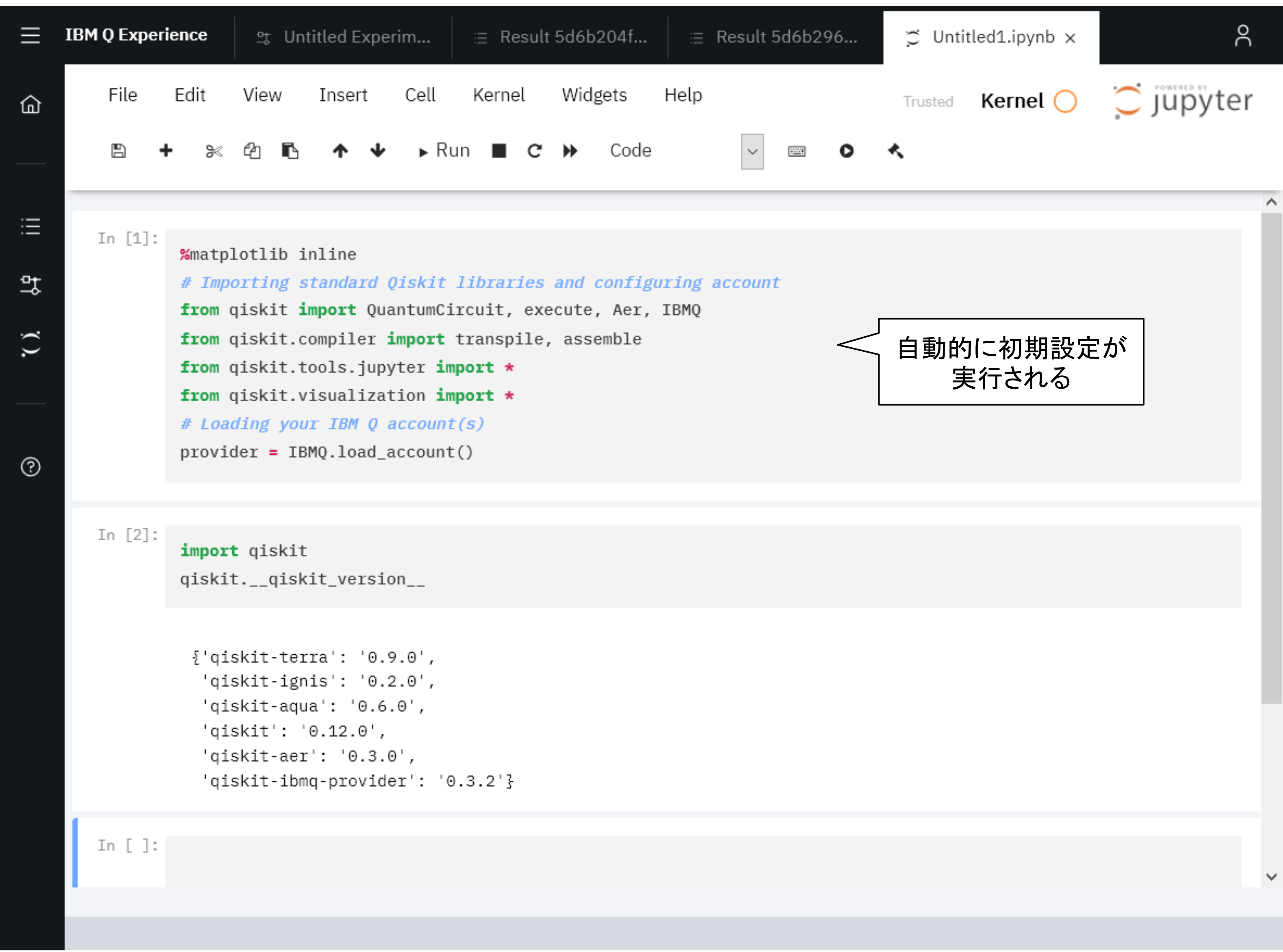
IBM Q Experience 実行



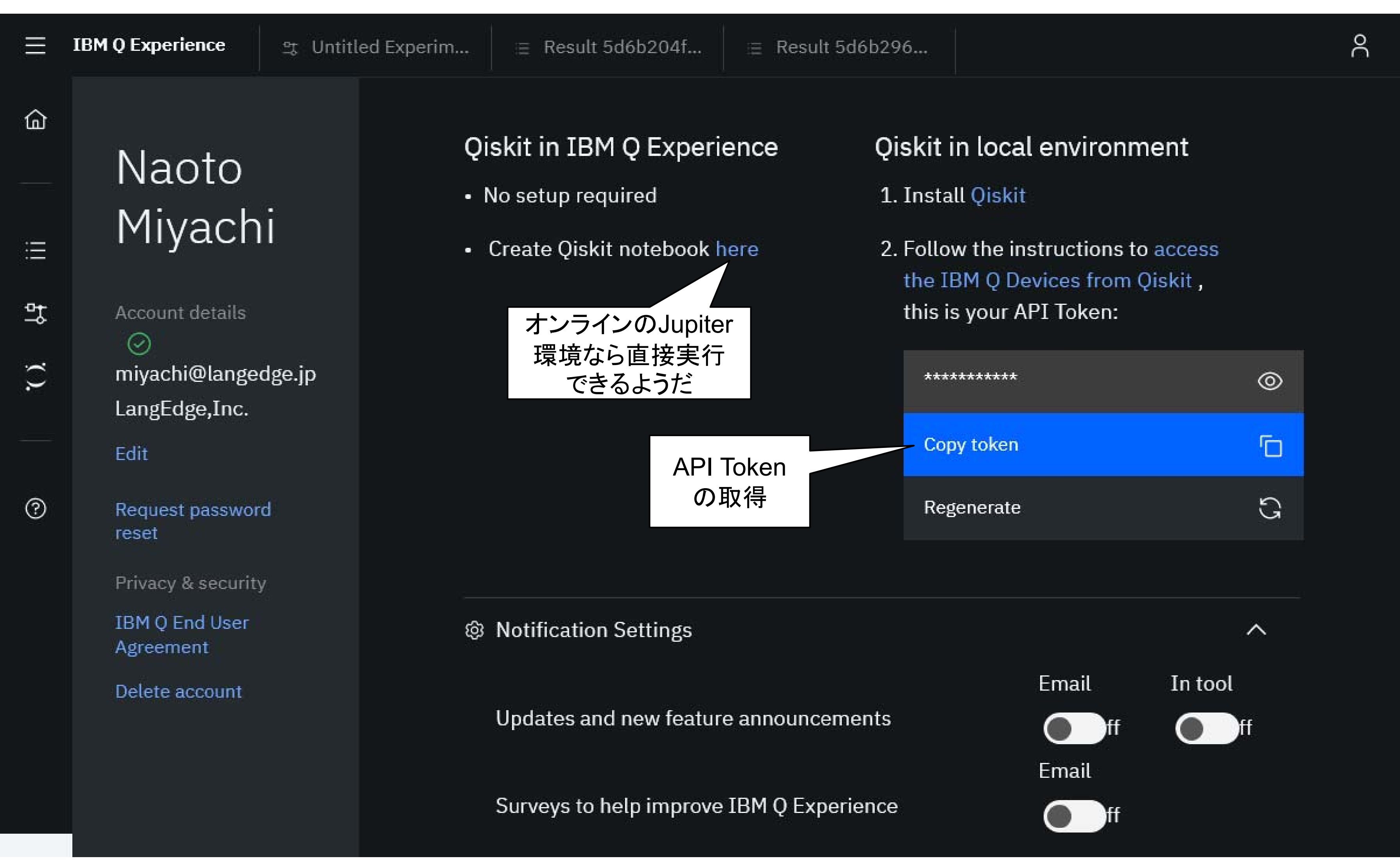
IBM Q Experience 実行



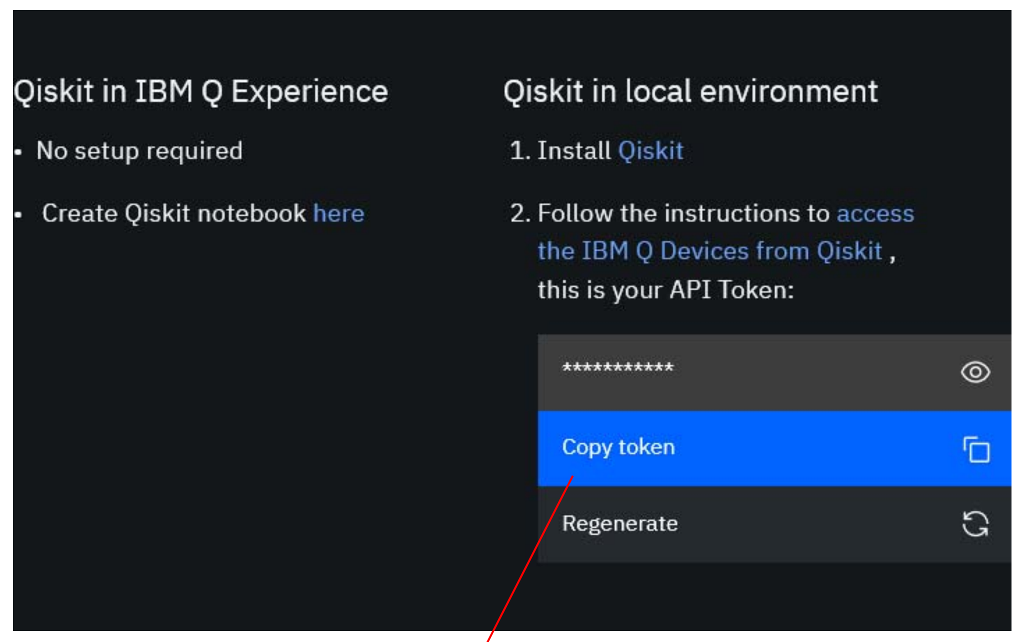
IBM Q Experience の notebook 画面



IBM Q Experience の API Token 取得



Local Qiskit で IBM Q 実機を使う(準備)



登録: 1回のみ	from qiskit import IBMQ IBMQ.save_account('MY_API_TOKEN')
読み込み: セッション毎	from qiskit import IBMQ prov = IBMQ.load_account()
指定:	from qiskit import IBMQ backend = prov.get_backend('ibmqx4')
バージョン:	import qiskit qiskit.__version__

.qiskit

```
[ibmq]
token = MY_API_TOKEN
url = https://quantumexperience.ng.bluemix.net/api
verify = True
```

ここまでが前準備。
OKなら次ページのプログラム実行。

Qiskit で IBM Q 実機を使う(実行)

```
from qiskit import *                                # 量子計算用
from qiskit import IBMQ                             # 実機利用用
print("Start: Load accounts")
prov = IBMQ.load_account()                                # 実機用にアカウントロード
backend = prov.get_backend('ibmq_ourense')               # 実機指定：IBM Q 5 qubit
#backend = prov.get_backend('ibmq_qasm_simulator')       # 量子シミュレータ指定
q = QuantumRegister(1)        # 量子ビットを1つ用意
c = ClassicalRegister(1)      # 古典ビットを1つ用意
qc = QuantumCircuit(q, c)     # 量子回路に量子ビットと古典ビットをセット
qc.h(q[0])                    # 量子重ね合わせ（アダマール演算）
qc.measure(q[0], c[0])        # 量子ビットq[0]を観測し古典ビットc[0]へ
print("Run: Start")
r = execute(qc, backend, shots=100).result() # 回路を100回実行する
print("Run: End:")
rc = r.get_counts()           # 結果取得
print(rc)                     # 結果表示
```

この部分は実機でもシミュレータでも同じ。

```
Start: Load accounts
Run: Start
Run: End:
{'1': 48, '0': 52}
```

Run: Start 後に数分～10数分かかる場合があります(キュー実行の為)

IBM Q 実機での実行結果！

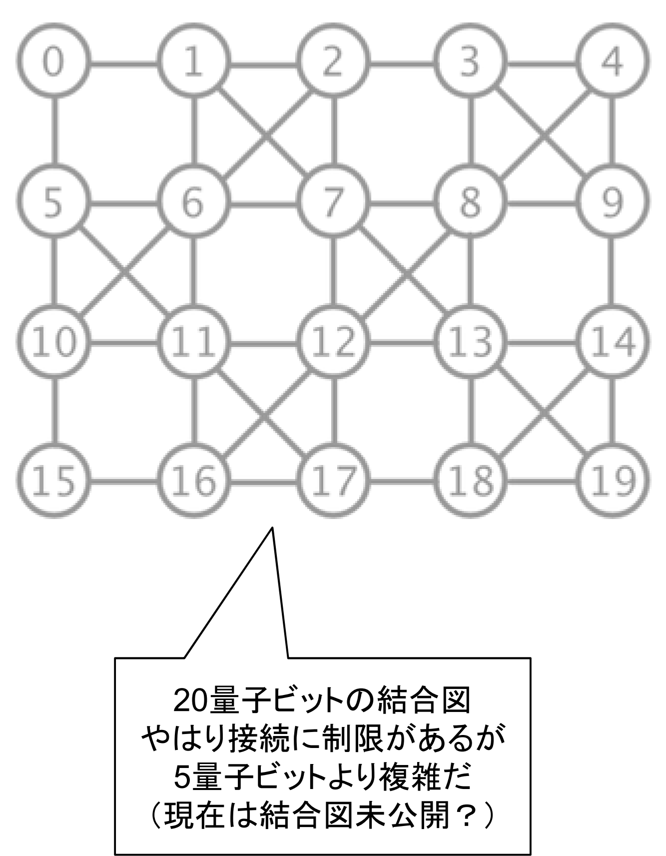
※ 実行状況は自分の Dashboard で確認できます。

IBM Q の 20量子ビット機

IBM Q 20 Tokyo

20

qubits



※ IBM Q Network で利用可能

Availability & status

For IBM Q clients

● Online

Last calibration occured

2019-03-24 8:14:51 pm

Average measurements

Frequency (GHz)	4.97
T1 (μs)	81.75
T2 (μs)	51.07
Gate error (10^{-3})	1.88
Readout error (10^{-2})	7.44

https://www.research.ibm.com/ibm-q/technology/devices/

参考：QASM（OpenQASM 2.0）

IBMが公開している量子回路アセンブラ言語。

Cirq（Google）、Q#（MS）、Blueqat（MDR）等のほぼ全ての量子プラットホームがサポートしているので相互運用ができる。

Visual Studio Code用の拡張も公開されている。

https://marketplace.visualstudio.com/items?itemName=qiskit.qiskit-vscode

|0⟩ H
c[0]

The IBM Q Experience は QASMの回路エディタにも使える

QASM

```
include "qelib1.inc";
qreg q[1];
creg c[1];
h q[0];
measure q[0] -> c[0];
```

Part 2：量子ゲート型のプログラミング

GoogleのCirqを使う

環境： Anaconda3（Python3.5）

以下より環境に合わせてダウンロードとインストール

https://www.anaconda.com/distribution/

ライブラリ： Cirq （シルク）

Windows版： Anaconda Prompt
MacOS版：ターミナル

インストール

```
pip install cirq
```

バージョン指定インストール

```
pip install cirq=0.5.0
```

アンインストール

```
pip uninstall cirq
```

※ Cirqのバージョン確認：

In:	import cirq cirq.__version__
Out:	'0.5.0‘

本資料のソースは 0.5.0 と表示される環境にて確認しています。

Cirq アダマール演算（量子 Hello World!）

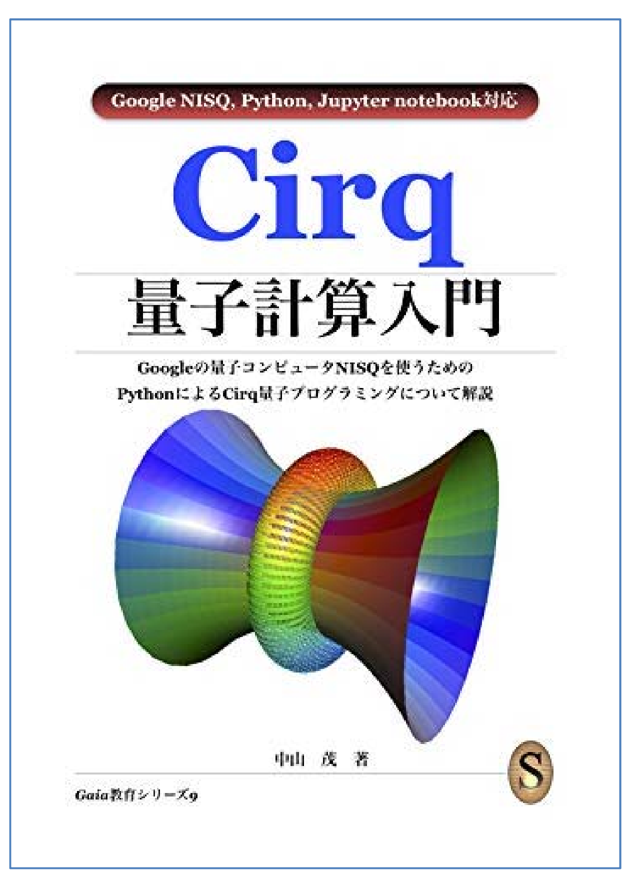
Jupiter Notebook から以下を実行（コピペで大丈夫です）。

```
import cirq                                    # cirq量子計算用
Q = cirq.LineQubit(0)                          # 量子ビットを1つ準備
qc = cirq.Circuit.from_ops(                    # 量子回路生成
  cirq.H(Q),                                   # 量子重ね合わせ（アダマール演算）
  cirq.measure(Q, key='m')                     # 量子ビットQを観測しkey名'm'へ.
)
print("Circuit:")
print(qc)                                      # 量子回路表示
qsim = cirq.Simulator()                        # 量子シミュレータ
result = qsim.run(qc, repetitions=1000)        # 回路を1000回実行する
print("Results:")
result.histogram(key='m')                      # key名'm'のヒストグラム表示
```

実行結果。

```
Circuit:
(0, 0): ───H───M('m')───
Results:
Counter({0: 494, 1: 506})
```

量子ゲート: 参考図書



Cirq量子計算入門

中山 茂 (著) - 209 ページ
Amazon Services International, Inc. (2018/8/4)

Kindle版：7560円
https://www.amazon.co.jp/gp/product/B07GJPPPJW/

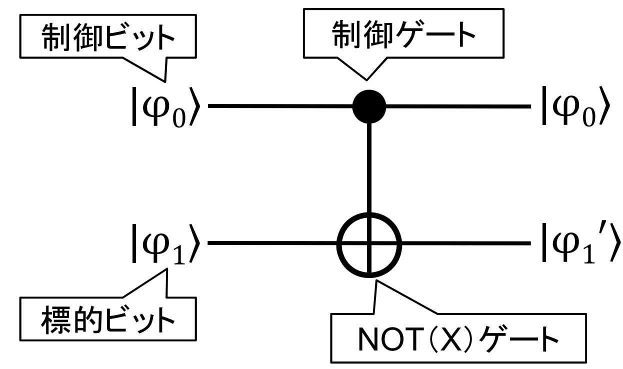
Amazonで検索すると同著者の図書が沢山ありますが、内容はほぼ同じで使っているSDK等が違うだけなので買うとしても1冊あれば充分。
Cirqがリリースされてすぐに書かれた本なので少し古く、推薦図書では無いが参考図書として利用は可能。

2-1: 複数量子ビット操作

Part1 では1量子ビット操作の話をしました。
（ここまでは大丈夫ですか？）

Part2 は量子ゲート型の複数の量子ビット操作
から説明して行きます。

制御反転CNOT（CX）ゲート



CNOTゲートの演算は、制御(Controlled)ビットが $|1\rangle$ の時のみ標的(Targeted)ビットが反転する。

$$制御反転演算\ \mathrm{CNOT}(q0, q1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

CXとしても良い

$\|\varphi_0\rangle$	$\|\varphi_1\rangle$	$\|\varphi_0\rangle$	$\|\varphi_1'\rangle$
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

テンソル積

$\mathrm{CNOT}\,|00\rangle = |00\rangle$
$\mathrm{CNOT}\,|01\rangle = |01\rangle$
$\mathrm{CNOT}\,|10\rangle = |11\rangle$
$\mathrm{CNOT}\,|11\rangle = |10\rangle$

反転

出力 $|\varphi_1'\rangle$ だけ見ると XOR ゲートと言える

ベル状態（量子もつれ）

ベル状態とは、2つの量子ビット間に量子もつれを生じている状態であり、1つを観測するともう片方の値が決まる。量子回路としてはアダマールとCNOTを使って実現可能。

$|\varphi_0\rangle$ —[H]—●— $|\varphi_0'\rangle$

$|\varphi_1\rangle$ ——————⊕— $|\varphi_1'\rangle$

ベル状態（量子もつれ）回路

$$|\varphi_0\varphi_1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\end{pmatrix} \\ \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\end{pmatrix} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

$$|\varphi_0'\varphi_1'\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

入力： $|\varphi_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, $|\varphi_1\rangle = |0\rangle$

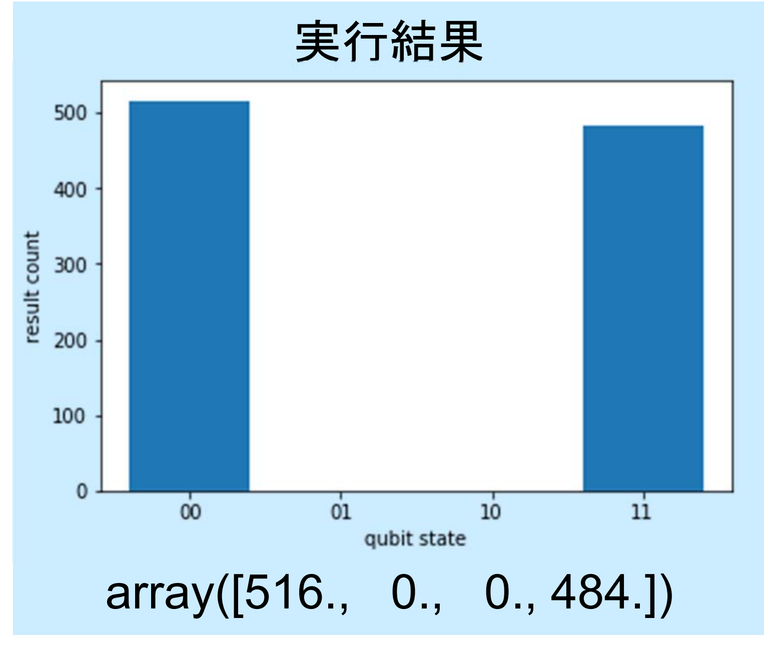
出力： $|\varphi_0'\varphi_1'\rangle = \frac{1}{\sqrt{2}}|00\rangle + \boxed{0|01\rangle + 0|10\rangle} + \frac{1}{\sqrt{2}}|11\rangle$ （囲み部分 $= 0\%$）

※ $|\varphi_0\rangle$が$|0\rangle$で観測されると$|\varphi_1'\rangle$も$|0\rangle$に、$|\varphi_0\rangle$が$|1\rangle$で観測されると$|\varphi_1'\rangle$も$|1\rangle$となる。

ベル状態回路の実行

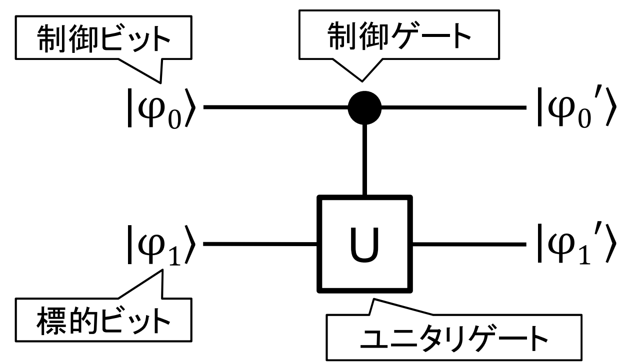
```
import cirq
Q = [cirq.LineQubit(i) for i in range(2)]
qc = cirq.Circuit.from_ops(
  cirq.H(Q[0]),
  cirq.CNOT(Q[0],Q[1]),
  cirq.measure(Q[0], key='q0'),
  cirq.measure(Q[1], key='q1')
)
print("Circuit:")
print(qc)
qsim = cirq.Simulator()
result = qsim.run(qc, repetitions=1000)
print("Result:")
cirq.plot_state_histogram(result)
```

```
Circuit:
0: ───H───@───M('q0')───
          │
1: ───────X───M('q1')───
```



q0が0ならq1も0に、
q0が1ならq1も1になる。
01と10にはならない。
ベル状態(量子もつれ)
を確認できる。

制御ユニタリ CU ゲート



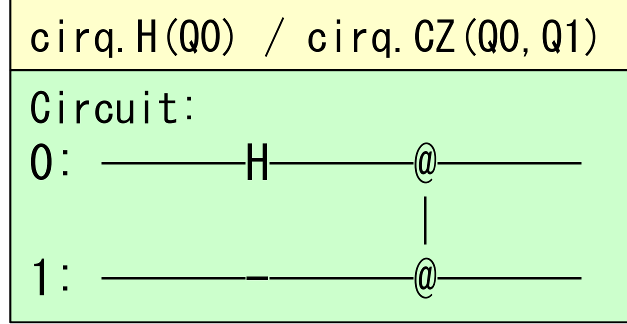
CX（CNOT）ゲートは標的ビット側をXゲートにしたが、YやZ,H,S等の任意ユニタリ演算も指定できる。

$$CU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{pmatrix}$$

$$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = Y \qquad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = Z$$

ここに任意のユニタリ演算行列を入れる

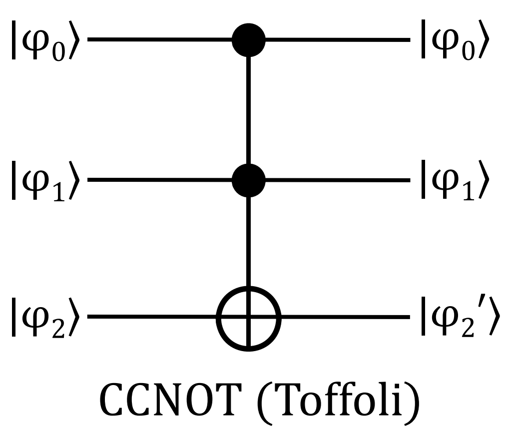
例）Zゲートの CZ(q0, q1) ベル演算結果： $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$

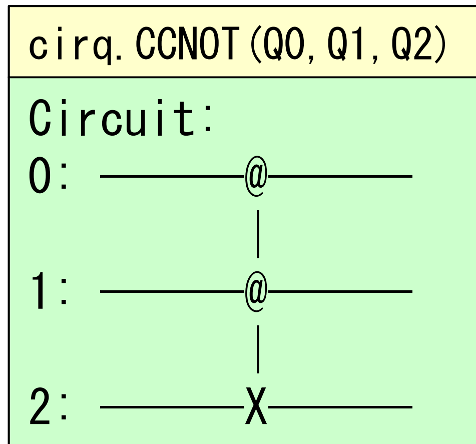


CZゲートは上下を入れ替えても同じ。回路ではどちらも“@” で表現される。

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

トフォリ CCNOT ゲート





制御反転ゲートに対しもう1つ制御ビットを追加したCCNOTゲート。

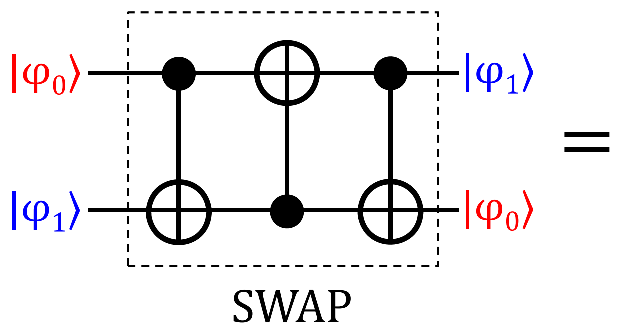
トフォリ演算 CCNOT (q0, q1, q2) ＝

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

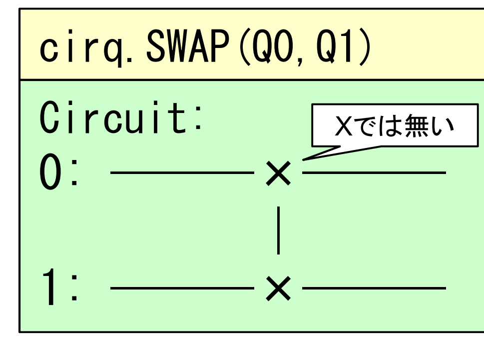
CSWAP |000⟩ = |000⟩　CSWAP |100⟩ = |100⟩
CSWAP |001⟩ = |001⟩　CSWAP |101⟩ = |101⟩
CSWAP |010⟩ = |010⟩　CSWAP |110⟩ = |111⟩
CSWAP |011⟩ = |011⟩　CSWAP |111⟩ = |110⟩
反転

※トフォリゲートはAND/OR/NOT等が実現可能な万能ゲートである。

交換 SWAP ゲート



=



CNOTゲートを3つ交互に重ねると量子ビットの値を交換できる。

交換演算 SWAP(q0, q1) = $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

SWAP $|00\rangle = |00\rangle$
SWAP $|01\rangle = |10\rangle$
SWAP $|10\rangle = |01\rangle$
SWAP $|11\rangle = |11\rangle$
（中央2行：反転）

計算過程：

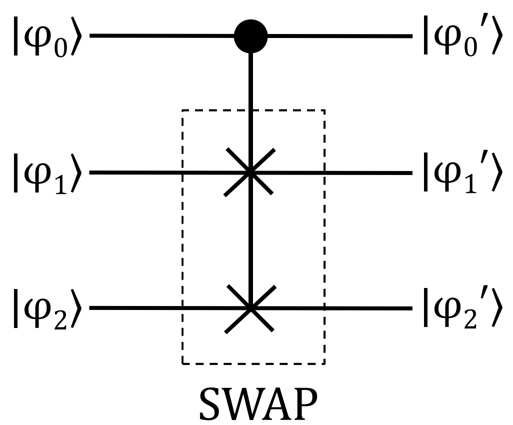
$|\varphi_0\rangle|\varphi_1\rangle \rightarrow$

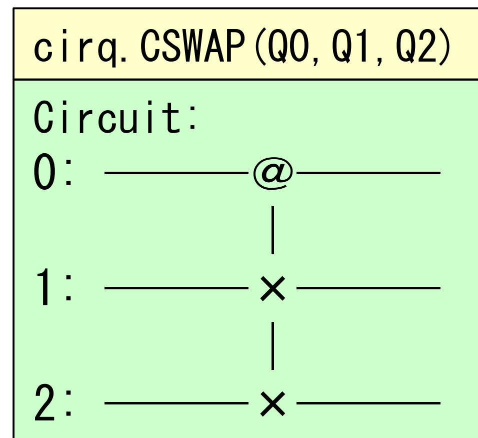
(CNOT$_{01}$)→ $|\varphi_0\rangle|\varphi_1 \oplus \varphi_0\rangle$ （⊕ は XOR）

(CNOT$_{10}$)→ $|\varphi_0 \oplus (\varphi_1 \oplus \varphi_0)\rangle|\varphi_1 \oplus \varphi_0\rangle = |\varphi_1\rangle|\varphi_1 \oplus \varphi_0\rangle \rightarrow$

(CNOT$_{01}$)→ $|\varphi_1\rangle|(\varphi_1 \oplus \varphi_0) \oplus \varphi_1\rangle = |\varphi_1\rangle|\varphi_0\rangle$

制御交換 CSWAPゲート





交換ゲートに対し
制御ビットを追加
したCSWAPゲート。

$$\text{制御交換演算 } CSWAP(q0, q1, q2) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$CSWAP\,|000\rangle = |000\rangle$　$CSWAP\,|100\rangle = |100\rangle$

$CSWAP\,|001\rangle = |001\rangle$　$CSWAP\,|101\rangle = |110\rangle$

$CSWAP\,|010\rangle = |010\rangle$　$CSWAP\,|110\rangle = |101\rangle$

$CSWAP\,|011\rangle = |011\rangle$　$CSWAP\,|111\rangle = |111\rangle$

反転

2-2: 量子アルゴリズムの基本

複数の量子ビットを使った演算も行えるようになりました。これをどう使えば量子計算ができるのかを疑問に思われているでしょう。

まず量子アルゴリズムの基本となる、位相に関係する仕組みを勉強します。

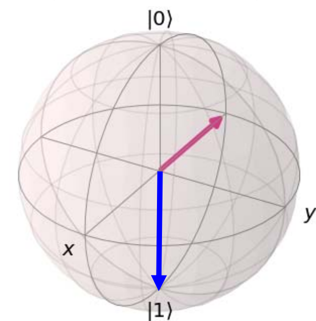
プラスケット $|+\rangle$ と マイナスケット $|-\rangle$

$|+\rangle$ は、$|0\rangle$ をアダマール変換した重ね合わせ状態。
$|+\rangle$ を再度アダマール変換すると $|0\rangle$ に戻る。

$$|+\rangle = \mathrm{H}|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$\mathrm{H}|+\rangle = |0\rangle$$

$|-\rangle$ は、$|1\rangle$ をアダマール変換した重ね合わせ状態。
$|-\rangle$ を再度アダマール変換すると $|1\rangle$ に戻る。
$|+\rangle$ と $|-\rangle$ は逆位相の関係。



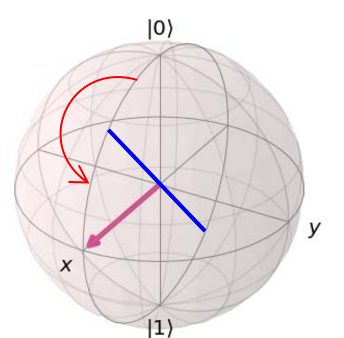
$$|-\rangle = \mathrm{H}|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$\mathrm{H}|-\rangle = |1\rangle$$

アダマール変換と位相

プラスケット $|+\rangle$ と マイナスケット $|-\rangle$ も考慮すると、
アダマール変換では、入力ケット（下の例では $|\varphi\rangle$）が、
位相に影響すると言う見方もできる。

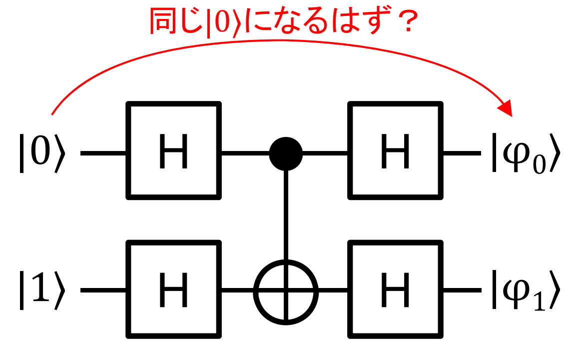
$$H|\varphi\rangle = \frac{1}{\sqrt{2}}|0\rangle + (-1)^{\varphi}\frac{1}{\sqrt{2}}|1\rangle$$



重要!

アダマール変換は、
重ね合わせ状態の生成と同時に、
振幅（入力ベクトル）を位相に変換する。

制御反転CNOTゲートと位相の反転



CNOTゲートの制御(Controlled)ビットは同じ値が出力される…はず？
ではアダマールで重ね合わせした値を入力するとどうなる？

答え：実行すると $|\varphi_0\rangle = |1\rangle$, $|\varphi_1\rangle = |1\rangle$ になる。
あれ？なぜ制御ビットの値が変化しているの？

$$|0\rangle \otimes |1\rangle \rightarrow (H \otimes H) \rightarrow |+\rangle \otimes |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |-\rangle$$

$$\rightarrow (CNOT) \rightarrow \frac{1}{\sqrt{2}}\left((-1)^{NOT(0)}|0\rangle + (-1)^{NOT(1)}|1\rangle\right) \otimes |-\rangle$$

$$= \frac{1}{\sqrt{2}}(-|0\rangle + |1\rangle) \otimes |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes |-\rangle = |-\rangle \otimes |-\rangle$$

$$\rightarrow (H \otimes H) \rightarrow |1\rangle \otimes |1\rangle$$

制御ビットの値は変化しないが位相が反転する。

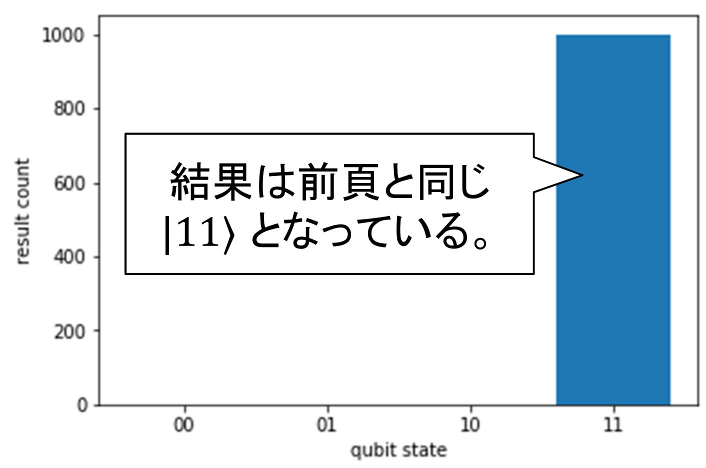
アダマールとCNOTの確認

Jupiter Notebook から以下回路を実行。

```
Q = [cirq.LineQubit(i) for i in range(2)]  # 2量子ビットを用意
qc = cirq.Circuit.from_ops(
  cirq.X(Q[1]),                          # Q1を反転して |1⟩ にする
  cirq.H.on_each(*Q),                    # Q0/Q1にアダマール変換
  cirq.CNOT(Q[0],Q[1]),                  # CNOT適用
  cirq.H.on_each(*Q),                    # Q0/Q1にアダマール変換
  cirq.measure(Q[0], key='q0'),          # Q0を観測
  cirq.measure(Q[1], key='q1')           # Q1を観測
)
```

実行結果。

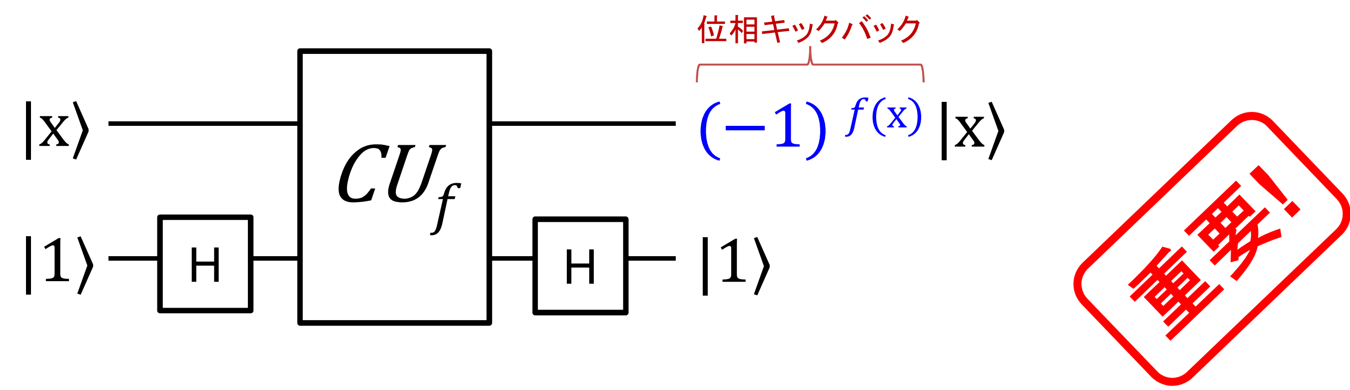
```
Circuit:
0: ─────────H───@───H───M('q0')───
                │
1: ───X─────H───X───H───M('q1')───
Results:
array([   0.,     0.,     0.,  1000.])
```



位相キックバック（Phase kick back）

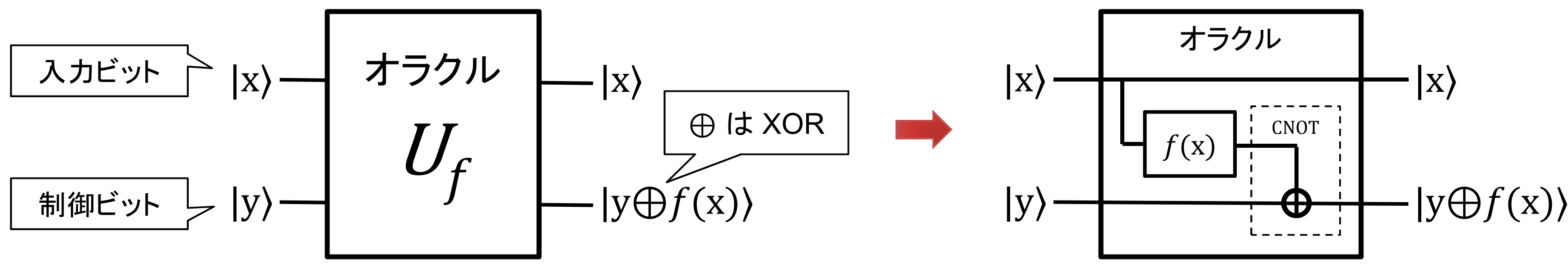
前頁のCNOT（CX）では位相が制御ビットに影響したが
これは一般的な制御ユニタリCUゲートにも適用可能。

CUゲートの動作を関数 $f(x)$ とすると、
関数を制御ビットの位相に反映する。
これを**位相キックバック**と呼ぶ。



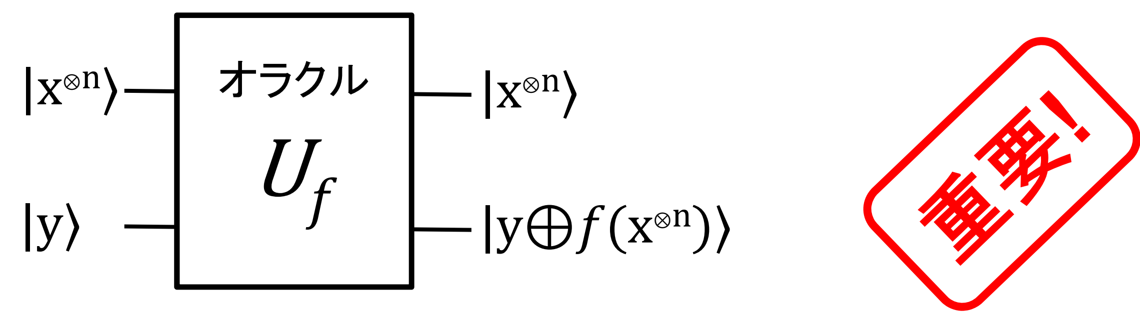
オラクル(Oracle：神託)ボックス

関数 $f(x)$ を計算するオラクル(神託)ボックス U_f を想定。



オラクルボックス U_f はユニタリ演算のブラックボックス。

入力は複数ビットに拡張できる。



2-3: ドイチェ アルゴリズム

位相キックバックとオラクルボックスが分かったところで、オラクルボックスを使う基本として、ドイチェアルゴリズムを見ます。

ドイチェ(Deutsch)問題

ドイチェ問題は**1ビット関数 $f(x)$ の性質(種類)を判定**。

性質A:一定(constant):出力は常に一定の値となる。

$f(0)=0$ かつ $f(1)=0$、または $f(0)=1$ かつ $f(1)=1$

性質B:均等(balanced):0 と 1 が均等(50%)の確率で出力される。

$f(0)=0$ かつ $f(1)=1$、または $f(0)=1$ かつ $f(1)=0$

	性質A:一定(constant)		性質B:均等(balanced)	
$f(0)$	0	1	0	1
$f(1)$	0	1	1	0

古典アルゴリズムでは $f(0)$ と $f(1)$ の両方結果を取得しないと、どちらの性質(種類)か判定ができない。古典アルゴリズムでは $f(0)=0$ だったとしても $f(1)$ の結果を見る必要がある。

つまり**問い合わせ**が $f(0)$ と $f(1)$ の**必ず2回必要**となる。

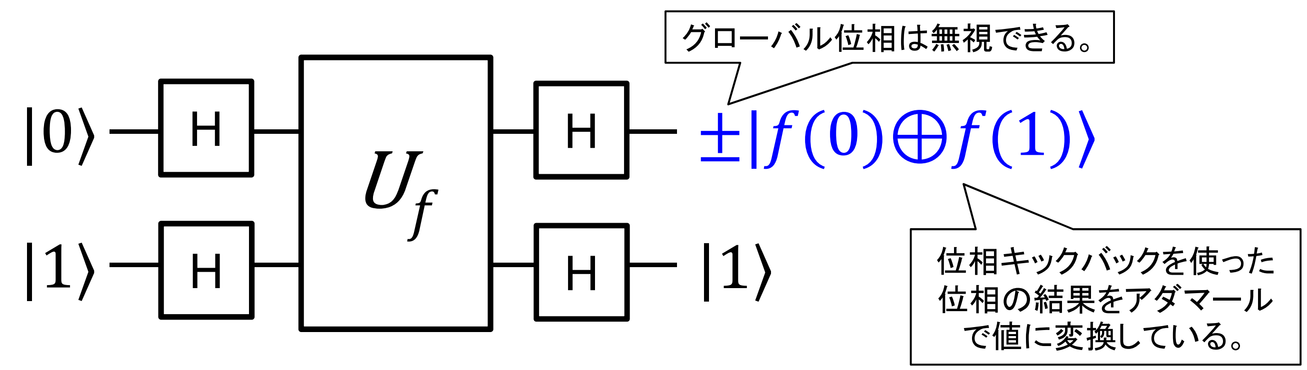
ドイチェ問題の量子計算

入力を $|0\rangle$ と $|1\rangle$ とする。

重ね合わせ状態を作る。

測定ユニタリの入出力にアダマールHを適用。

上位ビット出力は $f(0)$ と $f(1)$ の⊕(XOR)となる。



上位ビット出力 $f(0) \oplus f(1)$ を確認して判定。

一定(constant) : $f(0) \oplus f(1) = |0\rangle$

均等(balanced) : $f(0) \oplus f(1) = |1\rangle$

ドイチェ問題を計算する為の定義

	性質A：一定（constant）		性質B：均等（balanced）	
	$f_{00}(x)$	$f_{11}(x)$	$f_{01}(x)$	$f_{10}(x)$
$f(0)$	**0**	**1**	**0**	**1**
$f(1)$	**0**	**1**	**1**	**0**
f 変換	$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
U_f ゲート	U_{f00} (|x⟩ → |x⟩, |y⟩ → |y⊕f(x)⟩)	U_{f11}	U_{f01}	U_{f10}
cirq	def oracle_f00(x, y_fx) return yield ← 何もしない	def oracle_f11(x, y_fx) yield X(y_fx)	def oracle_f01(x, y_fx) yield CNOT(x, y_fx)	def oracle_f10(x, y_fx) yield X(y_fx) yield CNOT(x, y_fx)

ドイチェ問題の計算

Jupiter Notebook から以下回路を実行（左右は連続）。

```
from cirq import *
# ユニタリ定義
def oracle_f00(x, y_fx):
    return
    yield
def oracle_f01(x, y_fx):
    yield CNOT(x, y_fx)
def oracle_f10(x, y_fx):
    yield X(y_fx)
    yield CNOT(x, y_fx)
def oracle_f11(x, y_fx):
    yield X(y_fx)
```

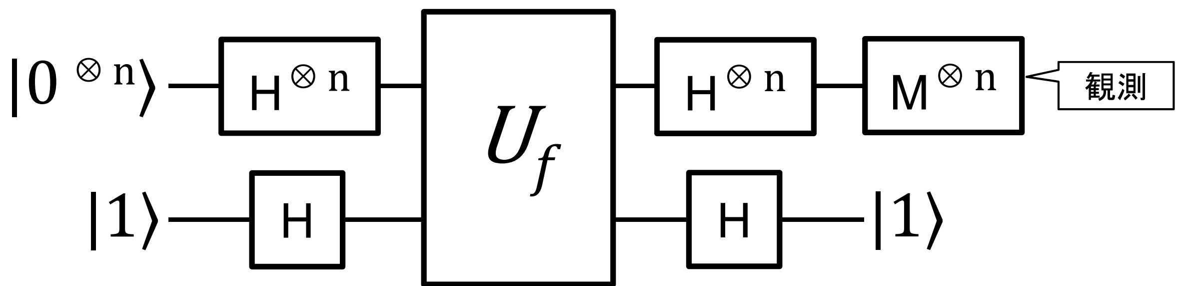
```
Q = [LineQubit(i) for i in range(2)]
qc = Circuit()
qc.append(X(Q[1]))
qc.append(H.on_each(*Q))
qc.append(oracle_f00(Q[0], Q[1]))
# qc.append(oracle_f01(Q[0], Q[1]))
# qc.append(oracle_f10(Q[0], Q[1]))
# qc.append(oracle_f11(Q[0], Q[1]))
qc.append(H.on_each(*Q))
qc.append(measure(Q[0], key='c'))
r = Simulator().run(qc)
print(r)
```

実行結果。

qc.append(oracle_f00(Q[0], Q[1])) ⇒ （実行結果） ⇒ c=0 （constant）
qc.append(oracle_f01(Q[0], Q[1])) ⇒ （実行結果） ⇒ c=1 （balanced）
qc.append(oracle_f10(Q[0], Q[1])) ⇒ （実行結果） ⇒ c=1 （balanced）
qc.append(oracle_f11(Q[0], Q[1])) ⇒ （実行結果） ⇒ c=0 （constant）

ドイチェ・ジョサ(Deutsch-Jozsa)問題

ドイチェ問題の入力を複数量子ビットにした問題。
2ビット以上だと一定・均等以外のケースもある。
例：2ビットの時に 0010 のように一定でも均等でも無いケースがあり得る。
この為に一定・均等いずれかであることを前提。



複数ビットに対しても適用可能とすることで、
ビット数が増えても演算は1回で済む。

2ビットのドイチェ・ジョサ問題

	一定（constant）		均等（balanced）					
	f_{C0}	f_{C1}	f_{B0}	f_{B1}	f_{B2}	f_{B3}	f_{B4}	f_{B5}
$f(00)$	0	1	0	0	0	1	1	1
$f(01)$	0	1	0	1	1	1	0	0
$f(10)$	0	1	1	0	1	0	1	0
$f(11)$	0	1	1	1	0	0	0	1

$f_{C0}(x1, x2) = 0$,　$f_{C1}(x1, x2) = 1$

$f_{B0}(x1, x2) = x1$,　$f_{B1}(x1, x2) = x2$

$f_{B2}(x1, x2) = x1 \oplus x2$,　$f_{B3}(x1, x2) = \mathrm{NOT}(x1)$

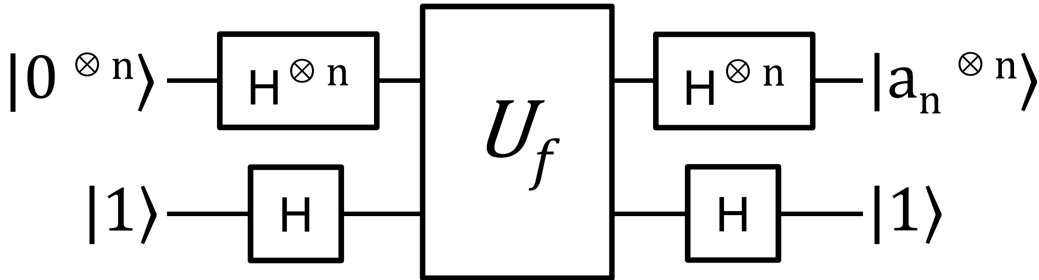
$f_{B4}(x1, x2) = \mathrm{NOT}(x2)$,　$f_{B5}(x1, x2) = \mathrm{NOT}(x1 \oplus x2)$

2ビットまではCNOT/NOTゲートで構成可能。
3ビット以上ではCCNOT（トフォリ）ゲートが必要。

その他の量子アルゴリズム問題

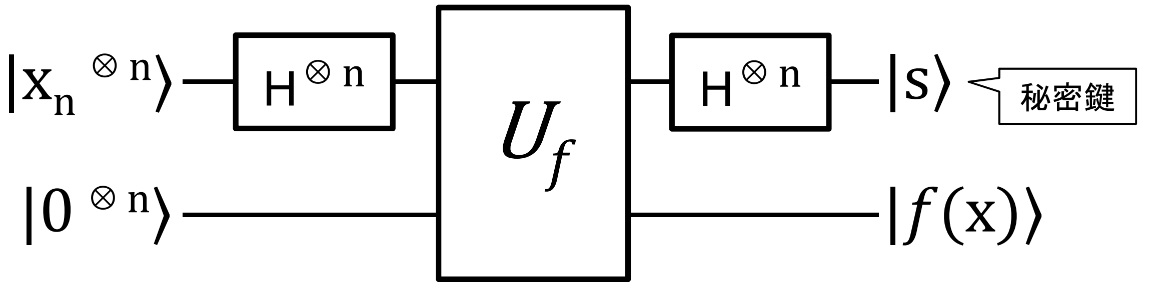
➢ ベルンシュタイン・ヴァジラニ(Bernstein-Vazirani)問題

nビットの変数xと定数aの内積を解とする関数f(x)がある時に定数aを求める。
$f(x)=x \cdot a=(x_0 \cdot a_0)\oplus(x_1 \cdot a_1)\oplus\ldots\oplus(x_n \cdot a_n)$



➢ サイモン(Simon)問題

nビット変数xと関数f(x)がある時に、$f(x)=f(x\oplus s)$となる秘密鍵sを得る問題。
ただし$s\neq0$とする。古典では2^n回かかるが量子ではn回で済む。



ドイチェ系 量子アルゴリズム

1. アダマールゲートによる重ね合わせを作る。
   - 複数の入力状態を同時に実現する。
   - 振幅(値)を位相に変換する。
2. ブラックボックスにて位相キックバックを使う。
   - 内容が分からないユニタリ変換関数の性質を得る問題用。
   - 関数を使ったオラクルボックスを設定する。
   - 関数の内容を位相にキックバックさせる。
3. アダマールゲートにより位相を振幅に戻す。
   - 位相を振幅(値)に変換する。

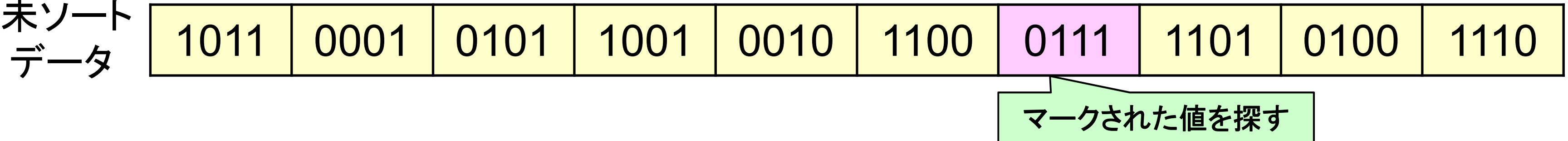
※ 量子アルゴリズムを使うことで、対象となるビット数が増えても1回の問合せで済む。これらの問題では確率的な解にはならず決定的な解となる。

2-4: グローバー検索（量子検索）

検索を高速化する重要な量子アルゴリズムが、
グローバー検索です。

グローバー(Grover)検索問題

n個の未ソート(ランダム)状態のデータがある時、解となる特定の値xを検索する問題である。



古典的な計算ではn回の検索が必要となるが、グローバー(量子)検索では$\sqrt{n}$回の検索で済む。
※ 量子検索でも1回では検索できないが充分早い。

検索と言っているが、関数 $y=f(x)$ がある時に、特定の解 y を与える x が存在するかどうかを、判断する逆関数の導出問題である。

振幅増幅手法

欲しい解の確率振幅をマイナスにマーキングして、全確率振幅の平均値で逆転させる計算手法で、グローバー検索で利用。

問題: $|\varphi\rangle = \frac{1}{2}(|00\rangle+|01\rangle+|10\rangle+|11\rangle)$ がある時に、この中から $|10\rangle$ を探す(2量子ビットにおける検索例)。

手順1: 解 $|10\rangle$ の確率振幅(位相)をマイナスにマーキングする。

$$|\varphi\rangle_{\text{marked}} = \frac{1}{2}(|00\rangle+|01\rangle-|10\rangle+|11\rangle)$$

手順2: 全確率振幅の平均値を求める。

$$\langle a\rangle = \left(\frac{1}{2}+\frac{1}{2}-\frac{1}{2}+\frac{1}{2}\right) / 4 = \frac{1}{4}$$

平均値の周りで逆転する為に、
平均値から確率振幅を引いた後で平均値を足す

手順3: 全確率振幅の平均値の周りで反転させる。

$$|\varphi\rangle'_{\text{marked}} = \left(\frac{1}{4}-\frac{1}{2}+\frac{1}{4}\right)|00\rangle + \left(\frac{1}{4}-\frac{1}{2}+\frac{1}{4}\right)|01\rangle + \left(\frac{1}{4}+\frac{1}{2}+\frac{1}{4}\right)|10\rangle + \left(\frac{1}{4}-\frac{1}{2}+\frac{1}{4}\right)|11\rangle = |10\rangle$$

求める $|10\rangle$ が
見つかった

|10⟩ のマーキング実行

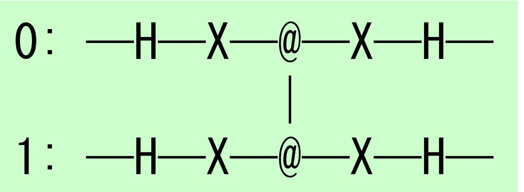
```
from cirq import *
import numpy as np          # 結果表示にnumpyを使う
Q = [LineQubit(i) for i in range(2)]
qc = Circuit()
qc.append(H.on_each(*Q))    # 均等な重ね合わせ状態を作る
qc.append(S(Q[0]))
qc.append(CZ(*Q))
qc.append(S(Q[0]))
print("Circuit:")
print(qc)
r = Simulator().simulate(qc)     # 位相を見る
print(np.around(r.final_state, 3))  # 丸めて結果表示
```

```
Circuit:
0: ───H───S───@───S───
              │
1: ───H───────@───────
[ 0.5+0.j   0.5+0.j  -0.5+0.j   0.5-0.j]
```

2量子ビット確率分布(位相)マーキング

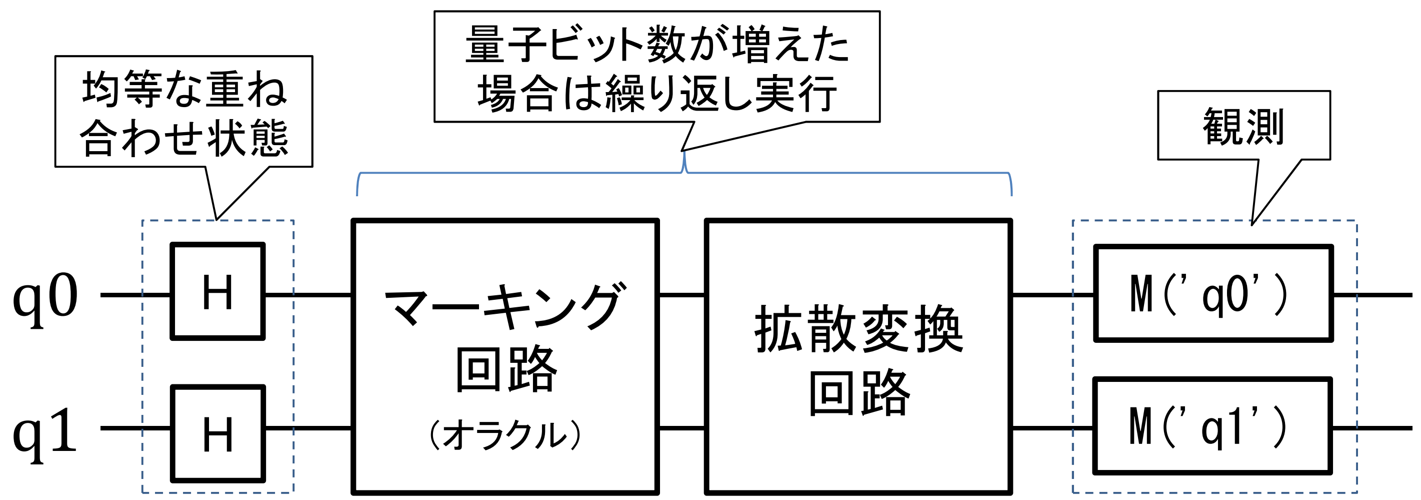
対象	回路	回路ソース	実行結果
$\|00\rangle$	Q0: —H—S—@—S— │ Q1: —H—S—@—S—	H.on_each(*Q) S.on_each(*Q) CZ(*Q) S.on_each(*Q)	[0.5+0.j -0.5+0.j -0.5+0.j -0.5+0.j]
$\|01\rangle$	Q0: —H———@——— │ Q1: —H—S—@—S—	H.on_each(*Q) S(Q[1]) CZ(*Q) S(Q[1])	[0.5+0.j -0.5+0.j 0.5+0.j 0.5-0.j]
$\|10\rangle$	Q0: —H—S—@—S— │ Q1: —H———@———	H.on_each(*Q) S(Q[0]) CZ(*Q) S(Q[0])	[0.5+0.j 0.5+0.j -0.5+0.j 0.5-0.j]
$\|11\rangle$	Q0: —H———@——— │ Q1: —H———@———	H.on_each(*Q) CZ(*Q)	[0.5+0.j 0.5+0.j 0.5+0.j -0.5+0.j]

平均値の周りで反転（拡散変換）



平均値の周りで反転させる（拡散変換）回路

グローバー検索の回路



2量子ビットのグローバー検索の実行

```
from cirq import *
Q = [LineQubit(i) for i in range(2)]
qc = Circuit()
# 均等な重ね合わせ状態を作る
qc.append(H.on_each(*Q))
# マーキング |10>
qc.append(S(Q[0]))
qc.append(CZ(*Q))
qc.append(S(Q[0]))
# 平均値の周りで反転 （拡散変換）
qc.append(H.on_each(*Q))
qc.append(X.on_each(*Q))
qc.append(CZ(*Q))
qc.append(X.on_each(*Q))
qc.append(H.on_each(*Q))
# 観測 （結果）
qc.append(measure(Q[0], key='q0'))
qc.append(measure(Q[1], key='q1'))
# 実行と結果表示
r = Simulator().run(qc, repetitions=1000)
plot_state_histogram(r)
```

解として|10⟩ が出力された

実行結果

array([0., 0., 1000., 0.])

繰り返し数の最適回数

今回は2量子ビットの問題を解いたので1回の拡散変換で結果が出たが、量子ビット数が増えると繰り返して拡散変換を実行する必要がある。なお最適回数をオーバーすると結果が悪くなる。

繰り返し数の最適回数Kは以下の式となる。

$$\text{最適回数 K} = \frac{\pi}{4}\sqrt{N} - \frac{1}{2}$$

N	計算結果	K
$2^2=4$	1.070796327…	1
$2^3=8$	1.721441469…	2
$2^4=16$	2.641592654…	3
$2^5=32$	3.942882938…	4
$2^6=64$	5.783185307…	6

N	計算結果	K
$2^8=256$	12.06637061…	12
$2^{10}=1024$	24.63274123…	25
$2^{12}=4096$	49.76548246…	50
$2^{14}=16384$	100.0309649…	100
$2^{16}=65536$	200.5619298…	201

グローバー検索の応用例

➢ 充足可能性(SAT：SATisfiability)問題

一つの命題論理式が与えられたとき、それに含まれる変数の値を偽 (False) あるいは真 (True) にうまく定めることによって全体の値を‘真’にできるか、という問題である。オラクルUを解きたいSATに合わせてセットすることで解ける。

SAT例題：$(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$ の時
解：x_1＝False, x_2＝True にすると全体が True になる。

➢ 素数探索問題

$|i\rangle$がある時に i が素数かどうかを判定するオラクルをグローバーのアルゴリズムを使って用意し、量子フーリエ変換と組み合わせて素数の探索を行う。

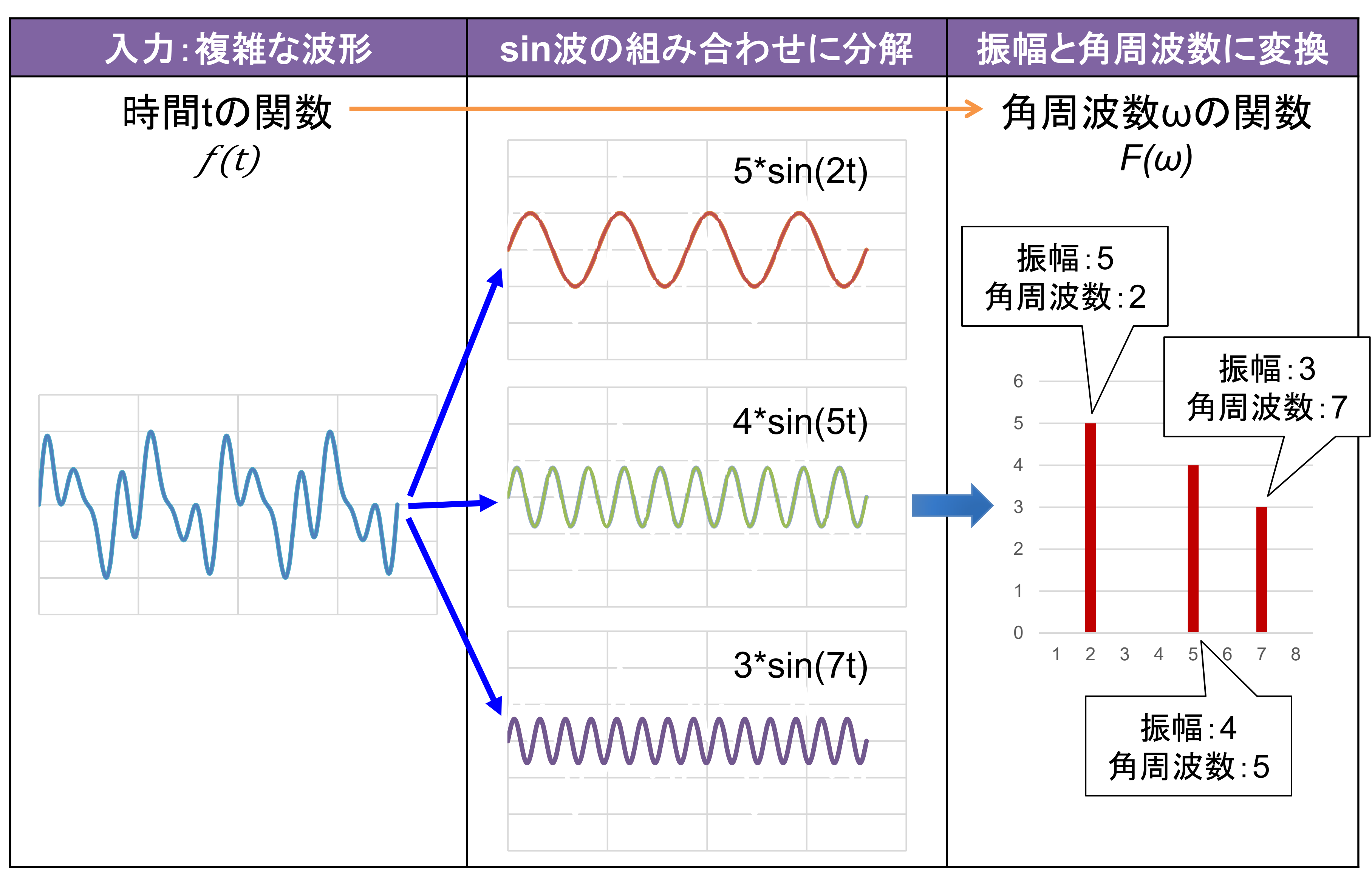
Jose I. Latorre, German Sierra
Quantum Computation of Prime Number Functions,
arXiv:1302.6245 [quant-ph] (2013)
https://arxiv.org/abs/1302.6245

※ グローバーのアルゴリズムを使った論文は多く、応用性の高いアルゴリズムです。

2-5: 量子フーリエ変換

アナログ計算であるフーリエ変換は量子計算と相性が良いアルゴリズムです。また可逆性よりフーリエ変換が可能なら逆フーリエ変換の量子回路も可能となります。

フーリエ変換（波形を振幅と角周波数に分解）



アダマールは 1量子ビット フーリエ変換

※ アダマール変換は2回適用すると元に戻るが位相が逆の逆フーリエ変換でもある。

Step1: アダマール変換を量子フーリエ変換的に解く。

$$H|x\rangle = \frac{1}{\sqrt{2}}|0\rangle + (-1)^x \frac{1}{\sqrt{2}}|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$$

$$= \frac{1}{\sqrt{2}} \sum_{y=0}^{1} (-1)^{x \cdot y} |y\rangle$$

$x = 0 : \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

$x = 1 : \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

Step2: N（2^n）数の量子フーリエ変換（QFT_N）に拡張する。

$$QFT_N |x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega^{x \cdot y} |y\rangle$$

$\omega = \exp\left(\frac{2\pi i}{N}\right)$

ωの例：アダマールH（N=2）の時 $\omega = \exp(\pi i) = 180°$

※ 組み合わせ数（N=2^{n}ビット）が増えると角度（位相）は半分ずつ減って行く。

制御位相回転ゲート cRn

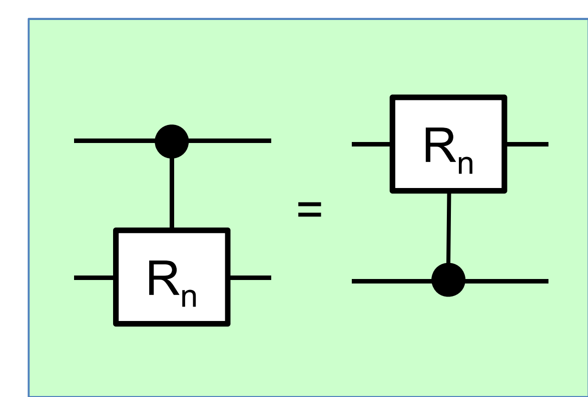
量子フーリエ変換は位相 $\exp\left(\frac{2\pi i}{2^n}\right)$ 計算が必要。

制御付きの位相回転ゲートRx の $x = 2^n$ として、

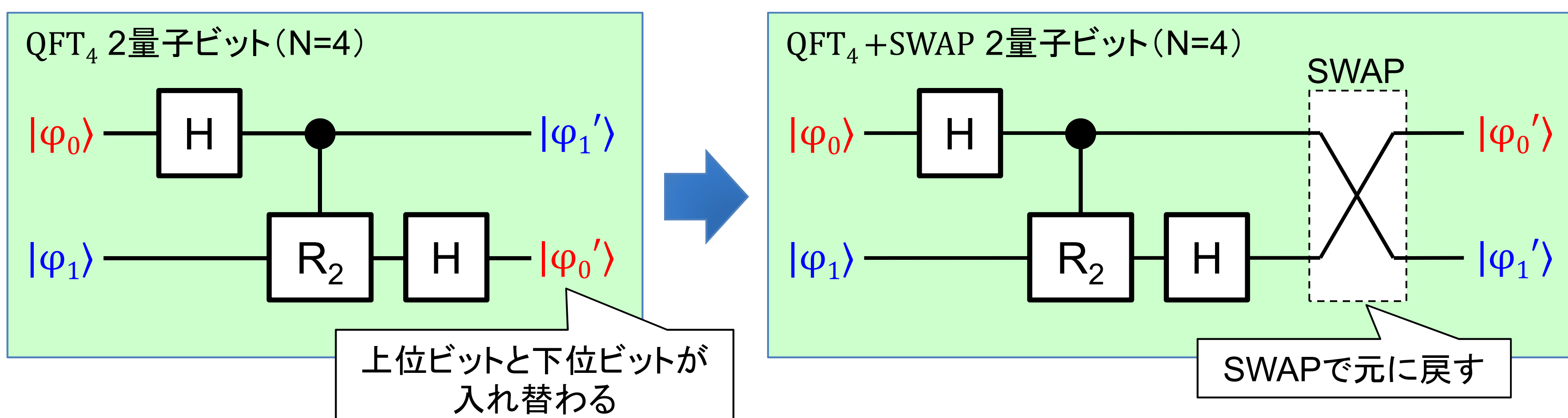
$cR_1 = CZ$, $cR_2 = CZ^{0.5}$, $cR_3 = CZ^{0.25}$,… $cR_n = CZ^{1/2^{n-1}}$

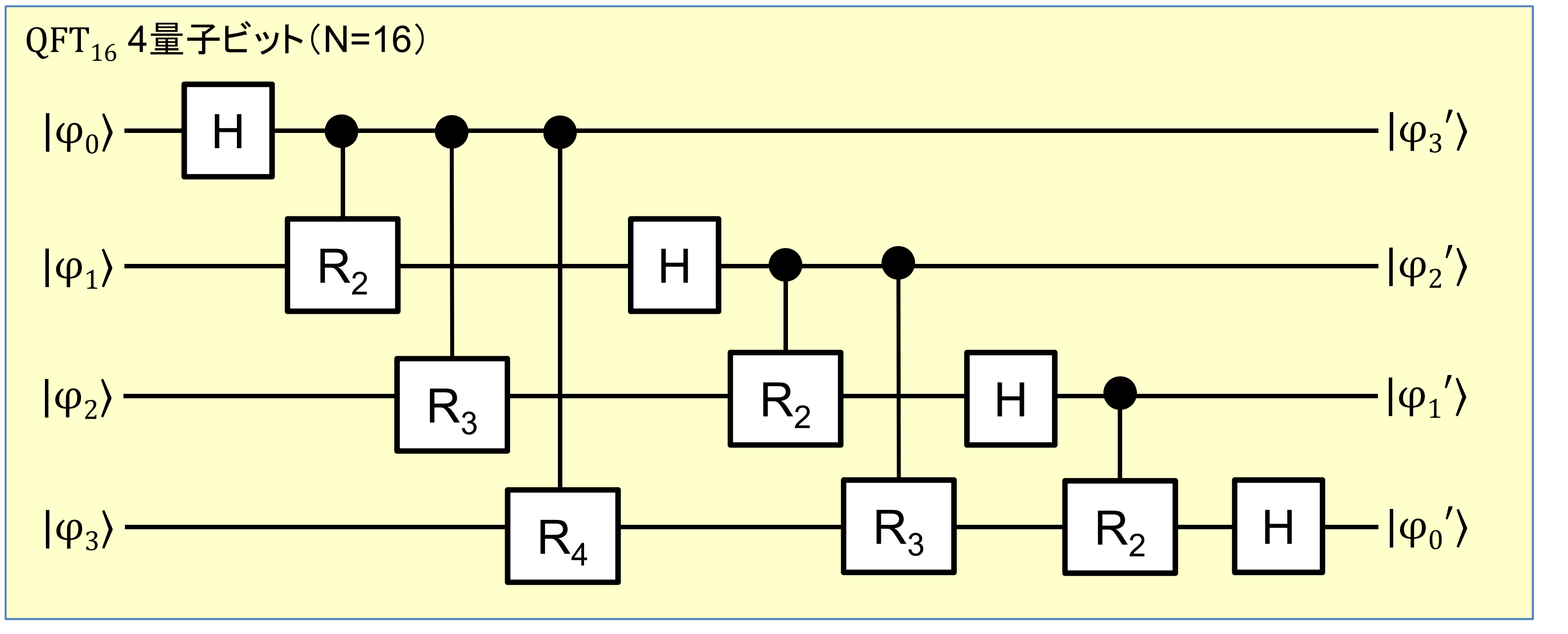
となる。Cirqでは $cR_2 = CZ^{0.5}$ = CZ(q)**0.5 とする。

$$cRn = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & (-1)^{1/2^{n-1}} \end{pmatrix} = CZ^{1/2^{n-1}}$$



n量子ビットの量子フーリエ変換 QFT_N





2量子ビットの量子フーリエ変換 計算

1. 入力が振幅(入力ベクトル)なら位相に変換して出力

$$\mathrm{QFT}_4 |00\rangle = \frac{1}{\sqrt{4}} \sum_{y=0}^{3} \omega^{0 \cdot y} |y\rangle$$

$$= \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \mathrm{H}\,|00\rangle$$

2. 入力が位相なら振幅(入力ベクトル)に変換して出力

$$\mathrm{QFT}_4\, \mathrm{H}\, |00\rangle = \frac{1}{4} \left(\sum_{y=0}^{3} \left(\sqrt{i}\right)^{0 \cdot y} |y\rangle + \sum_{y=0}^{3} \left(\sqrt{i}\right)^{1 \cdot y} |y\rangle + \sum_{y=0}^{3} \left(\sqrt{i}\right)^{2 \cdot y} |y\rangle + \sum_{y=0}^{3} \left(\sqrt{i}\right)^{3 \cdot y} |y\rangle \right)$$

$$= |00\rangle$$

量子干渉効果で打ち消し

※ $|00\rangle$ を使った場合は、2量子ビットのアダマール変換と同じ。

2量子ビットの量子フーリエ変換 実行1

```
from cirq import *
Q = [LineQubit(i) for i in range(2)]
qc = Circuit()
# 量子フーリエ変換
qc.append(H(Q[0]))
qc.append(CZ(*Q)**0.5)
qc.append(H(Q[1]))
# 上位と下位のビット入れ替え
qc.append(SWAP(Q[0],Q[1]))
# 観測（結果取得）
qc.append(measure(Q[0], key='q0'))
qc.append(measure(Q[1], key='q1'))
print("Circuit:")
print(qc)
# 実行と結果表示
r = Simulator().run(qc, repetitions=1000)
plot_state_histogram(r)
```

```
Circuit:
0: ──H──@──────────×──M('q0')──
        │          │
1: ─────@^0.5──H───×──M('q1')──
```

実行結果

array([264., 237., 250., 249.])

振幅（値）を位相に変換

2量子ビットの量子フーリエ変換 実行2

```
from cirq import *
Q = [LineQubit(i) for i in range(2)]
qc = Circuit()
# 重ね合わせ状態を入力
qc.append(H.on_each(*Q))
# 量子フーリエ変換
qc.append(H(Q[0]))
qc.append(CZ(*Q)**0.5)
qc.append(H(Q[1]))
# 上位と下位のビット入れ替え
qc.append(SWAP(Q[0],Q[1]))
# 観測（結果取得）
qc.append(measure(Q[0], key='q0'))
qc.append(measure(Q[1], key='q1'))
# 実行と結果表示
r = Simulator().run(qc, repetitions=1000)
plot_state_histogram(r)
```

実行結果

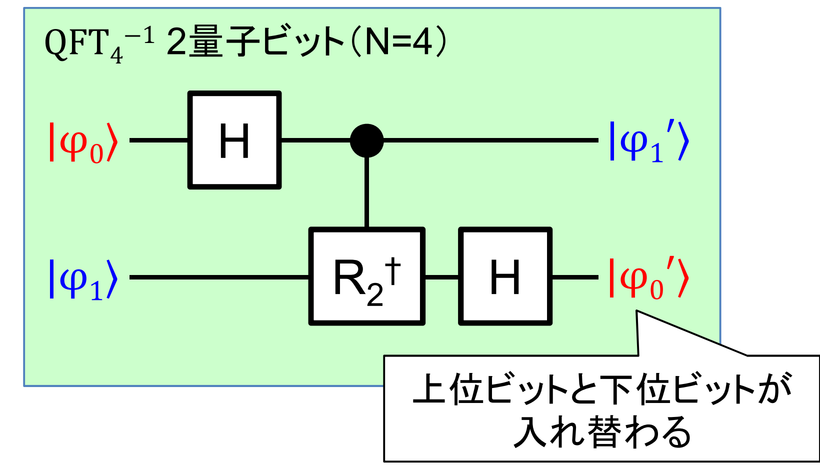
array([1000., 0., 0., 0.])

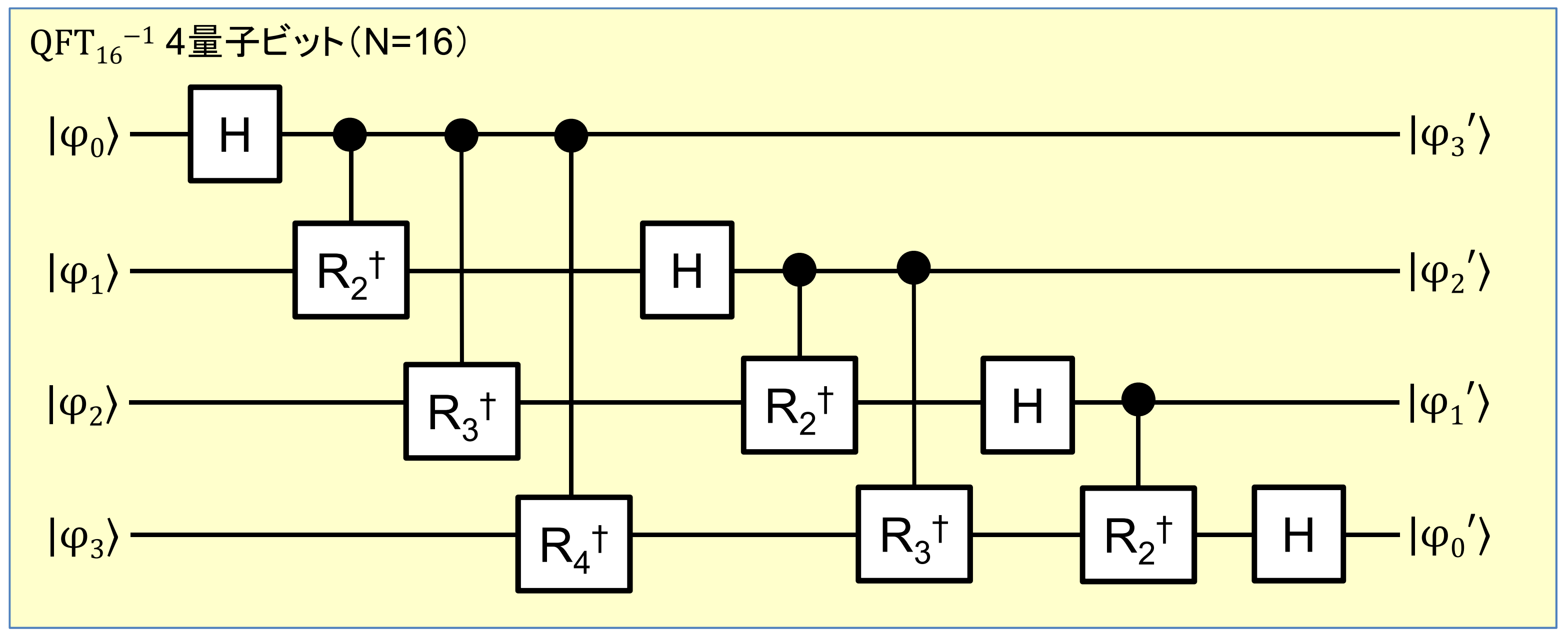
位相を振幅（値）に変換

- 量子干渉効果により重ね合わせ状態が打ち消されている。よく利用される変換。

n量子ビットの逆量子フーリエ変換 QFT_N^{-1}

逆量子フーリエ変換は位相の向きを逆にする。
R_n を $R_n^\dagger$ に変更する。
※アダマールは反転だった。





量子フーリエ変換･逆量子フーリエ変換

```
from cirq import *
# 量子フーリエ変換 定義 (inv=-1なら逆変換)
def qft(Q,n,inv):
    for i in range(n):
        for j in range(i):
            yield CZ(Q[i],Q[j])**(inv*1/2**(i-j))
        yield H(Q[i])
# 前準備
n = 3     # 3量子ビット(qbit)
Q = [LineQubit(i) for i in range(n)]
qc = Circuit()
# 入力として最後の量子ビットを反転して |001> に
qc.append(X(Q[2]))
# 量子フーリエ変換 inv = 1
qc.append(qft(Q,n,1))
qc.append(SWAP(Q[0],Q[2]))  # Q[1]は入れ替え不要
# 逆量子フーリエ変換 inv = -1
qc.append(qft(Q,n,-1))
qc.append(SWAP(Q[0],Q[2]))  # Q[1]は入れ替え不要
# 観測 (結果取得)
qc.append(measure(Q[0], key='q0'))
qc.append(measure(Q[1], key='q1'))
qc.append(measure(Q[2], key='q2'))
# 実行と結果表示
r = Simulator().run(qc, repetitions=1000)
plot_state_histogram(r)
print(r)
```

実行結果

初期値の $|001\rangle$ に戻っている

2-6：ショアのアルゴリズム

素因数分解を行うショアのアルゴリズムにより
公開鍵暗号が破れると言う注目をされました。

実際には必要なスケールを持つ量子ハードを
実現できないのですが正しく恐れる為に、
アルゴリズムを学びましょう。

素因数分解問題

ある正の整数を、素数の積の形で表わすこと。
ここでは正の整数の入力から、2つの素因数の積に分解する(2つの異なる素因数を得る)問題と考える。

※ 素因数:整数の因数である約数のうち素数であるもの。

例1:15 = 3 × 5　　(入力:15、出力:3と5)
例2:221 = 13 × 17　　(入力:221、出力:13と17)
例3:21631 = 97 × 223 (入力:21631、出力:97と223)

古典計算にて安直に解くには順番に素数を掛けて試して行く。
例2なら、221÷2、221÷3、221÷5、...、221÷13=17(正解!) とすることで解けるが元の整数が大きくなると指数的に困難になる。

※ RSA暗号は素因数分解が困難であることを前提とした暗号方式である。

ショア(Shor)のアルゴリズム

1994年にIBMのピーター・ショアが発表した、**素因数分解アルゴリズム。**(25年前なので結構古い)

全てを量子計算する訳では無く、位数発見部を量子計算する(他は古典計算する)ことで、多項式時間(n個の計算を n^c 回で計算、c は定数)で計算可能とする。

※ 古典解法では指数時間(n個の計算を c^n 回で計算)が必要だった。

量子と古典の両方の計算を使う、**ハイブリッド計算**が必要になるのでPythonによる計算は適している。

※ 素因数分解は可能だがビット数が増えると必要な量子ビット数が増える。この為、**実用にはハードウェアの進歩が必要。**

ショアのアルゴリズムの手順

ショアの計算では以下の3ステップが必要。

Step1：前処理（古典計算）

- 入力された整数 N から任意の数 a を選択。
- N と a は互いに素である必要がある。

Step2：周期発見（量子計算）

- 数 a を使って量子的に位数発見問題を解く。
- 位相の数から位数 r（周期T）を得る。

Step3：後処理（古典計算）

- 位数 r をチェックして正しくなければStep1へ、位数 r を使い2つの素因数を計算して終了。

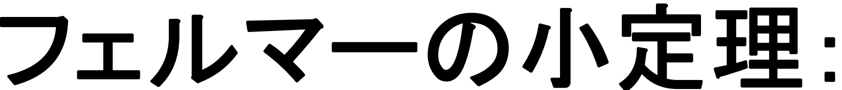
参考: フェルマー(Fermat)テスト

古典計算

互いに素:

2つの整数の最大公約数が1である。

2つの整数 a, b の**最大公約数 gcd(a, b) が 1** となる。



p が素数の時に互いに素な整数 a に以下が成り立つ。

$$a^p \equiv a \pmod p \Rightarrow a^{p-1} \equiv 1 \pmod p$$

フェルマーテスト(素数判定):

a と p が互いに素の時。

フェルマーの小定理の対偶を使うと以下が言える。

$a^{p-1} \not\equiv 1 \pmod p$ なら **p は素数ではない。**

Step1：前処理（古典計算）

入力値NからNより小さく互いに素な因数aを選択する。

Nと互いに素な素因数aの見つけ方：

1. Nより小さい整数aを選ぶ。
2. Nとaの最大公約数 gcd(N,a) が1ならaは素因数（※）。
3. 最大公約数が1以外なら別の整数aを選びやり直す。

※「ユークリッドの互除法」により最大公約数が1なら「互いに素」となる。

例：入力値N=15が与えられた場合
解：互いに素な値 a は、2, 4, 7, 8, 11, 13, 14 となる。

```
import math              # gcdを使う為にmathを利用
N = 15                   # 入力値N
for i in range(2,N):     # 1は除くので2からNまで
    r = math.gcd(N,i)    # 最大公約数の計算
    if r == 1:           # rが1なら互いに素
        print(i)         # 互いに素な値a表示
```

古典計算

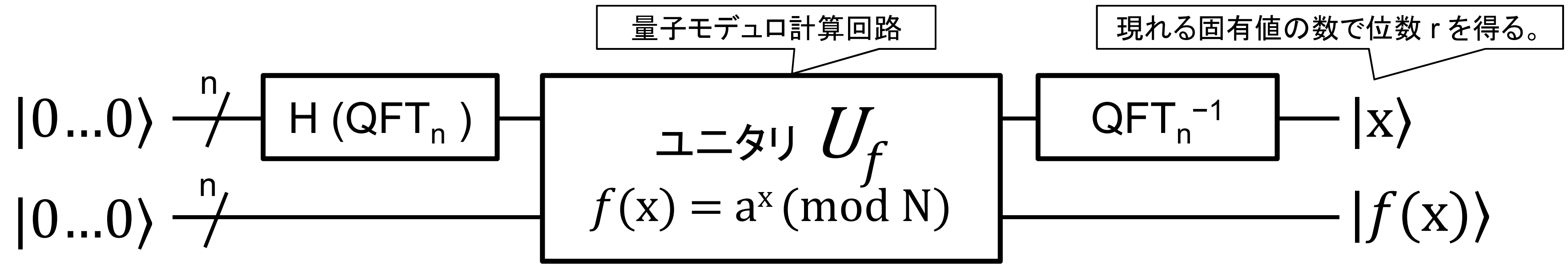
実行

```
2
4
7
8
11
13
14
```

Step2：周期(位数)発見 (量子計算)

入力値NとStep1で計算した値aから位数を計算。

適切なユニタリ回路を組み、位数発見問題を解く。



※ 量子フーリエ変換は全ビットをアダマール変換すれば良く、逆量子フーリエ変換は既出である。

例：入力値N=15の場合に、a=4と7を試してみる。

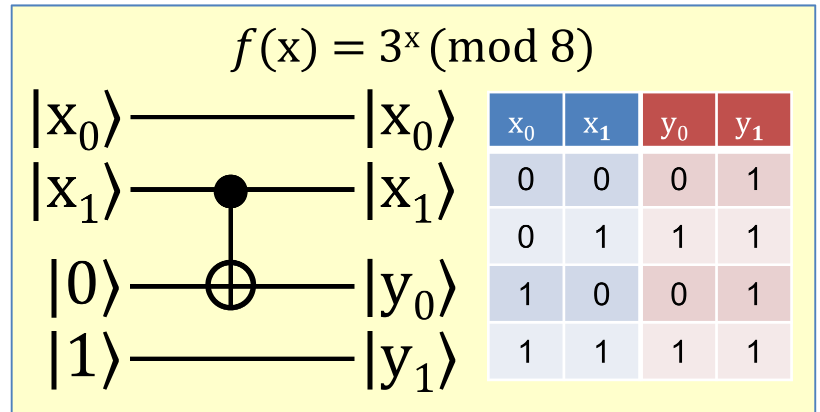
解：次ページ以降に記載の量子回路を実行する。

N=15, a=4 の場合の位数 $r = 2$ を得る

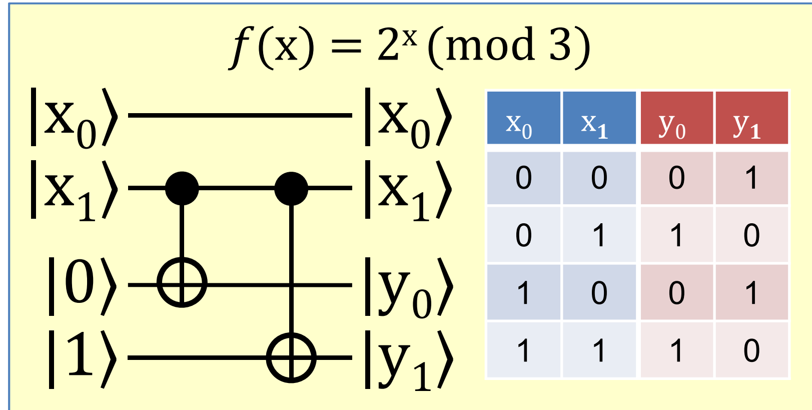
N=15, a=7 の場合の位数 $r = 4$ を得る

簡易: 位数発見問題のユニタリ回路1

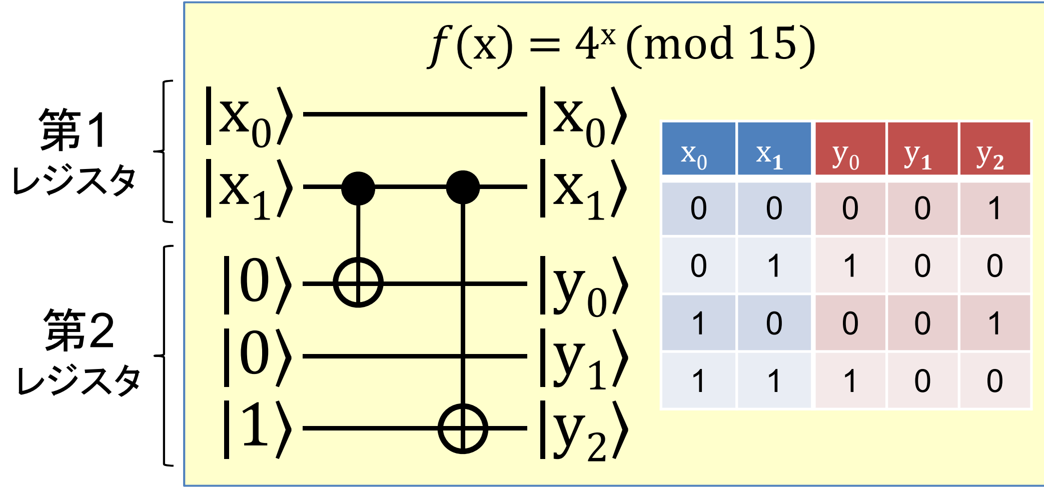
関数 $y = f(x) = a^x \bmod N$ (x=0,1,2,...) のユニタリ回路が必要。
まともにやるのは大変なので真理値表から作って試験する。



x_0	x_1	y_0	y_1
0	0	0	1
0	1	1	1
1	0	0	1
1	1	1	1



x_0	x_1	y_0	y_1
0	0	0	1
0	1	1	0
1	0	0	1
1	1	1	0

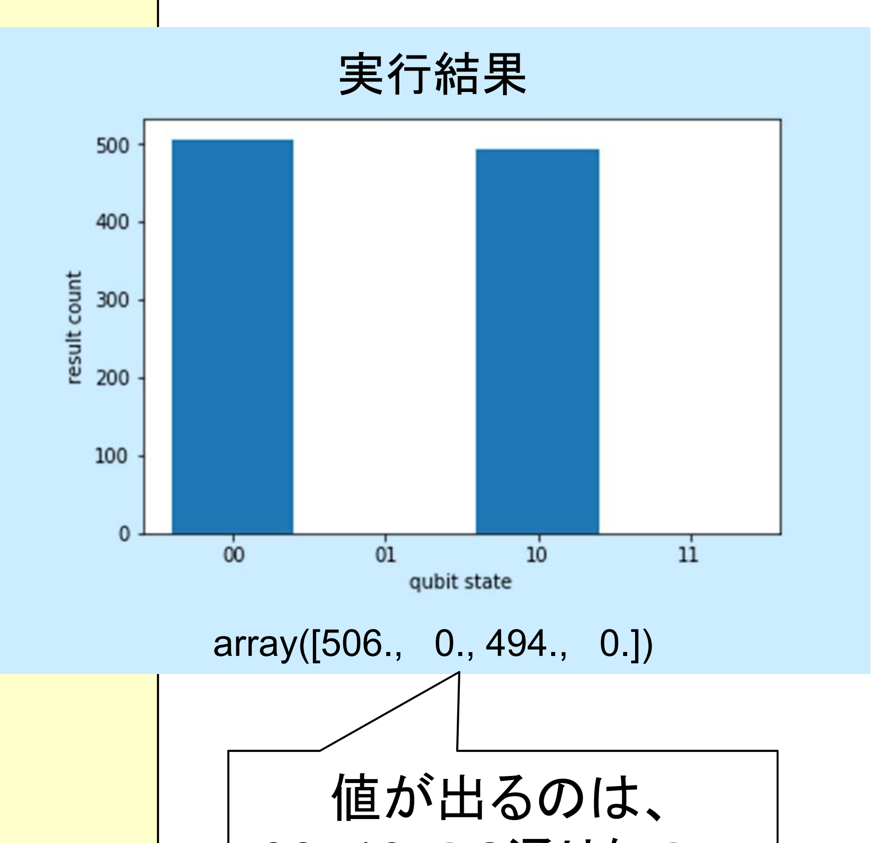


x_0	x_1	y_0	y_1	y_2
0	0	0	0	1
0	1	1	0	0
1	0	0	0	1
1	1	1	0	0

簡易方法では位数rを発見する回路の為に全ケースを先に計算（位数rは明確）をしているので実用的では無いが最小量子ビット数で回路を実現できるので学習用として利用されている。

周期発見 N=15, a=4 の実行

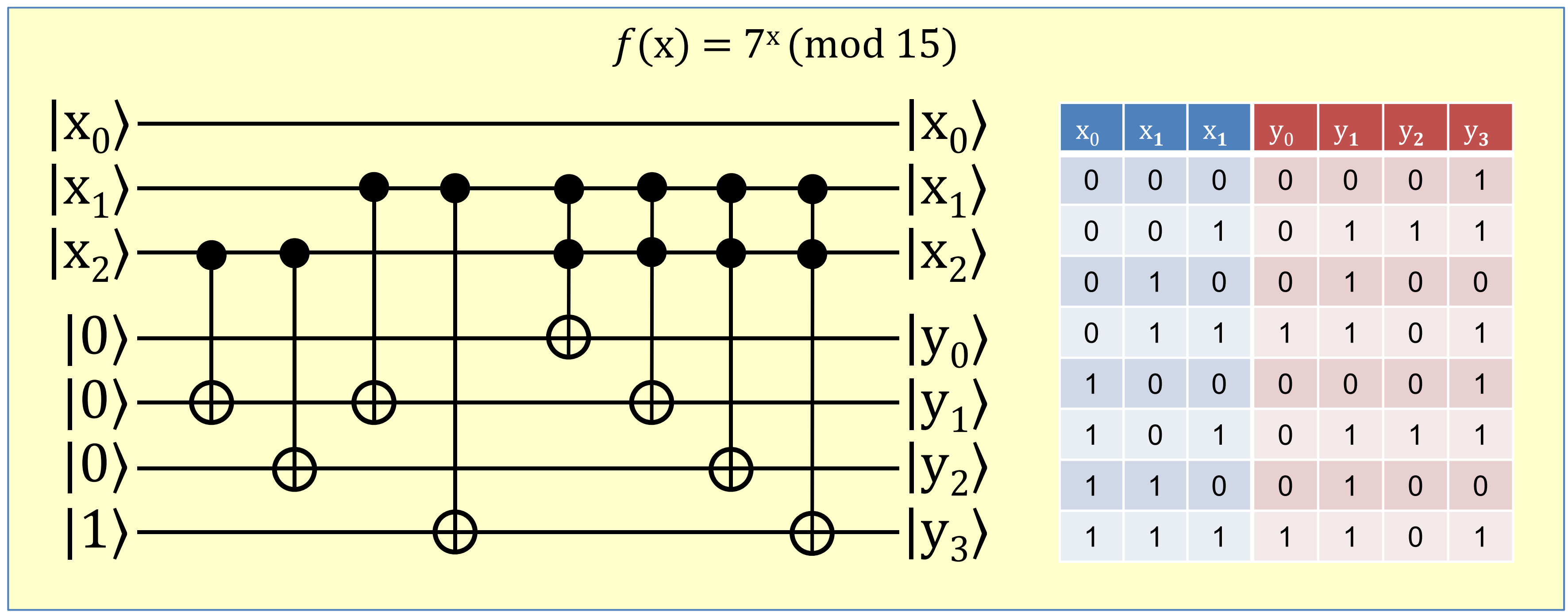
```
from cirq import *
# 量子フーリエ変換 定義 (inv=-1なら逆変換)
def qft(Q,n,inv):
    for i in range(n):
        for j in range(i):
            yield CZ(Q[i],Q[j])**(inv*1/2**(i-j))
        yield H(Q[i])
# 前準備
n = 3         # yは3qbit (入力N=15なので充分)
Qx = [GridQubit(0,i) for i in range(n-1)] # xは2qbit
Qy = [GridQubit(1,i) for i in range(n)]
qc = Circuit()
# 計算用Qxを重ね合わせ状態にする
qc.append(H.on_each(*Qx))
# オラクル計算
qc.append(X(Qy[n-1]))
qc.append(CNOT(Qx[1], Qy[0]))
qc.append(CNOT(Qx[1], Qy[2]))
# Qxを逆量子フーリエ変換 inv = -1
qc.append(qft(Qx,n-1,-1))
qc.append(SWAP(Qx[0],Qx[1]))
# Qxの観測 (結果取得)
qc.append(measure(Qx[0], key='q0'))
qc.append(measure(Qx[1], key='q1'))
# 実行と結果表示
r = Simulator().run(qc, repetitions=1000)
plot_state_histogram(r)
```



値が出るのは、
00, 10 の2通りなので
r=2

簡易: 位数発見問題のユニタリ回路2

$f(\mathrm{x}) = 7^{\mathrm{x}} \pmod{15}$ だとこんなに複雑に...

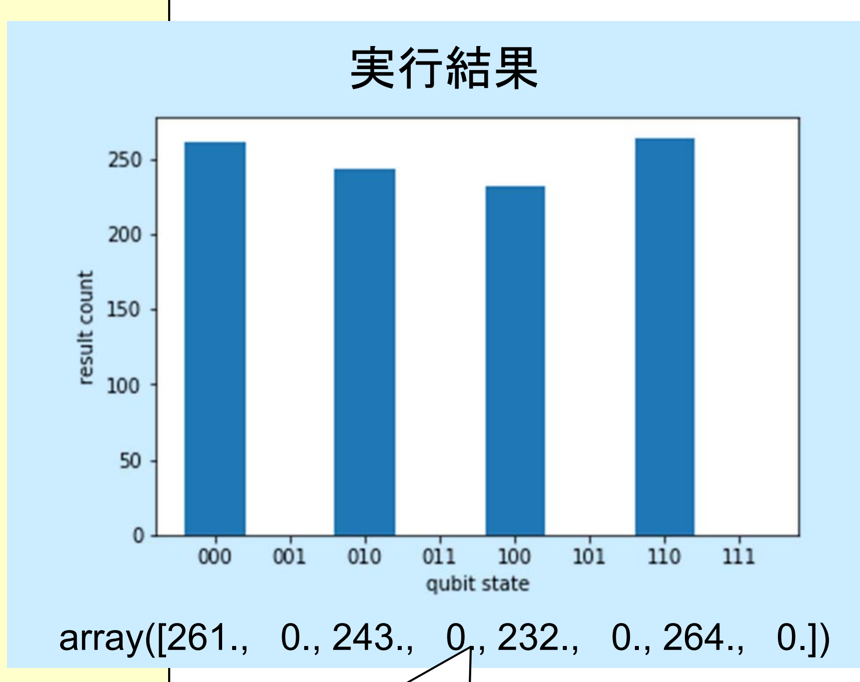


x_0	x_1	x_1	y_0	y_1	y_2	y_3
0	0	0	0	0	0	1
0	0	1	0	1	1	1
0	1	0	0	1	0	0
0	1	1	1	1	0	1
1	0	0	0	0	0	1
1	0	1	0	1	1	1
1	1	0	0	1	0	0
1	1	1	1	1	0	1

この程度でも汎用的な位数発見回路が必要と思える。
しかしまともにやると必要となる量子ビット数が増える…

周期発見 N=15, a=7 の実行

```
from cirq import *
# 量子フーリエ変換 定義 (inv=-1なら逆変換)
def qft(Q,n,inv):
    for i in range(n):
        for j in range(i):
            yield CZ(Q[i],Q[j])**(inv*1/2**(i-j))
        yield H(Q[i])
# 前準備
n = 4        # yは4qbit (入力N=15なので充分)
Qx = [GridQubit(0,i) for i in range(n-1)] # xは3qbit
Qy = [GridQubit(1,i) for i in range(n)]
qc = Circuit()
# 計算用Qxを重ね合わせ状態にする
qc.append(H.on_each(*Qx))
# オラクル計算
qc.append(X(Qy[n-1]))
qc.append(CNOT(Qx[2], Qy[1]))
qc.append(CNOT(Qx[2], Qy[2]))
qc.append(CNOT(Qx[1], Qy[1]))
qc.append(CNOT(Qx[1], Qy[3]))
qc.append(CCX(Qx[1], Qx[2], Qy[0]))
qc.append(CCX(Qx[1], Qx[2], Qy[1]))
qc.append(CCX(Qx[1], Qx[2], Qy[2]))
qc.append(CCX(Qx[1], Qx[2], Qy[3]))
# Qxを逆量子フーリエ変換 inv = -1
qc.append(qft(Qx,n-1,-1))
qc.append(SWAP(Qx[0],Qx[2]))
# Qxの観測 (結果取得)
qc.append(measure(Qx[0], key='q0'))
qc.append(measure(Qx[1], key='q1'))
qc.append(measure(Qx[2], key='q2'))
# 実行と結果表示
r = Simulator().run(qc, repetitions=1000)
plot_state_histogram(r)
```



値が出るのは、
000, 010, 100, 110
の4通りなので r=4

Step3：後処理（古典計算）

Step2で計算された位数 r を使って以下をチェック。

1. 位数rが奇数ならば、Step1から別のaでやり直し。
2. 位数rが偶数かつ、$a^{r/2}\pm 1$とNが互いに素であれば、Step1から別のaでやり直し。
3. 周期Tが偶数で、$a^{r/2}\pm 1$とNが互いに素で無ければ、以下の計算により2つの素因数が得られる。

素因数 $P1 = \gcd(a^{r/2}+1, N)$

素因数 $P2 = \gcd(a^{r/2}-1, N)$

例：$N=15, a=7, r=4$（$r/2=2$）の時、

素因数 $P1 = \gcd(7^2+1, 15) = \gcd(50, 15) = 5$

素因数 $P2 = \gcd(7^2-1, 15) = \gcd(48, 15) = 3$

解：15 の素因数は 5 と 3 である（5×3=15 なので正しい）。

補足: 素因数を得る為の計算

$f(x) = f(x+r)$ から、
$a^x \bmod N = a^{x+r} \bmod N$ となる。
これより $a^r \bmod N = 1$ が導かれる。

$a^r \bmod N = 1$ で位数 r が偶数なら、
$(a^{r/2}-1)(a^{r/2}+1) \bmod N = 0$ となり、
最大公約数 $\gcd(a^{r/2}\pm 1, N)$ により、
因数が高い確率(50%以上)で計算できる。

※ N と $a^{r/2}\pm 1$ が互いに素の場合は計算に失敗する。

N=15, a=2,4,7,8,11,13,14 の周期表

関数 $f(x) = a^x \bmod 15$ の計算結果：

$a^{r/2} \pm 1$とNが互いに素ならNGとする。

a	^1	^2	^3	^4	^5	^6	^7	^8	^9	^10	^11	^12	r	chk
2	2	4	8	1	2	4	8	1	2	4	8	1	4	OK
4	4	1	4	1	4	1	4	1	4	1	4	1	2	OK
7	7	4	13	1	7	4	13	1	7	4	13	1	4	OK
8	8	4	2	1	8	4	2	1	8	4	2	1	4	OK
11	11	1	11	1	11	1	11	1	11	1	11	1	2	NG
13	13	4	7	1	13	4	7	1	13	4	7	1	4	OK
14	14	1	14	1	14	1	14	1	14	1	14	1	2	NG

a=2, r=4 : $P1 = \gcd(2^2+1, 15) = \gcd(5, 15) = 5$ / $P2 = \gcd(2^2-1, 15) = \gcd(3, 15) = 3$

a=4, r=2 : $P1 = \gcd(4^1+1, 15) = \gcd(5, 15) = 5$ / $P2 = \gcd(4^1-1, 15) = \gcd(3, 15) = 3$

a=7, r=4 : $P1 = \gcd(7^2+1, 15) = \gcd(50, 15) = 5$ / $P2 = \gcd(7^2-1, 15) = \gcd(48, 15) = 3$

a=8, r=4 : $P1 = \gcd(8^2+1, 15) = \gcd(65, 15) = 5$ / $P2 = \gcd(8^2-1, 15) = \gcd(63, 15) = 3$

a=11, r=2 : $P1 = \gcd(11^1+1, 15) = \gcd(12, 15) = 12$ / $P2 = \gcd(11^1-1, 15) = \gcd(10, 15) = 10$

a=13, r=4 : $P1 = \gcd(13^2+1, 15) = \gcd(170, 15) = 5$ / $P2 = \gcd(13^2-1, 15) = \gcd(168, 15) = 3$

a=14, r=2 : $P1 = \gcd(14^1+1, 15) = \gcd(15, 15) = 0$ / $P2 = \gcd(14^1-1, 15) = \gcd(13, 15) = 13$

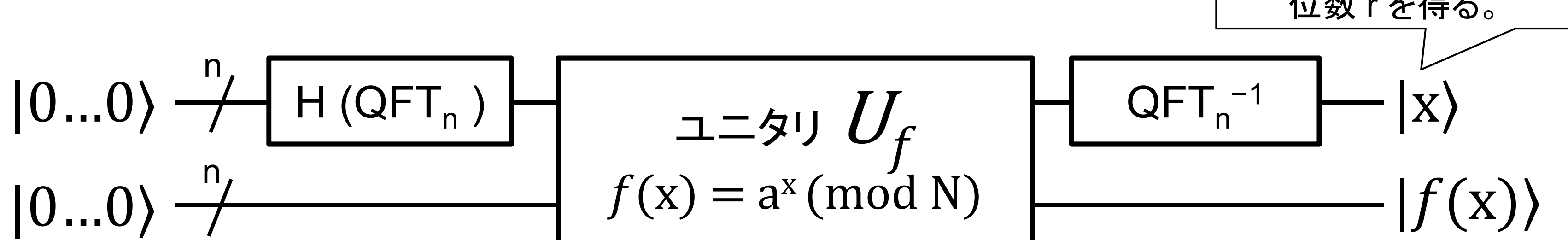
Re: 位数(周期)発見問題

関数 $f(\mathrm{x}) = \mathrm{a}^{\mathrm{x}} \pmod{\mathrm{N}}$ において、
$f(\mathrm{x}) = f(\mathrm{x+r})$ となる位数 r を発見する。

※ 位相が分かればその数から位数が高確率で取得できる。

⇒ (逆)量子フーリエ変換を使い、位数を発見する事が可能。

位数発見の量子回路:



※ 量子フーリエ変換は全ビットをアダマール変換すれば良く、逆量子フーリエ変換は既出である。

関数 $f(\mathrm{x}) = \mathrm{a}^{\mathrm{x}} \pmod{\mathrm{N}}$ の汎用ユニタリ回路が必要。

位数(周期)発見の汎用回路 : $a^x \pmod N$

入力:整数 N と任意の数 a を入力。← 2入力
出力:位数 r を得る。← 1出力
実現:位数発見の汎用的モジュロ回路が必要!

推薦資料「Shorのアルゴリズム」

加藤 拓己*, 湊 雄一郎*, 中田 真秀** (*MDR株式会社, **理化学研究所)
https://speakerdeck.com/gyudon/shorfalsearugorizumu

参考資料「量子計算機の到来を正しく恐れたい」

國廣 昇 (東京大学) : NICTサイバーセキュリティシンポジウム2019発表資料
※ 本資料の疑問への答えが、上記の「Shorのアルゴリズム」と言える。
https://www2.nict.go.jp/csri/plan/H31-symposium/pdf/kunihiro.pdf

以上の資料が参考になる。

$x^a \bmod N$ を計算する量子ゲート回路

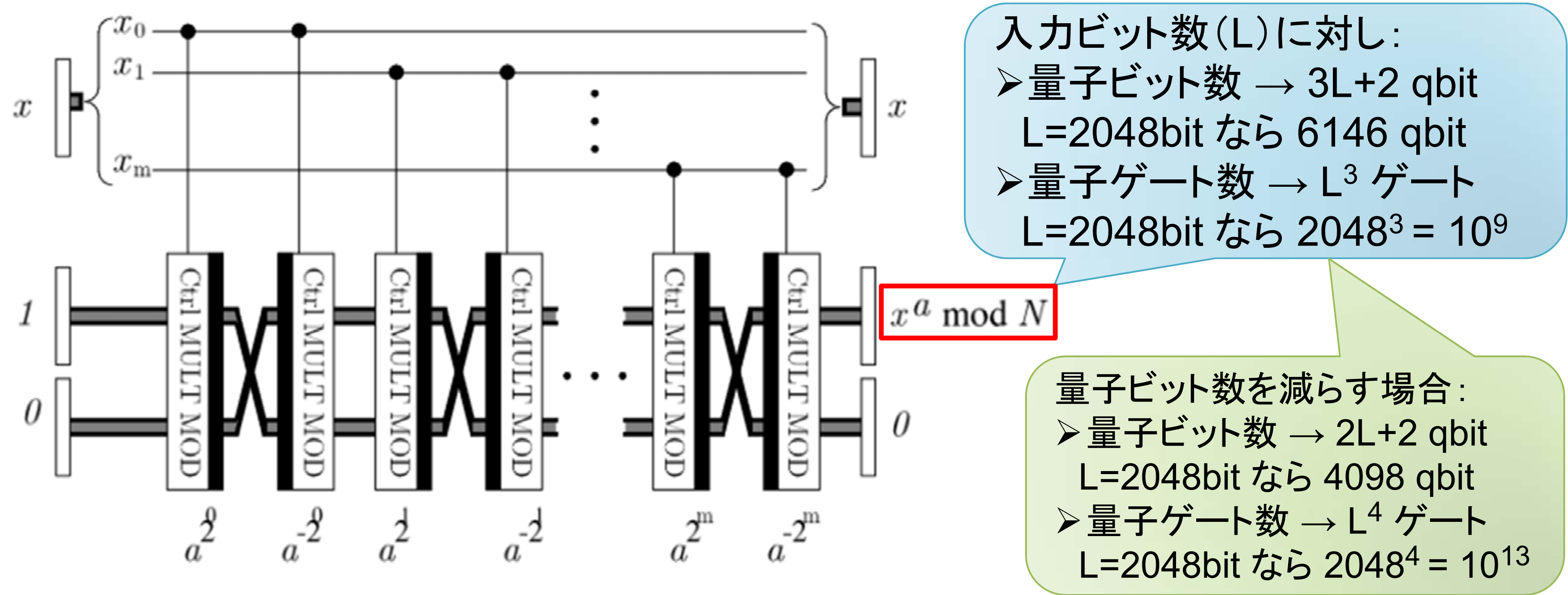


Figure 6)
V. Vedral, A. Barenco and A. Ekert

V. Vedral, A. Barenco, A. Ekert,
Quantum Networks for Elementary Arithmetic Operation,
arXiv:quant-ph/9511018 (1995)
https://arxiv.org/abs/quant-ph/9511018

「いちばんやさしい量子コンピュータで暗号を解くshorのアルゴリズム概要」
Blueqatを使ったモジュロ演算の実装例　@YuichiroMinato（MDR）
https://qiita.com/YuichiroMinato/items/5f98c467c006d7cb902c

現在のRSA暗号は解けるのか？

※ 現在2048～4096ビットのRSA暗号が使われている。

2048ビット（L=2048）を解こうとした場合：

- 量子ビット数：2048×3＋2 ＝ 6146（エラー無し）
- 量子ゲート数：2048^3 ＝ 86億（持続時間に影響）

が必要となる（量子ゲート数は古典だとステップ数に相当）。

エラー無しの6148量子ビットの実現：

- 量子ビット数もだがエラー無しハードルは高い。

86億ゲートを実現する重ね合わせ持続時間の実現：

- 量子重ね合わせ状態維持もハードルが高い。

正直言って実現することはかなり難しい？

2-7：エラー訂正問題

量子ゲート型の量子コンピュータの課題は、
量子ビット数を増やすこととエラー訂正である。

量子ビットエラー訂正の難しさ

古典計算での1ビットエラーは反転しかない。
0→1 になるか、1→0 になるかのどちらかのみ。
※ もちろん古典計算でも複数ビットへの影響の可能性はある。

- **量子計算1ビットエラーはX/Y/Zの3種類がある。**
  - ➢ ビット反転 X(q) エラー
  - ➢ 位相ビット反転 Y(q) エラー
  - ➢ 位相反転 Z(q) エラー
- **量子計算の途中でエラー訂正する必要がある。**
  - ➢ 重ね合わせ状態の量子ビットを測定することなく訂正する。
  - ➢ ショアのエラー訂正アルゴリズム以前は不可能と言われた。

物理量子ビットと論理量子ビット

必要なのは量子計算に利用するエラー訂正された
論理量子ビットである。論理量子ビットは複数の
物理量子ビットを組み合わせてエラー訂正して実現。

A）エラー率：1%

物理
量子
ビット

1論理量子ビットを
1物理量子ビットで構成。

論理
量子
ビット

B）エラー率：0.01%
（1%×1%）

物理
量子
ビット

物理
量子
ビット

物理
量子
ビット

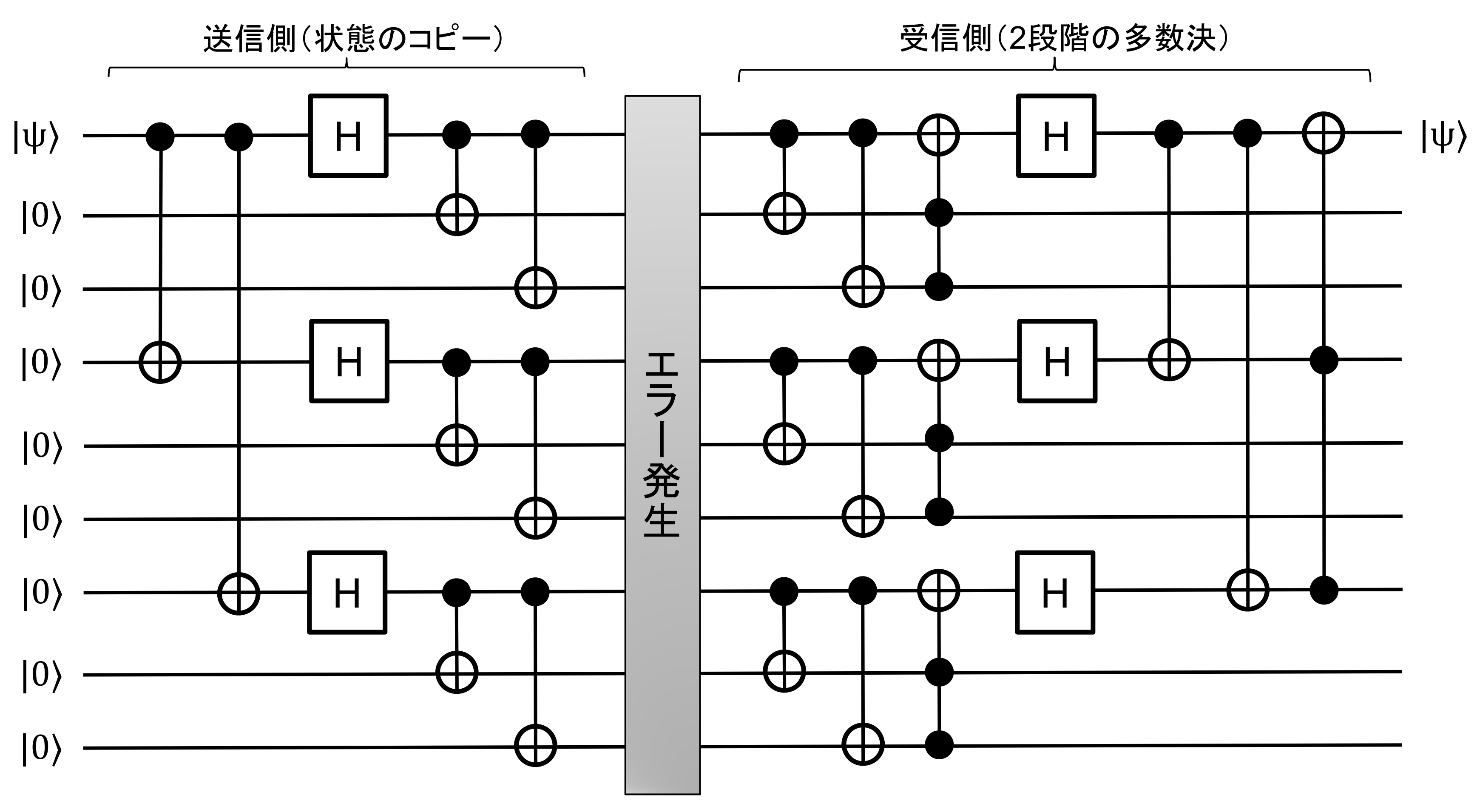
1論理量子ビットを
3物理量子ビットで構成
してもつれ合わせる。
2つ同時にエラーに
ならない限り補正可能。

C）エラー率：0.0001%
（0.01%×0.01%）

1論理量子ビットを
3物理量子ビットによる
B）を3セット利用。
2セット同時にエラーに
ならない限り補正可能。

量子ビット(ショア)のエラー訂正回路

ショアは量子パリティビットを導入して解決の道を示したが、必要となる量子ビット数は増大する(この場合9量子ビットが必要)。



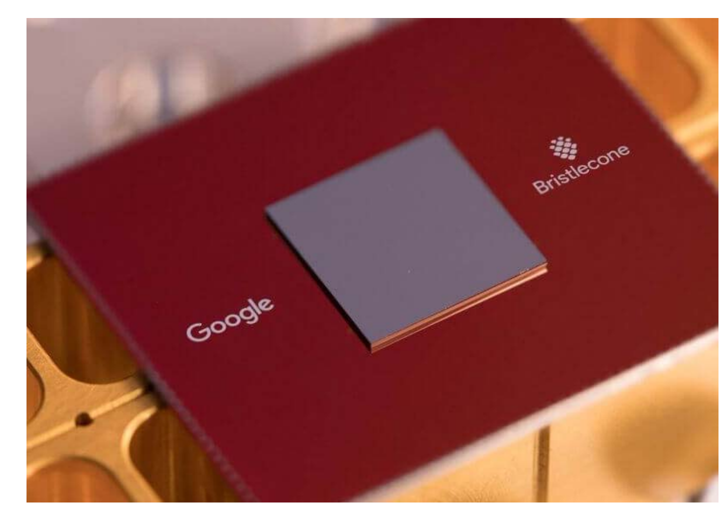
2-8： Cirq（Google）・Blueqat（MDR）

本資料ではGoogleのCirqを使って来たが少しCirqの補足説明をする。

また日本発の量子計算フレームワークとしてBlueqatが公開されており、簡単に紹介する。

Google 量子コンピュータへの取り組み

Googleは、2018年3月5日、**72量子ビット量子プロセッサBristlecone**(ブリッスルコーン)を発表。



2018年7月18日に、NISQ用フレームワーク Cirq を公開。

Bristleconeは極低温の超伝導状態で動作するタイプの量子計算用チップ。Googleは1チップにまとめることで低エラーの達成と実用を目指す。

Bristlecone以外にもイオントラップ方式のIonQにも投資中(Amazonも投資)で、色々手を打っている。

cirq.Simulatorのrunとsimulate

- cirq.Simulator.simulate(qc)
  - 重ね合わせ状態のまま、値を取得できる量子シミュレータ。
  - ノイズ無しの理論値（確率振幅）を取得できる。
  - 計算途中の重ね合わせ状態の確認時に利用する。

- cirq.Simulator.run(qc,...)
  - ノイズあり（NISQ用）の量子シミュレータ。
  - 観測しないと結果の取得はできない。

※ cirq.google.Bristlecone
- おそらく実機 Bristlecone 実行用。
- 現時点では未サポート。

Blueqatを使う（MDR）

環境：Anaconda3（Python3.5）

以下より環境に合わせてダウンロードとインストール

https://www.anaconda.com/distribution/

ライブラリ：Blueqat（ブルーキャット）

Windows版：Anaconda Prompt
MacOS版：ターミナル

インストール

```
pip install blueqat
```

バージョン指定インストール

```
pip install blueqat=0.3.9
```

アンインストール

```
pip uninstall blueqat
```

※ Blueqatのバージョン確認：

In:	import blueqat blueqat.__version__
Out:	'0.3.9‘

本資料のソースは 0.3.9 と表示される環境にて確認しています。

Blueqat 超入門（量子 Hello World!）

アダマールゲートHの実行

```
from blueqat import *
Circuit().h[0].m[:].run(shots=1000)
```

量子回路生成　アダマール　観測　実行（1000ショット）

実行結果：

```
Counter({'1': 488, '0': 512})
```

```
from blueqat import *
qc = Circuit(1)   # 1qbit指定
qc.h[0].m[:]      # 回路
r = qc.run(shots=1000) # 実行
print(r)          # 表示
```

複数ビット操作：

```
Circuit().z[1:3] # Z on 1,2
Circuit().x[:3]  # X on (0, 1, 2)
Circuit().h[:]   # H on all qubits
Circuit().x[1,2] # 1qubit gate
```

ローテーション操作：

```
Circuit().rz(math.pi/4)[0] # Z rotate pi/4
```

Blueqat 便利機能：ユニタリ行列表示

回路（ビット反転）指定してユニタリ行列を表示：

```
from blueqat import *
Circuit().x[0].run(backend="sympy_unitary")
```

実行結果（x[0]）：

```
Matrix([
[0, 1],
[1, 0]])
```

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

実行結果（アダマール h[0]）：

```
Matrix([
[1/sqrt(2),  1/sqrt(2)],
[1/sqrt(2), -sqrt(2)/2]])
```

$$= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

表現が違うが値は正しい

回路（複数ゲート）指定してユニタリ行列を表示：

```
from blueqat import *
Circuit().h[0].x[0].h[0].run(backend="sympy_unitary")
```

実行結果（HXHなのでZゲートと同じ）：

```
Matrix([
[1,  0],
[0, -1]])
```

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

複数ゲートでも
使えるので便利。

Blueqat で GPU を利用 (QGATE)

インストール: 要 Blueqat 0.3.9 以降

```
pip install numba
```

- numba(QGATE)を使った計算時間: run時のバックエンドで指定

```
from blueqat import Circuit
import numba
import time
start = time.time()
Circuit().h[20].x[:].y[:].z[:].h[:].h[:].h[:].run(backend='numba', shots=1000)
print(time.time() - start)
```

```
3.1740002632141113
```

※ Intel Core i7-6700 CPU @ 3.40GHz / NVIDIA GeForce GTX 950 - 4GB

- 比較の為に標準(GPU未使用)の計算時間:

```
from blueqat import Circuit
import time
start = time.time()
Circuit().h[20].x[:].y[:].z[:].h[:].h[:].h[:].run(shots=1000)
print(time.time() - start)
```

```
9.470999956130981
```

※ Intel Core i7-6700 CPU @ 3.40GHz / NVIDIA GeForce GTX 950 - 4GB

※ 非力なGPUでも3倍速い。高価なGPUボードなら桁違いに速いらしい。

Bluegatの情報

ソースリポジトリ:
https://github.com/Blueqat/

開発者が日本人なので日本語の情報が多い。

オンライン・ドキュメント（日本語）：
https://blueqat.readthedocs.io/ja/latest/

ブログ情報（日本語）:
https://qiita.com/tags/blueqat

MDR株式会社:
https://mdrft.com/?hl=ja

MDR開発中の実機が完成したらBlueqatから使える予定。

2-9：量子ゲート編 付録

付録1：Cirq/Qiskit/Blueqat対応表（基本編）

Gate	Cirq	Qiskit	Blueqat
恒等演算		iden	i
ビット反転演算	X	x	x
位相ビット反転演算	Y	y	y
位相反転演算	Z	z	z
アダマール演算	H	h	h
$\frac{\pi}{4}$位相シフト演算	S	s	s
$\frac{\pi}{8}$位相シフト演算	T	t	t
$-\frac{\pi}{4}$位相シフト演算	inverse(S)	sdg	sdg
$-\frac{\pi}{8}$位相シフト演算	inverse(T)	tdg	tdg
測定	measure	measure	m, measure
制御反転演算	CNOT	cx	cx, cnot
交換演算	SWAP	swap	swap
トフォリ演算	CCX, TOFFOLI	ccx	ccx, toffoli

付録2：Cirq/Qiskit/Blueqat対応表（拡張編）

Gate	Cirq	Qiskit	Blueqat
X軸任意回転演算	Rx	rx	rx
Y軸任意回転演算	Ry	ry	ry
Z軸任意回転演算	Rz	rz	rz
制御位相ビット反転演算		cy	
制御位相反転演算	CZ	cz	cz
トフォリ位相反転演算	CCZ		ccz
制御交換演算	CSWAP	cswap	
指定角λ演算		u1	u1
指定角φ,λ演算		u2	u2
指定角θ,φ,λ演算		u3	u3
制御指定角λ演算		cu1	cu1
制御指定角φ,λ演算		cu2	cu2
制御指定角θ,φ,λ演算		cu3	cu3
量子コンピュータ実機	**Google**（予定）	**IBM**	**MDR**（予定）

量子ゲート型の状況は...

量子アルゴリズムは現在盛んに論文が出ているので日進月歩の状況です。

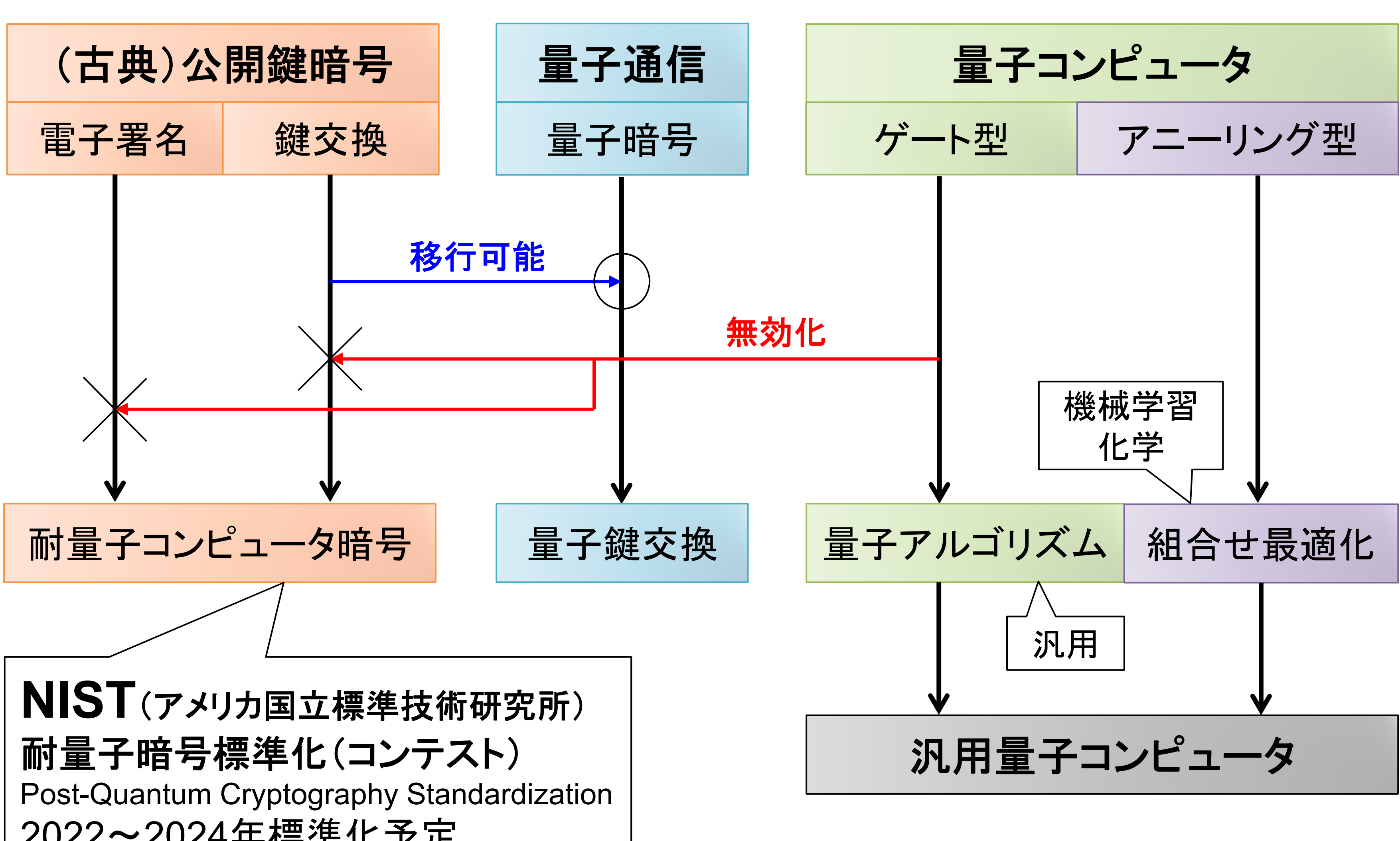
本資料もすぐに陳腐化する可能性があります。新しい手法が開発されていないか確認しましょう。

とは言え位相キックバックやオラクル等のブラックボックスを使った手法は基本となります。

実機によるプログラミングはまだ制限が多いのでしばらくは量子シミュレータを活用しましょう。

次回はアニーリング計算プログラミングに関して学びます。是非次回もご参加ください。

暗号と量子の関連



量子アルゴリズムの古典暗号への影響

種類	暗号方式	量子アルゴリズムの影響
公開鍵暗号	RSA暗号 楕円曲線暗号	ショアのアルゴリズム（サイモンのアルゴリズム） ➢ 状況：n^c から c^n へ回数を減らして解ける ➢ 対策：耐量子コンピュータ暗号へ移行 ※
共通鍵暗号	AES暗号	グローバーのアルゴリズム ➢状況：2^{100}から2^{50}程度へ回数が減らせる ➢対策：暗号鍵サイズを2倍にする サイモンのアルゴリズム ➢状況：CBC/CFB/CBC-MAC/GCMは危険 ➢対策：安全な他のモードを利用する
ハッシュ計算	SHA-2	グローバーのアルゴリズム ➢状況：2^{100}から2^{50}程度へ回数が減らせる ➢対策：ハッシュサイズを2倍にする

※ 新しい耐量子コンピュータ公開鍵暗号として、格子暗号方式・多変数多項式方式・ハッシュ関数方式・符号ベース方式等がNISTの耐量子暗号標準化（PQCS）で検討中。

Part 3：量子アニーリング型の プログラミング

次回！サル量子オフ#2　10月9日 18:10～
古典プログラマ向け量子プログラミング入門
[アニーリング編]
https://ossal.connpass.com/event/146249/

古典プログラマ向け量子プログラミング入門 [量子アニーリング編] 目次:

Part 3: 量子アニーリング型のプログラミング

3−1: ハミルトニアンとQUBO（Blueqat）

3-2: イジングモデル

3-3: グラフ理論

3-4: 巡回セールスマン問題

3-5: 多体相互作用

3-6: アニーリング計算まとめ

3-7: D-Wave（Ocean SDK）

3-8: 量子アニーリング編 付録

http://scienceinoh.jp/schrodinger/