日本語訳『Qiskit Textbook Machine Learning』勉強会
・データの符号化

Kifumi Numata (Qiskit Advocate)

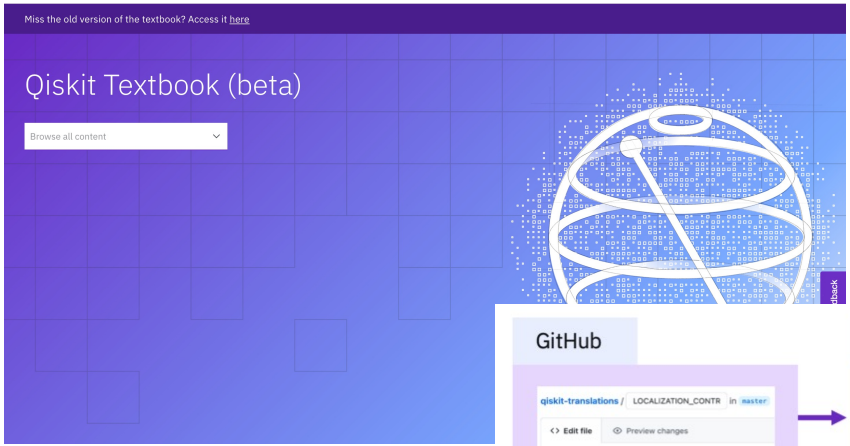
May 30, 2022

本文：https://learn.qiskit.org/course/machine-learning/data-encoding

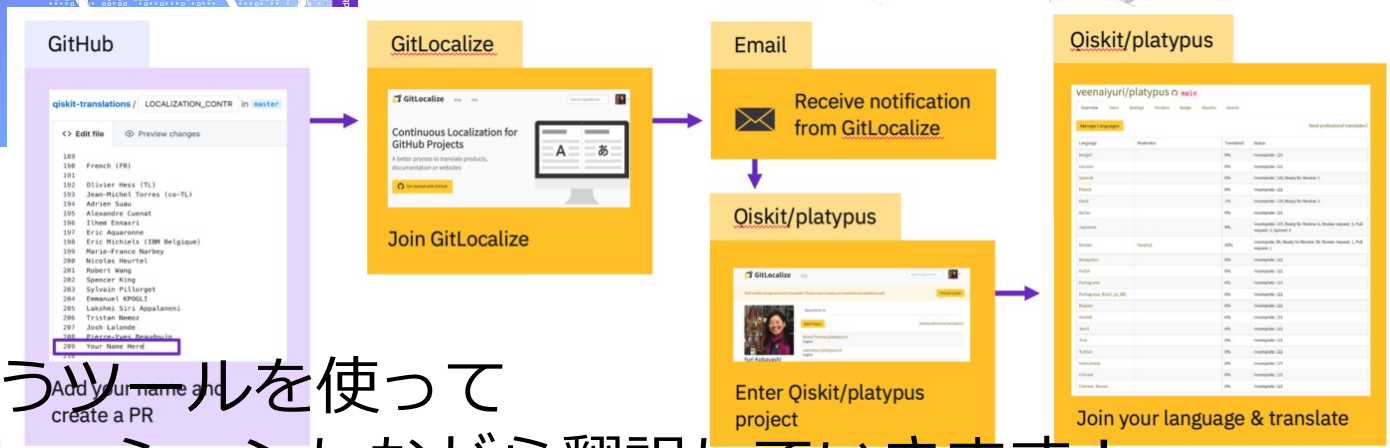
一時的な和訳： https://github.com/quantum-tokyo/introduction/blob/main/qiskit_textbook/New_textbook/quantum-machine-learning/encoding.ipynb

新版Qiskit Textbookの翻訳協力者募集中！

https://github.com/qiskit-community/qiskit-translations



Help make Qiskit accessible to non-English speaking communities



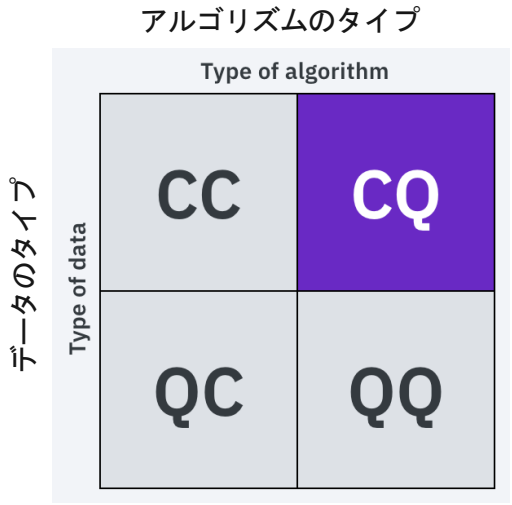
GitLocalizeというツールを使って
複数人でコラボレーションしながら翻訳していきます！

前回の復習

量子機械学習の4つのアプローチ

データの生成と処理の方式の組み合わせ

- CCアプローチ
  従来の機械学習
- QCアプローチ
  機械学習の量子計算への応用
- **CQアプローチ**　（古典データで量子アルゴリズム）
  従来の機械学習の問題に量子アルゴリズムを適用
- QQアプローチ
  量子データに量子アルゴリズムを適用



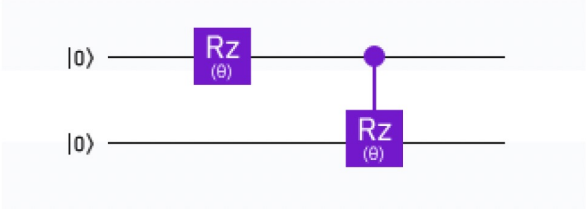
C: Classical (古典的)
Q: Quantum (量子的)

前回の復習

パラメータ化された量子回路とは

チューニング可能なパラメータを持つ量子回路のこと。

特にNISQ(Noisy Intermediate-Scale Quantum)量子アルゴリズムの根幹を担う概念です。



パラメータ化された量子回路の特徴づけ

- **Expressibility**(表現能力)
- **EntanglinigCapability**(エンタングリング能力)

分類タスクにおいて、Expressibilityと分類精度は強い相関があることが報告されています。

データの符号化

量子機械学習のためのデータ符号化を紹介します。

1. 計算基底符号化
2. 振幅符号化
3. 角度符号化
4. 任意の符号化

はじめに

機械学習モデルを成功させるには、**データの表現**が非常に重要。

- **古典的な機械学習**の場合の課題
  データを数値的に表現し、古典機械学習アルゴリズムで**最適に処理できるように**する。

- **量子機械学習**の場合の課題
  データを効率的に量子システムに**入力**し、量子機械学習アルゴリズムで**処理できるように**する。
  → つまり、
  データを量子コンピューターで処理できるような形にすることが課題

データの符号化、データエンコーディング、データの埋め込み、**ロード**などと呼ばれる。
このプロセスは、量子機械学習アルゴリズムの重要な部分であり、
その計算能力に直接影響します。

古典データセット

それぞれが N 個の**特徴(feature)**を持つM個のサンプルで構成される古典的なデータセット$\mathcal{X}$について考えます:

$$\mathcal{X} = \{x^{(1)}, \ldots, x^{(m)}, \ldots, x^{(M)}\}$$

ここで、$x^{(m)}$はm=1,...,MのN 次元ベクトルです。

このデータセットを量子ビットで表す手法はいろいろありますが、
今回は４つの手法を紹介します。

(*) 特徴(feature)：特徴とは、学習しようとするものの性質で、数値を割り当てることができるものです。例えば、猫について学習する場合、「身長」、「年齢」、「おやつを食べる傾向」などが特徴として挙げられます。

1. 計算基底符号化

古典的なNビット文字列をN量子ビットの**計算基底状態**に関連付けます。

(*) 計算基底状態：Z基底状態とも呼ばれ、Z（または計算）基底で測定したときの状態。|00⟩や|00110100⟩のようなラベルを持つ状態です。IBMのシステムは常にZ基底で測定します。

Nビット文字列
$x^{(m)} = (b_1, b_2, \cdots, b_N)$

計算基底
符号化

N量子ビット
$|x\rangle = |b_1, b_2, \cdots, b_N\rangle$

（ここで、n=1,⋯,Nに対して$b_n \in \{0,1\}$ ）

例）x=5の場合、0101なので、4量子ビットの|0101⟩で表す。

古典的なデータセット$\mathcal{X}$の場合は、

$$\mathcal{X} = \{x^{(1)}, \cdots, x^{(m)}, \cdots, x^{(M)}\}$$

各データポイント(Nビット)

$$x^{(m)} = (b_1, b_2, \cdots, b_N)$$

計算基底符号化

量子状態(N量子ビット)

$$|x\rangle = |b_1, b_2, \cdots, b_N\rangle$$

(n=1,…,Nにおいて、 $b_n \in \{0,1\}$、
また、m=1,…,Mです。)

よって
データセット$\mathcal{X}$全体を以下のような計算基底状態の重ね合わせとして
表すことができます。

$$|\mathcal{X}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^m\rangle$$

例）データセット

$$\mathcal{X} = \{x^{(1)} = 101, x^{(2)} = 111\}$$

計算基底符号化

量子状態

$$|\mathcal{X}\rangle = \frac{1}{\sqrt{2}}(|101\rangle + |111\rangle)$$

参照：Textbook Tool

計算基底符号化の実装

例）データセット$\mathcal{X} = \{x^{(1)} = 101, x^{(2)} = 111\}$ ➡ $|\mathcal{X}\rangle = \frac{1}{\sqrt{2}}(\overset{|5\rangle}{|101\rangle} + \overset{|7\rangle}{|111\rangle})$

Qiskitでは、どのような状態でデータセットを符号化するかを計算したら、**initialize関数**で準備します。

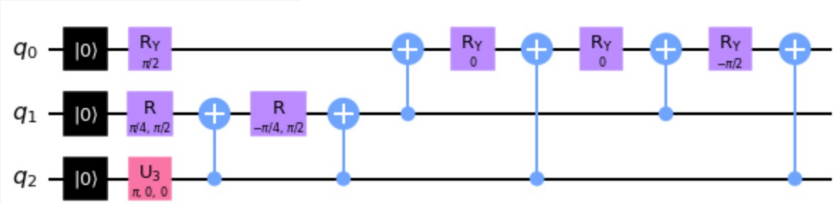
```
import math
from qiskit import QuantumCircuit

desired_state = [
    0,
    0,
    0,
    0,
    0,
    1 / math.sqrt(2),
    0,
    1 / math.sqrt(2)]

qc = QuantumCircuit(3)
qc.initialize(desired_state, [0,1,2])

qc.decompose().decompose().decompose().decompose().decompose().draw("mpl")
```

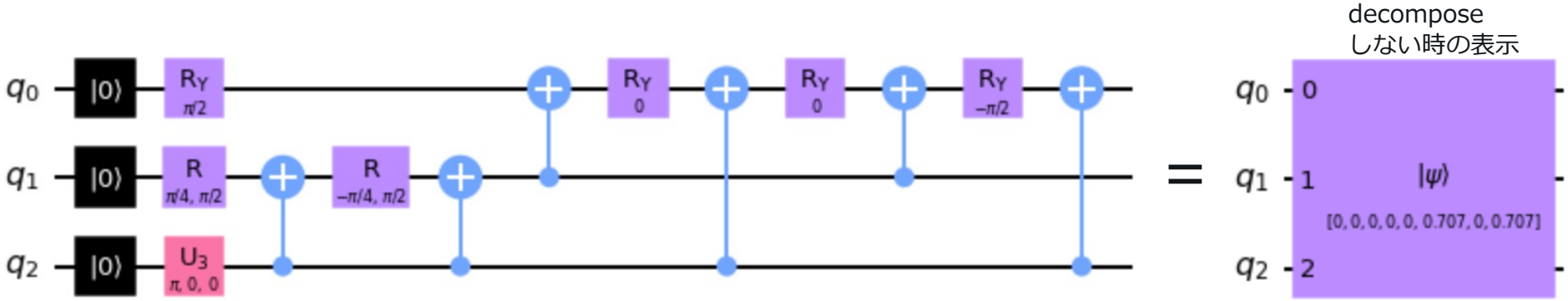
$$|5\rangle = \begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}, |7\rangle = \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{pmatrix}$$



もう一つdecompose().が必要。
decomposeがない場合の量子回路はかなりハイレベルな表現。

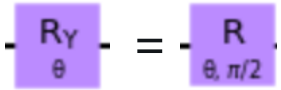
計算基底符号化の欠点：
理解するのは簡単だが、状態ベクトルが非常に疎になることが多く、効率的に実装できない。

Initialize関数とは



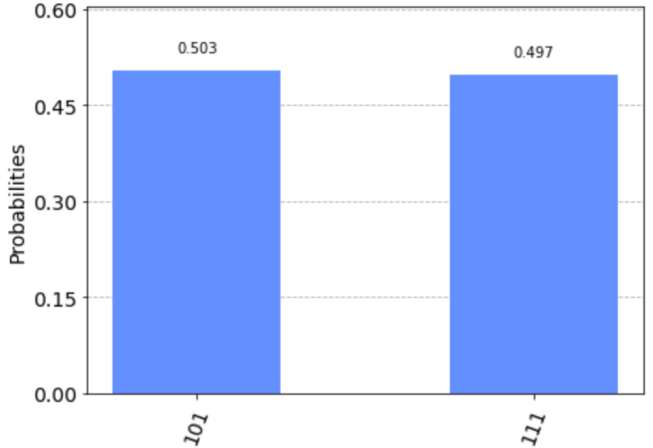

```
desired_state = [
    0,
    0,
    0,
    0,
    0,
    1 / math.sqrt(2),
    0,
    1 / math.sqrt(2)]

qc = QuantumCircuit(3)
qc.initialize(desired_state, [0,1,2])
```



{'101': 503, '111': 497}

シミュレーターでの実行結果



Initialize関数とは

任意の量子状態$|\psi\rangle$を作ってくれるのがInitialize関数。

量子状態$|\psi\rangle$は、ユニタリー行列Uで、 $|0\rangle$に戻すことができます。

$$U|\psi\rangle = |0\rangle$$

Uの逆行列U^{-1}が作ることができれば、$|\psi\rangle$を実装できます。

$$|\psi\rangle = U^{-1}|0\rangle$$

Initialize関数は、 Uを量子ゲートで実装し、それを逆行列にすることで任意の量子状態$|\psi\rangle$を実装します。

参考：@yoshi_mino さんのQiita記事「量子の重ね合わせを使って、任意の確率分布を表現する Initializer を実装する」

任意の1量子ビットの量子状態$|\psi\rangle$は、Z軸回転、 Y軸回転で、 $|0\rangle$に戻る

$$R_z(\phi) = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix} \equiv u1(\phi) \quad = e^{-i\phi/2} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$$

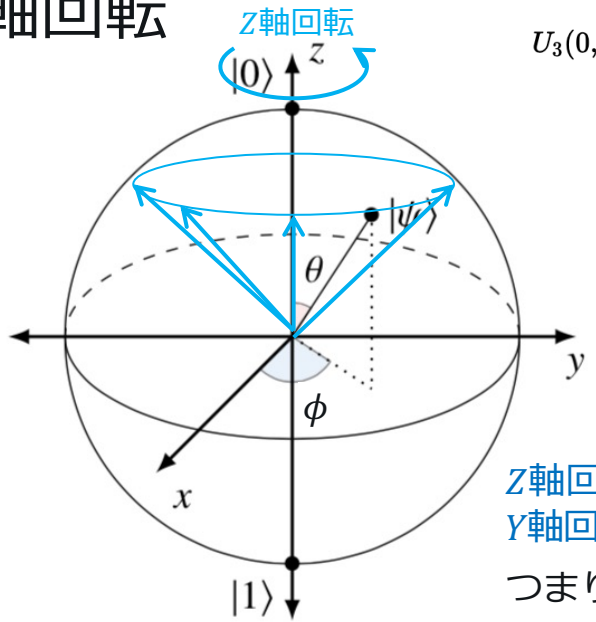
グローバル・フェーズ $e^{-i\phi/2}$だけ u1
と異なるので、同じとみなしています。

$$U_3(0,0,\lambda) = U_1 = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}$$

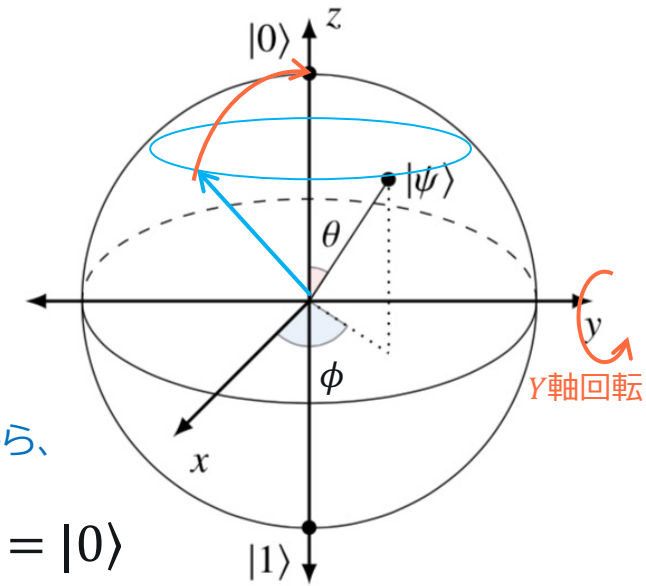
位相ゲート

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2). \end{pmatrix} = u3(\theta, 0, 0)$$

Z軸回転



Y軸回転



Z軸回転でXZ平面に合わせてから、
Y軸回転すると$|0\rangle$になる

つまり、$R_y(\theta)R_z(\phi)|\psi\rangle = |0\rangle$

$R_y(\theta)R_z(\phi)|\psi\rangle = |0\rangle$になる角度$\theta, \phi$の求め方

この θ, ϕ が求まれば、逆回転させて、$|\psi\rangle = U^{-1}|0\rangle$となる$U^{-1}$ が求められる。

ただし、 $|\psi\rangle$は1量子ビット状態。

$R_y(\theta)R_z(\phi)|\psi\rangle = |0\rangle$ より

$$R_z(\phi)|\psi\rangle = R_y(-\theta)|0\rangle$$

$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}$とすると

$$\underbrace{\begin{pmatrix}1 & 0\\ 0 & e^{i\phi}\end{pmatrix}}_{R_z(\phi)}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \underbrace{\begin{pmatrix}\cos(\theta/2) & \sin(\theta/2)\\ -\sin(\theta/2) & cos(\theta/2)\end{pmatrix}}_{R_y(-\theta)}\begin{pmatrix}1\\ 0\end{pmatrix}$$

より　$\alpha = \cos(\theta/2)$なので

$\theta = 2\arccos(\alpha)$

また　$R_z(\phi)\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \begin{pmatrix}\cos\frac{\theta}{2}\\ -\sin\frac{\theta}{2}\end{pmatrix}$　から

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = R_z(-\phi)\begin{pmatrix}\cos\frac{\theta}{2}\\ -\sin\frac{\theta}{2}\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & e^{-i\phi}\end{pmatrix}\begin{pmatrix}\cos\frac{\theta}{2}\\ -\sin\frac{\theta}{2}\end{pmatrix} = \begin{pmatrix}\cos\frac{\theta}{2}\\ -e^{-i\phi}\sin\frac{\theta}{2}\end{pmatrix}$$

より　ϕは、αとβの相対位相

3量子ビットで$U|\psi\rangle = |0\rangle$になる$U$を作る

目標： n量子ビットで U^{-1}を作って、状態$|\psi\rangle = U^{-1}|0\rangle$を作りたい。
まず3量子ビットで考えてみる。

$|\psi\rangle = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix}$ で、 $U|\psi\rangle = |0\rangle$ となるUを作りたい。

$|\psi\rangle$を４つの2次元縦ベクトルに分解。

$$|\psi_1\rangle = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, |\psi_2\rangle = \begin{pmatrix} a_3 \\ a_4 \end{pmatrix}, |\psi_3\rangle = \begin{pmatrix} a_5 \\ a_6 \end{pmatrix}, |\psi_4\rangle = \begin{pmatrix} a_7 \\ a_8 \end{pmatrix}$$

$R_1|\psi_1\rangle = r_1|0\rangle$ となるR_1 は、 $R_y(\theta)R_z(\phi)$ のθ, ϕを求めることで求められる。
同じようにして、 R_2, R_3, R_4 も求める。

$$\begin{bmatrix} R_1 & & & \\ & R_2 & & 0 \\ & & R_3 & \\ & 0 & & R_4 \end{bmatrix} |\psi\rangle = \begin{bmatrix} R_1|\psi_1\rangle \\ R_2|\psi_2\rangle \\ R_3|\psi_3\rangle \\ R_4|\psi_4\rangle \end{bmatrix} = \begin{bmatrix} r_1|0\rangle \\ r_2|0\rangle \\ r_3|0\rangle \\ r_4|0\rangle \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} \otimes |0\rangle$$

8x8の行列

8次元縦ベクトル(3量子ビット)の$|\psi\rangle$が
8x8の行列で、4次元ベクトル(2量子ビット)
と$|0\rangle$に分解された。

3量子ビットで$U|\psi\rangle = |0\rangle$になる$U$を作る (続き)

同じようにして、 $\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}$も$\begin{bmatrix} r'_1 \\ r'_2 \end{bmatrix} \otimes |0\rangle$に分解でき、 $\begin{bmatrix} r'_1 \\ r'_2 \end{bmatrix}$も$R_y(\theta)R_z(\phi)$ で$|0\rangle$にすることができるので、

$$R''_1 \begin{bmatrix} R'_1 & 0 \\ 0 & R'_2 \end{bmatrix} \begin{bmatrix} R_1 & & & 0 \\ & R_2 & & \\ & & R_3 & \\ 0 & & & R_4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix} = R''_1 \begin{bmatrix} R'_1 & 0 \\ 0 & R'_2 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} \otimes |0\rangle = R''_1 \begin{bmatrix} r'_1 \\ r'_2 \end{bmatrix} \otimes |0\rangle \otimes |0\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle$$

2x2の行列　4x4の行列　8x8の行列　$|\psi\rangle$

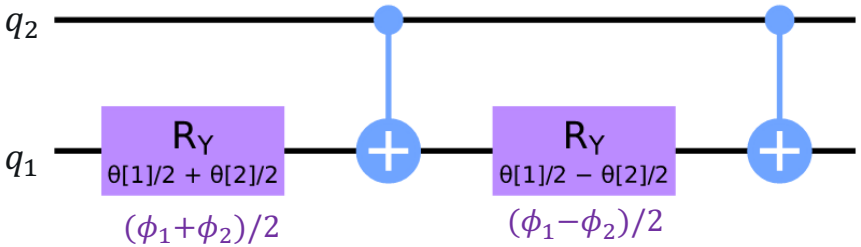
このようにして、 $U|\psi\rangle = |0\rangle$　となるUを作ることができる。
（n量子ビットでも同様。）

2x2の行列 R''_1は、 $R''_1|\psi_1\rangle = r_1|0\rangle$ となる$R_y(\theta)R_z(\phi)$ のθ, ϕを求めることで求められるが、
4x4の行列、8x8の行列 の作り方は？？

3量子ビットで$U|\psi\rangle = |0\rangle$になる$U$を作る（続き）

$\begin{bmatrix} R(\phi_1) & 0 \\ 0 & R(\phi_2) \end{bmatrix}$は以下のように実装できます。

4x4の行列



- q_2 =0の場合： $R\left(\frac{\phi_1}{2}\right) + R\left(\frac{\phi_1}{2}\right) = R(\phi_1)$ がかかる。
- q_2 =1の場合：2個目のRの前後のXゲートで回転の向きが反転するので $R\left(\frac{\phi_2}{2}\right) + R\left(\frac{\phi_2}{2}\right) = R(\phi_2)$になる。

参考）Qiskit テキストブック 2.4.3

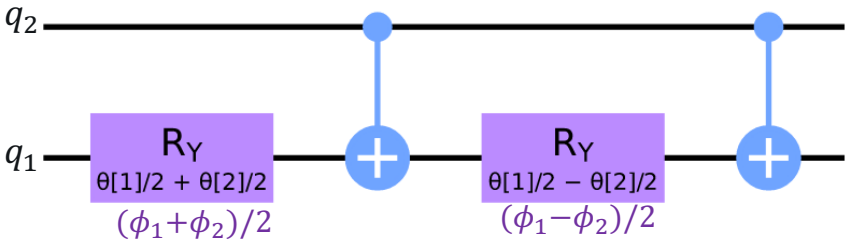
確かめ

$$\begin{bmatrix} R(\phi_1) & 0 \\ 0 & R(\phi_2) \end{bmatrix} = \begin{bmatrix} R(\phi_1)_{11} & R(\phi_1)_{12} & 0 & 0 \\ R(\phi_1)_{21} & R(\phi_1)_{22} & 0 & 0 \\ 0 & 0 & R(\phi_2)_{11} & R(\phi_2)_{12} \\ 0 & 0 & R(\phi_2)_{21} & R(\phi_2)_{22} \end{bmatrix}$$ と書くと

$$\begin{bmatrix} R(\phi_1)_{11} & R(\phi_1)_{12} & 0 & 0 \\ R(\phi_1)_{21} & R(\phi_1)_{22} & 0 & 0 \\ 0 & 0 & R(\phi_2)_{11} & R(\phi_2)_{12} \\ 0 & 0 & R(\phi_2)_{21} & R(\phi_2)_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = R(\phi_1)|00\rangle$、また $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle$ （ここで$|q_2 q_1\rangle$）

を使うと確かめられる。

(補足)



$= U'$ とおく

この回路は、

- q_2=0の場合： $R\left(\frac{\phi_1}{2}\right) + R\left(\frac{\phi_1}{2}\right) = R(\phi_1)$がかかる。
- q_2 =1の場合： $R\left(\frac{\phi_2}{2}\right) + R\left(\frac{\phi_2}{2}\right) = R(\phi_2)$がかかる。

というもの($*$)

$$U' = \begin{bmatrix} R(\phi_1)_{11} & R(\phi_1)_{12} & & 0 \\ R(\phi_1)_{21} & R(\phi_1)_{22} & & \\ & 0 & R(\phi_2)_{11} & R(\phi_2)_{12} \\ & & R(\phi_2)_{21} & R(\phi_2)_{22} \end{bmatrix}$$

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = |00\rangle, \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = |01\rangle, \begin{bmatrix}0\\0\\1\\0\end{bmatrix} = |10\rangle, \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = |11\rangle$$

U'が($*$)であることの確かめ

$|q_2\rangle \otimes |q_1\rangle$

$|0\rangle \otimes |0\rangle \rightarrow U' \rightarrow |0\rangle \otimes R(\phi_1)|0\rangle \rightarrow R(\phi_1)|00\rangle$

$|0\rangle \otimes |1\rangle \rightarrow U' \rightarrow |0\rangle \otimes R(\phi_1)|1\rangle \rightarrow R(\phi_1)|01\rangle$

$|1\rangle \otimes |0\rangle \rightarrow U' \rightarrow |1\rangle \otimes R(\phi_2)|0\rangle \rightarrow R(\phi_2)|10\rangle$

$|1\rangle \otimes |1\rangle \rightarrow U' \rightarrow |1\rangle \otimes R(\phi_2)|1\rangle \rightarrow R(\phi_2)|11\rangle$

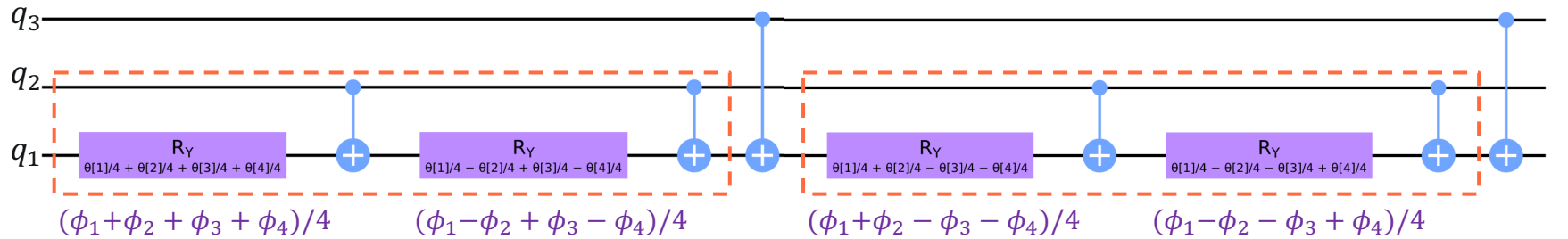
3量子ビットで$U|\psi\rangle = |0\rangle$になる$U$を作る (続き)

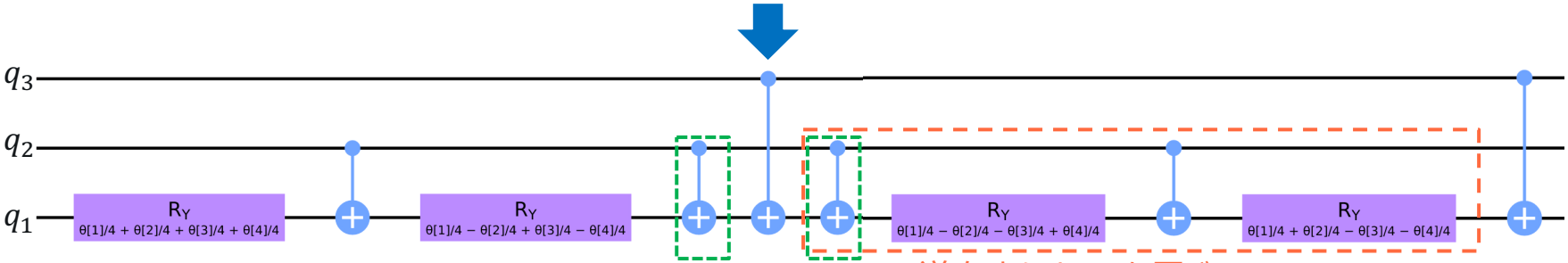
同様にして　　8x8の行列

$$\begin{bmatrix} R(\phi_1) & & & & 0 & & & \\ & R(\phi_2) & & & & & & \\ & & R(\phi_3) & & & & & \\ & & & R(\phi_4) & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{bmatrix}$$

は以下のように実装できます。

$$\begin{bmatrix} R_1 & & & 0 \\ & R_2 & & \\ & & R_3 & \\ 0 & & & R_4 \end{bmatrix} |\psi\rangle = \begin{bmatrix} R_1|\psi_1\rangle \\ R_2|\psi_2\rangle \\ R_3|\psi_3\rangle \\ R_4|\psi_4\rangle \end{bmatrix}$$



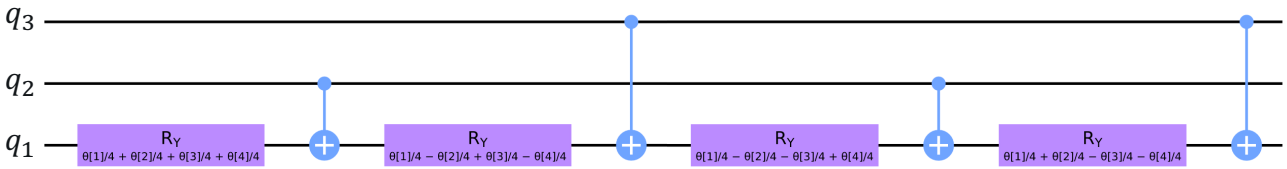


2 つのCNOTがキャンセルできる

逆向きにしても同じ

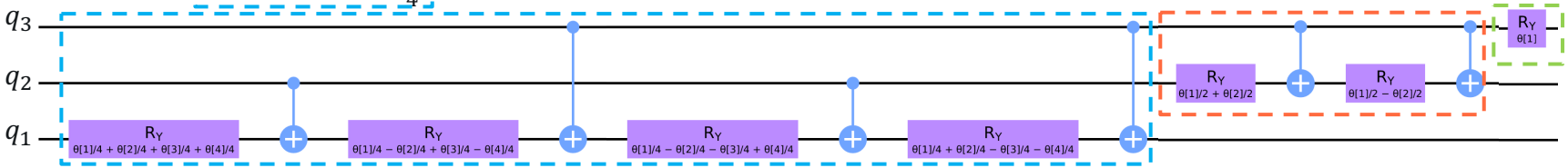
References: [1] Shende, Bullock, Markov. Synthesis of Quantum Logic Circuits (2004) [*https://arxiv.org/abs/quant-ph/0406176v5*]

２つのCNOTを消去



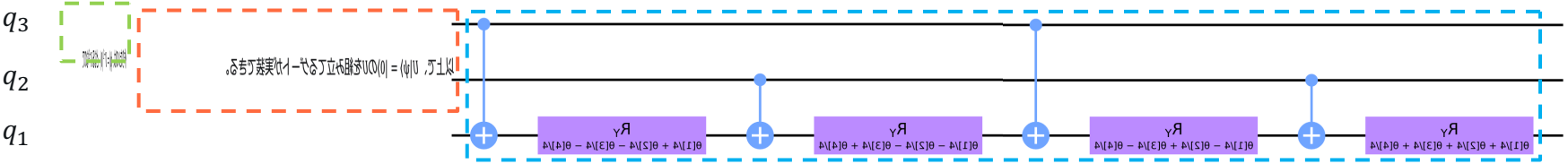
以上で、$U|\psi\rangle = |0\rangle$の$U$を組み立てるゲートが実装できる。

$R''_1 \begin{bmatrix} R'_1 & 0 \\ 0 & R'_2 \end{bmatrix} \begin{bmatrix} R_1 & & 0 & \\ & R_2 & & \\ & & R_3 & \\ 0 & & & R_4 \end{bmatrix} |\psi\rangle = |0\rangle$　を並べると(行列と回路は逆順なことに注意)



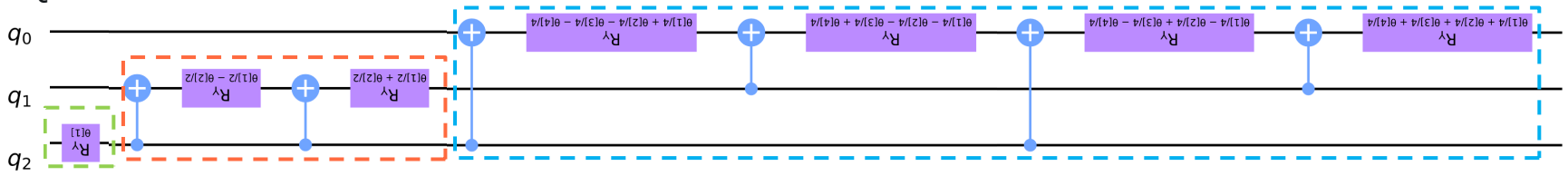
作りたいのは、$|\psi\rangle = U^{-1}|0\rangle$　となる U^{-1}なので

上記を逆行列（逆順に並べて、回転の向きを逆に）にしたものがInitialize関数で作ってくれるもの

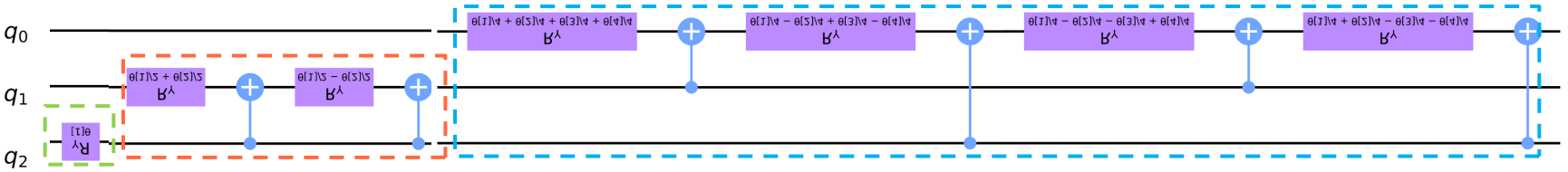


さらにQiskitのビット配列では上下逆転なので

Qiskitのビット配列で



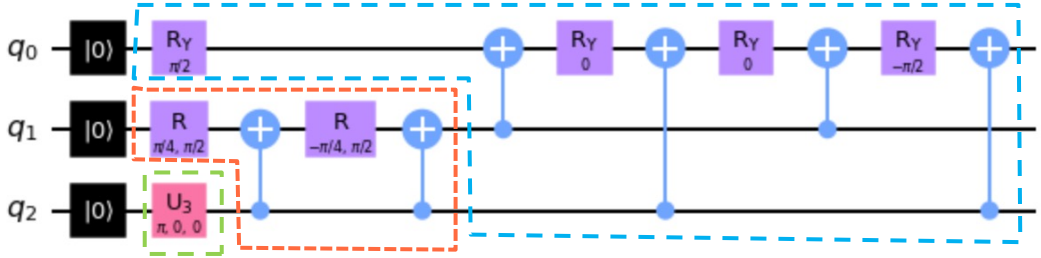
先程と同じく点線の中は逆向きにしても同じ法則を使うと



References: [1] Shende, Bullock, Markov. Synthesis of Quantum Logic Circuits (2004) [*https://arxiv.org/abs/quant-ph/0406176v5*]

相対位相が入る場合、Rzも同じように入れていく。

Initialize関数で作ったものと同じ形になりました。



計算基底符号化のまとめ

- 計算基底符号化は、NビットデータをN量子ビットの計算基底状態に変換する。
- Qiskitでは、Initialize関数で計算基底符号化を行うことができる。
- Initialize関数は、任意の量子状態が、Z、Y軸回転、CNOTで$|0\rangle$にできることより、その逆行列を作って、$|0\rangle$から任意の量子状態をセットする。

例）データセット$\mathcal{X} = \{x^{(1)} = 101, x^{(2)} = 111\}$ ➡ $|\mathcal{X}\rangle = \frac{1}{\sqrt{2}}(|101\rangle + |111\rangle)$

($|101\rangle$ = $|5\rangle$, $|111\rangle$ = $|7\rangle$)

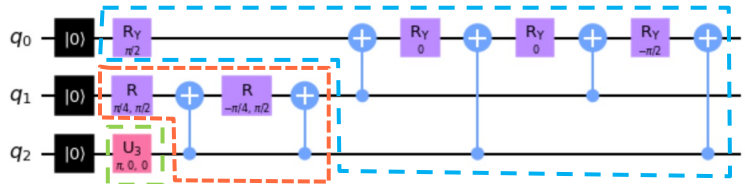
```
import math
from qiskit import QuantumCircuit

desired_state = [
    0,
    0,
    0,
    0,
    0,
    1 / math.sqrt(2),
    0,
    1 / math.sqrt(2)]

qc = QuantumCircuit(3)
qc.initialize(desired_state, [0,1,2])

qc.decompose().decompose().decompose().decompose().decompose().draw("mpl")
```

$$|5\rangle = \begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}, |7\rangle = \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{pmatrix}$$



計算基底符号化の欠点：
理解するのは簡単だが、状態ベクトルが非常に疎になることが多く、効率的に実装できない。

2. 振幅符号化

振幅符号化は、データを量子状態の振幅に符号化します。
つまり、正規化された古典N次元データポイント$\boldsymbol{x}$を、**n量子ビットの量子状態$|\psi_x\rangle$の振幅**として表します。

N次元データポイント x

振幅符号化

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle$$

ここで、$N = 2^N$、x_iはxのi番目の要素であり、
$|i\rangle$は i番目の計算基底状態です。

データセットの場合

$$\mathcal{X} = \{x^{(1)}, \cdots, x^{(m)}, \cdots, x^{(M)}\}$$

MN次元のすべてのデータポイントを
長さ $N\times M$ の1つの振幅ベクトルに連結。

$$\alpha = A_{\text{norm}}(x_1^{(1)}, \ldots, x_N^{(1)}, \ldots, x_1^{(m)}, \ldots, x_N^{(m)}, \ldots, x_1^{(M)}, \ldots, x_N^{(M)})$$

ここでA_{norm}は正規化定数であり、$|\alpha|^2 = 1$。

振幅符号化

$$|\mathcal{X}\rangle = \sum_{i=1}^{N} \alpha_i |i\rangle$$

ここで、α_iは振幅ベクトルの要素であり、
$|i\rangle$は計算基底状態です。
符号化される振幅の数は$N\times M$です。

n量子ビットのシステムには2^n個の振幅がのあるで、振幅の埋め込みには**$n \geq log_2(NM)$個の量子ビット**が必要。

振幅符号化の例

例）データセットデータセット $\mathcal{X} = \{x^{(1)} = (1.5, 0), x^{(2)} = (-2, 3)\}$を符号化すると、

$$\alpha = \frac{1}{\sqrt{15.25}}(1.5, 0, -2, 3)$$

（両方のデータポイントを連結し、結果のベクトルを正規化）

よって振幅符号化された2量子ビットの量子状態は

$$|\mathcal{X}\rangle = \frac{1}{\sqrt{15.25}}(1.5|00\rangle - 2|10\rangle + 3|11\rangle)$$

この例では、振幅ベクトルの要素の総数$N \times M$(つまり、2ビットx2データポイント=4)は2の累乗なので、必要な量子ビット数は$\boldsymbol{n \geq log_2(NM)}$よりピッタリ2。
$N \times M$が2の累乗でない場合、 $2^n \geq NM$となるようにnの値を選び、振幅ベクトルに情報量の少ない定数を埋め込む（残りの量子状態の振幅はゼロにする）。

参照：Textbook Tool

振幅符号化の実装

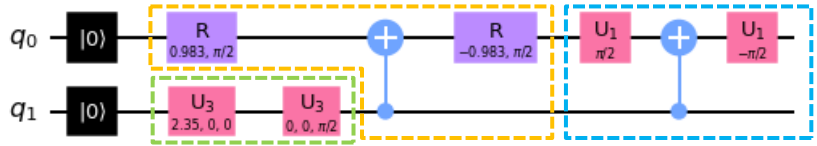
計算基底符号化と同様に、データセットを符号化する状態を計算して、initialize関数で準備できます:

$$|\mathcal{X}\rangle = \frac{1}{\sqrt{15.25}}(1.5|00\rangle - 2|10\rangle + 3|11\rangle)$$ なので

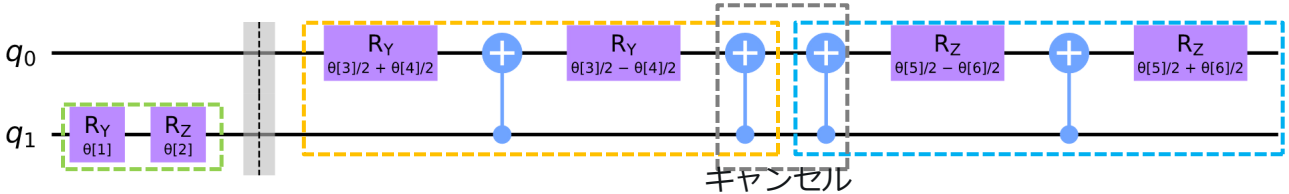
```
desired_state = [
    1 / math.sqrt(15.25) * 1.5,
    0,
    1 / math.sqrt(15.25) * -2,
    1 / math.sqrt(15.25) * 3]

qc = QuantumCircuit(2)
qc.initialize(desired_state, [0,1])

qc.decompose().decompose().decompose().decompose().decompose().draw("mpl")
```



先程のinitialize関数の2量子ビット版にZ回転も入れたもの　と同じ形



振幅符号化のまとめ

- 振幅符号化は、データを量子状態の振幅に符号化する。（実装にはInitialize関数を使う）

古典データセット

$$\mathcal{X} = \{x^{(1)}, \cdots, x^{(m)}, \cdots \mathrm{z}, x^{(M)}\}$$

MN次元のすべてのデータポイントを
長さ $N \times M$ の1つの振幅ベクトルに連結。

$$\alpha = A_{\mathrm{norm}}(x_1^{(1)}, \ldots, x_N^{(1)}, \ldots, x_1^{(m)}, \ldots, x_N^{(m)}, \ldots, x_1^{(M)}, \ldots, x_N^{(M)})$$

ここでA_{norm}は正規化定数であり、$|\alpha|^2 = 1$。

振幅符号化

$$|\mathcal{X}\rangle = \sum_{i=1}^{N} \alpha_i |i\rangle$$

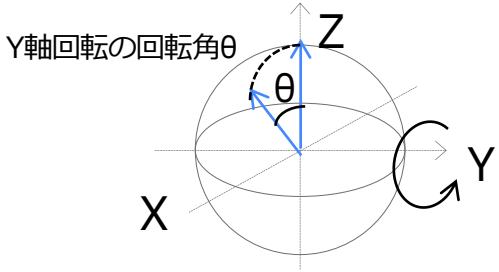
ここで、α_iは振幅ベクトルの要素であり、
$|i\rangle$は計算基底状態です。
符号化される振幅の数は$N \times M$です。

- 振幅符号化の利点：符号化に必要なのは$\boldsymbol{log_2(NM)}$ 量子ビットのみ。
- ただし、その後の計算に使うアルゴリズムが量子振幅の操作になり、量子状態を準備し測定する方法が効率的ではない傾向があります。（振幅増幅や位相推定には多くの制御ゲートが必要）

3. 角度符号化

角度符号化は、N個の特徴(feature)をn量子ビットの回転角に符号化します。(ここで、$N \leq n$)
たとえば、データポイント$x = (x_1, \cdots, x_N)$は次のように符号化できます。

$$|x\rangle = \bigotimes_{i=1}^{N} \cos(x_i)|0\rangle + \sin(x_i)|1\rangle$$



(*) $\bigotimes_{i=1}^{N}$ ：大きなシグマ（Σ）が異なる値の総和を表すのと同じように、この大きな「オータイムス」は、ある値の集合に対して、その右側にある式のクロネッカー積を取ることを意味します。

例）データポイント$x = (x_1,\ x_2, x_3)$は、以下のように３量子ビットで角度符号化できる。

$$|x\rangle = \bigotimes_{i=1}^{3} \cos(x_i)|0\rangle + \sin(x_i)|1\rangle$$

$$= (\cos(x_1)|0\rangle + \sin(x_1)|1\rangle) \otimes (\cos(x_2)|0\rangle + \sin(x_2)|1\rangle) \otimes (\cos(x_3)|0\rangle + \sin(x_3)|1\rangle)$$

$$= \begin{pmatrix}\cos(x_1)\\ \sin(x_1)\end{pmatrix} \otimes \begin{pmatrix}\cos(x_2)\\ \sin(x_2)\end{pmatrix} \otimes \begin{pmatrix}\cos(x_3)\\ \sin(x_3)\end{pmatrix}$$

データポイント
$x = (x_1, \cdots, x_N)$

角度符号化

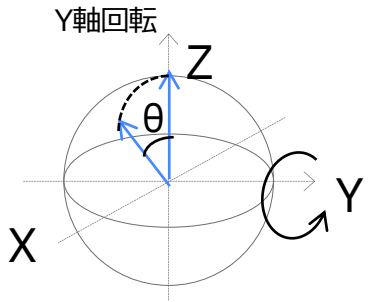
N量子ビット

$$|x\rangle = \bigotimes_{i=1}^{N} \cos(x_i)|0\rangle + \sin(x_i)|1\rangle$$

- 計算基底符号化、振幅符号化と違って、データセット全体ではなく、一度に1つのデータポイントのみを符号化する方法。
- 使用するのはN量子ビットで量子回路の深さも一定であるため、現在の量子ハードウェアに適用可能。

Y軸周りの単一量子ビットの回転を使って実装します。

$$RY(\theta) = \exp(-i\frac{\theta}{2}Y) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

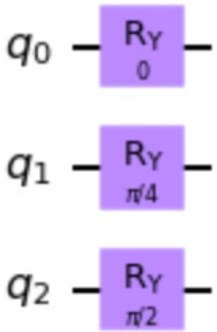


角度符号化の実装

データポイント$x = (0, \pi/4, \pi/2)$の時、

```
qc = QuantumCircuit(3)

qc.ry(0, 0)
qc.ry(math.pi/4, 1)
qc.ry(math.pi/2, 2)
qc.draw("mpl")
```



高密度角度符号化

高密度角度符号化は、角度符号化を少しだけ一般化したもの。
相対位相を使って1量子ビットごとに2つの特徴を符号化します。

データポイント$x = (x_1, \cdots, x_N)$は次のように符号化されます:

$$|x\rangle = \bigotimes_{i=1}^{N/2} \cos(x_{2i-1})|0\rangle + e^{ix_{2i}} \sin(x_{2i-1})|1\rangle$$

実装には、Phaseゲートを使えます。

$$p(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$$

4. 任意の符号化

角度符号化における関数をさらに一般化したもの。

- 角度符号化： sin関数
- 高密度角度符号化： sin関数とexp関数
- **任意の符号化**：任意のユニタリー演算（CNOT含む）

N個の特徴(feature)をn量子ビット上のN個のパラメーター化されたゲートの回転として符号化。（ここで、$n \leq N$）

データセット全体ではなく、一度に1つのデータポイントのみを符号化。

一定の深さの量子回路で$n \leq N$ であるn量子ビットを使用しているため、現在の量子ハードウェアで実行しやすい。

任意の符号化の例：EfficientSU2

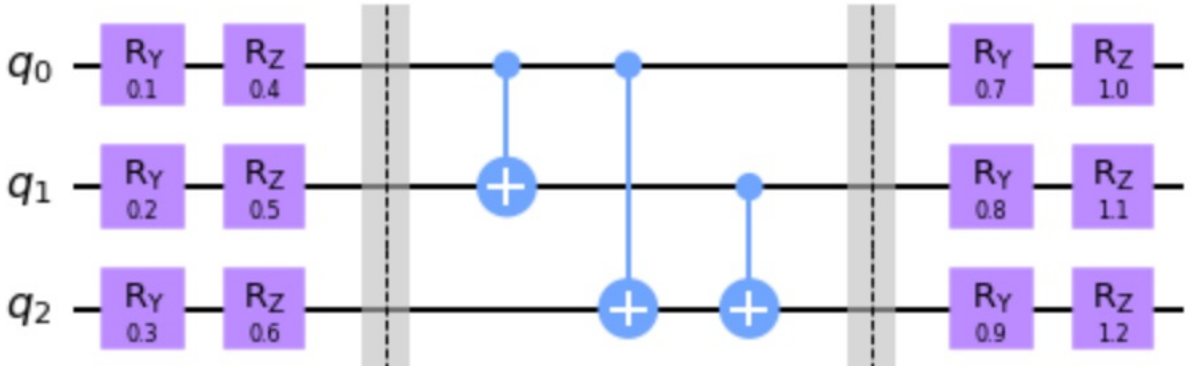
データポイント $x = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2]$を12の特徴(feature)で符号化し、
パラメーター化された各ゲートを使用して符号化します。

```
from qiskit.circuit.library import EfficientSU2

circuit = EfficientSU2(num_qubits=3, reps=1, insert_barriers=True)
circuit.decompose().draw("mpl")
```

```
x = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2]

encode = circuit.bind_parameters(x)
encode.decompose().draw("mpl")
```



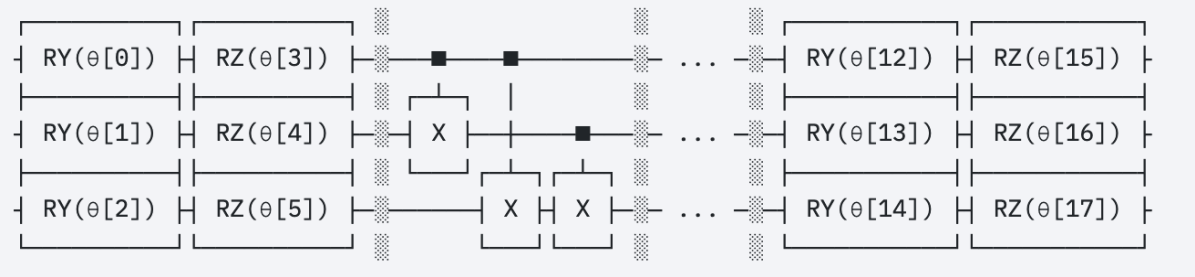
EfficientSU2とは

ハードウェア的に効率的なSU(2)の回路です。

SU(2) :群の分類で、2 行 2 列のユニタリー行列で行列式が 1 であるもの

SU(2)とCXの層で構成されています。
変分量子アルゴリズムの試行波動関数や機械学習の分類回路を準備するための経験的な形式です。

3量子ビットの場合、 SU(2) ゲートとしてYゲートとZゲートを用いると、

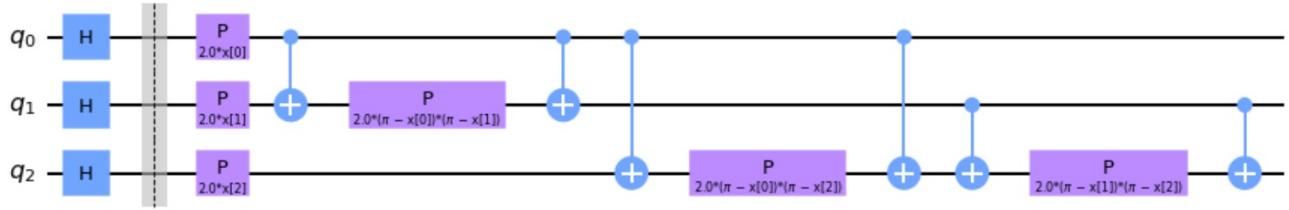


任意の符号化の例：ZZFeatureMap

3量子ビットのZZFeatureMap回路は、6つのパラメーター化ゲートがあるが、
3つの特徴(feature)のデータポイントしか符号化しません:

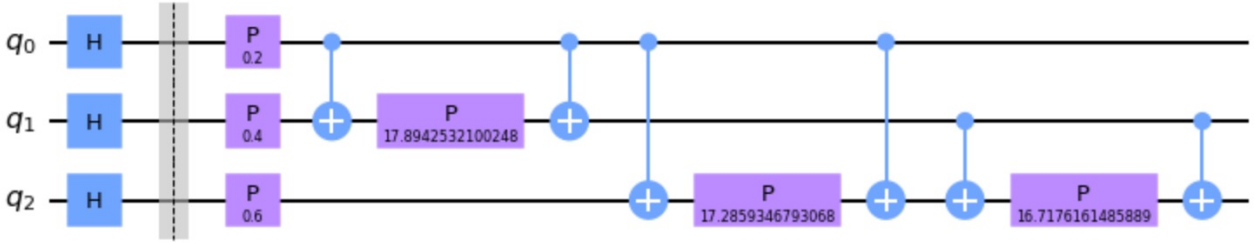
```
from qiskit.circuit.library import ZZFeatureMap

circuit = ZZFeatureMap(3, reps=1, insert_barriers=True)
circuit.decompose().draw("mpl")
```




```
x = [0.1, 0.2, 0.3]

encode = circuit.bind_parameters(x)
encode.decompose().draw("mpl")
```



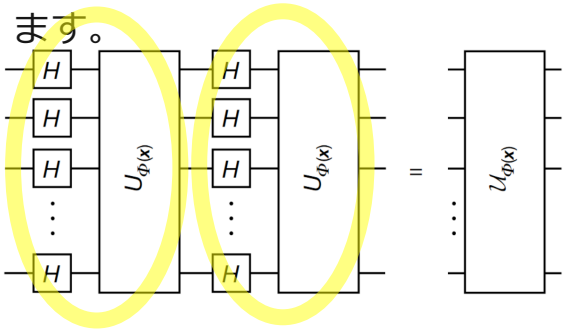
量子特徴量マップ

後ろの章（Quantum feature maps and kernels）で詳しく説明されます

ZZFeatureMap以外にも様々なFeature Map(特徴量マップ)があります。

基本形は、エンタングルメントを生成するブロック$U_{\Phi(x_k)}$とHゲートの層が交互に連なった形をしています。

（d回繰り返し）



$$|\Phi(\mathbf{x})\rangle = \prod_{d} \mathrm{U}_{\Phi(\mathrm{x})}\mathrm{H}^{\otimes \mathrm{n}} \ |0\rangle$$ ：$\mathrm{U}_{\Phi(\mathrm{x})}\mathrm{H}^{\otimes \mathrm{n}}$ をd回繰り返す

$$U_{\Phi(x)} = exp\left(i\sum_{i}\phi_i(\boldsymbol{x})P_i + i\sum_{i,j}\phi_{ij}(\boldsymbol{x})P_{ij} + \sum_{i,j,k}\phi_{ijk}(\boldsymbol{x})P_{ijk} + \cdots\right)$$

- $\mathrm{i,j,k}$ … はエンタングルさせるビットの組み合わせ
- $\mathrm{P_i} \in \{\mathrm{I,X,Y,Z}\}$　　:パウリ行列
- $\phi_i(\boldsymbol{x})$はパウリ行列の係数(位相)

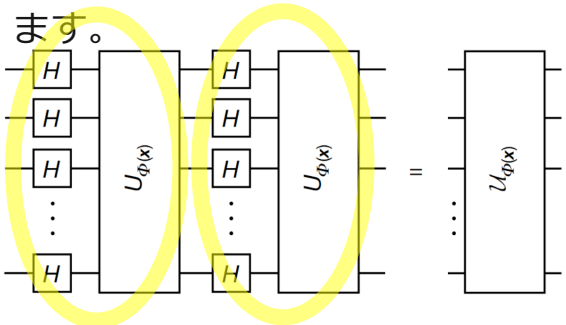
量子特徴量マップ

後ろの章（Quantum feature maps and kernels）で詳しく説明されます

ZZFeatureMap以外にも様々なFeature Map(特徴量マップ)があります。

基本形は、エンタングルメントを生成するブロック$U_{\Phi(x_k)}$とHゲートの層が交互に連なった形をしています。

（d回繰り返し）



$|\Phi(\mathbf{x})\rangle = \prod_d \mathrm{U}_{\Phi(\mathrm{x})}\mathrm{H}^{\otimes \mathrm{n}} |0\rangle$ ：$\mathrm{U}_{\Phi(\mathrm{x})}\mathrm{H}^{\otimes \mathrm{n}}$ をd回繰り返す

$$U_{\Phi(x)} = exp\left(i\sum_i \phi_i(\boldsymbol{x})P_i + i\sum_{i,j} \phi_{ij}(\boldsymbol{x})P_{ij} + \sum_{i,j,k} \phi_{ijk}(\boldsymbol{x})P_{ijk} + \cdots\right)$$

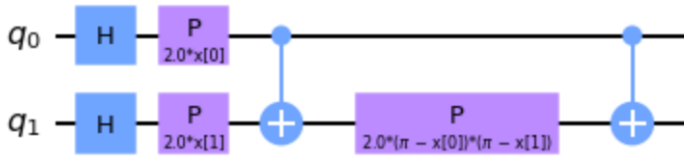
ZZ feature mapの場合

$$\begin{cases} \phi_i(\boldsymbol{x}) = x_i \\ \phi_{ij}(\boldsymbol{x}) = (\pi - x_i)(\pi - x_j) \end{cases}$$

$$\begin{cases} P_i = Z_i \\ P_{ij} = Z_{ij} = Z_i\, Z_j \end{cases}$$

2量子ビットの時：

$$\mathrm{U}_{\Phi(\mathrm{x})} = exp(ix_0Z_0 + ix_1Z_1 + i(\pi - x_0)(\pi - x_1)Z_0Z_1)$$



データの符号化

1. 計算基底符号化

例）データセット$\mathcal{X} = \{x^{(1)} = 101, x^{(2)} = 111\}$ ➡ $|\mathcal{X}\rangle = \frac{1}{\sqrt{2}}(|101\rangle + |111\rangle)$

2. 振幅符号化

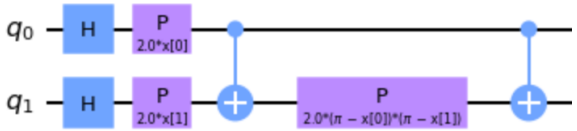
例）$\mathcal{X} = \{x^{(1)} = (1.5, 0), \quad x^{(2)} = (-2, 3)\}$ ➡ $|\mathcal{X}\rangle = \frac{1}{\sqrt{15.25}}(1.5|00\rangle - 2|10\rangle + 3|11\rangle)$

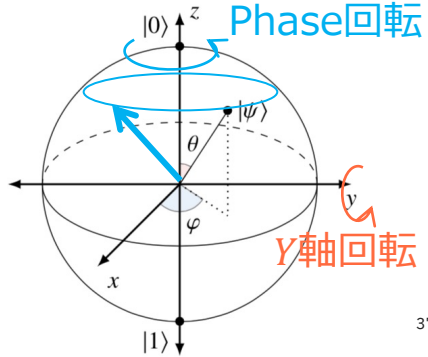
3. 角度符号化

例）データポイント$x = (x_1,\ x_2)$ ➡ $S_x = RY(2x_1) \otimes RY(2x_2)$

4. 任意の符号化(角度符号化の拡張)

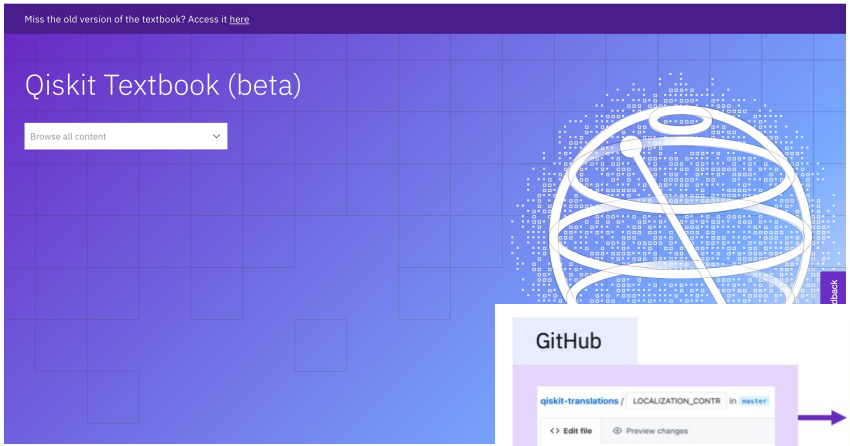
例）データポイント$x = (x_1,\ x_2)$ ➡



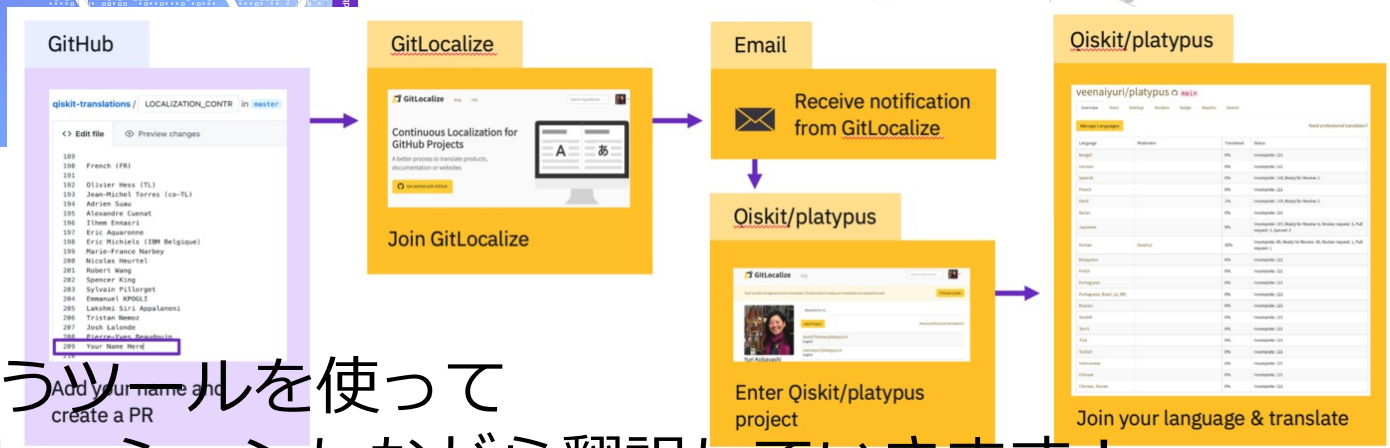


新版Qiskit Textbookの翻訳協力者募集中！

https://github.com/qiskit-community/qiskit-translations



Help make Qiskit accessible to non-English speaking communities



GitLocalizeというツールを使って
複数人でコラボレーションしながら翻訳していきます！

量子コンピューター夏の学校

～今年は量子シミュレーションです！

開催期間：7/18(月)~7/29(金)

第2段申し込み：6/3(金)朝7:00(日本時間) Open

Qiskit Global Summer School 2022: Quantum Simulations

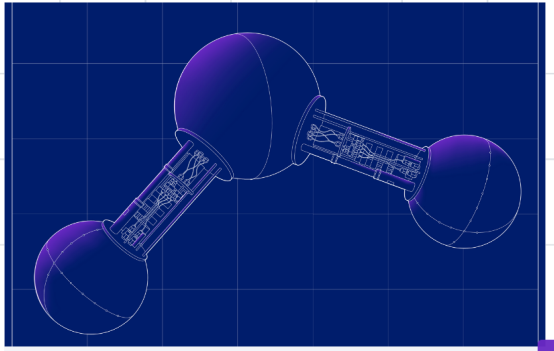
The Qiskit Global Summer School 2022 is a two-week intensive summer program designed to empower the quantum researchers and developers of tomorrow with the skills and know-how to explore the world of quantum computing and its applications. This third-annual summer school will provide a focused introduction to quantum computing and its applications to quantum simulation, with a specific focus on quantum chemistry.

Please follow Qiskit Twitter for more details and updates. For any questions, please check out our FAQ below!

Early Bird Registration will open at 12:00 PM EST on May 26, 2022.

https://qiskit.org/events/summer-school/

Register now!



Qiskit Global Summer School 2022: Quantum Simulations

The Qiskit Global Summer School returns as a two-week intensive course focused on Quantum Simulations and more!

Online

July 18 - 29, 2022

Learn more

Feedback