**Quantum Phase Estimation**

The goal of quantum phase estimation aims to calculate the rotation around the Z axis on the Bloch sphere or (Phase) induced by applying a unitary gate on a qubit.

In short: it will estimate the change in phase from an unknown unitary gate.

This algorithm uses the inverse of the Quantum Fourier Transform (QFT†) as a building block for the quantum phase estimation algorithm.

**Properties:**

1. When estimating the phase of a unitary gate, if QPE is ran with different amounts of estimation qubits, the QPE run with more estimation qubits will be closer to the expected value.
2. When estimating the phase of a unitary gate, if QPE is ran with different amounts of estimation qubits, the QPE run with less estimation qubits will be further to the expected value.
3. The majority of results generated by the algorithm should be for the qubit string that generates the closest phase to the one entered into the algorithm.
4. The same bitstring should be generated (100%) with certainty when phase entered is an exact multiple of where N = amount of estimation qubits and N > 0. (Assuming the simulator is without noise)

**Similar implementations of the Quantum Phase Estimation algorithm**

|  |  |  |
| --- | --- | --- |
| Qiskit | CirQ | Q# |
|  |  |  |
| To begin, we write a method that performs the inverse QFT algorithm. Moving from the Fourier basis to the computational basis to measure the qubit states.  In the controlled phase gate, we use -math.pi, unlike the normal QFT algorithm, as we are trying to apply the inverse.  The n parameter is the amount of estimation qubits that are used for the phase estimation algorithm. | The inverse QFT algorithm in CirQ, is using the same structure for the code, with some slightly different syntax to set up the circuit.  We require both the qubit object array, as well as the quantum circuit variable.  Again, the controlled phase gate does not exist in CirQ, so we use a CZ gate and replace -pi, with -1. | In Q# we use “operations” as functions, as mentioned in the QFT algorithm, Q# does not use the concept of quantum circuits.  The main structure of the functions on all three is the same, though the syntax in Q# is quite different. |
|  |  |  |
| We require the same parameters in all three implementations to perform the QPE algorithm.  We use the number of qubits to use for estimation, as well as the angle to estimate. | 🡨 | 🡨 |
|  |  |  |
| In the main body of the function, we create the quantum circuit object, passing in the amount of estimation qubits +1 (for the lower register, that we apply rotations on).  For the second parameter, we pass in the integer containing only the amount of estimation qubits. The classical register determines how many qubits can be measured at the end, and we are only interested in the values of the estimation qubits. | We define an array linequbit objects, of length estimation qubits + 1.  We also need to create the circuit object. | In Q# we the “use” keyword is used to allocate qubit registers, and again we allocate estimation qubits + 1. |
|  |  |  |
| We apply the Hadamard gate to each estimation qubit location in the circuit. | We apply the Hadamard gate to the qubit object in the array, then add the gate to the circuit. | We apply the Hadamard gate to all estimation qubits in the qubit array. |
|  |  |  |
| Apply the X gate to the lower register qubit location in the circuit.  (By lower register, I mean the last qubit in the circuit/array. We apply rotations to it, but are not interested in its final value) | Apply the X gate to the lower register qubit, then add the gate to the circuit. | Apply the X gate to the lower register qubit. |
|  |  |  |
| Apply many Z rotations using controlled phase gates, depending on the entered angle to estimate. | Same, but using CirQ syntax, and using a CZ instead of CP gate. | We use the Controlled R1 gate, which is the same as the CP gate like Qiskit.  We use a ‘**mutable’** as the counter variable, and use ‘**set’** to reassign the value. |
|  |  |  |
| Call the premade QFT dagger circuit to move from Fourier basis to computational basis, so we can perform measurements | 🡨 | 🡨 |
|  |  |  |
| Apply measurement gates on all estimation qubit locations in the circuit, and map the qubit register to the classical register for each estimation qubits. | Apply the measurement gate to each estimation qubit and add it to the circuit. Apply the key ‘m’ so that we can easily select the results later. | Apply the measurement gate to each estimation qubit in the qubit array.  It is a requirement to reset the value to the 0 state on all qubits after performing measurement, so we call ResetAll() |
|  |  |  |
| This code just sets up the run, runs it 4096 times, accumulates the results, and formats them | 🡨 | For Q# we use a host file in C# to run the Q# QPE algorithm. This is because the Q# language is very limited and hard to work with.  We essentially run the Q# version of the algorithm, accumulate the results for each run, format them, and apply the algorithm (below) to convert the results to an angle. |
|  |  |  |
| This is the main algorithm to turn the results of estimation qubits back into an angle/ phase of rotation.  We convert the qubit values as binary to an integer value, then divide it by where N is the amount of estimation qubits. | 🡨 similar code can be found on both CirQ and Q#, but its too long to fit in a good screenshot. | 🡨 |