README

Sun's Algorithm Implementation

February 2025

1 Description

This repository contains all the files related to the implementation of Sun's Algorithm for the problem of quantum state preparation of an n-qubit quantum state, using $2n$ ancillary qubits, that is a particular subcase of the range presented in the first theorem of Sun et al. original paper.

All modules have been developed in Python with the support of the PennyLane library, which enables quantum computing simulations.

The repository includes various files containing classes and functions essential for the complete implementation:

- **utils.py:** Contains all the functions used to operate with vectors, print results, and compute parameters using traditional computing algorithms.
- **circuit_classes.py:** Contains all the classes used to generate the desired quantum circuit and to simulate quantum states.
- **lambda_n.py:** Contains a scalable implementation of the Λ_n circuit described in the original paper. It uses an initial random complex vector and a variable N that represents the number of qubits of the input register, but all parameters can be changed as a preference.
- **quantum_state_preparation.py:** Contains the implementation of the whole quantum state preparation circuit, using a random complex vector as the desired final quantum state.
- **phases_linear_system.py:** Contains all the functions needed to generate the coefficients related to the phases of the QSP.

2 Requirements

The project has been implemented using Python 3.12 (Python 3.10 or later should also work) and the latest version of the *PennyLane* library. The implementation also makes use of the *math*, *operator*, *numpy*, and *matplotlib.pyplot* modules, which should already be present in the latest version of Python.

3 Usage

To test the implementation, it is sufficient to run *"quantum_state_preparation.py"* to generate a desired random N-qubit quantum state. It is also possible to generate a specific, real or complex, N-qubit quantum state by modifying the *coefficients* variable with a 2^n-element vector of l_2-norm = 1:

```
coefficients = complex_numbers.tolist() # Modify this to insert the
    desired vector
```

3.1 Quantum Circuit Implementation

Every implemented circuit makes use of some derived classes from the abstract class *Stage*, to define the position of the gates and the values of some useful parameters, and to instantiate a *QuantumCircuit* object that is basically a collection of stages and allows displaying the circuit and the quantum state in various ways using the `QuantumCircuit.printCircuit(params)` method.

It is also possible to obtain directly the output state of the circuit by calling the `QuantumCircuit.computeQuantumState(params)` method or to display the circuit metadata using the `QuantumCircuit.getCircuitInfos()` method.

3.2 Calculation of Parameters

Multiple functions are used to calculate the parameters needed to make all the necessary phase shifts in the computational basis; this process is done using traditional computation and can therefore be executed separately from the quantum computation part. A user can simply call the `qspParameters(coeff_vector, num_of_qubits)` function to generate $3 * (2^n - 1)$ α angles from the desired coefficient vector, these angles are then split and divided between the 3 Λ_n components used to implement the UCG_n.

3.3 Λ Circuit

With the current implementation, it is possible to build the circuit for Λ_n, with n as a variable, thanks to the adopted stage structure.

The actual script works by initializing the variable n and then randomly generating the complex coefficient vector, whose values correspond to the probability amplitudes of the desired final quantum state. Then, the necessary angles are calculated from the initial vector and associated with each computational basis string. Finally, using the *Stage* classes, the quantum circuit is constructed and the final state - in this case, the values of the diagonal of the matrix associated with each multiplexor - can be retrieved.