Quantum Walks and Monte Carlo Methods for the Quantum Galton Board (QGB) Problem

Jibril Abdullahi, Krystal Maughan, Otene Sunday ThankGod*

Abstract

We present a quantum Galton board implementation, starting with a classical implementation, a *1 and 2-layer* Quantum Galton Box implementation, and then a generalized algorithm. We then modify the function to obtain a different target distribution. Finally, we propose an optimized implementation of this problem for the previous distributions. We demonstrate our solution by computing the distances between the obtained distributions and the target distributions, accounting for stochastic uncertainty. We show a solution that maximizes the accuracy and number of board layers for the Quantum Galton Board problem.

1 The Galton Board Problem

A Galton Box is a simple device invented by Sir Francis Galton that demonstrates how individual random events aggregate into a predictable distribution. At each level, a particle randomly chooses left or right. The final position depends on how many right moves were made, producing a binomial (or approximately normal) distribution. It models how particles randomly fall through pegs, making left/right decisions at each level, ultimately forming a binomial distribution — a foundational concept in both classical and quantum statistics. In a classical Galton Board, one can calculate the output by the combinatorial probabilility:

$$\binom{n}{k} p^k q^{n-k} = \frac{1}{2^n}\binom{n}{k} \tag{1}$$

2 Background for Quantum Methods

2.1 Mid-Circuit Measurement and Reuse (MCMR)

Although the problem does not extensively contain quantum implementations, a few works have explored this topic. In [RSMP21], a quantum algorithm for efficiently simulating Galton machines with exponential quantum advantage over classical algorithms by assuming a polynomial depth in terms of the number of qubits, with applications to option pricing and machine learning. They propose a *repeat-until-success* scheme that only requires a constant-bounded number of repetitions, claiming that their algorithm is resistant to both phase-flip and bit-flip errors. Their algorithm leverages *Mid-Circuit Measurement and Reuse* (*MCMR*) technology such that it is applicable to state-preparation problems, with a specific focus on finance.

2.2 The Universal Simulator

In [CV22] they propose an implementation that is exponentially faster than a classical calculation using an example that can be understood without requiring an understanding of computational complexity. To do this, they use 3 types of quantum gates to demonstrate a straightforward implementation of $O(n^2)$ complexity. In comparison to previous research, they create a quantum circuit which creates a superposition of all possible trajectories. Their design can be extended to a universal statistical simulator by modifying peg arrangements. Their applications are focused on quantum Galton boards for quantum optics, computing, and statistical simulations. In our review of the literature, [CV22] and [RSMP21] were two most relevant works.

*Authors are denoted in alphabetical order

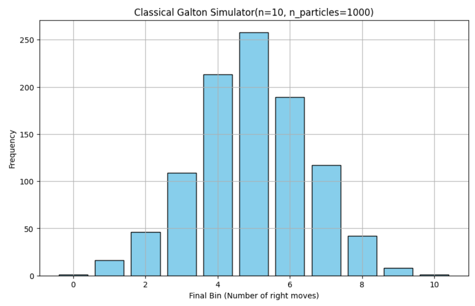

Figure 1: Final Binomial distribution confirming a Gaussian distribution

3 Classical Implementation

The classical simulation for a Galton Box (also known as the Plinko game) uses classical Monte Carlo methods in Python. This implementation is located in [Ote]. Using 10 levels and 1000 particles, we observe a classical implementation that simulates a Binomial distribution.

4 Modelling a Quantum Implementation

4.1 A Quantum Peg

The first goal of the quantum peg construction is that of simulating a superposition equivalent to that of the classical probability distribution function (PDF). Fortunately, there is literature on how this is done. [GR02]. Our implementation is at [Kry]. One of the challenges of quantum implementations detailed in [CV22] was that of noise. We implemented the 1-level board for 2 pegs, and then the 2-level board with 3-pegs. As we scaled the solution from that of 2-level board with 3-pegs to that of an n-level board with n-pegs, we noticed that one contributing factor to noise is circuit depth. Therefore, in our implementation, we aim to minimize the depth of the circuits needed for implementation. Our basic process is as follows:

- **Draw the Ball and Pegs:** We represent the ball and pegs with qubits and initialize the quantum circuits. The number of qubits would depend on the number of layers in the Galton Box needed. For example, for n layers, you would need n qubits to encode the path choices, with potentially additional qubits for auxiliary operations or to represent the final bins.
  - To do this, we initalize a set of qubits to represent the path of the ball. For each pin, we apply a Hadamard ($\mathcal{H}$) gate to a qubit. This places the qubit into a superposition of 0 and 1, representing the equal probability of going left or right at that pin.
  - We then model the pegs as interactions between the qubits.
- **Introduce Quantumness:** As stated above, we introduce superposition via Hadamard gates, allowing the ball to be in multiple paths simultaneously.
  - We use controlled-SWAP or other control gates to simulate the ball interacting with the peg and changing its path based on the state of other qubits. This mimics the left and right decision at each peg.
- **Create the Board:**We then create the circuit object in Qiskit using QuantumCircuit. We add the gates needed, which define the quantum box.
- **Measure Step:** We measure the qubits at the end of the circuit to determine the final position of the quantum ball. We retrieve the counts from the execution result, which shows the frequency of each measured outcome (i.e. which bin the particle landed in, and how many times).
  - We repeat this execution of the circuit multiple times to observe the probabilities of the distribution of the final positions, which represents a classical Galton box's bell-curve distribution. We plot these results in a histogram to visualize the resulting distribution

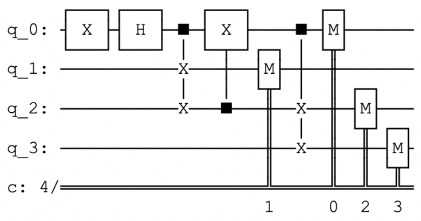

Figure 2: Simple Quantum Galton Board

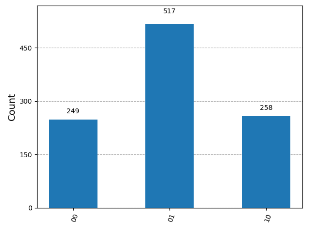

Figure 3: Two-Level Quantum Galton Board

4.2 3-Pegs, 2-levels Quantum Galton Board

We initialized the Hadamard gate and simulated the first peg row by using a control qubit, where our targets are the next positions for the qubit. We similarly simulate the second row and measure and obtain results. This demonstrates a binomial distribution.

4.3 n pegs, n-level Quantum Galton Board: The Hadamard Walk

For our n-level, n-peg board, we used the Hadamard gate in a slightly different construction from simply extending our initial methods. Initially, we did attempt to extend a construction of our 1 and 2 layer board. What we found is that an initial similar implementation of our Quantum n-level board resulted in a noisy result. Instead, we used a simulation in which we counted the bit string of the number of 1s in our distribution, which most closely simulated a binomial distribution and was scalable without being overly noisy. We call this the *Hadamard Walk* construction.

4.4 Different Rotation of Distribution via Biasing

A paper by Aaronson and Arkhipov describes a quantum bosonic analog of Galton board's board [AA10], which gives some insight on the process of biasing a distribution. For this, we use $R(\theta)$ instead of a Hadamard

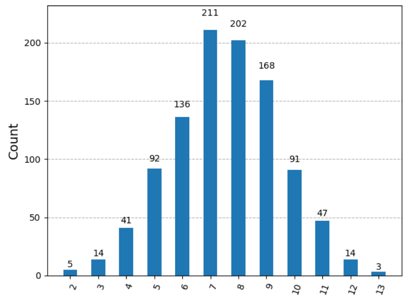

Figure 4: n-Level Quantum Galton Board with Normal Distribution

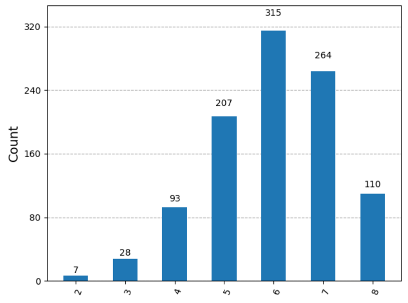

Figure 5: n-Level Quantum Galton Board with Right-Skew using RY(θ) gate

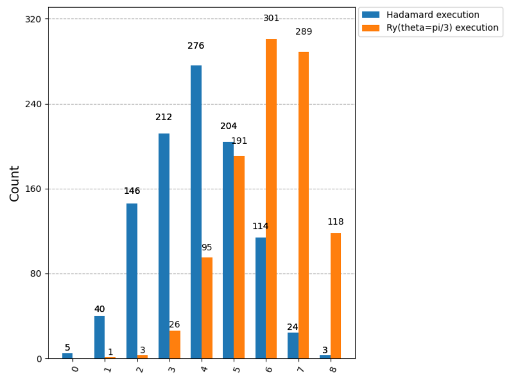

Figure 6: n-Level Biased Quantum Galton Board (orange) with Normal Gaussian distribution (blue)

gate, but the construction is the same as the *Hadamard Walk* construction. For our angle, we used $\theta = \frac{\pi}{3}$, which gives us a right-skewed distribution. In effect, we are using an RY gate to perform a rotation about the Y-axis of the Bloch Sphere by $\frac{\pi}{3}$.

We then plotted the differences between the two distributions on the same plot, which is represented in *Figure 6*.

5 Noise Model from Hardware

As we did not have access to a hardware noise model, we created a custom noise model, which can be found in our github repository with the Quantum Board codebase [Kry], in the Noise Model Backend notebook. Here, we used depolarizing errors and readout errors so that we added specific noise to the circuits, transpiled the circuit, and then ran our model through IBM's *Aer simulator*. Here, we add depolarizing errors to our $\mathcal{H}$ gates. We additionally add readout errors to our measurements.

```
## 2. Add depolarizing error to all H gates
depolarizing_prob = 0.01
error_h = depolarizing_error(depolarizing_prob, 1)
noise_model.add_all_qubit_quantum_error(error_h, ['h'])
```

```
## 3. Add a readout error to measurements
readout_error_matrix = [[0.95, 0.05], [0.03, 0.97]]
readout_error = ReadoutError(readout_error_matrix)
noise_model.add_all_qubit_readout_error(readout_error)
```

We plotted the effect of this noise side by side by looking at the original Gaussian model that we used for our Hadamard walk, and our new custom-noisy model. This result is represented in *Figure 8*.

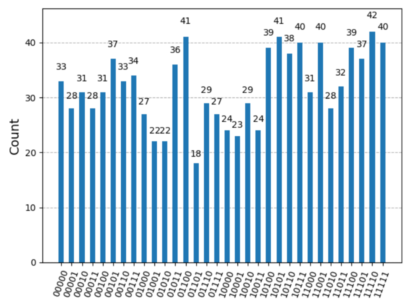

Figure 7: Noise for Custom Back End Model

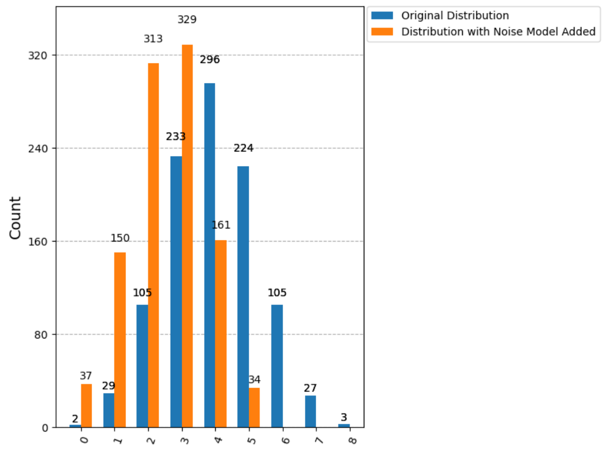

Figure 8: Distribution Comparison of Model with and without Noise Model

5.1 Results of our Custom Noise Model

In *Figure 6*, we can see that the right-skewed model still follows a Gaussian distribution, even though the tail of the distribution is truncated and is not fully represented on our plot. We compared this to the custom-noise model, in which it can be observed that the model starts to slightly deviate from its statistical mean μ, as well as lose its Gaussian Bell-curve shape. With added noise, the Gaussian property of the model may very well begin to deteriorate, as the model begins to look more *laplacian*. However, this observation will need more data to test. Since we were unable to obtain access to IBM's Qiskit Hardware, we could not test the effects of our circuit with a hardware noise model that was not custom simulated. For our custom noise model, we used a mixture of depolarizing and readout errors.

6 Future Work and Summary

As discussed, one of our challenges was running our circuits on actual quantum hardware. We were able to simulate a Quantum Galton Board using a Hadamard Walk on an n-level board, and to demonstrate a biased distribution by rotating about the Y-axis of the Bloch sphere. In the future, we hope to run further experiments on hardware versus quantum simulation and observe noise with a noisy backend versus a custom noisy model.

References

[AA10] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics, 2010.

[CV22] Mark Carney and Ben Varcoe. Universal statistical simulator, 2022.

[GR02] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions, 2002.

[Kry] Krystal. Quantum monte carlo galton box simulation. `https://github.com/kammitama5/quantum-monte-carlo/blob/main/Quantum-Galton-Box_pegs_and_levels.ipynb`.

[Ote] Otene. Quantum walks and monte carlo galton box simulation. `https://github.com/stotene/Quantum-Walks-and-Monte-Carlo-Galton-Box-Simulation-/tree/main`.

[RSMP21] Arthur G. Rattew, Yue Sun, Pierre Minssen, and Marco Pistoia. The efficient preparation of normal distributions in quantum registers. *Quantum*, 5:609, December 2021.