Modeling a Discrete Pseudo-Gaussian as a Method of Benchmarking Noisy Quantum Chips

Aurelia Brook

Professor Dries Sels

Computational Physics: PHYS-GA 2000

Final Project

December 14, 2021

Contents

1 Abstract

While advancements in quantum algorithms have been significant, there is still much to be done in terms of quantum computing hardware. There are few practical uses of current NISQ (Noisy Intermediate-Scale Quantum) era quantum computer chips. An errorless or "noiseless" quantum computer, however, could solve classically intractable problems which span from protein-folding to code-cracking. In order to make use of our current technology, one must be able to identify error types and quantify noise on quantum hardware. This can be done through the use of benchmarking algorithms, where a probability distribution is modeled on a quantum chip and then compared to a noiseless simulation. For this project, a symmetrized exponential probability distribution was modeled as a pseudo-Gaussian distribution with Qiskit[3] on IBM's quantum chips to make observations on the noise levels and dominant error types found on NISQ era processors.

2 Preparing a Pseudo-Gaussian Wavefunction

2.1 The Klco-Savage State Preparation Scheme

Given some wavefunction of the form $\Psi(x) = e^{\alpha x}$, one could prepare this as an exponential probability distribution with positive and real amplitudes[2]:

$$|\Psi\rangle = \frac{1}{\sqrt{(\sum_{y=0}^{2^n-1} \Psi^2(y))}} \sum_{y=0}^{2^n-1} \Psi(x)\,|x\rangle \tag{1}$$

Here, n is the number of qubits, and α is an exponential growth factor. This can be performed by applying a series of rotation matrices, represented by the R_y gate:

$$R_y(\theta) = e^{-i\theta\sigma_y} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2}$$

$$\theta_\ell = \arctan e^{2^\ell \alpha} \tag{3}$$

The ℓ index on the angle corresponds to each of the qubits that will be included in the implementation of this scheme. The series of rotation gates applied creates the exponential probability distribution. Symmetrizing the exponential requires reversing the binary interpretation of the second half of the Hilbert space, creating a peak in the middle of the probability distribution. This is done by adding a series of two-qubit CNOT gates, applied to the 0^{th} qubit and the ℓ qubit, then the 0^{th} and $\ell - 1$ up until the 0^{th} and 1^{st} qubit.

2.2 Implementation

Due to the low circuit depth of the scheme proposed in the paper "Minimally-Entangled State Preparation of Localized Wavefunctions on Quantum Computers", it was easiest to simply prepare the circuit using IBM's Quantum Composer and make adjustments from there to properly implement the pseudo-Gaussian distribution. The circuit used is here:

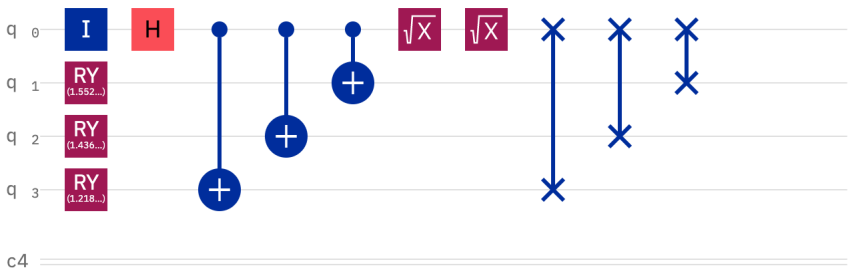

Figure 1: Quantum circuit used in the experiment.

The circuit used in the experiment is slightly different from the one posed in the paper, as there were some issues implementing the given circuit. When put into code, the scheme posed in the paper formed a symmetric exponential function with the exponential rise at the ends of the list of possible computational basis states. Solutions to this include either implementing a Gaussian reshuffling method prior to measurement, or applying a series of swap gates. As seen in Figure 1, the latter was chosen.

3 Results and Discussion

The implementation of the pseudo-Gaussian was done with IBM Composer and Quantum Lab. When the circuit was formed, the following probability distribution was produced:

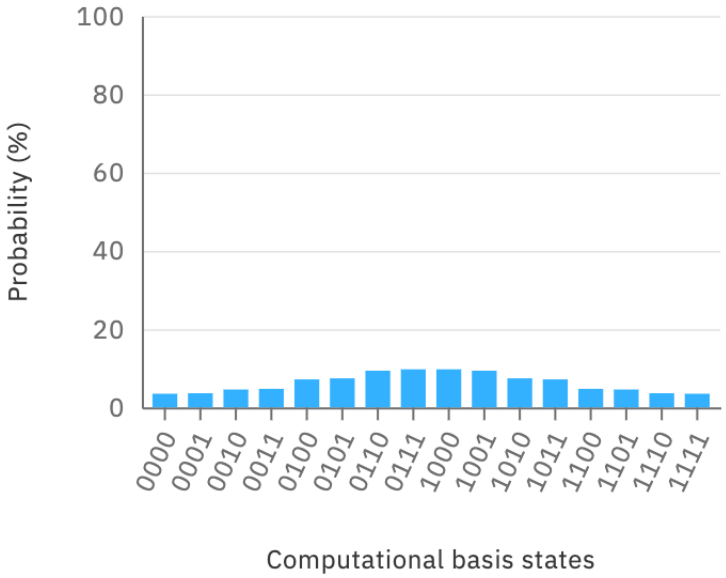

Figure 2: Probabilities of each computational basis state prior to run on a quantum chip.

This was modeled on the "Q-Sphere", a version of the Bloch sphere which visualizes a vector representation of each possible computational basis state.

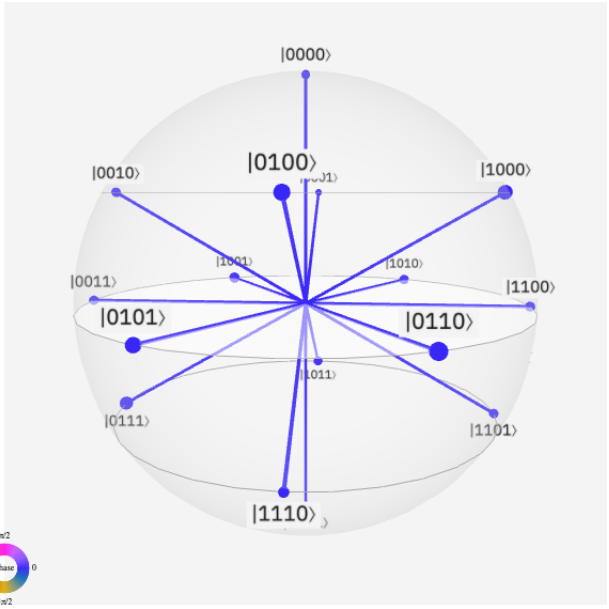

Figure 3: A visual representation of each possible 4-qubit combination.

3.1 IBMQ Qinto

The code listed in the Appendix was pushed on to IBM's Qinto chip. When this is done, the circuit composed for the probability distribution is transpiled to match the topology of the Qinto chip.

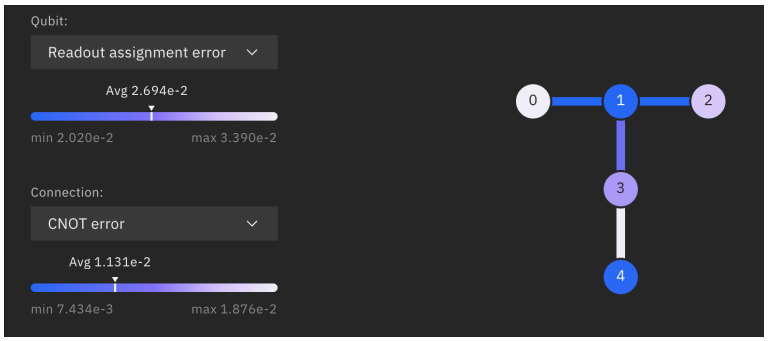

Figure 4: Details on the makeup of the IBMQ Qinto chip.

Figure 5: The same circuit as Figure 1 after undergoing transpilation.

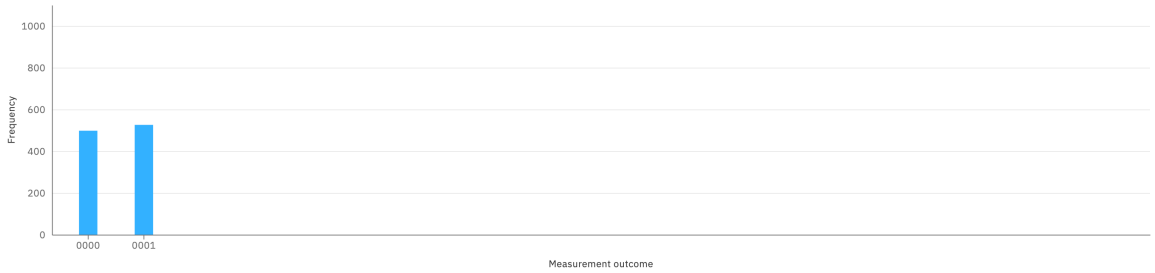

Figure 6: The probabilities of wavefunction collapse onto either the $|0\rangle$ or $|1\rangle$ state after measurement on Qinto.

3.2 QASM Simulator

The same code for the state preparation algorithm was sent to be run on IBM's Qiskit QASM noise simulator, which generates a noise model based on error rate data generated from IBM's quantum processors. Selecting common error sources such as depolarizing error allow for the ability to spot trends in post-measurement data due to specific error types.

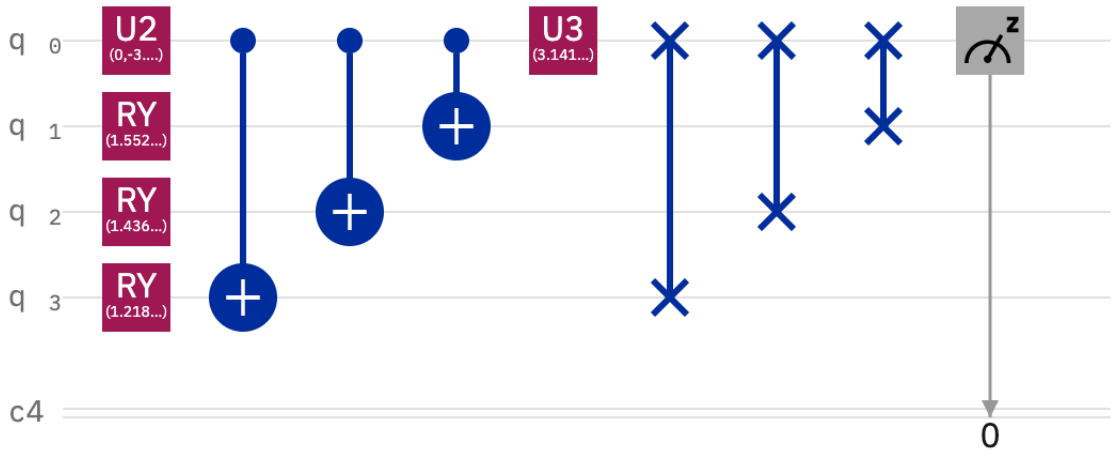

Figure 7: The same circuit as Figure 1 after undergoing transpilation.

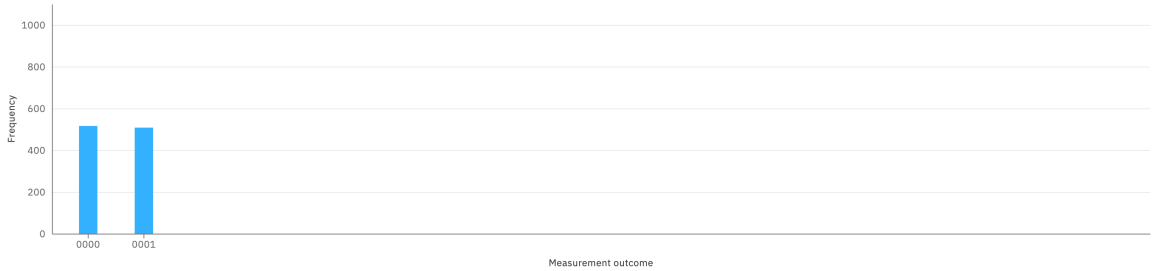

Figure 8: The probabilities of wavefunction collapse onto either the $|0\rangle$ or $|1\rangle$ state after measurement with QASM.

3.3 Comparison to Other Benchmarking Methods

Other benchmarking methods include creating a 1D Gaussian distribution out of the probability amplitudes and randomized benchmarking. Using the Gaussian distribution as a benchmarking method works well as skews to the $|0\rangle$ state are very clearly visible, and its likeness to a uniform distribution can be easily quantified. However, since the pseudo-Gaussian is symmetrized at the center and can be prepared without entanglement, the amount of noise that will affect the system is quite minimal. Whatever noise that appears within the binary interpretation of the first half of the Hilbert space will be reflected into the second half. Because IBM's systems only allows use of chips with low numbers of possible qubits, the number of qubit combinations is quite small. Therefore, any noise that appears tends to dampen the appearance of the probability distribution, as the probabilities of sampling each computational basis state must still sum to one.

Due to the implementation of the pseudo-Gaussian through IBM's quantum computing lab services, the scheme could not be implemented without the inclusion of Z-measurement where the wavefunction must collapse onto either the $|0\rangle$ or $|1\rangle$ state. However, this made the scheme much easier to implement and runs much faster than an alternative method of recreating the circuit from scratch and asking the chip to compute the probability of sampling each individual qubit combination. Implementation of randomized sampling as a benchmarking method is quite simple with Qiskit Aer[1] as there exists functions that will execute this for you. Potential next steps for this project would be implementing both a longer way of preparing a wavefunction as a discrete pseudo-Gaussian probability distribution, and comparing to randomized benchmarking to generate comparisons of these methods on each available chip. Additionally, executing the same program through an account that has full access to chips with higher quantum volume than Qinto (QV = 16) and higher number of qubits available (Qinto only has 5) would be beneficial.

4 Conclusion

The modeling of a discrete pseudo-Gaussian probability distribution on IBM's quantum chips was successful. As can be seen from the data, the use of QASM with the created program for this project can help identify how certain error types affect measurement readings. It is clear that running on Qinto did not generate much noise, as the result was very close to being 50/50. Depolarizing error and bit reset errors are not prevalent on this chip, as there was no skew towards the $|0\rangle$ state. As such observations were made by using the symmetrized exponential as a tool for benchmarking, this program can then be applied to various problems in helping to generate better noise data for each chip in conjunction with QASM and any method of quantifying noise.

5 References

1. IBM. “Simulators.” Simulators - Qiskit 0.24.1 Documentation, https://qiskit.org/documentation/stable/0.24/tutorials/s

2. Klco, Natalie, and Martin J. Savage. ”Minimally entangled state preparation of localized wave functions on quantum computers.” Physical Review A 102.1 (2020): 012612.

3. The Qiskit Team. “Learn Quantum Computation Using Qiskit.” Learn Quantum Computation Using Qiskit, Data 100 at UC Berkeley, 18 Oct. 2021, https://qiskit.org/textbook/preface.html.

6 Appendix

```
#IMPORTS
from ibm_quantum_widgets import CircuitComposer
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi
from qiskit import Aer, IBMQ, execute
from qiskit.providers.aer import noise
from qiskit.tools.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

provider = IBMQ.load_account()
provider.backends()
backend = provider.get_backend('ibmq_quito')
noise_model = noise.NoiseModel.from_backend(backend)

device = provider.get_backend('ibmq_quito')
properties = device.properties()
coupling_map = device.configuration().coupling_map


#Form Circuit
qreg_q = QuantumRegister(4, 'q')
creg_c = ClassicalRegister(4, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

#Gate Operations
circuit.ry(0.909116063, qreg_q[0])
circuit.ry(1.0255887, qreg_q[1])
circuit.ry(1.21828291, qreg_q[2])
circuit.h(qreg_q[3])
circuit.z(qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[2])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.measure(qreg_q[3], creg_c[3])


editor = CircuitComposer(circuit=circuit)
editor

#TAKEN FROM QISKIT AER EXAMPLE GIVEN BY IBM
# Get the basis gates for the noise model
basis_gates = noise_model.basis_gates

# Select the QasmSimulator from the Aer provider
simulator = Aer.get_backend('qasm_simulator')

# Execute noisy simulation and get counts
result_noise = execute(circuit, simulator,
                       noise_model=noise_model,
                       coupling_map=coupling_map,
                       basis_gates=basis_gates).result()
counts_noise = result_noise.get_counts(circuit)
plot_histogram(counts_noise, title="Counts for 4-qubit GHZ state with depolarizing noise model")
```