量子コンピューティング向けテストフレームワークの開発

1. 背景

近年、クラウドの普及と量子コンピュータの性能向上に伴い、誰もが容易に高度な量子計算に挑戦できるようになってきた。一方、プログラム実装は困難である。量子ビットを操作するプログラムを書く必要があることに加え、多くの場合量子回路が複雑となることは避けられない。そのため、量子計算に慣れた開発者であっても、計算結果に誤りがないことの確認は容易ではない。量子コンピュータの利用者増加に伴い、開発者が自身の書いた量子プログラムの正しさを確認したいという需要は増えていくと考えられる。

古典コンピュータのプログラムが正しいことの担保としてはテストコードを書くことが一般的に有効である。テストコードは、開発者が自分のプログラムコードに誤りが無く想定通りに動くことを確認するために開発者自身によって書かれる。具体的には、入力とそれに対する出力が想定通りであることを確認する。テストコードを書く際には以下の苦労が伴う：

- ✓ テスト用に余分にコードを書く必要がある。
- ✓ 規約を設けずに複数人で開発をすると、テストコードの書き方がバラバラになり、後で確認が困難なテストが散在することになる。

これらを軽減する目的で、主要なプログラミング言語にはテスト作成用のフレームワークが存在する。フレームワークは、入出力の関係が開発者の想定通りであることの確認処理を抽象化する複数のアサートメソッド、テスト実行前および完了後の処理を簡易化する機能、テストの実行順番を管理する機能等を備えている。古典コンピュータ用のプログラミングに対するテストフレームワークは充実しており、またテストのベストプラクティスやナレッジも多くある。

2. 目的

開発者が自分の量子計算プログラムコードに誤りが無いことを確認したい場合には、古典の場合に倣って、開発者がテストコードを作成することが有効だと我々は考えている。既存の古典的なフレームワークをそのまま使えることが理想的だが、量子コンピュータでは観測値は確率論的に決まるので、観測値をテストの入力および出力として使うことができない。この課題は、シミュレータを使うことで解決される。すなわち、量子回路をシミュレートすることで、状態ベクトルやユニタリ行列をテストの入力および出力として決定論的に扱うことができる。しかし、依然として下記の課題があり、既存の古典的なフレームワークをそのまま使うことは手間がかかる：

課題①　古典コンピュータでの開発は高級言語で書かれるが、量子コンピュータはまだ発展が十分ではなくアセンブリ言語に近い表現で量子回路をプログラムする必要がある。この抽象化レイヤーの違いのため、古典的なテストフレームワークをそのまま適用することができない。

課題②　複雑な大規模量子回路においては、出力として想定される状態ベクトルを事前に知ることは容易でない。

そこで、本プロジェクトでは、上記の課題をクリアすることを念頭に置いて、量子コンピュータに特化したテストのフレームワークを開発することを目的とした。

3.　ソフトウェア開発内容

3.1 動作環境

Python3.7 以上

3.2 構成

ソフトウェアは 4 つのブロックで構成される。内容・意義等は下記：

① アサートメソッド

入出力の確認処理を抽象化する関数であり、ユーザーにとってのインターフェイス。状態ベクトル、行列、または量子回路をテスト対象のオブジェクトとして引数に受け取る。具体的なアサートメソッドのリストは「3.3 機能」を参照。

② コンバータ

入力される量子回路を独自実装の量子回路シミュレータクラス(QuantestPyCircuit)に変換する。Qiskit と OpenQASM に対応済み。

③ シミュレータ

量子回路をシミュレーションする。 QuantestPyCircuit を親クラスとして 2 つ存在し、アサートメソッドの内部で適切に選択および実行される：

名称	機能	特徴
状態ベクトルシミュレータ(StateVectorCircuit)	一般的なゲートは全て使用可能	何でも計算できる一方で数十量子ビットが上限。古典計算機で効率的にテストできないモジュールのシミュレーションでの使用を想定。
多制御パウリシミュレータ(PauliCircuit)	多制御パウリ・Swap ゲートのみ使用可能で入力は計算基底に限られる	計算可能な対象物が限られる一方で数百万量子ビットの処理が可能。古典計算機で効率的にテストできるモジュールのシミュレーションでの使用を想定。

④ 可視化ツール

量子回路の作図をサポートする。バックエンドで多制御パウリシミュレータが実行されるアサートメソッドは、エラー発生時に可視化ツールを利用して彩色された回路図を出力できる。これは、エラー箇所の特定に大いに役立つ。

上記 4 つのブロックの位置関係を明らかにするために、図 1 にユースケース例を示したシーケンス図を載せる。

3.3 機能

ユーザーはアサートメソッドの機能を理解し、適切な場所で適切なアサートメソッドを呼び出す必要がある。アサートメソッド名と機能の一覧を表 1 に示す。

3.4 ソフトウェアで解決する課題

一部のアサートメソッドは引数に量子回路を受け取ることが出来る。これは計画時の課題①を解決する。現在多くの開発者はアセンブリ言語に近い表現で開発を行っているため、この仕様はこれら多くの開発者にとって都合が良い。例えば、補助量子ビットが役目を終えた後に 0 に初期化されていることをテストするテストコードは、フレームワーク無しでは、数十行にもなる。assert_ancilla_reset というアサートメソッドを使うことで、引数に量子回路を渡すたった一行のコードでテストが可能となる。その他のアサートメソッドも適切な抽象度で作成されており、実装済みの 10 個のアサートメソッドは多くのテストケースに対応できると考えている。

4. 新規性・優位性

量子計算プログラミングに利用できるテストフレームワークとして標準的といえるものがまだ存在しないことを考えると、本プロジェクトで開発したソフトウェアの新規性は高いと言える。

本プロジェクトでは、そのままではテスト困難な大規模量子回路をテストする方法の指針を示し、その指針に基づいたテストを支援するツールを開発した。これは計画時の課題②を解決する。Q#や Cirq でも既に複数のアサートメソッドが単体テスト用に提供されているが、本プロジェクトが大規模回路のテストもターゲットとしている点はこれら類似のものと比較した場合の優位性と言える。

5. 期待されるユーザー価値と社会へのインパクト

本プロジェクトの成果を活用することで、量子計算のプログラマーはテストを書きやすくなる。また、誤ったプログラムを実行することが減るので、量子コンピュータ実行コスト削減になり、それは量子コンピュータのリソースを社会全体として有効活用していくことにも繋がる。結果的に、開発者は開発における本質的な部分に注力し、量子コンピュータを活用したソフトウェア開発の速度が高まることが期待される。

6. 氏名

松本光洋　中村純也

（参考）

QuantestPy: https://github.com/QuantestPy/quantestpy

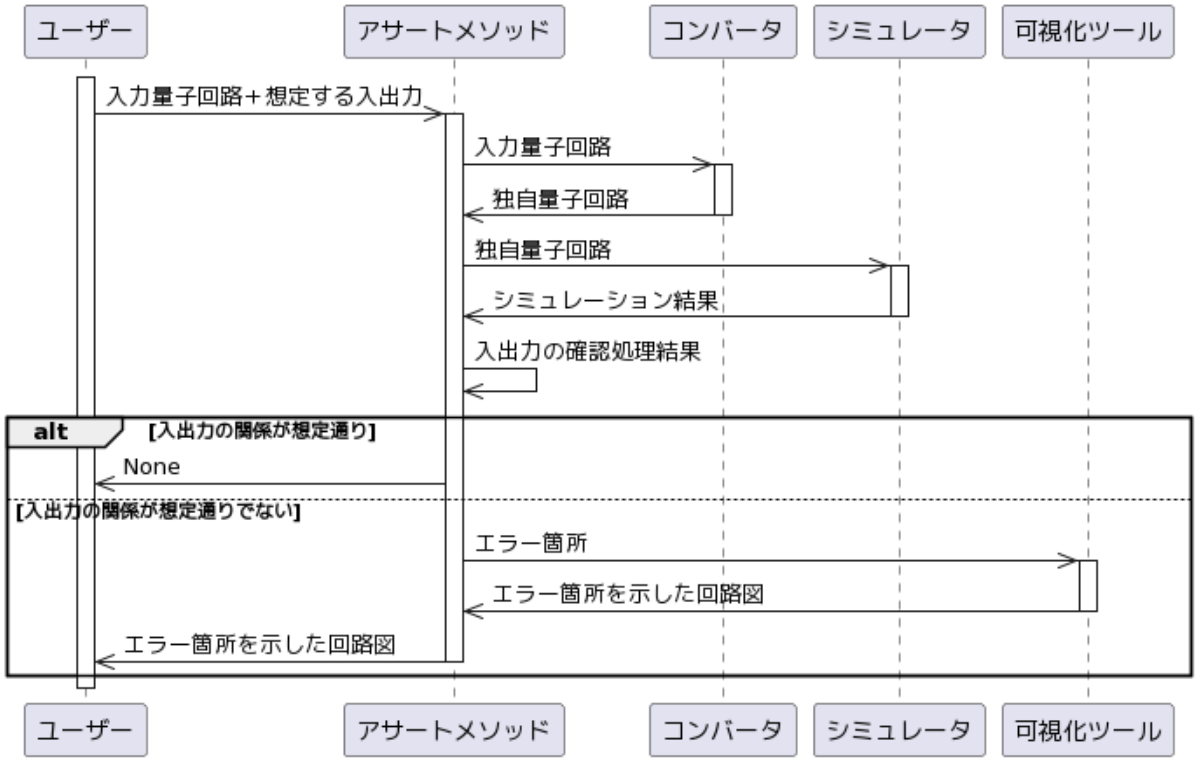

図１　ユースケース例を示したシーケンス図

メソッド名	機能
assert_normalized_state_vector	状態ベクトルのノルムが 1 であることを確認する。
assert_equivalent_state_vectors	二つの状態ベクトルが等しいことを確認する。
assert_unitary_operator	行列がユニタリであることを確認する。
assert_equivalent_operators	二つの行列が等しいことを確認する。
assert_circuit_equivalent_to_operator	量子回路とユニタリ行列が等しいことを確認する。
assert_qubit_reset_to_zero	指定の量子ビットが 0 に初期化されていることを確認する。
assert_ancilla_reset	補助量子ビットが 0 に初期化されていることを全計算基底の初期状態において確認する。
assert_equivalent_circuits	二つの量子回路が等しいことを確認する。
assert_unary_iteration	インデックス付き量子回路を想定通り組めていることを確認する。エラー時に回路図出力のオプションあり。
assert_circuit_equivalent_to_output_qubit_state	終状態の量子ビットが想定と等しいことを確認する。エラー時に回路図出力のオプションあり。

表１　アサートメソッド一覧