CSC299 - Report 1 - Simple Implementation - One Qubit Ancilla

September 1, 2021

This shall be the first report of the CSC299 project about the automatized implementations of nonunitary operations using quantum algorithms. This report is meant to serve as an introduction to the concept of implementing any operation using its decomposition as a linear combinations of unitaries.

The primary objective of these reports is to implement the algorithm described by Berry, Childs, Cleve, Kothari, Somma (2015) which aims to encode an arbitrary operation inside a larger unitary operation.

Reference used: Simulating Hamiltonian Dynamics with a truncated Taylor Series (BCCKS)

[1]:
```
import tequila as tq
import copy
from typing import Iterable
import numpy as np
```

1 Introduction

Quantum computing refers to using the properties of quantum states to perform computation; popular examples of such properties include superposition, interference and entanglement. It is believed that quantum computing can enable more efficient algorithms for certain problems than can be possible using solely classical methods, and one such supposed application involves the simulation of quantum systems.

Over the past thirty years, multiple developments have been made in this field which demonstrate that quantum computers can efficiently simulate Hamiltonian dynamics and much of current ongoing work involves improving the performance of these methods and expanding the scope of such simulations.

We wish to have a method that can simulate the effect of running any non-unitary operation H, if we know how to write it as a linear combination of unitary (LCU) operations U_j if we already have some mechanisms for implementing each of the U_j.

One of the many practical uses of such a technique is as follows. Suppose that we wish to simulate the evolution under some Hamiltonian H for time t within a maximum error of some $\epsilon > 0$ with the operator $U = e^{-iHt}$. Divide the time t into r equal time-segments of length t/r. For each segment, define the operator for time-evolution within that segment as follows:

$$U_r = e^{-iHt/r} \approx \sum_{j=0}^{K} \frac{1}{j!}(-iHt/r)^j$$

Here, we wish to truncate the Taylor series expansion of the exponential at some order K such that the maximum error for each segment is ϵ/r, which would imply that the maximum result across all the segments is ϵ. Assuming that $r > ||H||\, t$ (where $||\cdot||$ denotes the usual matrix norm), it has been shown by Berry et. al (2014) [insert citation] that we must have:

$$K \in \mathcal{O}\left(\frac{\log(r/\epsilon)}{\log(\log(r/\epsilon))}\right)$$

The rest of this report now continues with the goal of implementing the LCU algorithm in general, and not just focus on this particular example of a potential application.

2 Procedure

Consider the Hamiltonian H which acts over n qubits. Let $N = 2^n$ denote the dimension of the Hilbert space $\mathcal{H}$. Without loss of generality, we can assume that H is Hermitian. Indeed, suppose that H was not Hermitian. Then, as shown by Harrow, Hassidim, Lloyd (2009) while solving linear systems of equations, we would simply define $\tilde{H}$ such that:

$$\tilde{H} = \begin{bmatrix} 0 & H^\dagger \\ H & 0 \end{bmatrix}$$

In this case, $\tilde{H}$ acts over $2n$ qubits, where the first n qubits are ancillae. Furthermore, as is evident by the construction above, $\tilde{H}$ is Hermitian, even though H is not. And so, if H is not Hermitian, we would simply proceed with considering $\tilde{H}$ instead.

Reference used: Quantum Algorithm for solving linear systems of equations (HHL)

Now, suppose that we have the decomposition $H = \sum_{j=0}^{m-1} \alpha_j U_j$ over n qubits, where each of the U_j's are unitary. We know that any arbitrary Hamiltonian can be expressed in such a form due to the fact that any element of the standard orthonormal basis can be expressed as a linear combination of such unitary operators. While it is possible to mathematically prove this statement, the derivation of such a proof possesses less importance to our purposes and it suffices to demonstrate that it is possible to decompose any arbitrary qubit Hamiltonian in Tequila into its constituent Pauli-strings using the paulistrings attribute in code, as follows.

[3]:
```
H = tq.QubitHamiltonian()
paulis = H.paulistrings
```

In order to have a convenient notation for normalization factors in later derivations, we define $s = \sum_{k=0}^{m-1} \alpha_k$ to denote the sum of all coefficients in the linear combination of unitaries. Moreover, without loss of generality, we can assume that $\alpha_j > 0$ for all j. Indeed, suppose that there exists some $\alpha_j < 0$: we can then replace the term $\alpha_j U_j$ with the term $\alpha'_j U'_j$, where $(\alpha'_j, U'_j) = (-\alpha_j, -U_j)$.

Given m unitaries in the expansion of H, we shall require $\lceil \log_2 m \rceil$ qubits in the ancillary register, along with as many qubits as required by H for the state register. We define two oracles: the Prepare oracle shall only act on the ancilla, while the Select operator shall act on both the ancillary and state registers. We use this approach because we want to have Prepare assemble a state in the ancilla register that is representative of the coefficients α_j in the LCU decomposition, and we want Select to consequently assemble the LCU on the state register with the help of the ancilla state obtained after applying Prepare.

We define the Prepare oracle such that:

$$\text{Prepare}\,|0\rangle = \frac{1}{\sqrt{s}} \sum_{j=0}^{m-1} \sqrt{\alpha_j}\,|j\rangle$$

We also define the Select oracle as follows:

$$\text{Select}\,|j\rangle\,|\psi\rangle = |j\rangle\,U_j\,|\psi\rangle$$

Next, we define the operator W using Prepare and Select as shown below:

$$W = (\text{Prepare}^\dagger \otimes I_N)\text{Select}(\text{Prepare} \otimes I_N)$$

Therefore, we see that:

$$\begin{aligned}
W\,|0\rangle\,|\psi\rangle &= (\text{Prepare}^\dagger \otimes I_N)\text{Select}(\text{Prepare} \otimes I_N)\,|0\rangle\,|\psi\rangle \\
&= (\text{Prepare}^\dagger \otimes I_N)\text{Select}\left(\frac{1}{\sqrt{s}} \sum_{j=0}^{m-1} \sqrt{\alpha_j}\,|j\rangle\,|\psi\rangle\right) \\
&= (\text{Prepare}^\dagger \otimes I_N)\left(\frac{1}{\sqrt{s}} \sum_{j=0}^{m-1} \sqrt{\alpha_j}\,|j\rangle\,U_j\,|\psi\rangle\right) \\
&= \frac{1}{s}\,|0\rangle\,H\,|\psi\rangle + \sqrt{1 - \frac{1}{s^2}}\,|\Phi\rangle
\end{aligned}$$

where $|\Phi\rangle$ is some state whose ancillary component lies in a subspace orthogonal to $|0\rangle$. Hence, if $P_0 = |0\rangle\langle 0| \otimes I_N$ is the projector onto the zero-subspace of the ancillary register, then we have that:

$$P_0 W\,|0\rangle\,|\psi\rangle = \frac{1}{s}\,|0\rangle\,H\,|\psi\rangle$$

This implies that if we measure the ancillary register after applying the W operator to $|0\rangle\,|\psi\rangle$ and we find that the ancillary qubits are all in the 0 state, then we trigger the succesful preparation in the state register, as desired. Hence, the success probability for this algorithm is currently $\frac{1}{s_0^2} = \frac{1}{s^2}\,||H\,|\psi\rangle||^2$, which we can improve upon as detailed below.

2.1 Notes on procedure

Notice that when H can be written as a linear combination of m unitary operators with all positive coefficients, then we need a total of $k = \lceil \log_2 m \rceil$ ancillary qubits to implement the Prepare operator described in the above procedure. This is because we need enough qubits in the ancillary register to define a unique labeling for each unitary in the linear combination.

Also note that the success probability of emulating H by the above procedure is $\frac{1}{s_0^2} = \frac{||H|\psi\rangle||^2}{s^2}$. This probability depends on the size of the ancilla and decreases exponentially in the number of qubits contained in the ancilla. We can apply the procedure of amplitude amplification to increase this probability up to 1. Ideally, if $s = 2$, then this fits the procedure of oblivious amplitude amplification. However, since H is not unitary itself, we need to slightly alter the approach of oblivious amplitude amplification that we use here, which is further explained in a later section of this report.

2.2 One qubit example

Suppse that we are given a Hamiltonian H as follows:

$$H = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Then, clearly, H is non-unitary, since $HH^\dagger \neq I_2$. However, we can write $H = \frac{1}{2}I - \frac{1}{2}Z$, thus expressing H as the sum of two unitary operations. Because we need all coefficients to be real and positive, we shall instead use the expansion $H = \frac{1}{2}I + \frac{1}{2}(-Z)$.

Now, we define the Prepare operator as follows:

$$\text{Prepare}\,|0\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$$

It turns out that, in this case, since both coefficients are equal, the Prepare operator is equivalent to a Hadamard operation.

We also define the Select operator as follows:

$$\begin{aligned} \text{Select}\,|0\rangle\,|\psi\rangle &= |0\rangle\,|\psi\rangle \\ \text{Select}\,|1\rangle\,|\psi\rangle &= |1\rangle\,(-Z)\,|\psi\rangle \end{aligned}$$

This simply corresponds to controlled $-Z$ operation.

Next, we define the operator W using Prepare and Select as shown below:

$$W = (\text{Prepare}^\dagger \otimes I_N)\text{Select}(\text{Prepare} \otimes I_N)$$

Following the same derivation as above:

$$W\,|0\rangle\,|\psi\rangle = |0\rangle\,H\,|\psi\rangle$$

Notice that, in this case, since $s = 1/2 + 1/2 = 1$, this implies that there is no other orthogonal state, and so, the success probability is 1.

3 Implementation of one qubit ancilla

In this section, we wish to create an implementation of the LCU procedure using Tequila.

3.1 Implementing Select operator

We first need to define a subroutine to return the controlled version of the unitary. We do so by naturally translating the definition of the Select operator into Python, as follows. We first need to create a way of adding extra controls into each unitary operation, as required by the algorithm, since this is not a part of Tequila's built-in features at the moment. Note that this was included into the library in version > 1.5.1 via the method U = U.add_controls(…), where U is an object of class QCircuit. To do so, we construct the following function which returns a new circuit with additional control qubits.

[3]:
```
def control_unitary(ancilla, unitary: tq.QCircuit) -> tq.QCircuit:
    """Return controlled version of unitary

    SHOULD NOT mutate unitary

    Preconditions:
        - ancilla and unitary have no common qubits
    """
    gates = unitary.gates
    cgates = []
    for gate in gates:
        cgate = copy.deepcopy(gate)
        if isinstance(ancilla, Iterable):
            control_lst = list(cgate.control) + list(ancilla)
        else:
            control_lst = list(cgate.control) + [ancilla]
        cgate._control = tuple(control_lst)
        cgate.finalize()
        cgates.append(cgate)

    return tq.QCircuit(gates=cgates)
```

The select oracle is naturally translated into code as follows.

[4]:
```
def select_1ancilla(ancillary, unitary_0: tq.QCircuit, unitary_1: tq.QCircuit)␣
↪-> tq.QCircuit:
    """
    Select operator, when the Hamiltonian can be expressed as the linear␣
↪combination of two
    unitary operators.
    Requires only one ancillary qubit.

    Returns the select oracle.
    """
    impl_1 = control_unitary(ancilla=ancillary, unitary=unitary_1)

    x_gate = tq.gates.X(target=ancillary)
    control = control_unitary(ancilla=ancillary, unitary=unitary_0)

    impl_0 = x_gate + control + x_gate

    return impl_1 + impl_0
```

3.2 Implementing Prepare operator

Next, we implement the prepare oracle for the single-qubit ancilla case in the form of an R_y rotation as follows.

We are given the values of α_0, α_1 in the expansion, and we wish to create a circuit that can be expressed as follows:

$$\text{Prep} = \begin{bmatrix} \sqrt{\frac{\alpha_0}{\alpha_0+\alpha_1}} & \sqrt{\frac{\alpha_1}{\alpha_0+\alpha_1}} \\ x_0 & x_1 \end{bmatrix}$$

Here, x_0 and x_1 denote arbitrary elements of the matrix with which we are not concerned about. This is because the Prep circuit shall only ever be applied to the zero-state on the ancilla, and so, the mapping for the other basis states of the ancilla is of no importance.

Now, recall that the general form for a R_y rotation of angle θ is as follows:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

Hence, if we wish to express the Prep operator in the form of a R_y rotation, we would hvae the following:

$$\cos(\theta/2) = \sqrt{\frac{\alpha_0}{\alpha_0+\alpha_1}}$$
$$-\sin(\theta/2) = \sqrt{\frac{\alpha_1}{\alpha_0+\alpha_1}}$$

This implies that:

$$\theta = -2 \cdot \arcsin\left(\sqrt{\frac{\alpha_1}{\alpha_0+\alpha_1}}\right)$$

Therefore, we now see that $\text{Prep} = R_y\left(-2 \cdot \arcsin\left(\sqrt{\frac{\alpha_1}{\alpha_0+\alpha_1}}\right)\right)$ and we can naturally translate this into Python as shown in the following cell.

[5]:
```python
def prepare_1ancilla(ancillary, sum_of_unitaries: list[tuple[float, tq.QCircuit]]) -> tq.QCircuit:
    """
    Prepare operator, when the Hamiltonian can be expressed as the linear combination of two
    unitary operators.
    Requires only one ancillary qubit.

    Preconditions:
        - alpha_0 > 0 and alpha_1 > 0
    """
    alpha_0, alpha_1 = sum_of_unitaries[0][0], sum_of_unitaries[1][0]

    theta = -2 * np.arcsin(np.sqrt(alpha_1 / (alpha_0 + alpha_1)))

    return tq.gates.Ry(target=ancillary, angle=theta)
```

[11]:
```
def lcu_1ancilla(unitaries: list[tuple[float, tq.QCircuit]]) -> tq.QCircuit:
    """Return the circuit (Prep)(Select)(Prep.dagger()) according to the LCU␣
↪algorithm,
    with 0 as the ancilla qubit.

    Preconditions:
        - all(0 not in unitary.qubits() for unitary in [u[1] for u in␣
↪unitaries])
    """
    prepare = prepare_1ancilla(0, unitaries)
    circ = prepare + select_1ancilla(0, unitaries[0][1], unitaries[1][1]) +␣
↪prepare.dagger()
    return circ
```

3.3 Notes on implementation

The input and the output of the functions are circuits in Tequila. This implies that, assuming we know how to efficiently implement the circuits corresponding to each unitary in the linear combination, then we can prepare a circuit for the operator W defined by the algorithm. This shows that the circuit is independent of starting state, so we can efficiently simulate the Hamiltonian H with any initial state $|\psi\rangle$ that we know how to construct.

3.4 Example

In this section, we translate the example case shown above into Python using the functions we defined.

[12]:
```
def example_function() -> tq.QCircuit:
    """Test example for LCU with two unitary matrices"""
    identity = tq.QCircuit()
    unitaries = [(0.5, identity), (0.5, tq.gates.Z(1))]
    return lcu_1ancilla(unitaries=unitaries)
```

We can draw the result as follows.

[13]:
```
result = example_function()
tq.draw(result, backend='qiskit')
```

[13]:
```
q_0:  RY(-/2)    X   X   RY(/2)

q_1:

c: 2/
```

4 Amplitude amplification

In this section, we focus on generalizing the procedure of oblivious amplitude amplification for the case where H is not a unitary operator itself. This procedure will boost the success probability of the algorithm used, since it aims to increase the probability of measuring the ancilla in the zero-state to as close to 1 as possible, in the best case.

To be more precise with what is meant by "best case", recall that the success probability of the LCU operator, W, was shown to be $\frac{1}{s_0^2} = \frac{||H|\psi\rangle||^2}{s^2}$. The procedure of amplitude amplification detailed below aims to boost the value of s to 1, so in the case that $||H|\psi\rangle||^2 = 1$, we have that the success probability is $\frac{1}{s_0^2} \approx 1$.

4.1 Theoretical derivations

Firstly, define the k-dimensional ancillary reflection $R = I_k - 2P_0$ about the zero-state of the ancillary register. Note that this operation is called a reflection because of the way that it operates with respect to elements in the zero state in the ancillary register. For instance, consider the $|j\rangle$ for the ancillary register where $j \neq 0$ and define the state $|\phi\rangle = |j\rangle|\psi\rangle$. Then, we have that:

$$\begin{aligned} R|\phi\rangle &= (I_k - 2P_0)|\phi\rangle \\ &= I_k|\phi\rangle - 2P_0|\phi\rangle \\ &= |\phi\rangle - 0 \\ &= |\phi\rangle \end{aligned}$$

However, now consider the state $|0\rangle$ for the ancillary register, and consider $|\phi\rangle = |0\rangle|\psi\rangle$. Then, we have that:

$$\begin{aligned} R|\phi\rangle &= (I_k - 2P_0)|\phi\rangle \\ &= I_k|\phi\rangle - 2P_0|\phi\rangle \\ &= |\phi\rangle - 2|\phi\rangle \\ &= -|\phi\rangle \end{aligned}$$

Next, define the amplitude amplification operator $A = -WRW^\dagger R$.

Recall the following facts. Firstly, note that $P_0^2 = P_0$ since P_0 is a projection operator and that $P_0|0\rangle|\psi\rangle = |0\rangle|\psi\rangle$. Also note that, by definition, W is a unitary operator. Moreover, recall our assumption that H is Hermitian. Finally, by construction, we have that:

$$P_0 W \propto \frac{1}{s}(|0\rangle\langle 0| \otimes H)$$

Now, we shall look at the result of applying a single step for the amplitude amplification procedure.

For this derivation, define the angle α such that $\frac{1}{s} = \sin\alpha$. We have:

$$\begin{aligned}
AW\left|0\right\rangle\left|\psi\right\rangle &= A\left(\frac{1}{s}\left|0\right\rangle H\left|\psi\right\rangle + \sqrt{1-\frac{1}{s^2}}\left|\phi\right\rangle\right) \\
&= A\left(\sin\alpha\left|0\right\rangle H\left|\psi\right\rangle + \cos\alpha\left|\phi\right\rangle\right) \\
&= WRW^\dagger R\left(\sin\alpha\left|0\right\rangle H\left|\psi\right\rangle + \cos\alpha\left|\phi\right\rangle\right) \\
&= WRW^\dagger\left(-\sin\alpha\left|0\right\rangle H\left|\psi\right\rangle + \cos\alpha\left|\phi\right\rangle\right) \\
&= ... \\
&= \sin(3\alpha)\left|0\right\rangle H\left|\psi\right\rangle + \cos(3\alpha)\left|\Phi\right\rangle \\
&= \left(\frac{3}{s}-\frac{4}{s^3}\right)\left|0\right\rangle H\left|\psi\right\rangle + \left(4\sqrt{\left(1-\frac{1}{s^2}\right)^3} - 3\sqrt{1-\frac{1}{s^2}}\right)\left|\Phi\right\rangle
\end{aligned}$$

Therefore, we observe that:

$$P_0 AW\left|0\right\rangle\left|\psi\right\rangle \propto \left(\frac{3}{s}-\frac{4}{s^3}\right)\left|0\right\rangle H\left|\psi\right\rangle$$

Note that, for this amplitude amplification step, there may be some angles α, or equivalently, some values of s, for which the probability of measuring the ancilla in the zero-state remains unchanged. We can find out what these values of s are by simply equating the probabilities as follows:

$$\begin{aligned}
\left(\frac{3}{s}-\frac{4}{s^3}\right)^2 &= \frac{1}{s^2} \\
s^4 - 3s^2 + 2 &= 0
\end{aligned}$$

Since we assumed that $s > 0$, this implies that $s = 1$ or $s = \sqrt{2}$. Note that $s = 1$ is a trivial solution, as in this case, we do not need to apply the amplitude amplification procedure at all.

Now, we handle the $s = \sqrt{2}$ case by simply adding another qubit to the ancilla register, which stays zero throughout the entire process. This, in effect, is equivalent to manually lowering the initial success probability to some other value so that we may then proceed with amplitude amplification as usual.

After applying A for a total of p times, we note that the resulting state is as follows:

$$A^p W\left|0\right\rangle\left|\psi\right\rangle = \sin((2p+1)\alpha)\left|0\right\rangle\left|\psi\right\rangle + \cos((2p+1)\alpha)\left|\Phi\right\rangle$$

The proof of this claim is presented in the Appendix.

Thus, to get the success probability close to 1, we need a value of p such that $\sin^2((2p+1)\alpha) = 1$ and so, we define:

$$p = \left\lfloor \frac{1}{2}\left(\frac{\pi/2}{\arcsin(1/s)} - 1\right)\right\rfloor$$

4.2 Implementing amplitude amplification

We simply translate the definitions from the previous section into Python as follows. Firstly, we define a function to return the value p, which denotes the number of times we need to apply the amplitude amplification operator, A, to get the probability of measuring the ancilla in the $|0\rangle$ state as close to 1 as possible.

[14]:
```
def _num_iter(unitaries: list[tuple[float, tq.QCircuit]]) -> int:
    """Return the number of times to apply the amplitude amplificiation to␣
↪maximize
    success probability"""
    s = sum(pair[0] for pair in unitaries)
    alpha = arcsin(1 / s)
    frac = (pi / 2) / alpha
    return floor(0.5 * (frac - 1))
```

Next, we can define the reflection operator, R, in Python as follows:

[2]:
```
def reflect_operator(state_qubits, ancilla) -> tq.QCircuit:
    """
    Return the reflection operator R = (I - 2P) \\otimes I_N,
    where:
        - I is the identity operator over the ancilla,
        - P is the projector onto the 0 state for the ancilla,
        - I_N is the identity operator over the state register

    """
    if isinstance(state_qubits, list) and not isinstance(ancilla, list):
        qubits = list(set(state_qubits + [ancilla]))
    else:
        qubits = list(set([state_qubits] + [ancilla]))

    z_gate, cz_gate = tq.gates.Z(target=qubits), tq.gates.Z(control=ancilla,␣
↪target=state_qubits)

    return z_gate + cz_gate
```

Finally, we can define the amplification operator, A, as well as the entire amplitude amplification procedure as follows:

[16]:
```
def amp_amp_op(walk_op: tq.QCircuit, ancilla) -> tq.QCircuit:
    """Return W R W.dagger() R,
     where R is the reflect operator returned by the function␣
↪reflect_operator"""
    anc_qubits = ancilla if isinstance(ancilla, list) else [ancilla]
    state_qubits = [qubit for qubit in walk_op.qubits if qubit not in␣
↪anc_qubits]

    reflect = reflect_operator(state_qubits=state_qubits, ancilla=ancilla)

    return reflect + walk_op.dagger() + reflect + walk_op
```

```
def amp_amp(unitaries: list[tuple[float, tq.QCircuit]], walk_op: tq.QCircuit,␣
↪ancilla) \
        -> tq.QCircuit:
    """Amplitude amplification procedure obtained by repeating the amplitude␣
↪amplification
    step for a total of s times where s is the result of function _num_iter()
    """
    amplification_operator = amp_amp_op(walk_op, ancilla)
    s = _num_iter(unitaries)

    sum_of_steps = tq.QCircuit()
    for _ in range(s):
        sum_of_steps += amplification_operator

    return walk_op + sum_of_steps
```

4.3 Further notes on implementing amplitude amplification

4.3.1 Reusing ancillary qubits

Note that, in our implementation, we have reused the qubits in the ancilla for all the rounds of amplitude amplification. While this is foolproof in the case where H is unitary, it may not guarantee functionality in other cases since the ancillary qubits may still not be returned to the $|0\rangle$ state at all times. To resolve this issue, we recommend altering the definitions of functions to be able to pass in a list of lists for the ancilla and implementing a qubit mapping similar to the one demonstrated in Report 4 to completely guarantee the success of the algorithm.

4.3.2 Stationary angles

As explained earlier, in some cases, the amplitude amplification does not result in any change in the state. Handling such cases is explored in greater depth in Report 3.

5 References

- Berry, D. W., Childs, A. M., Cleve, R., Kothari, R., Somma, R. D. (2014). Simulating Hamiltonian dynamics with a truncated Taylor series. arXiv:1412.4687
- Harrow, A. W., Hassidim, A., Lloyd, S. (2009). Quantum algorithm for solving linear systems of equations. arXiv:0811.3171