Quantum state preparation for bell-shaped probability distributions using deconvolution methods

Madhav Sharma K.N[1], Camille de Valk[2], Ankur Raina*[3], and Julian van Velzen[4]

[1]Department of Physics, Indian Institute of Science Education and Research Bhopal, India
[1,2,4]Capgemini Quantum Lab
[3]Department of EECS, Indian Institute of Science Education and Research Bhopal, India

Abstract

Quantum systems are a natural choice for generating probability distributions due to the phenomena of quantum measurements. The data that we observe in nature from various physical phenomena can be modelled using quantum circuits. We present a hybrid approach to loading probability distributions by performing deconvolution as a pre-processing step before the quantum circuit. To quantify the closeness of the distribution of outcomes from the hybrid classical-quantum block and the target distribution, we use the Jensen-Shannon distance as the cost function. The chosen cost function is symmetric and allows us to improve the deconvolution step before the use of quantum circuits leading to an overall reduction of the circuit depth. The deconvolution step consists of splitting a bell-shaped probability mass function into smaller probability mass functions. The classical step paves the way for parallel data processing in the quantum hardware that consists of a quantum adder circuit as the penultimate step before measurement. We test the algorithm on IBM Quantum simulators and IBMQ Kolkata, a 27-qubit quantum processor, and validate the hybrid Classical-Quantum algorithm by loading two different distributions of bell shape. We load 7 and 15-element PMF of (i) Standard Normal distribution and (ii) Laplace distribution.

1 Introduction

Traditional methods to model various real-world phenomena use random processes or random variables whose distribution is to be learnt. In classical computing machines or systems, the process of learning the distribution of a random source is termed stochastic modelling. Or, the problem can be alternatively seen as generating a given probability distribution, namely the target distribution. In such cases, it becomes important to learn the parameters of that distribution to generate samples from it easily. Seen from a quantum computing lens, efficient data generation using quantum measurements on qubits can solve problems in many areas, particularly finance. For example, quantum computers can be used for derivative pricing, risk modelling, and portfolio optimization [1]. In the

*ankur@iiserb.ac.in

study of the efficiency of Quantum computers in finance, there exists an important algorithm called Quantum amplitude estimation (QAE) [2].

QAE promises quadratic speed-up over the classical Monte Carlo algorithm, which has applications in finance to price an option [3] or to calculate risk metrics like Value at Risk (VaR) and Conditional Value to Risk (CVaR) [4]. Fig. 1 shows the block diagram of the steps taken in the calculation of VaR with amplitude estimation. First, we load the probability distribution of interest into the quantum computer and implement the objective function step. Then, the QAE block estimates the value of the objective function at each value of x (for this, we use bisection search). The second, third and fourth steps are iterative and stop when the desired value of x is reached, which is the required VaR value.

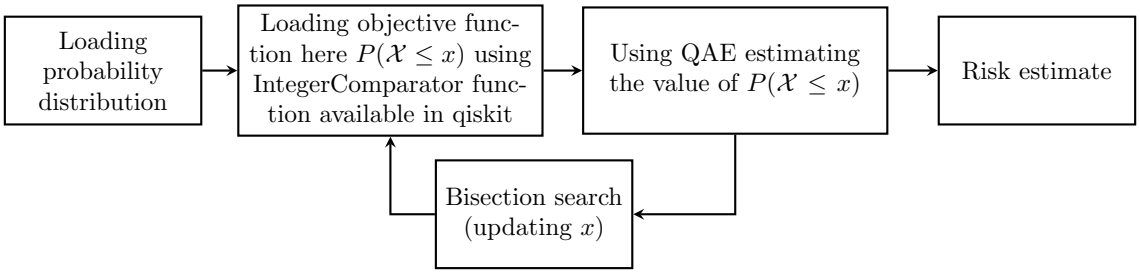


Figure 1: Flowchart representing all the steps involved in the calculation of VaR using a quantum computer.

As explained in Fig. 1, the QAE algorithm can be used to calculate the risk metric VaR, which can be extended to calculate CVaR [4]. Apart from finance and risk management, the QAE algorithm finds application in all other fields where the Monte Carlo algorithm is used for calculation. However, the success of the QAE algorithm depends on the fact that an algorithm with very low circuit depth can prepare a given quantum register in a given target probability distribution.

The first step to using the QAE algorithm is to prepare a given quantum register in a probability distribution specific to the problem of interest, which we refer to as the loading problem. Suppose the probability distribution of interest comes under the family of log-concave probability distributions. We can use the well-known Grover-Rudolph (GR) [5] state preparation method to load the probability distribution. However, the GR state preparation method uses many multi-controlled single-target Rotational Y (R_y) gates. Implementing these gates on hardware requires many small gates that eventually increase circuit depth. If the probability distribution does not come under the log-concave probability distributions, we require an exponential number of gates to load the distribution into the quantum register [6, 7].

In this paper, we discuss a new approach inspired by the principle of deconvolution of a discrete-time sequence into two discrete-time sequences as part of the classical pre-processing step. Our work uses a classical pre-processing step of deconvolution of Probability Mass Function (PMF), which decreases the circuit depth by using more qubits and is compatible with any state preparation method. This method is more compatible with today's Noisy Intermediate-Scale Quantum computer (NISQ) because it is envisaged to have more qubits but a lower operating time.

The structure of this paper is as follows. We discuss the motivation for using deconvolution in Section 2. Section 3.1 discusses how we intend to deconvolve a given discrete probability mass function. Section 4 discusses the experiments and their results. Particularly, we look at the circuit depth required to load the Gaussian and Laplace distribution using our method compared to GR's state preparation. This is followed by conclusions in Section 5.

1.1 Notation

1. Random variables by $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$
2. Pauli gates by X, Y, Z
3. Probability mass functions by $\boldsymbol{P}, \boldsymbol{R}, \boldsymbol{q}$
4. Number of qubits in a quantum register $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{n}$.
5. $\lceil\ \rceil$ represents the ceiling function.
6. $\lfloor\ \rfloor$ represent the floor function.

2 Hybrid classical quantum algorithm

Recently, there has been a lot of activity in the quantum computing community to solve the data loading problem using techniques from other fields like Tensor Networks [8]. A divide-and-conquer approach to the problem was proposed by Araujo *et al.*, providing an exponential time advantage with a quantum circuit having poly-logarithmic circuit depth [9]. But the algorithm scales in space (no.of qubits) as $O(N)$, where N is the dimension of the state vector in which the quantum state should be prepared. In this work, we try to solve this problem for bell-shaped distribution important to risk management using concepts like convolution from signal processing. An important algorithm that depends on the efficient loading of an arbitrary probability distribution is quantum amplitude estimation (QAE). We explain the QAE algorithm in more detail and stress its importance The complexity of creating an arbitrary quantum state is exponential in the number of qubits. However, there exists a trade-off between the number of qubits required for loading and time in terms of circuit depth.

QAE provides quadratic speed-up over classical systems that use Monte Carlo simulations [7]. This can lead to promising solutions when the loading problem is efficiently solved. Our approach uses a hybrid scheme consisting of a classical pre-processing step followed by a quantum circuit. In this way, we attempt to integrate classical and quantum computers. We use the classical constraint optimization algorithm, namely the trust region method, to deconvolve the PMF and design a quantum circuit for loading probability distribution that uses more qubits but fewer gates. The classical pre-processing step runs iteratively until the chosen cost function is minimized. This cost-function-dependent outcome works with the quantum circuit meant for implementing a quantum algorithm. Fig. 2 depicts the hybrid classical-quantum algorithm we propose in this article. The classical step

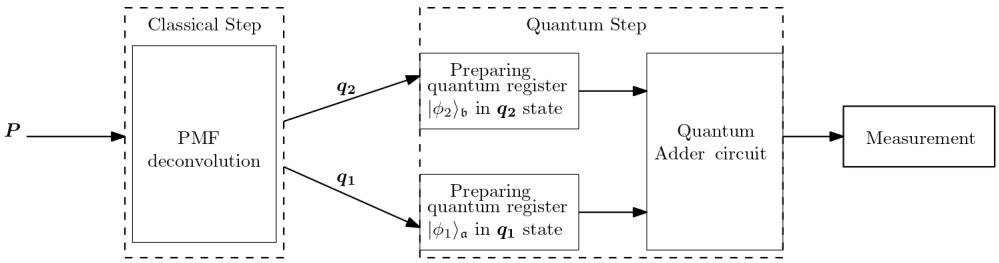


Figure 2: Flowchart representing all the steps involved. Here $\boldsymbol{P}$ (input) is the target probability distribution.

consists of deconvolving the target PMF into smaller PMFs sharing the task of generating a bell-shaped probability distribution. The classical step is explained in detail in Section 3. The quantum step is presented next.

2.1 Preparation of quantum registers

To load a given distribution into a quantum register having $\mathfrak{n}$ qubits, we have to discretize the probability distribution into $2^{\mathfrak{n}}$ regions. This will produce a Probability Mass function (PMF) having $2^{\mathfrak{n}}$ entries, and in terms of quantum computing, it will correspond to a state vector of a $\mathfrak{n}$ qubit system. In this article, we will concentrate on log-concave probability distributions. The log-concave probability distribution is a family of distributions efficiently integrable using existing classical integration techniques and can be discretized efficiently using a classical computer. Once we have discretized the probability distribution and we have $2^{\mathfrak{n}}$ length state vector $|\psi\rangle$, the next step is to prepare a quantum circuit that will transform the state $|0\rangle_{\mathfrak{n}}$ to $|\psi\rangle_{\mathfrak{n}}$.

In this work, we consider a circuit library consisting of one qubit gate plus the Controlled-Not gate (CX gate) to calculate the circuit depth. Common methods, like the GR method and those based on the GR method, use multi-controlled R_y gates one after another to load a given distribution into a $\mathfrak{n}$ qubit system. Since these multi-controlled R_y gates are applied sequentially, the decomposition of the multi-controlled gates is required for implementation at the hardware level, leading to an increase in the circuit depth. To demonstrate the scaling of circuit depth for different

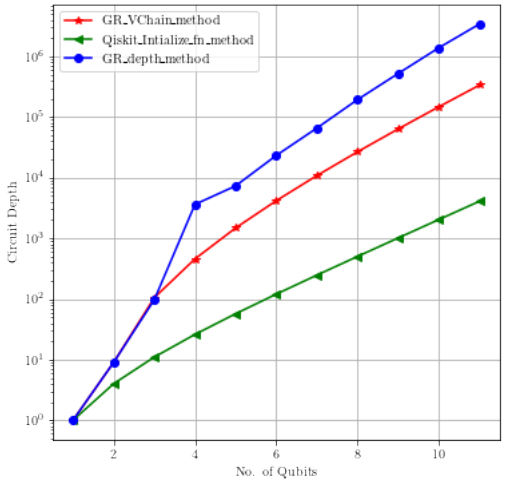


Figure 3: On the X-axis, we plot the number of qubits that we have to prepare in a given superposition state, and on the Y-axis, we plot the circuit depth required to prepare these qubits in a given superposition state. We use the Grover Rudolph state preparation method to prepare the state of the qubits in a given probability distribution. To implement the multi-controlled R_y gate, we use the VChain method depicted in Fig. 14 in Appendix A.3.

state preparation methods, in Fig. 3, we plot a graph depicting how the circuit depth increases as we increase the number of qubits in state preparation for different state preparation methods. The blue, green, and red plots represent the scaling of circuit depth concerning several qubits that are being prepared in a given target probability state using different state preparation methods like the Grover Rudolph (GR) state preparation method, qiskit's built-in `initialize` function [10] and the

Grover Rudolph method with VChain implementation, respectively. The difference between the GR and GR with VChain implementation is how we decompose the multi-controlled R_y gate. In the normal GR implementation, we do not use ancilla qubits and decompose the multi-controlled R_y using single qubit and CNOT gates.

We can also use another approach in which we use ancilla qubits to decompose the multi-controlled R_y, depicted in Fig. 14 called the VChain method explained in the Appendix A.3. In this approach, the Toffoli (CCX) gates are further decomposed into single qubit and CNOT gates for hardware implementation. But still, as expected from the MCMT documentation available on the Qiskit website[11], we see a circuit depth reduction. To prepare the state of one qubit, we need only one R_y gate. Still, to prepare a superposition state of 5 qubits where we use all the 2^5 computational basis available in the 5 qubit Hilbert space, we need a quantum circuit having a circuit depth = $O(2^5)$. Similarly, for preparing a 13 qubit system in a given distribution using the GR method, we need $O(2^{13})$ gates, which is huge for current NISQ computers. Using Qiskit's in-built function called "initialize" instead of the Grover Rudolph method results in a relatively reduced circuit depth. The Qiskit's in-built function `qiskit.extensions.Initialize()` is based on the method described in [12]. However, it's important to note that as the number of qubits increases, the depth of the circuit prepared by the Qiskit initialize function also scales up rapidly. The complexity of the GR method and Initialize function is similar to $O(2^{\mathfrak{n}})$ but differ in a small constant factor.

The current NISQ computers, based on superconducting qubits technology, are expected to scale up to 10k-100k qubits by 2026. At the same time, the decoherence time for the qubits is going to be limited [13]. This inspires us to develop methods that reduce circuit depth and utilize more qubits. Hence, in this work, we have devised a scheme to reduce the circuit depth requirement for bell-shaped distributions like the normal distribution by adding an extra classical step that deconvolves the PMF. To understand deconvolution, we first define what is convolution. Suppose we have two independent random variables $\mathcal{X}$ and $\mathcal{Y}$ and we define another random variable $\mathcal{Z}$ as the sum of the random variables $\mathcal{X}$ and $\mathcal{Y}$. Then, the probability distribution of the random variable $\mathcal{Z}$ is given by the convolution of the probability distributions of random variables $\mathcal{X}$ and $\mathcal{Y}$, respectively [14]. deconvolution is the exact opposite of convolution. Given a probability distribution, can we split it into two probability distributions, each corresponding to an independent random variable? In Section 3.1, we discuss an optimization approach to deconvolve a given PMF of length N into two smaller PMFs of length $\lfloor \frac{N+1}{2} \rfloor$ and $\lceil \frac{N+1}{2} \rceil$. Loading these two smaller distributions requires fewer gates. In Section 4, where we discuss our results, we load PMFs of length 7 and 15. Since the lengths are odd, the value of $\frac{N+1}{2}$ is an integer, and hence for the odd length cases, we have $\mathfrak{a} = \mathfrak{b} = \left\lceil \log_2 \left(\frac{N+1}{2} \right) \right\rceil$.

We load these two in parallel on two different quantum registers, then add the quantum data from these two registers and store the result of the adder in any of the registers using an additional qubit.

We take a $\mathfrak{a}$-qubit quantum register and a $\mathfrak{b}$-qubit quantum register, where $\mathfrak{a} = \left\lceil \log_2 \left(\lfloor \frac{N+1}{2} \rfloor \right) \right\rceil$ and $\mathfrak{b} = \left\lceil \log_2 \left(\lceil \frac{N+1}{2} \rceil \right) \right\rceil$. Here, note that $\mathfrak{b} \geq \mathfrak{a}$ since in $\mathfrak{b}$ formula $\frac{N+1}{2}$ is rounded of by the ceiling function. Then we prepare one quantum register in the state $|\phi_1\rangle_{\mathfrak{a}}$ and the other in the state $|\phi_2\rangle_{\mathfrak{b}}$ where

$$|\phi_1\rangle_{\mathfrak{a}} = \sum_{i=0}^{L(\boldsymbol{q_1})-1} \sqrt{q_{1_i}} \, |i\rangle_{\mathfrak{a}} = A \, |0\rangle_{\mathfrak{a}} \,, \tag{1}$$

$$|\phi_2\rangle_{\mathfrak{b}} = \sum_{j=0}^{L(\boldsymbol{q_2})-1} \sqrt{q_{2_j}} \, |j\rangle_{\mathfrak{b}} = B \, |0\rangle_{\mathfrak{b}} \,, \tag{2}$$

where $L(\boldsymbol{q_1})$ and $L(\boldsymbol{q_1})$ represent the lengths of PMFs $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ respectively. In the above equations 1, 2, the vectors $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ are normalized. Using a quantum adder circuit represented by the unitary U_A, we add two quantum states defined in Equation 1 and 2, then we get a quantum state of length $\mathfrak{n}+1$ as shown in Fig. 4. The construction of Quantum adder circuit U_A is discussed in detail in the next Section 2.2.

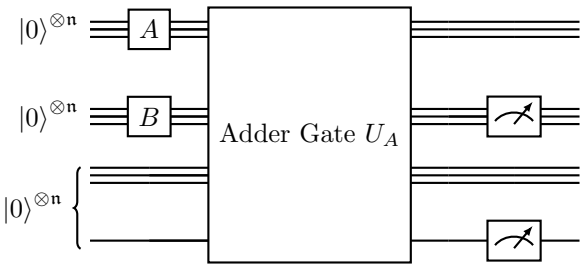


Figure 4: Quantum Circuit Loading probability distribution using the deconvolution method. A gate transforms the state $|0\rangle_{\mathfrak{n}}$ to state $\sum_{i=0}^{L(\boldsymbol{q_1})-1} \sqrt{q_{1_i}}\,|i\rangle_{\mathfrak{n}}$, similarly B gate transforms the state $|0\rangle_{\mathfrak{n}}$ to state $\sum_{j=0}^{L(\boldsymbol{q_2})-1} \sqrt{q_{2_j}}\,|j\rangle_{\mathfrak{n}}$. A and B are constructed using the GR method. The outcome of these two gates is passed into the adder circuit.

Mathematically we can write

$$(A \otimes B \otimes \mathrm{I}_{\mathfrak{n}})\,|0\rangle_{\mathfrak{n}}\,|0\rangle_{\mathfrak{n}}\,|0\rangle_{\mathfrak{n}} = \left(\sum_{i=0}^{L(\boldsymbol{q_1})-1} q_{1_i}\,|i\rangle_{\mathfrak{n}}\right) \otimes \left(\sum_{j=0}^{L(\boldsymbol{q_2})-1} q_{2_j}\,|j\rangle_{\mathfrak{n}}\right) |0\rangle_{\mathfrak{n}} \tag{3}$$

$$U_A \left(\sum_{i=0}^{L(\boldsymbol{q_1})-1} \sum_{j=0}^{L(\boldsymbol{q_2})-1} q_{1_i} q_{2_j}\,|i\rangle_{\mathfrak{n}}\,|j\rangle_{\mathfrak{n}}\,|0\rangle_{\mathfrak{n}}\right) = \sum_{i=0}^{L(\boldsymbol{q_1})-1} q_{1_i}\,|i\rangle_{\mathfrak{n}} \sum_{h=0}^{L(\boldsymbol{q_1})+L(\boldsymbol{q_2})-2} k_h\,|h\rangle_{\mathfrak{n}+1}\,|0\rangle_{\mathfrak{n}-1}\,. \tag{4}$$

The quantum state obtained after the addition has a probability distribution obtained by the convolution of probability distribution $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$, i.e., $\boldsymbol{k} = \boldsymbol{q_1} * \boldsymbol{q_2}$ and $h = i + j$ and is overwritten on the quantum register $|j\rangle_{\mathfrak{n}}$ in equation 4. Therefore, if we break down the probability distribution to be loaded into two smaller probability distributions, then all we need to do is load these two distributions using two separate GR state preparation circuits and add the two resulting states using the quantum adder circuit shown in Fig. 2.

2.2 Quantum adder

We present the design for the Quantum adder circuit used in Fig. Fig. 2 and 4, used for adding two quantum states,

$$|\phi_1\rangle_{\mathfrak{a}}\,|\phi_2\rangle_{\mathfrak{b}}\,|0\rangle_{\mathfrak{b}} \xrightarrow{\text{Adder Circuit}} |\phi_1\rangle_{\mathfrak{a}}\,|\phi_1 \oplus \phi_2\rangle_{\mathfrak{b}+1}\,|\text{ancilla}\rangle_{\mathfrak{b}-1}\,, \tag{5}$$

where, ϕ_1 and ϕ_2 refer to the computational basis state of $\mathfrak{a}$ qubits and $\mathfrak{b}$ respectively. The "$\bigoplus$" in equation 5 represents bit-wise modulo-2 addition. The design for the quantum adder circuit used in this work is inspired by the VBE adder circuit described in [15], but using this algorithm, we can add quantum registers of the same sizes. This implies we can add quantum registers having $\mathfrak{a} = \mathfrak{b}$ and in Algorithm 1, we assume $\mathfrak{a} = \mathfrak{b} = \mathfrak{n}$. Algorithm 1 shows how to construct an adder circuit, and Fig. 5 gives this circuit for the case $\mathfrak{n} = 2$. In this design, we note that we do not reset the ancilla qubits used by the adder circuit. For resetting the ancilla qubit, we need extra gates. For example, for $\mathfrak{a} = \mathfrak{b} = 2$ in Algorithm 1, we need one extra CCX gate for resetting the ancilla qubit. Equation 5 gives the mathematical representation for the action of the quantum adder circuit:

Algorithm 1 Algorithm for designing Quantum adder Circuit

Input: $\mathfrak{n}$ qubit quantum state $|\phi_1\rangle_\mathfrak{n}$ and $|\phi_2\rangle_\mathfrak{n}$. 1 extra qubit $|0\rangle$, which is included with the ancilla qubit.

Require: $\mathfrak{n} - 1$ ancilla qubits

$i \leftarrow 0$
while $i \neq \mathfrak{n}$ **do**
  CCXgate(Control qubit = $(|\phi_1\rangle_\mathfrak{n}[i], |\phi_2\rangle_\mathfrak{n}[i])$,Target = $|\text{ancilla}\rangle_\mathfrak{n}[i]$)
  $i \leftarrow i + 1$
end while
$i \leftarrow 0$
while $i \neq \mathfrak{n}$ **do**
  CXgate(Control qubit = $|\phi_1\rangle_\mathfrak{n}[\mathfrak{n} - i - 1]$,Target = $|\phi_2\rangle_\mathfrak{n}[\mathfrak{n} - i - 1]$)
  $i \leftarrow i + 1$
end while
$i \leftarrow 0$
while $i \neq \mathfrak{n}$ **do**
  CCXgate(Control qubit = $(|\phi_2\rangle_\mathfrak{n}[i]$,$|\text{ancilla}\rangle_\mathfrak{n}$[i-1]),Target = $|\text{ancilla}\rangle_\mathfrak{n}[i]$)
  $i \leftarrow i + 1$
end while
$i \leftarrow 0$
while $i \neq \mathfrak{n}$ **do**
  CXgate(Control qubit = $|\text{ancilla}\rangle_\mathfrak{n}[i]$,Target = $|\phi_2\rangle_\mathfrak{n}[i + 1]$)
  $i \leftarrow i + 1$
end while

The GR state preparation method scales as $\mathcal{O}(2^\mathfrak{n})$, and the complexity of the adder circuit scales linearly with the number of qubits [15]. So, instead of using a big GR state preparation circuit to prepare a state in $|\psi\rangle_{\mathfrak{n}+1}$, we make use of two smaller GR state preparation circuits and make two different quantum states $|\phi_1\rangle_\mathfrak{n}$ and $|\phi_2\rangle_\mathfrak{n}$. Then we pass these two states through the quantum adder circuit to achieve the state $|\psi\rangle_{\mathfrak{n}+1}$. We note in the current context from [14] that addition in basis space is convolution in probability space, i.e., when we perform an addition operation on two quantum states, the probability amplitudes corresponding to each basis state of the two quantum states will convolve with each other. Therefore, we need only $\mathcal{O}(2^{\mathfrak{n}-1})$ gates for the GR part and $\mathcal{O}(\mathfrak{n})$ gates for the adder circuit. In the Algorithm 1 the notation $|\phi_1\rangle_\mathfrak{n}[i]$ refers to the i^{th} qubit of the quantum register $|\phi_1\rangle_\mathfrak{n}$. In the next section, we will discuss two approaches to deconvolution, one in which use an optimization technique called trust region and the second in which we use polynomial factorization.

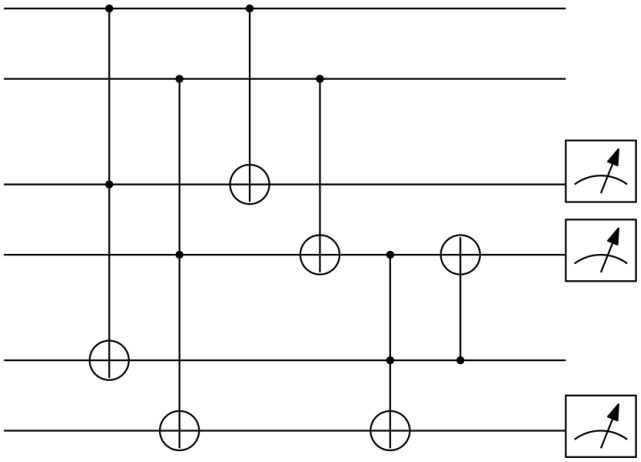


Figure 5: Design for Quantum adder Circuit constructed using Algorithm 1 for $\mathfrak{n} = 2$.

3 Deconvolution of Target PMF

We present the classical part of our hybrid algorithm, which employs classical techniques to accelerate the functioning of quantum circuits. We present two approaches to deconvolve a target PMF that we envisage to generate as the end goal of the quantum hardware.

3.1 Deconvolution using trust region Method

In this section, we describe how to perform deconvolution of a given PMF using the trust region method [16]. This is a constraint optimisation problem and hence can be mathematically formulated as :

$$\min_{\boldsymbol{q_1},\boldsymbol{q_2}} f(\boldsymbol{q_1},\boldsymbol{q_2}) = JS(\boldsymbol{P}||\boldsymbol{Q}) \text{ where } \boldsymbol{Q} = \boldsymbol{q_1} * \boldsymbol{q_2}, \tag{6}$$

where, $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ represents normalized vectors and $*$ represents convolution of $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ [17]. Also, $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ vectors in Equation 6 corresponds to a PMF and hence $0 < q_{1_i} < 1, 0 < q_{2_j} < 1$. Here $\boldsymbol{Q}$ is a PMF of length $L(\boldsymbol{q_1}) + L(\boldsymbol{q_2}) - 1$. $\boldsymbol{P}$ represents the probability mass function that we require to deconvolve and hence is also a vector of length $L(\boldsymbol{q_1}) + L(\boldsymbol{q_2}) - 1$. The function $JS(\boldsymbol{P}||\boldsymbol{Q})$ is called the Jensen Shannon distance and is given by

$$JS(\boldsymbol{P}||\boldsymbol{Q}) = DS(\boldsymbol{P}||\boldsymbol{R}) + DS(\boldsymbol{Q}||\boldsymbol{R}), \tag{7}$$

where

$$\boldsymbol{R} = \frac{1}{2}\left(\boldsymbol{P} + \boldsymbol{Q}\right). \tag{8}$$

Further, $DS(\boldsymbol{P}||\boldsymbol{R})$ can be defined as

$$DS(\boldsymbol{P}||\boldsymbol{Q}) = \sum_{i=0}^{L(\boldsymbol{q_1})+L(\boldsymbol{q_2})-2} P_i \log \frac{P_i}{Q_i}, \tag{9}$$

where P_i refers to the i^{th} element of the PMF $\boldsymbol{P}$ and Q_i refers to the i^{th} element of the PMF $\boldsymbol{Q}$. We use the trust region method to find the optimum $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ such that it minimises the

Jensen–Shannon distance function $f(\boldsymbol{q_1}, \boldsymbol{q_2})$. We chose the Jensen-Shannon function as the cost function due to its symmetric nature. We calculate the gradient of the Jensen–Shannon distance with respect to both the probability vector $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$, which have $L(\boldsymbol{q_1})$ and $L(\boldsymbol{q_2})$ independent variables, respectively, because they both represent a PMF with number of elements equal to $L(\boldsymbol{q_1})$ and $L(\boldsymbol{q_2})$, respectively. Since they are PMFs, they have to satisfy the constraint $\sum_{i=0}^{L(\boldsymbol{q_1})-1} q_{1_i} = 1$ and $\sum_{i=0}^{L(\boldsymbol{q_2})-1} q_{2_i}$, which reduces the number of independent variables from $L(\boldsymbol{q_1})$ to $L(\boldsymbol{q_1}) - 1$ and $L(\boldsymbol{q_2})$ to $L(\boldsymbol{q_2}) - 1$ respectively.

We pass the Jensen–Shannon distance function, the gradient, the Hessian matrix and the method we intend to use to the Python package `scipy.optimize.minimize`. We explain the calculation of the Gradient and Hessian functions in detail in Appendix A.4. We use a standard Trust region optimizer from `scipy` [18] to find the minimum of the Jensen–Shannon Distance function. We explain the sequence in which we perform these steps in Algorithm 2.

Algorithm 2 Algorithm for deconvolution of PMF

Input: P
variable declare $\boldsymbol{q_1}$ = Random Guess.
variable declare $\boldsymbol{q_2}$ = Random Guess.
Calculate the Gradient using 26.
Calculate the Hessian Matrix using the equations from 28 to 31.
Pass the cost function, gradient vector, and Hessen matrix to the `scipy.optimize` package of Python.
Terminate after 1000 iterations.

3.2 Deconvolution Using Polynomial Factoring

We know that any given PMF can be represented using a probability-generating function (PGF) [14]. The PGF for a discrete distribution is given by a polynomial function. We can calculate Probability density function (PDF) $f(x)$ corresponding to a discrete probability distribution PMF $\boldsymbol{P}$ as

$$f(x) = \sum_{i=0}^{L(\boldsymbol{P})-1} P_i x^i. \tag{10}$$

This implies if we have PMF of length $2^{\mathfrak{n}}$, then we have a polynomial of degree $2^{\mathfrak{n}-1}$ corresponding to the PGF of the given PMF. It is also possible to calculate the PMF from the PGF of a distribution as:

$$P_i = \left. \frac{\partial^i f(x)}{\partial^i x} \right|_{x=0}. \tag{11}$$

In this context, we can reformulate the deconvolution of probability distribution as follows. Let $f(x)$ be the polynomial function representing the PGF of a given probability distribution. We need to factorize the function $f(x)$ into polynomials which have non-negative coefficients i.e.,

$$f(x) = \prod_{i=1}^{K} \tilde{f}_i(x), \tag{12}$$

where $\tilde{f}_i(x)$ represents polynomial function of degree $\deg(\tilde{f}_i(x)) < 2^{\mathfrak{n}-1}$ having non negative coefficients such that

$$\sum_{i=1}^{K} \deg\left(\tilde{f}_i(x)\right) = 2^{\mathfrak{n}-1}. \tag{13}$$

Since polynomial $f(x)$ have all positive coefficients, then according to Theorem 3 in [19], $f(x)$ does not have a root in $\mathbb{R}^+$ including zero. This implies that the linear factors of the polynomial will be of the form $(x + c)$ where c is a real positive constant i.e., polynomial $f(x)$ will have linear factors with positive coefficients alone. It will also have quadratic polynomial factors of the form $x^2 - bx + c$ and $x^2 + bx + c$. Now, our task is to get rid of the quadratic polynomial of the form $x^2 - bx + c$ by multiplying it with the appropriate $x^2 + bx + c$ polynomial and forming higher order polynomials having positive coefficients. We have developed an algorithm based on calculating all the roots of the polynomial and taking into account that there can be multiple possible combinations for multiplying polynomials with all positive coefficients that produce the target polynomial. Therefore factorization of the target polynomial into polynomials of smaller degrees having all positive coefficients is not unique.

The algorithm for deconvolution using polynomials is given in Algorithm 3. The Python function FindRoots defined in the Algorithm 3 returns a 2D array of roots where the conjugate pair complex roots are stored as a 1D array. In Algorithm 3, $basket_1$ and $basket_2$ are also 2D arrays of roots, and we randomly shuffle the 1D set of roots stored in the $basket_2$ and not the individual roots inside the 1D array. The numpy.polyroot python function used in Algorithm 3 takes the coefficients of a polynomial as input and returns all the roots of the entered polynomial $f(x)$ [20]. Similarly, the Python function numpy.polyfromroots takes roots as input and returns the coefficients of a monic polynomial with the roots we gave as input [21]. In Appendix A.5, we explain this algorithm in detail using an example. In the next Section 4, we will discuss the results that we obtained by deconvolving the given PMF using the optimization technique discussed in Section 3.1. However, using the approach discussed in this section to speed up the classical process and get more accurate results is possible. Deconvolution performed by this approach is more accurate since algorithms based on an optimization approach sometimes give solutions up to an error bound. The optimization approach is efficient when the parameters we optimize are less, and the cost function is smooth without any local minimum. When the parameter space is too big, the algorithm based on the optimization technique terminates based on the iteration limit we imposed, and the algorithm outputs an approximate result for deconvolution.

4 Discussion of the experiments and results

An extra classical deconvolution layer can reduce the circuit depth at the cost of using more ancillary qubits. To demonstrate this, in 7, we plot circuit depth vs. the number of qubits that will be prepared in a given target probability state with and without using the deconvolution layer. Also, Fig. 7 shows that the state preparation method with the deconvolution layer performs better in terms of the circuit depth than the one without it. Hence, if the deconvolution layer is implemented, then the state preparation method scales as $O(2^{\mathfrak{n}-1} + \mathfrak{n})$, and hence approximately halves the circuit depth. In Fig. 7a, 7b and 7c we plot the circuit depth scaling of Qiskit Initialize function, GR State preparation method and GR state preparation method with VChain implementation Vs the number of qubits whose state is being prepared in a target probability distribution with and without the deconvolution layer. The difference between the GR state preparation method and the GR state preparation method with VChain implementation is in how we implement the multi-controlled R_y

Algorithm 3 Algorithm for deconvolution using polynomial factorization

Input: Target Polynomial $f(x)$.
procedure FINDROOTS(coefficients of the target polynomial$f(x)$)
    Use the `numpy.polyroot` function to find all the roots of the polynomial $f(x)$ given by the coefficients.
    return Roots
end procedure
procedure GROUPROOTS(Roots)
    Group the roots into three: Complex roots with real part positive, Complex roots with real part negative and real roots.
    return Comp_root_real_pos,Comp_root_real_neg,real_root
end procedure
Arrange the Complex root with a positive real part in ascending order with respect to the real term and call it basket_1.
Merge the group of Complex roots with the negative real part and the group of real roots and call it basket_2.
Randomly shuffle the elements of the basket_2.
while length(basket_1) ≥ 0 **do**
    temp_array = $\text{basket}_1[0]$
    temp_boolean = **True**
    while temp_boolean **do**
        random_index = generate random integer between 0 and length(basket_2).
        temp_array = temp_array + basket_2[random_index]
        poly_coeff = `numpy.polynomial.polyfromroots`(temp_array)
        Delete the element basket_2[random_index] from the list basket_2
        if if any(c.real ≤ 0 for c in poly_coeff) **then**
            temp_boolean = **False**
        else
            temp_boolean = **True**
        end if
    end while
    Delete the first element from basket_1.
    basket_2.append(temp_array)
end while
Create an empty array variable and name it list_of_poly_factors = [ ]. This list holds the factors of the target polynomial.
for `i in` basket_2 **do**
    temp = `numpy.polynomial.polyfromroots`(i)
    list_of_poly_factors.append(temp/sum(temp))
end for

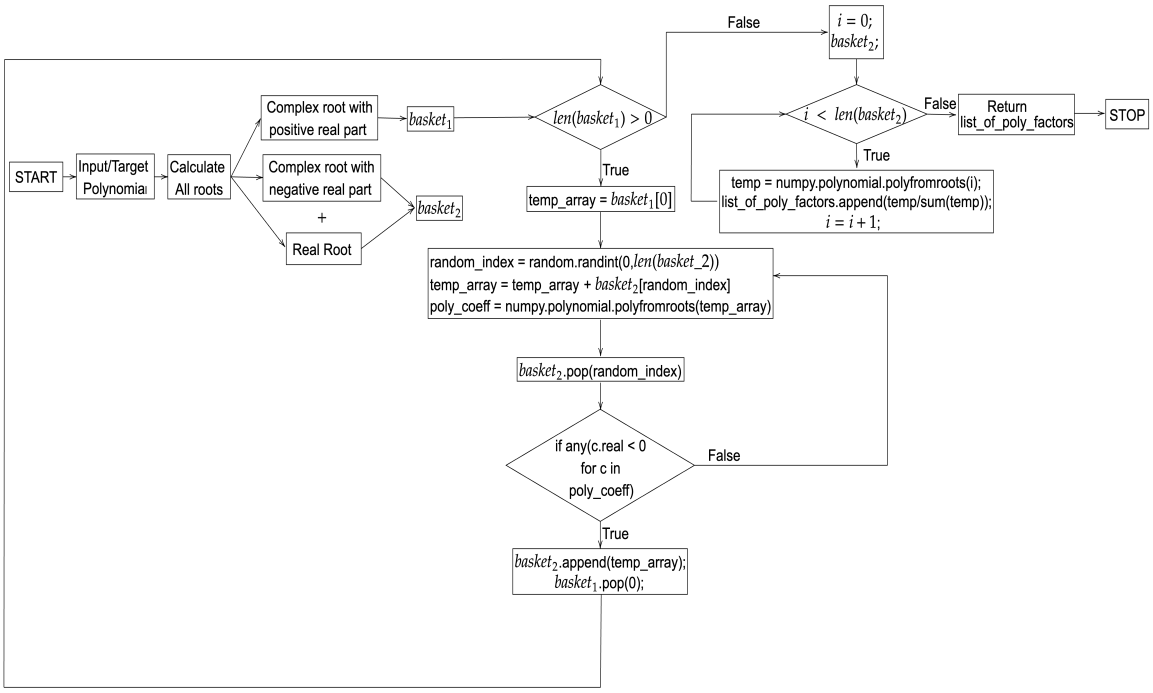


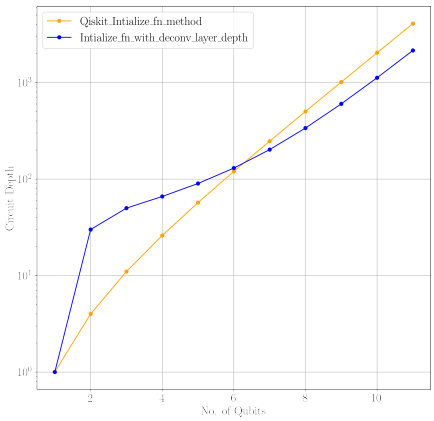
Figure 6: Flowchart for Algorithm 3.

gate. In the first case, we let Qiskit break down the big gates into smaller gates without using any ancilla qubits. In the next case, we explicitly use the feature of multi-control multi-target V-Chains (MCMTVChain) in Qiskit to implement the multi-controlled R_y-gates as shown in Fig. 14. Here we use ancilla qubits in a V-Chain structure to create a multi-controlled single target $R_y(\theta)$-gate.

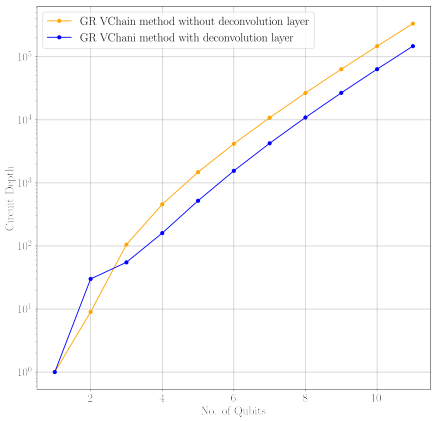
To test and compare the different state preparation methods with and without the deconvolution layer, we load Gaussian and Laplacian distributions on a QASM simulator and real IBM quantum hardware. We also calculate a metric called Quantum Circuit Volume (QCV) [22] for each implementation. The mathematical definition of QCV is given below :

$$\mathrm{QCV}(C) = s(C) \times d(C). \tag{14}$$

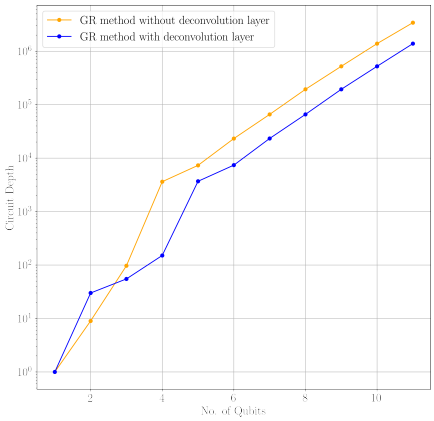
In equation 14, C represents the quantum algorithm whose quantum circuit we are implementing, $s(C)$ represents the number of qubits we used for this implementation and $d(C)$ represents the depth of the quantum circuit implementation. QCV is used to quantify the amount of quantum resources used to implement an algorithm. To test the method's performance on quantum hardware, real and simulated, we measured the quantum circuit 2048 times (shots). Then, the JS distance between the empirical PMF and the input PMF is measured. Here, JS distance is used as a metric to quantify the differences between the measured and target PMF. The circuits are run on IBMQ's noiseless QASM simulator and on *ibmq_kolkata*, which is one of the IBM Falcon Processors. IBMQ Kolkata has 27 qubits and a measured Quantum Volume of $128 = 2^7$. Quantum Volume is different from Quantum Circuit Volume. Quantum volume is a metric that is used to compare different NISQ devices. The largest random circuit of equal depth and width that can be successfully implemented on a quantum computer is termed quantum volume [23].



(a) Qiskit Initialize function

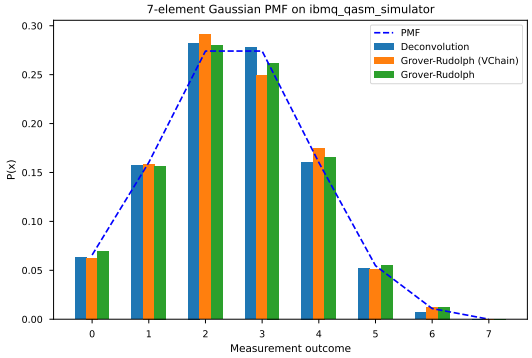


(b) GR State preparation with VChain

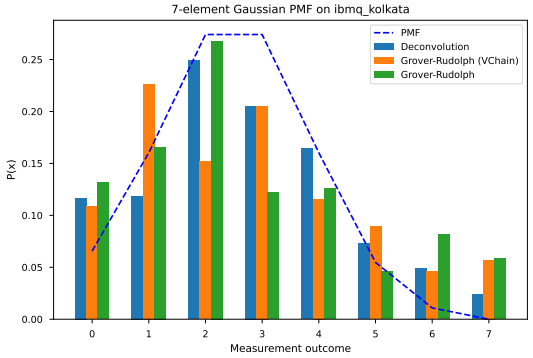


(c) GR State preparation

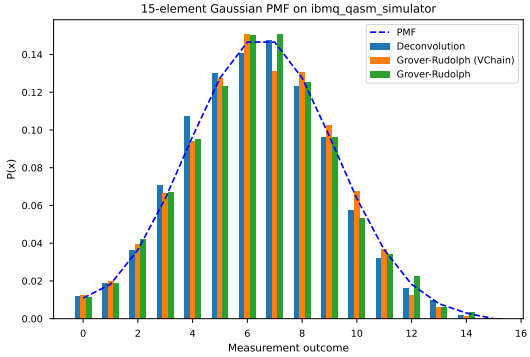
Figure 7: We plot the Log of circuit depth Vs the number of qubit graphs for different state preparation methods. In Fig. (a), we use the Qiskit initialize function to prepare the qubits in a given state, (b) we use the GR state preparation method with VChain implementation for state preparation, and (c) we use the GR state preparation method. In the above figures, the orange line represents state preparation using the state preparation method without the deconvolution step, and the blue line represents state preparation using state preparation methods with an extra classical step of deconvolution.



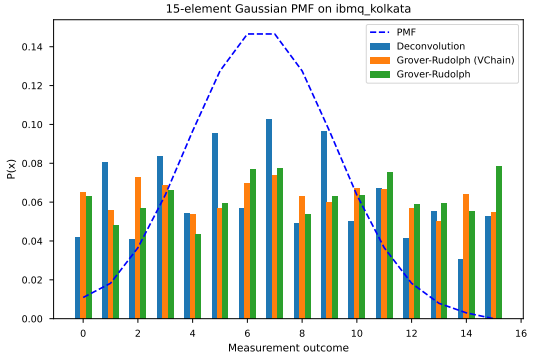
(a) 7-element Gaussian PMF generated using IBMQ QASM simulator (Table 1a).



(b) 7-element Gaussian PMF generated using IBMQ Kolkata (Table 1b).



(c) 15-element Gaussian PMF generated using IBMQ QASM simulator (Table 1c).



(d) 15-element Gaussian PMF generated using IBMQ Kolkata (Table 1d).

Figure 8: Discretized Gaussian PMF on a quantum computer: Fig.8a and Fig. 8c show results on a noiseless simulator. The PMF is very well approximated by all methods, as seen in Table 1. The novel deconvolution method produces the best results (lowest distance) with the fewest gates. The same holds for the results on IBMQ Kolkata (Figures 8b and 8d), but it should be noted that noise limits the performance of the 15-element PMF.

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolution	$\mathbf{8.84 \cdot 10^{-4}}$	**10**	6	**60**
Grover Rudolph (VChain)	$1.93 \cdot 10^{-3}$	30	4	120
Grover Rudolph	$4.99 \cdot 10^{-4}$	33	**3**	99

(a) 7-element Gaussian PMF on IBMQ QASM simulator (Fig. 8a).

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolution	$\mathbf{6.80 \cdot 10^{-2}}$	**54**	6	**324**
Grover Rudolph (VChain)	$1.34 \cdot 10^{-1}$	153	**4**	612
Grover Rudolph	$1.66 \cdot 10^{-1}$	134	**4**	536

(b) 7-element Gaussian PMF on IBMQ Kolkata (Fig. 8b).

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolution	$\mathbf{1.69 \cdot 10^{-3}}$	**42**	11	**462**
Grover Rudolph (VChain)	$2.37 \cdot 10^{-3}$	104	6	624
Grover Rudolph	$1.70 \cdot 10^{-3}$	413	**4**	1652

(c) 15-element Gaussian PMF on IBMQ QASM simulator (Fig. 8c).

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolved	$\mathbf{2.35 \cdot 10^{-1}}$	**298**	11	**3278**
Grover Rudolph (VChain)	$2.94 \cdot 10^{-1}$	823	**6**	4938
Grover Rudolph	$3.13 \cdot 10^{-1}$	796	**6**	4776

(d) 15-element Gaussian PMF on IBMQ Kolkata (Fig. 8d).

Table 1: Results of preparing Gaussian PMFs on the quantum simulators and IBMQ Kolkata. The JS distance is measured between the original discretized PMF and the measurement results. The deconvolution method has the lowest JS distance and circuit depth for all experiments.

Using the Qiskit primitive sampler, we can optimise the circuit and make it more resilient to noise [24]. We used optimisation level 3, which means the circuit is transpiled with 1Q gate optimisation, dynamical decoupling, commutative cancellation, and 2 qubit KAK optimisation. We used resilience level 1, which means that errors associated with readout errors are mitigated with matrix-free measurement mitigation (M3). For more information on these techniques, we refer to qiskit runtime [25]. The results for a Gaussian PMF are depicted in Fig. 8 and Table 1, and the results for Laplacian PMF is shown in Fig. 9 and Table 2. It can be seen that the deconvolution method is well able to approximate the PMF on the simulator. The deconvolution method performs best on the quantum computer compared to the other methods. Due to noise and circuit depth ($\mathcal{O}(10^2)$ for the 15-element PMF), the JS distance is on the order of 10^{-1} for all methods, but lowest for the deconvolution method. From the Table 1 and Fig. 8, it is clear that the deconvolution method has a shallower circuit and the best outcome compared to the GR state preparation method.

To further ascertain the advantage of the deconvolution layer, we also present the results of loading Laplacian distribution using the deconvolution approach to show that the method is not only confined to the standard normal distribution but as described in Section 2 is applicable for all bell state distribution. The Laplacian distribution also comes under the log-concave probability

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolution	$\mathbf{5.88 \cdot 10^{-4}}$	**10**	6	**60**
Grover Rudolph (VChain)	$7.25 \cdot 10^{-4}$	30	4	120
Grover Rudolph	$8.10 \cdot 10^{-4}$	33	**3**	99

(a) 7-element Laplace PMF on IBMQ QASM simulator (Fig. 9a).

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolution	$\mathbf{4.88 \cdot 10^{-2}}$	**54**	6	**324**
Grover Rudolph (VChain)	$9.76 \cdot 10^{-2}$	159	**4**	636
Grover Rudolph	$1.50 \cdot 10^{-1}$	133	**4**	532

(b) 7-element Laplace PMF on IBMQ Kolkata (Fig. 9b).

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolution	$\mathbf{3.00 \cdot 10^{-3}}$	**42**	11	**462**
Grover Rudolph (VChain)	$2.53 \cdot 10^{-3}$	104	6	624
Grover-Rudolph	$2.56 \cdot 10^{-3}$	413	**4**	1652

(c) 15-element Laplace PMF on IBMQ QASM simulator (Fig. 9c).

State preparation method	JS-distance	Circuit Depth	Active qubits	QCV
Deconvolution	$\mathbf{1.75 \cdot 10^{-1}}$	**279**	11	**3069**
Grover-Rudolph (VChain)	$3.40 \cdot 10^{-1}$	815	**6**	4890
Grover-Rudolph	$3.76 \cdot 10^{-1}$	798	**6**	4788

(d) 15-element Laplace PMF on IBMQ Kolkata (Fig. 9d).

Table 2: Results of preparing Laplace PMFs on the quantum simulator and IBMQ Kolkata. The JS distance is measured between the original discretized PMF and the measurement results. The deconvolution method has the lowest JS distance and lowest circuit depth for all experiments.

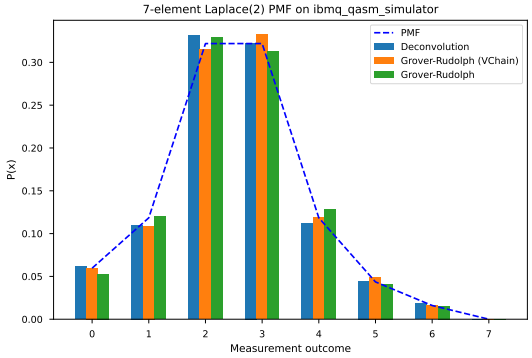
distribution, and the probability density function corresponding to a random variable $\mathcal{X}$ that follows the Laplace distribution is [26]:

$$f(x|\mu, \theta) = \frac{1}{2\theta} e^{-\frac{|x-\mu|}{\theta}}, \tag{15}$$

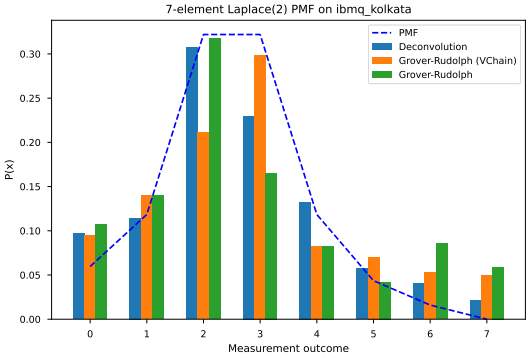
where the parameter μ is the mean value of the random variable $\mathcal{X}$, θ is called the scale parameter. $\mu \in \mathbb{R}$ is also called the location parameter, whereas the scale parameter θ can only take positive real values i.e., $\theta > 0$. We choose the parameters $\theta = 2$ and $\mu = 0$ to validate our findings.

5 Conclusion and future work

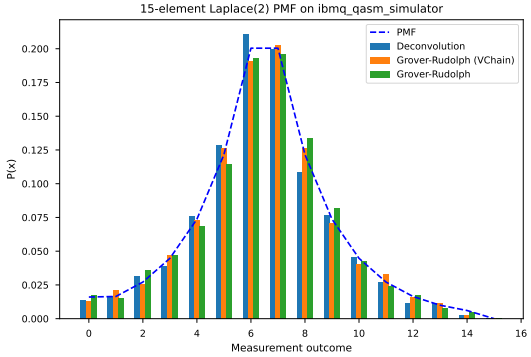
In this article, we discuss a hybrid classical-quantum algorithm whose classical step is based on the principle of deconvolution from signal processing. This classical preprocessing step helps in reducing the circuit depth required to load a given bell-shaped probability distribution. The deconvolution step breaks down the target PMF into smaller PMFs, which can be loaded in parallel into different quantum registers, reducing the circuit depth. The quantum part consists of (controlled) rotation



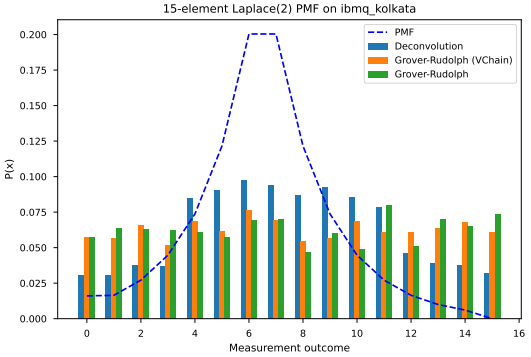
(a) 7-element Laplace PMF generated using IBMQ QASM simulator (Table 2a).



(b) 7-element Laplace PMF generated using IBMQ Kolkata (Table 2b).



(c) 15-element Laplace PMF generated using IBMQ QASM simulator (Table 2c).



(d) 15-element Laplace PMF generated using IBMQ Kolkata (Table 2d).

Figure 9: Preparing a discretized Laplace ($\theta = 2$) PMF on a quantum computer (analogous to Fig. 8, but with other PMF). Figures 9a and 9c are the results on IBM's QASM simulator (noiseless). All methods very well approximate the PMFs. When looking at Table 2, it is clear that the proposed deconvolution method results in the lowest distance with the lowest number of gates. When running on IBMQ Kolkata (figures 9b and 9d), the results are similar.

gates followed by a quantum adder circuit leading to reduced circuit depth. The algorithm reduces the dependency of the state preparation algorithms on big multi-controlled single-target gates used in traditional approaches for loading bell-shaped distributions. We propose the deconvolution step discussed in this article as an additional classical step that can be merged with any state preparation algorithm for loading bell-shaped distributions. This leads to a further reduction in the circuit depth. Deconvolution is the inverse of convolution and is a kind of inverse problem which we translated into a constrained optimization problem. We defined the JS function as the cost function of the optimization problem. We use the trust region method to find the optimum value of $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ that minimizes the cost function. This deconvolution algorithm is developed as part of the proof of concept for our approach, and we discuss a more efficient deconvolution algorithm based on polynomial factorization. To verify state preparation using the deconvolution approach, we load 7 and 15-element PMF constructed by discretizing the two different probability distributions (i) Standard Normal Distribution and (ii) Laplace distribution. The algorithm's results are positive on the QASM simulator and show a reduction in the circuit depth. Hence, the outcomes of the deconvolution method agree well with the expected outcome. But on the real hardware, still more circuit depth reduction is required since the noise is taking over.

We believe this work is the first where a theoretical concept from the signal processing field is used to solve state preparation problems in quantum computation. Due to a shared foundation in the mathematical framework of probability and statistics, signal processing and quantum computation have many overlapping and complementary topics. In future, we would like to dig further into different concepts of signal processing that can be used in quantum computation to design classical techniques that can efficiently load more complex problems into quantum processing units (QPUs) and provide us with more accurate results. We want to extend this method to solve the more complex problem of stochastic volatility modelling in finance, where the probability distribution is more complex. Calculations of some quantities of interest have no closed-form solutions, so they must be modelled using Monte Carlo simulation. Once we can load the required probability distribution into the QPU with minimum circuit depth, we can use the remaining available time to calculate complex financial quantities. Also, loading required probability distribution with minimum circuit depth into the current NISQ QPUs also finds application in other fields.

References

[1] D. Herman, C. Googin, X. Liu, A. Galda, I. Safro, Y. Sun, M. Pistoia, and Y. Alexeev, "A survey of quantum computing for finance," *arXiv preprint arXiv:2201.02773*, 2022.

[2] Y. Suzuki, S. Uno, R. Raymond, T. Tanaka, T. Onodera, and N. Yamamoto, "Amplitude estimation without phase estimation," *Quantum Information Processing*, vol. 19, pp. 1–17, 2020.

[3] N. Stamatopoulos, D. J. Egger, Y. Sun, C. Zoufal, R. Iten, N. Shen, and S. Woerner, "Option pricing using quantum computers," *Quantum*, vol. 4, p. 291, 2020.

[4] S. Woerner and D. J. Egger, "Quantum risk analysis," *npj Quantum Information*, vol. 5, no. 1, p. 15, 2019.

[5] L. Grover and T. Rudolph, "Creating superpositions that correspond to efficiently integrable probability distributions," *arXiv preprint quant-ph/0208112*, 2002.

[6] M. Plesch and i. c. v. Brukner, "Quantum-state preparation with universal gate decompositions," *Phys. Rev. A*, vol. 83, p. 032302, Mar 2011.

[7] A. C. Vazquez and S. Woerner, "Efficient state preparation for quantum amplitude estimation," *Physical Review Applied*, vol. 15, no. 3, p. 034027, 2021.

[8] A. A. Melnikov, A. A. Termanova, S. V. Dolgov, F. Neukart, and M. Perelshtein, "Quantum state preparation using tensor networks," *Quantum Science and Technology*, 2023.

[9] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva, "A divide-and-conquer algorithm for quantum state preparation," *Scientific reports*, vol. 11, no. 1, pp. 1–12, 2021.

[10] Q. contributors, "Intiliaze."

[11] Q. contributors, "Mcmtvchain."

[12] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum logic circuits," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pp. 272–275, 2005.

[13] IBM, "The ibm quantum development roadmap."

[14] B. Hajek, *Random processes for engineers*. Cambridge university press, 2015.

[15] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, no. 1, p. 147, 1996.

[16] N. I. Gould, S. Lucidi, M. Roma, and P. L. Toint, "Solving the trust-region subproblem using the lanczos method," *SIAM Journal on Optimization*, vol. 9, no. 2, pp. 504–525, 1999.

[17] A. V. Oppenheim, *Discrete-time signal processing*. Pearson Education India, 1999.

[18] T. S. community, "scipy.optimize.minimize."

[19] E. I. Verriest1 and N. seung Patrick Hyun2, "Roots of polynomials with positive coefficients," *23rd International Symposium on Mathematical Theory of Networks and Systems, Hong Kong University of Science and Technology*, 2018.

[20] N. Developers, "numpy.polynomial.polynomial.polyroots - numpy v1.26 manual."

[21] N. Developers, "numpy.polynomial.polynomial.polyfromroots."

[22] G. Park, K. Zhang, K. Yu, and V. Korepin, "Quantum multi-programming for grover's search," *Quantum Information Processing*, vol. 22, no. 1, p. 54, 2023.

[23] P. Jurcevic, D. Zajac, J. Stehlik, I. Lauera, and R. Mandelbaum, "Pushing quantum performance forward with our highest quantum volume yet," 2022. Accessed on 03 14, 2023.

[24] I. Q. (2023), "sampler primitive."

[25] I. Q. (2023), "Qiskit."

[26] S. Kotz, T. Kozubowski, and K. Podgórski, *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. No. 183, Springer Science & Business Media, 2001.

[27] Q. contributors, "Mcmt."

A Appendix

A.1 Quantum Amplitude Estimation Algorithm

The Quantum Amplitude Estimation (QAE) algorithm estimates the value of 'a' in the state $G_a \left|0\right\rangle_{\mathfrak{n}+1} = \sqrt{1-a}\left|\psi_0\right\rangle_{\mathfrak{n}}\left|0\right\rangle + \sqrt{a}\left|\psi_1\right\rangle_{\mathfrak{n}}\left|1\right\rangle$, where G_a is an operator acting on $\left|0\right\rangle_{\mathfrak{n}+1}$ state. One major application of QAE is calculating the expectation value of functions with random variables as their input. The calculation has three major steps, briefly described in the calculation VaR. But to re-emphasize, we outline them here more concretely.

- Step 1 Load the required Probability distribution that the random variable follows into an $\mathfrak{n}$-qubit quantum register. For this, we can map the random variable $\mathcal{R}$ in the interval $\{0, 1, 2, \ldots, N-1\}$ using $\mathfrak{n}$ qubits, where $N = 2^{\mathfrak{n}}$. Then, using any state preparation method discussed in this article, we can prepare a quantum operation that performs the following transformation

$$\mathcal{R}\left|0\right\rangle_{\mathfrak{n}} = \sum_{i=0}^{N-1} \sqrt{p_i}\left|i\right\rangle_{\mathfrak{n}} = \left|\psi\right\rangle_{\mathfrak{n}}, \tag{16}$$

  where $\left|i\right\rangle_n$ belongs to the computational basis set. In Equation 16, p_i represents the probability of superposition state $\left|\psi\right\rangle$ being in the basis state $\left|i\right\rangle_{\mathfrak{n}}$ and it also means the probability of random variable $\mathcal{R}$ taking a value $i \in \{0, 1, 2, \ldots, N-1\}$.

- Step 2 Load the function whose expectation is to be calculated using an extra qubit. We map the function to be calculated on the random variable to an operator F that performs the following transformation

$$F\left|\psi\right\rangle_{\mathfrak{n}}\left|0\right\rangle = \sum_{i=0}^{N-1} \sqrt{1-f(i)}\sqrt{p_i}\left|i\right\rangle_{\mathfrak{n}}\left|0\right\rangle + \sum_{i=0}^{N-1} \sqrt{f(i)}\sqrt{p_i}\left|i\right\rangle_{\mathfrak{n}}\left|1\right\rangle, \tag{17}$$

  where $f(i)$ represents the function whose expectation we aim to calculate.

- Step 3 Use the QAE algorithm to estimate the expectation value of the function, which is nothing but the probability of measuring the last qubit in state $\left|0\right\rangle$. For this estimation, the QAE algorithm requires only $O(M^{-1})$ samples, unlike $O(M^{-1/2})$ that other classical algorithms like the Monte Carlo require.

Hence the QAE algorithm gives a quadratic speed-up compared to the classical Monte Carlo algorithm. This advantage makes the QAE algorithm a very important algorithm for finance, particularly in the field of risk management.

A.2 Grover Rudolph State Preparation Method

In this section, we discuss the most commonly used state preparation method for loading log-concave probability distribution. Log-concave probability distributions are a set of probability distributions whose probability density functions are easily integrable using existing classical techniques [5]. Let us assume that we want to prepare a given quantum register in a certain superposition state following some specific log-concave probability distribution. In other words, we want to transform a quantum register that is initially in the state $\left|0\right\rangle_n$ to a superposition state $\left|\psi\right\rangle_n$ defined as

$$\left|\psi\left(\{p_i\}\right)\right\rangle = \sum_i \sqrt{p_i}\left|i\right\rangle_{\mathfrak{n}}, \tag{18}$$

where $|i\rangle_n$ are computational basis states. They denote the values a given random variable $\mathcal{R}_i = \{0, 1, 2, \ldots, i, \ldots, 2^n - 1\}$ can take, and p_i represents the probability of the random variable $\mathcal{R}_i$ to take each value. We do this for particular cases where an efficient classical algorithm exists to integrate the given probability density function $p(x)$ corresponding to the given random variable. Hence, we can create the superposition state as shown in Equation 18 for a discretized version $\{p_i\}$ of $p(x)$.

Here, we take a single random variable $\mathcal{X}$ for illustration purposes, but we can easily modify the method for a multivariate random variable. Let us assume that we have to discretize the probability distribution corresponding to the random variable $\mathcal{X}$ over N points, then the number of qubits needed to load this distribution is equal to $\mathfrak{n} = \lceil \log_2 N \rceil$. This means, as shown in Fig. 10, if we have $\mathfrak{m}$ qubits, then we can divide the probability distribution into $2^{\mathfrak{m}}$ regions.

The quantum superposition state of this m qubits can be written as

$$|\psi\rangle_{\mathfrak{m}} = \sum_{i=0}^{2^{\mathfrak{m}}-1} \sqrt{p_i^{(\mathfrak{m})}}\, |i\rangle\,, \tag{19}$$

where $p_i^{(\mathfrak{m})}$ represents the probability of the random variable $\mathcal{X}$ to lie in the i^{th} region [5]. If we add one more qubit, we can subdivide this region further and have a total of $2^{\mathfrak{m}+1}$ regions. That is,

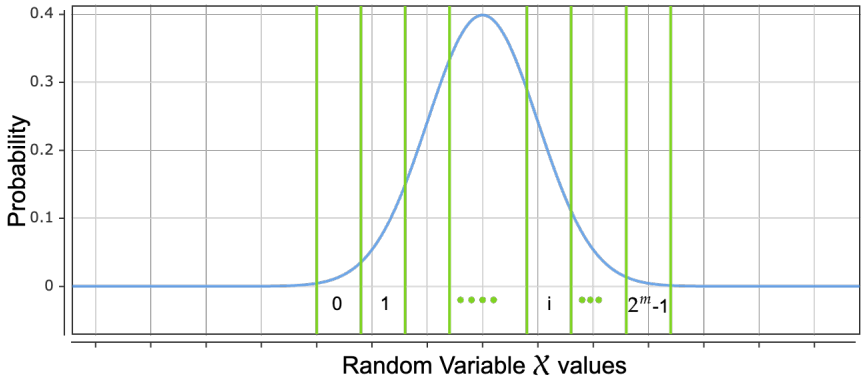


Figure 10: Normal distribution divided into $2^{\mathfrak{m}}$ regions. The x-axis of the plot represents the continuous values a random variable can take, and the y-axis represents the probability density of the random variable $\mathcal{X}$ taking each value. $p(x)$ of the normal distribution is given by $\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$. In the context of plotting, we set the standard deviation value (σ) equal to 1. However, it is important to note that we do not fix the value of the parameter μ, which results in the absence of marking along the X-axis. Based on the value of μ, the whole plot shifts right or left along the x-axis.

we can further split the region $2^{\mathfrak{m}}$ into two halves, the left half and the right half. Let the probability of random variable $\mathcal{X}$ lying in the left and right half of the region be e and f, respectively. Then using the R_y rotation operator on the extra qubit controlled on the previous m qubits, we can perform the following transformation

$$\sum_{i=0}^{2^{\mathfrak{m}}-1} \sqrt{p_i^{(\mathfrak{m})}}\, |i\rangle\, |0\rangle = \sum_{i=0}^{2^{\mathfrak{m}}-2} \sqrt{p_i^{(\mathfrak{m})}}\, |i\rangle\, |0\rangle + \sqrt{p_{2^{\mathfrak{m}}-1}^{(\mathfrak{m})}}\, |2^{\mathfrak{m}} - 1\rangle \left(\sqrt{e_i}\, |0\rangle + \sqrt{f_i}\, |1\rangle\right). \tag{20}$$

We can do this for other regions and transform the state to a desired particular state using a combination of X and multi-controlled R_y gates. Then we repeat this step, and in each step, we

increase the qubit by one. Once the number of qubits reaches the desired value $\mathfrak{n}$, we stop. At the end of this iterative step, we would have constructed a circuit (A) consisting of X and multi-controlled R_y gates that would perform the following transformation

$$A\left|0\right\rangle_n = \sum_{i=0}^{2^n-1} \sqrt{p_i^{(n)}}\left|i\right\rangle_n . \tag{21}$$

If the start and end point of the $(2^m-1)^{\text{th}}$ region is marked as x_{low} and x_{upp} then we can calculate the value of e and f using the formula

$$e_{2^m-1} = \frac{\int_{x_{\text{low}}}^{\frac{x_{\text{upp}}-x_{\text{low}}}{2}} p(x)}{\int_{x_{\text{low}}}^{x_{\text{upp}}} p(x)}, \tag{22}$$

$$f_{2^m-1} = \frac{\int_{\frac{x_{\text{upp}}-x_{\text{low}}}{2}}^{x_{\text{upp}}} p(x)}{\int_{x_{\text{low}}}^{x_{\text{upp}}} p(x)}. \tag{23}$$

The angle (θ_i) by which the last qubit must be rotated changes for each value the first m qubits $(|i\rangle)$ takes and can be calculated using a simple mathematical equation

$$\theta_i = \cos^{-1}(e_i) \;\text{ or }\; \theta_i = \sin^{-1}(f_i). \tag{24}$$

To illustrate this algorithm further, we explain it using an example in the next section.

A.2.1 Example for GR state preparation method

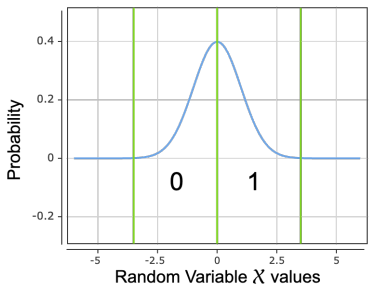
Using the GR state preparation method, we demonstrate how to build a quantum circuit using multi-controlled R_y and X gates to load standard normal distribution into a quantum register consisting of $n = 3$ qubits. The probability density function for standard normal distribution can be obtained by putting $\mu = 0$ and $\sigma = 1$ in the $p(x)$ formula corresponding to normal distribution. Hence $p(x)$ for normal distribution can be explicitly written as

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}, \tag{25}$$

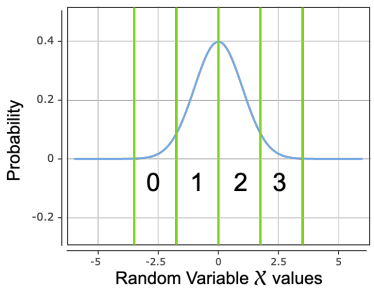
where x corresponds to different values random variable $\mathcal{X}$ can take along the x-axis. The plot for standard normal distribution is similar to the plot shown in Fig. 10, the only difference being the peak for the standard normal distribution will be located specifically at the origin since the value of the parameter μ equals to 0 for standard normal distribution as shown in the Fig. 11.

We start by dividing the normal distribution into two regions, as shown in Fig. 11a and the load the probability of being in the left half or right half as the probability of the first qubit being in the state $|0\rangle$ and $|1\rangle$ respectively. Next, we further subdivide this region into two halves resulting in a total of four regions. The probability of the random variable being in the left half of the region marked zero in Fig. 11a, which is the same as the region marked zero in Fig. 11b, is equal to the probability of second qubit being in state $|0\rangle$ given that the first qubit is in state $|0\rangle$. Similarly, we can define the probability of the first and second qubits being in the state $|01\rangle$,$|10\rangle$ and $|11\rangle$. The quantum circuit constructed following the GR state preparation method to prepare the first and second qubits in this state is shown in Fig. 12.

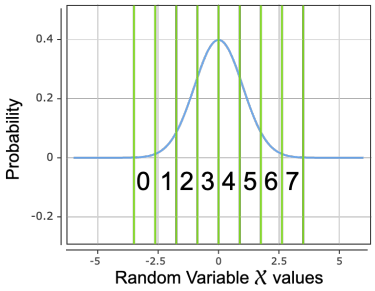
The values of θ_0, θ_1 and θ_2 can be calculated using the formal given in Equation 22, 23 and 24. For $\mathfrak{n} = 3$, we repeat this step just one more time and for any arbitrary value of $\mathfrak{n}$, we repeat



(a) Standard Normal Distribution plot divided into 2 regions.



(b) Standard Normal Distribution plot divided into 4 regions.



(c) Standard Normal Distribution plot divided into 8 regions.

Figure 11: Standard Normal distribution plot subdivided into different regions as we recursively progress in the GR state preparation.

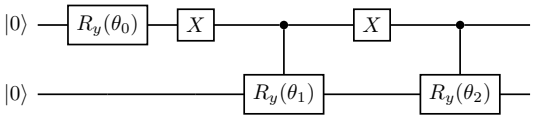


Figure 12: Quantum circuit for loading a given probability distribution into a 2 qubit system using GR method.

the step until the number of regions is equal to $2^{\mathfrak{n}}$. Fig. 13 shows the quantum circuit to load the discretized version of the normal distribution.

In Fig. 13, we can calculate the values of the θ's by following the Equations given in 22, 23 and 24.

A.3 VChain

The section discusses the VChain implementation of a multi-controlled single target gate. If the gate has control over the $\mathfrak{n}$ qubit, then we need $\mathfrak{n}-1$ ancilla qubit for the VChain implementation of the gate.

A.4 Gradient and Hessian matrix calculation for deconvolution using Trust region method

The gradient of the JS function is as given below :

$$\frac{\partial JS(\boldsymbol{P}||\boldsymbol{Q})}{\partial q_{1_k}} = \sum_{i=0}^{L(\boldsymbol{q_1})+L(\boldsymbol{q_2})-1} \log\frac{Q_i}{R_i}\frac{\partial Q_i}{\partial q_{1_k}} \tag{26}$$

$$Q_i = \sum_{u=\max(0,i-L(\boldsymbol{q_1})+1)}^{\min(i,L(\boldsymbol{q_2})-1)} q_{1_u} q_{2_{i-u}} \implies \frac{\partial Q_i}{\partial q_{1_k}} = \sum_{u=\max(0,i-L(\boldsymbol{q_1})+1)}^{\min(i,L(\boldsymbol{q_2})-1)} q_{2_{i-u}}\frac{\partial q_{1_u}}{q_{1_k}} \tag{27}$$

In equation 26 R_i represents the i^{th} element of the PMF $\boldsymbol{R}$ defined using the equation 8. In the equations 28, 29, 30 and 31, the index i runs from $i=0$ to $i=L(\boldsymbol{q_1})+L(\boldsymbol{q_2})-1$ and the index u

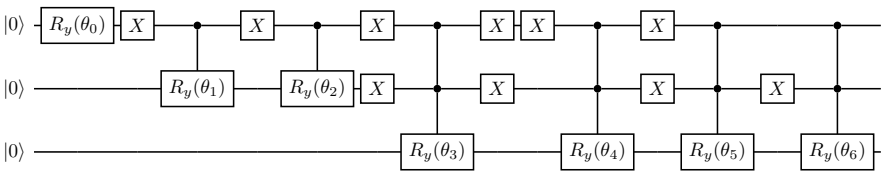


Figure 13: Quantum circuit built using GR method for loading required probability distribution into a 3 qubit system.

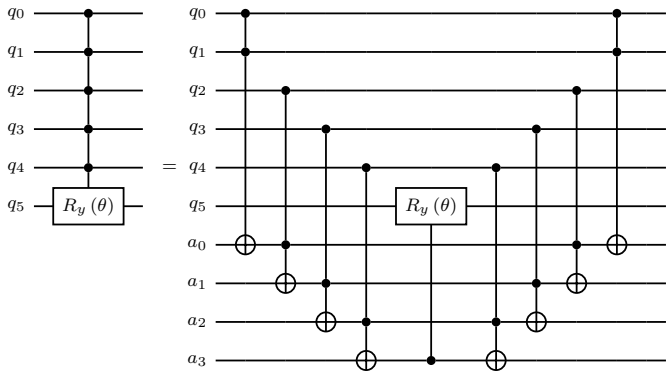


Figure 14: V-Chain structure to create a multi-controlled $R_y(\theta)$-gate. The multi-controlled $R_y(\theta)$-gate does not exist on current hardware and thus will need to be decomposed. The decomposition in basis gates without ancilla qubits will have more circuit depth compared to the decomposition in which we use ancilla qubits [27]. Using ancilla qubits and Toffoli gates to create the $R_y(\theta)$-gate will result in a linear-depth decomposed circuit at the expense of adding qubits. One can visually verify the two circuits are equivalent by considering the state of $q_0, q_1, ..., q_4$. The $R_y(\theta)$-gate will only be applied with input $|11111\rangle$, as should be the case for a multi-controlled gate.

runs from $u = \max(0, i - L(\boldsymbol{q_1}) + 1)$ to $\min(i, L(\boldsymbol{q_2}) - 1)$. The second-order derivative for the JS distance function is

$$A_{lk} = \frac{\partial JS(\boldsymbol{P}||\boldsymbol{Q})}{\partial q_{1_l} \partial q_{1_k}} = \sum_i \left(\left(\frac{1}{Q_i} - \frac{1}{2R_i} \right) \left(\sum_u q_{2_{i-u}} \frac{\partial q_{1_u}}{\partial q_{1_k}} \right) \left(\sum_u q_{2_{i-u}} \frac{\partial q_{1_u}}{\partial q_{1_l}} \right) \right), \tag{28}$$

$$\begin{aligned} B_{lk} = \frac{\partial JS(\boldsymbol{P}||\boldsymbol{Q})}{\partial q_{2_l} \partial q_{1_k}} = &\sum_i \log \frac{Q_i}{R_i} \left(\sum_u \frac{\partial q_{2_{i-u}}}{\partial q_{2_l}} \frac{\partial q_{1_u}}{\partial q_{1_k}} \right) + \\ &\sum_i \left(\left(\frac{1}{Q_i} - \frac{1}{2R_i} \right) \left(\sum_u q_{2_{i-u}} \frac{\partial q_{1_u}}{\partial q_{1_k}} \right) \left(\sum_u q_{1_u} \frac{\partial q_{2_{i-u}}}{\partial q_{2_l}} \right) \right), \end{aligned} \tag{29}$$

$$C_{lk} = \frac{\partial JS(\boldsymbol{P}||\boldsymbol{Q})}{\partial q_{2_l} \partial q_{2_k}} = \sum_i \left(\left(\frac{1}{Q_i} - \frac{1}{2R_i} \right) \left(\sum_u q_{1_u} \frac{\partial q_{2_{i-u}}}{\partial q_{2_k}} \right) \left(\sum_u q_{1_u} \frac{\partial q_{2_{i-u}}}{\partial q_{2_l}} \right) \right), \tag{30}$$

$$D_{lk} = \frac{\partial JS(\boldsymbol{P}||\boldsymbol{Q})}{\partial q_{1_l} \partial q_{2_k}} = \sum_i \log \frac{Q_i}{R_i} \left(\sum_u \frac{\partial q_{1_u}}{\partial q_{1_l}} \frac{\partial q_{2_{i-u}}}{\partial q_{2_k}} \right) + \sum_i \left(\left(\frac{1}{Q_i} - \frac{1}{2R_i} \right) \left(\sum_u q_{2_{i-u}} \frac{\partial q_{1_u}}{\partial q_{1_l}} \right) \left(\sum_u q_{1_u} \frac{\partial q_{2_{i-u}}}{\partial q_{2_k}} \right) \right). \tag{31}$$

The Hessian matrix, can be constructed using the equations 28, 29, 30 and 31 as given below :

$$H_{(m+n)\times(m+n)} = \left(\begin{array}{c|c} A_{m\times m} & B_{m\times n} \\ --- & --- \\ C_{n\times m} & D_{n\times n} \end{array} \right). \tag{32}$$

A.5 Example for Algorithm 3

In this Section, we take the example of a PMF $\boldsymbol{P}$ that we obtain by discretizing the normal distribution over 2^5 points. This implies the length/number of elements in the PMF vector $\boldsymbol{P}$ is 32 and the PGF $f(x)$ obtained using the Equation 10 corresponds to a polynomial of degree $len(\boldsymbol{P}) - 1 = 32 - 1 = 31$,

$$\begin{aligned} f(x) =& 0.001111 + 0.00187962x + 0.00307045x^2 + 0.00484294x^3 + 0.00737552x^4 + 0.01084556x^5 \\ &+ 0.01539884x^6 + 0.02111057x^7 + 0.02794398x^8 + 0.0357152x^9 + 0.04407519x^{10} \\ &+ 0.05251843x^{11} + 0.06042348x^{12} + 0.06712375x^{13} + 0.07199846x^{14} + 0.074567x^{15} \\ &+ 0.074567x^{16} + 0.07199846x^{17} + 0.06712375x^{18} + 0.06042348x^{19} + 0.05251843x^{20} \\ &+ 0.04407519x^{21} + 0.0357152x^{22} + 0.02794398x^{23} + 0.02111057x^{24} + 0.01539884x^{25} \\ &+ 0.01084556x^{26} + 0.00737552x^{27} + 0.00484294x^{28} + 0.00307045x^{29} + 0.00187962x^{30} \\ &+ 0.001111x^{31}. \end{aligned} \tag{33}$$

The first step of Algorithm 3 is to calculate all the roots of the polynomial $f(x)$ and then segregate them into three categories (i) Complex roots with non-negative real,(ii) Complex roots with a negative real part and (iii) real roots. Then we arrange the category (i) roots in ascending order based on the real part and name it $basket_1$. $basket_1$ is a list of 1D arrays, where the 1D arrays containing the conjugate pair and for this particular example

$$\begin{aligned} basket_1 = [&[(0.1750446257566166 + 0.9845605004232698j), (0.17504462575661892 - 0.9845605004232723j)] \\ &[(0.3597590272791502 - 0.9330452520061215j), (0.359759027279154 + 0.9330452520061235j)] \\ &[(0.5302711299415993 + 0.8478281245337772j), (0.5302711299415998 - 0.8478281245337765j)] \\ &[(0.6820882381676622 - 0.7312698786039119j), (0.6820882381676232 + 0.7312698786039119j)] \\ &[(0.7059889885210886 - 0.4103126556472708j), (0.7059889885210903 + 0.4103126556472708j)] \\ &[(0.776903022928539 - 0.6296202768053927j), (0.7769030229285399 + 0.6296202768053933j)] \\ &[(1.058808365464415 + 0.6153672073063402j), (1.0588083654644178 - 0.6153672073063433j)]]. \end{aligned} \tag{34}$$

Also, merge the categories (ii) and (iii) roots and store them in a list variable $basket_2$ and for this particular example

$$\begin{aligned} basket_2 = [&[(-0.016895893550830186 + 0.9998572542023804j), (-0.016895893550830644 - 0.9998572542023763j)] \\ &[(-0.20875549693350764 - 0.9779678637358403j), (-0.20875549693350828 + 0.9779678637358418j)] \\ &[(-0.393198331956103 + 0.9194536811318694j), (-0.3931983319561042 - 0.9194536811318671j)] \\ &[(-0.563157764980587 + 0.8263494005213874j), (-0.5631577649805874 - 0.8263494005213892j)] \\ &[(-0.7121153156206508 + 0.7020625166311788j), (-0.7121153156206516 - 0.7020625166311799j)] \\ &[(-0.8343547541714113 + 0.551227851429491j), (-0.8343547541714125 - 0.5512278514294874j)] \\ &[(-0.9251836312976823 - 0.379519760195477j), (-0.9251836312976831 + 0.37951976019547573j)] \\ &[(-0.9811146055103414 + 0.1934273270618093j), (-0.9811146055103458 - 0.19342732706180849j)] \\ &[(-1.0000000000000024 + 0j)]]. \end{aligned} \tag{35}$$

$basket_2$ list is randomly shuffled, and after this step, we start a while loop as shown in Fig. **??**. In the first iteration, we take the first element from $basket_1$ which is the 1D array $[(0.1750446257566166 + 0.9845605004232698j), (0.17504462575661892 - 0.9845605004232723j)]$. Then we start another while, in which we select a random element from $basket_2$ say $[(-0.563157764980587 + 0.8263494005213874j), (-0.5631577649805874 - 0.8263494005213892j)]$ and both the element into one single 1D array, i.e.,

$$\begin{aligned} \text{temp\_array} = [&(0.1750446257566166 + 0.9845605004232698j), (0.17504462575661892 - 0.9845605004232723j), \\ &(-0.563157764980587 + 0.8263494005213874j), (-0.5631577649805874 - 0.8263494005213892j)]. \end{aligned} \tag{36}$$

Using the Python function numpy.polynomial.polyfromroots, we calculate a monic polynomial that has roots mentioned in temp_array. We delete the random element we choose from the $basket_2$ list and check if all the coefficients of the monic polynomial are greater than zero. If yes, we terminate the current while loop and append the temp_array to the $basket_2$ list and also delete the element we choose from $basket_1$ from $baket_1$ itself. Otherwise, we select another random element and repeat the whole procedure. The outer while loop ends when we have exhausted all the elements from $baslet_1$. We run Algorithm 3 in parallel in many cores and select the result which would have factorised the target polynomial $f(x)$ into the most number of factors. Among this result, we select the one whose biggest factor has the least degree compared to other biggest factors belonging to other results. For our specific example using Algorithm 3, we get the following result

$$\begin{aligned} f(x) =&(1 + 0.77622628x + 1.60568904x^2 + 0.77622628x^3 + 1x^4)(1 + 0.06005415x + 0.05709808x^2 \\ &+ 0.06005415x^3 + 1x^4)(1 + 0.65352995x + 0.14493192x^2 + 1.16946487x^3 + 0.93393201x^4 \\ &+ 0.93393201x^5 + 1.16946487x^6 + 0.14493192x^7 + 0.65352995x^8 + 1x^9)(1 + 0.20201441x \\ &+ 0.06184121x^2 + 0.39708601x^3 + 0.23166121x^4 + 0.98690859x^5 + 1.56270069x^6 \\ &+ 1.38605043x^7 + 1.56270069x^8 + 0.98690859x^9 + 0.23166121x^{10} + 0.39708601x^{11} \\ &+ 0.06184121x^{12} + 0.20201441x^{13} + 1x^{14}). \end{aligned} \tag{37}$$

For this example, Algorithm 3 ran over 1000 runs, which were parallelized. The result was obtained in $50s$, which is much faster compared to the deconvolution algorithm developed using optimization techniques in Section 3.1.