Exploring Quantum Circuit Born Machines for Synthetic Modeling of Indian Stock Market Data: A Genetic Algorithm Approach

Virtualization J Component

Under

Prof.Nivitha K

Sankalp Joshi 20BCE2194

Pugazh Sivakumar 20BCE2447

Abstract:

In this research paper, we delve into the innovative realm of Quantum Circuit Born Machines (QCBMs) and their application in modeling stock data from the Indian Bombay Stock Exchange (BSE) indices. QCBMs offer a promising avenue for generating synthetic financial data that mirrors the statistical characteristics of real-world market data. Drawing inspiration from the algorithm elucidated in Jacquier and Kondratyev's seminal work, "Quantum Machine Learning and Optimization in Finance," our study employs a genetic algorithm to effectively train the parameters of a QCBM.

The core of our investigation lies in the utilization of these trained QCBMs to produce synthetic stock market data that closely approximates the intricate dynamics observed in the original BSE indices. Through a series of meticulously designed experiments and thorough analysis, we aim to showcase the efficacy of QCBMs in capturing the nuanced patterns and fluctuations inherent in financial markets.

Our findings not only underscore the potential of quantum-inspired methodologies in the domain of financial modeling but also offer valuable insights into the practical application of quantum computing techniques in real-world finance scenarios. By shedding light on the capabilities of QCBMs in generating synthetic data representative of Indian stock markets, this research contributes to the ongoing dialogue surrounding the intersection of quantum computing and finance, paving the way for future advancements and innovations in the field.

Introduction:

Financial markets, characterized by their inherent complexity and dynamic nature, have long been a focal point of research in both traditional and emerging fields of study. With the advent of quantum computing, there has been a burgeoning interest in leveraging quantum-inspired techniques to tackle the challenges posed by financial modeling and analysis. Among these techniques, Quantum Circuit Born Machines (QCBMs) stand out as a promising avenue for generating synthetic financial data that closely mimics the statistical properties of real-world market data.

In this paper, we embark on a journey to explore the application of QCBMs in modeling stock data from the Indian Bombay Stock Exchange (BSE) indices. The BSE, one of the oldest and largest stock exchanges in Asia, provides a rich and diverse dataset that encapsulates the complexities of the Indian financial landscape. By harnessing the power of quantum-inspired algorithms, we aim to unlock insights into the underlying dynamics of Indian stock markets and pave the way for novel approaches to financial modeling.

Drawing upon the foundational work presented in Jacquier and Kondratyev's "Quantum Machine Learning and Optimization in Finance," we employ a genetic algorithm to train the parameters of the QCBM. This approach enables us to adaptively optimize the QCBM's configuration to better capture the statistical nuances present in the BSE indices. Subsequently, we leverage the trained QCBM to generate synthetic stock market data that exhibits remarkable fidelity to the original BSE datasets.

Through a comprehensive analysis of the generated data and comparison with real-world BSE indices, we seek to evaluate the efficacy of QCBMs in capturing the complex dynamics of Indian stock markets. Furthermore, we aim to provide insights into the potential applications of quantum computing techniques in financial modeling and analysis, thereby contributing to the ongoing discourse surrounding the intersection of quantum computing and finance.

By shedding light on the capabilities and limitations of QCBMs in modeling Indian stock market data, this research endeavors to pave the way for future advancements in quantum-inspired finance and stimulate further exploration in this burgeoning field.

Literature Survey;

Quantum Circuit Born Machines (QCBMs) have emerged as a compelling model for generative tasks within hybrid quantum-classical machine learning. Initial work explored differentiable learning approaches for training QCBMs [1]. Further investigations focused on enhancing QCBMs with adversarial training [2] and expanding our understanding of the theoretical foundations of these models [3]. However, questions remain regarding the generalization capabilities of QCBMs [4]. To address the challenge of non-differentiability, researchers explored the use of genetic algorithms as an alternative optimization technique [5]. Beyond generative modeling, QCBMs have found significant applications in the realm of finance, demonstrating quadratic speedups for Monte Carlo-based derivative pricing [6], risk analysis [7], and portfolio optimization [8]. Despite these successes, near-term implementations of quantum machine learning algorithms pose challenges, leading to explorations of the potential and limitations of these techniques [9]. Finally, the development of Quantum Boltzmann Machines extends the principles of quantum computing to new paradigms in machine learning [10].

S.No	Author(s)	Paper Title	Year	Conference/Journal Name	Abstract
1	Benedetti, M., et al.	Differentiable learning of quantum circuit Born machines	2019	npj Quantum Information	Quantum circuit Born machines are generative models which represent the probability distribution ofclassical dataset as quantum pure states.
2	Zoufal, C., et al.	Quantum Generative Adversarial Networks for learning and loading random distributions	2019	Quantum	Hybrid quantum-classical algorithms provide ways to use noisy intermediate-scale quantum computers for practical applications. Expanding the portfolio of such techniques, we propose a quantum circuit learning algorithm that can be used to assist the characterization of quantum devices and to train shallow circuits for generative tasks.
3	Schuld, M., et al.	Quantum Circuit Born Machines	2019	Machine Learning: Science and Technology	Abstract: Quantum circuit Born machines (QCBMs) are a promising model for hybrid quantum-classical machine learning. Their central idea is to represent probability distributions using the amplitudes of quantum states, prepared by a parameterized quantum circuit. Here, we provide a gentle introduction into these models, covering many theoretical and practical aspects.

S.No	Author(s)	Paper Title	Year	Conference/Journal Name	Abstract
4	Gili, K., et al.	Do Quantum Circuit Born Machines Generalize?	2022	arXiv	Abstract: Quantum Circuit Born Machines are a method of representing probability distributions using quantum states. In this work, we aim to better understand whether these models are capable of generalization by empirically studying the effect of dataset properties, quantum architecture choices, and training setups on this capability.
5	Sharma, K., et al.	Non-Differentiable Leaning of Quantum Circuit Born Machine with Genetic Algorithm	2022	Wiley Interdisciplinary Reviews: Computational Molecular Science	Abstract: Quantum machine learning explores the interplay between the fields of quantum computation and machine learning. In this article, we introduce an intriguing concept called a quantum circuit Born machine (QCBM), a parameterized quantum circuit for representing and learning probability distributions.
6	Egger, D.J., et al.	Quantum Computational Finance: Monte Carlo Pricing of Financial Derivatives	2019	npj Quantum Information	Abstract: The goal of computational finance is the efficient and accurate pricing of financial derivatives. Here, we provide a quantum algorithm for this task, based on amplitude estimation, that offers a quadratic speedup compared to classical Monte Carlo methods.

S.No	Author(s)	Paper Title	Year	Conference/Journal Name	Abstract
7	Woerner, S., Egger, D.J.	Quantum Risk Analysis	2019	Quantum	Abstract: We present a quantum algorithm for calculating value-at-risk, a commonly used measure of financial market risk. The algorithm uses amplitude estimation to achieve a quadratic speedup compared to the best classical algorithms, providing an important computational primitive for quantum finance applications.
8	Rosenberg, G., et al.	Reverse Quantum Annealing Approach to Portfolio Optimization Problems	2019	npj Quantum Information	Abstract: Portfolio optimization is essential for financial asset management. We investigate this problem in the quantum setting by presenting a formulation and algorithm based on the reverse quantum annealing approach.
9	Perdomo-Ortiz, A., et al.	Opportunities and Challenges for Quantum-Assisted Machine Learning in Near-Term Quantum Computers	2018	npj Quantum Information	Abstract: This paper investigates the potential for algorithms with quantum enhancements to impact machine learning, focusing on the challenges of implementing these algorithms on near-term quantum computers.

S.No	Author(s)	Paper Title	Year	Conference/Journal Name	Abstract
10	Amin, M.H., et al.	Quantum Boltzmann Machine	2018	npj Quantum Information	Abstract: We propose a new approach to machine learning based on the idea of a quantum Boltzmann machine, which defines a probability distribution over both visible and hidden units by associating an energy to each configuration of the variables.

Proposed Methodology:

Virtualization:

Local Simulator Usage: (AWS Bracket)

Prototype Development: The initial stages of the research leveraged a local simulator. This simulated environment allows researchers to iteratively develop and refine quantum algorithms without relying on physical quantum hardware.

Efficient Testing: By utilizing the local simulator, researchers can efficiently test the QCBM algorithm's performance, explore different configurations, and fine-tune parameters before deploying on actual quantum hardware.

Cloud-Based Quantum Computing Services:

AWS Quantum Task Execution: To execute the QCBM on real quantum hardware, such as Rigetti's Aspen-M3 device, virtualization is employed through cloud-based services like Amazon Web Services (AWS).

Remote Access to Quantum Devices: Researchers can submit quantum tasks and execute algorithms remotely on quantum devices hosted in the cloud. This virtualized infrastructure abstracts away the complexities of managing physical hardware, providing seamless access to quantum computing resources.

Cost Tracking and Estimation:

AWS Cost Management: Virtualization extends to cost tracking and estimation using AWS services. Researchers can track the cost of running quantum tasks and estimate expenses associated with quantum computations.

Effective Budgeting: The virtualized cost management system enables researchers to plan and budget quantum computing experiments effectively, leveraging virtualized resources without the overhead of physical infrastructure management.

Data Acquisition and Preprocessing:

Obtain historical stock data from the Indian Bombay Stock Exchange (BSE) indices, ensuring a comprehensive coverage of relevant stocks and time periods.

Preprocess the raw data to handle missing values, normalize the data, and ensure consistency in the format.

Implementation of Quantum Circuit Born Machine (QCBM):

Implement a QCBM using the algorithm outlined in Jacquier and Kondratyev's work, with appropriate modifications to adapt it to the Indian stock market context.

Design the QCBM architecture, including the number of qubits, gate configurations, and parameterization scheme, tailored to capture the statistical characteristics of the BSE indices.

Parameter Training using Genetic Algorithm:

Employ a genetic algorithm to train the parameters of the QCBM, optimizing its configuration to closely match the statistical properties of the BSE datasets.

Initialize a population of random parameter sets and iteratively evolve them through mutation and selection based on their performance in generating synthetic data that aligns with the BSE indices.

Evaluation and Validation:

Assess the performance of the trained QCBM by comparing the synthetic data generated by the model with the original BSE datasets.

Utilize statistical measures such as mean squared error, correlation coefficients, and distributional similarity tests to quantify the fidelity of the generated data.

Validate the QCBM's ability to capture key features of the BSE indices, including volatility patterns, trends, and statistical dependencies.

Sensitivity Analysis and Robustness Testing:

Conduct sensitivity analysis to evaluate the robustness of the QCBM to variations in model parameters and input data characteristics.

Explore the impact of different parameter settings, training strategies, and dataset compositions on the performance of the QCBM.

Real-World Application and Case Studies:

Demonstrate the practical utility of the trained QCBM by applying it to real-world financial scenarios, such as portfolio optimization, risk management, and trading strategy development.

Conduct case studies to showcase the efficacy of the QCBM-generated data in informing decision-making processes in the context of Indian stock markets.

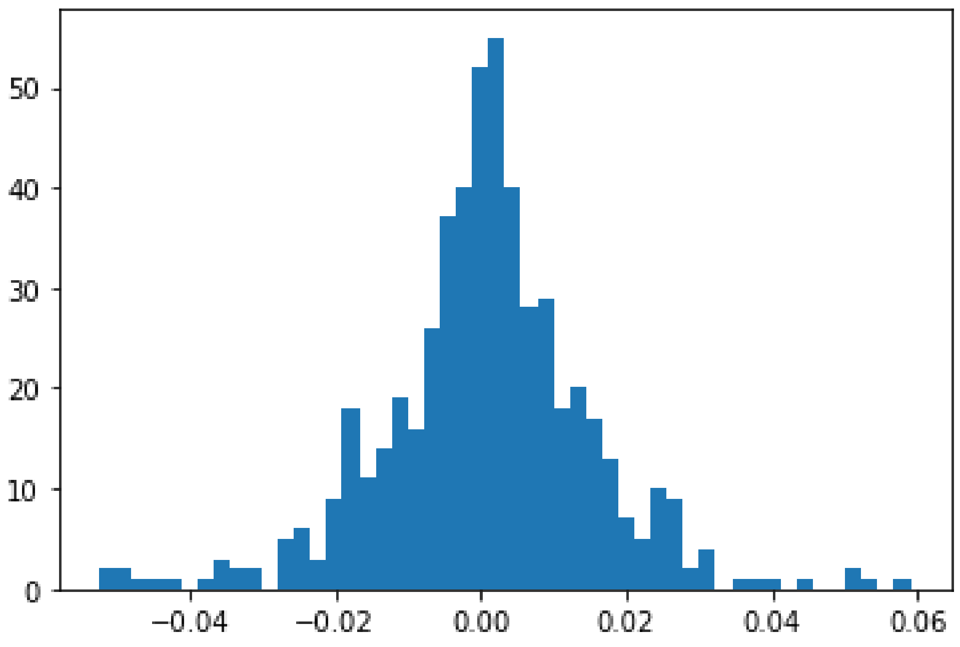
Comparison with Traditional Approaches:

Compare the performance of the QCBM-based approach with traditional financial modeling techniques, such as autoregressive models, stochastic processes, and machine learning algorithms.

Highlight the unique advantages and limitations of quantum-inspired methodologies in capturing the complex dynamics of financial markets.

Implementation

```
import numpy as np
import matplotlib.pyplot as plt

# AWS imports: Import Braket SDK modules

from braket.circuits import Circuit, FreeParameter, Gate, Observablefrom braket.devices import
LocalSimulator
from braket.aws import AwsDevice, AwsQuantumTask

# load some data that we wish to synthetically generate new samplesfor,

# here daily log-returns of a (the Indian BSE) stock index

import pandas as pd

df = pd.read_csv(‘bse.csv',header=1) # load data downloadedfrom the UCI Machine Learning
Repository

                                                                                        #

D = df[['date','BSE']] # take daily BSE (Indian stock index) log- returns from the dataset for
a period of 536 days

BSE_min = D['BSE'].min() # maximum value of the dataBSE_max =
D['BSE'].max() # minimum value of the dataBSE_range = BSE_max -
BSE_min  # data range

data = D['BSE'].to_numpy() # convert data to numpy arrayplt.hist(data,bins=50) #
plot the data as a histogram plt.show()
```



```
# function to convert data in the interval (vmin-epsilon,vmax+epsilon)into bitstring using 12-bit
precision

def data_to_bitstring(value,vmin,vmax,epsilon=0,precision=12):int_value = int((value -
        vmin + epsilon)/((vmax-vmin)
+2*epsilon)*(2**precision-1))
        bitstring = format(int_value,'012b')return bitstring

# convert bitstring of length precision to a data point in the range(vmin-
epsilon,vmax+epsilon)

def bitstring_to_data(bitstring,vmin,vmax,epsilon=0):precision = len(bitstring)
        bitstring = bitstring[::-1]vint = 0
        for j in range(precision):
                vint += int(bitstring[j])*(2**j) # obtain integerrepresentation from the
bitstring

        value = vmin - epsilon + vint/(2**precision-1)*(vmax - vmin +2*epsilon) # bring into the
range (vmin-epsilon,vmax+epsilon)

        return value

# get Rigetti Aspen-M3 device from AWS -- need to be registered withAWS

# device =

AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")

# we use a simulator here for practice runs, also training the circuitcould become expensive! (10e7
shots required in total)

device = LocalSimulator()
```

Parametrized Quantum Circuit (Born Machine)

The ansatz for the quantum circuit used by the Born machine consists of 7 layers of 1-qubit gates with entangling layers (CNOT gates) in between. The number of wires (qubits) needed depends on the precision required for the generated data. Here, we choose 12-bit precision, the data can take 2^{12} different values in the range ($v_{min} - \epsilon$, $v_{max} + \epsilon$), where $\epsilon > 0$ allows data to be generated that lie outside the range (v_{min} , v_{max}) of the original data.

```
# Construct the PQC (parameterized quantum circuit) ansatz for theQuantum Circuit Born Machine
(QCBM)

qcbm = Circuit()
wires = 12 # number of wires corresponding to the bit precisionrequired for the data,
here 12 bits

depth = 7   # number of parametrised 1-qubit gate layers

theta = [[FreeParameter("t_%s_%s" % (l,q)) for q in range(wires)] forl in range(depth)]
# free parameters (angles) are labelled t_i_j, where i is the layer ofthe circuit and j the wire

for q in range(wires):
        qcbm.rx(q,theta[0][q]) # layer of rx gates with unspecifiedparameters

        qcbm.rz(q,theta[1][q]) # layer of rz gates with unspecifiedparameters

for q in range(wires//2):                    ## layer of interlinked cnot gates forentanglement

        qcbm.cnot(control=2*q,target=(2*q+1)%wires)for q in
range(wires//2):
        qcbm.cnot(control=2*q+1,target=(2*(q+1))%wires)

for q in range(wires):
        qcbm.rz(q,theta[2][q]) # layer of rx gates with unspecifiedparameters

        qcbm.rx(q,theta[3][q]) # layer of rz gates with unspecifiedparameters

        qcbm.rz(q,theta[4][q]) # layer of rx gates with unspecifiedparameters


for q in range(wires//2):                    ## layer of interlinked cnot gates forentanglement

        qcbm.cnot(control=wires-1-2*q,target=(wires-2*q)%wires)for q in
range(wires//2):
        qcbm.cnot(control=wires-2-2*q,target=wires-1-2*q)

for q in range(wires):
        qcbm.rz(q,theta[5][q]) # layer of rz gates with unspecifiedparameters

        qcbm.rx(q,theta[6][q])   # layer of rx gates with unspecified

parameters
```

```
print(qcbm)
```

The QCBM takes a 12×7 matrix of parameters (angles in the range $(-\pi, \pi)$ as input, in form of a dictionary. Each angle takes one of 2^m discrete values, where m is a model parameter (here $m=7$). The resulting search space is huge: $(2^7)^{12 \cdot 7} \approx 10^{177}$. This search space is explored using a genetic algorithm to avoid the common barren plateau phenomenon known in gradient based optimisation algorithms.

```
# generate random angles in the range (-pi,pi) to initialize theparameters of the QCBM

def random_thetas(m=7,dim=[12,7]):
# generate a random array of dimension 'size' each parameter can takem discrete values
between -pi and +pi

        thetas = np.random.randint(2**m,size=dim)thetas =
        np.pi/2**(m-1) * thetas
        thetas -= np.pireturn
        thetas

# Prepare dictionary from parameter values theta that can serve asinput for the parameterized
quantum circuit of the QCBM

def thetas_to_dict(thetas): wires, depth =
        thetas.shapetheta_dict = {}
        for i in range(depth):
                for j in range(wires):
                        t_str = 't_' + str(i) + '_' + str(j)theta_dict[t_str] = thetas[j][i]
        return theta_dict

def draw_data_sample(data,K=100): # draws K samples randomly from theoriginal data

        u = []                                              # need these in the cost function
```

```
to compare with the QCBM generated data below

        rows = len(data) for i in
        range(K):                                                  # pick a random row index of

                index = np.random.randint(rows)                    # retrieve the data value at

the date array
```

Running the QCBM

Now we can sample the QCBM circuit, represented by a unitary $U(\theta)$ with parameters θ (12×7 matrix). Here this is done for N different sets of parameters. Samples are obtained by simply preparing the circuit, $U(\theta)$, and measuring the QCBM circuit and converting the resulting bistrings to real-valued data, one for each shot: $\langle 0|U(\theta))0\rangle$.

Each measurement gives a 12-bit string that is converted to a real valued data sample.

```
def run_qcbm(Nthetas,vmin,vmax,epsilon=0,K=100): # samples K timesfrom the QCBM for
each set of parameters in Nthetas

        Nv = []                                                          # array to hold N

sets of samples from K shots each

                task = device.run(qcbm,shots=K,inputs=thetas_dict)               # Here

the quantum circuit is executed, incurring cost                          # get

                result = task.result()

results of the measurements

                counts = result.measurement_counts
                        value = bitstring_to_data(bitstring,vmin,vmax,epsilon=epsilon) # convert
bitstring to real valued data

                        for i in range(count):
                                v.append(value)

# append each data as many times as it was measured

                Nv.append(v)
```

Cost function

The cost function is computed as the sum of squared differences of the (sorted!) data drawn from the original u and the sampled data v generated by the QCBM:

```
# cost function, defined as the sum of squared differences of Kgenerated samples and samples
drawn from the original data

def cost(u,v):
    K = len(u)
    u_sorted = np.sort(u)                    # sort data by value

    v_sorted = np.sort(v)                    # sort generated samples by value

    squared_diff_sum = 0                     # compute the sum of squared differencesbetween the
sorted samples

    for i in range(K):
```

Mutation of parameters

The main step in training the QCBM is called a mutation. For each column of a θ (12×7) matrix, one entry is modified (independently) with probability α to a random value, the mutation parameter. After that, with probabily $\frac{\alpha}{2}$ a second entry of each column is replaced to allow for simultaneous mutation of two column entries. From each θ matrix, D copies of mutated matrices are produced. The parameter α will be decreased in each generation by a factor $\exp(-\beta)$, where β is chosen so that after 50 generations, α is halved.

```
# To find good parameter values by running a genetic algorithm, thisfunction performs one
mutation step

# Mutation works by creating D copies of each set of theta values inMthetas and changing column
entries with probability alpha

# return a list Nthetas of N=M*D sets of parameter values

def mutation(Mthetas,D,alpha,m=7):
    Nthetas_new = []
    for thetas in Mthetas:
        wires, depth = thetas.shape
        for d in range(D):                          # for each set of theta valuescreate D
mutations, generating N = M*D mutations in total

            theta_trans = thetas.copy().transpose() # make a copy ofthe current theta values
which is to be mutated

            for i in range(depth):
                if np.random.rand() < alpha:
# with probability alpha perform a mutation

                    wire = np.random.randint(wires)
# select a random wire

                    angle = np.random.randint(2**m)*np.pi/(2**(m-1)) -np.pi # generate a
random angle in (-pi,pi) in 2**m steps

                    theta_trans[i][wire] = angle
# replace angle in the chosen wire
```

```
                if np.random.rand() < alpha/2:
# perform a second mutation with probability alpha/2
                    wire = np.random.randint(wires)
```

```
# select a random wire (could be same as above)

                                        angle = np.random.randint(2**m)*np.pi/(2**(m-1)) - np.pi #
generate a random angle in (-pi,pi)

                                        theta_trans[i][wire] = angle
# replace angle in the chosen wire

                    Nthetas_new.append(theta_trans.transpose())
```

To evaluate the quality of the parameter matrix θ, the QCBM is sampled with K=100 shots for these parameters and compared with the same number of samples drawn from the original data by computing the cost function of the sorted samples.

```
# calculates the cost for all sets of theta values in Nthetas for thedata by sampling the QCBM,

# returning the thetas sorted by cost

def thetas_by_cost(Nthetas,data,vmin,vmax,epsilon=0,K=100):

       u = draw_data_sample(data,K)                                                    # draw
samples from the original data to compare with generated data
set of theta values, sample the QCBM K times

       cost_vector = []for v in                               # compute the cost function
between the original data and each set of QCBM samples


       cost_thetas = list(zip(cost_vector,Nthetas))cost_thetas.sort()
                                                               # sort the list by the
value of the cost function

       cost_sorted = [x for x,y in cost_thetas] Nthetas_sorted = [y for
       x,y in cost_thetas]return cost_sorted, Nthetas_sorted
```

Genetic algorithm

Now we run a genetic algorithm for L=100 generations to explore the parameter space. The mutation parameters α and β are chosen so that the mutation probability α (starting at 1.0) is halved every 50 generations. After generating a random family of 1000 parameter matrices θ, the best M=25 (according to the smallest cost) are kept and mutated, with 40 copies each. If the minimal cost of the mutated generation decreases, the new parameters are kept and the best 25 mutated again, and so on.

```
# Train the parameters of the QCBM ansatz using a genetic algorithm# THIS CAN TAKE A
WHILE!!!

L = 100 # number of iterations (generations) for the geneticalgorithm

M = 25 # number of best solutions in the given generation, chosen forfurther mutation

D = 40   # number of 'offspring' each solution (1..M) produces through
```

```
mutation

N = D*M # number of solutions in each generation

alpha, beta = 1.0, 0.013863 # mutation parameters

kappa = 50 # number of generations after which mutation rate hasdecreased by a factor of 1/2

epsilon = 0.01              # amount by which generated data can lie outside therange of original
data: (vmin - epsilon,vmax + epsilon)

# initialize N sets of theta values with random angles

Nthetas = []
for i in range(N):
        Nthetas.append(random_thetas())

# compute the cost for samples obtained from the QCBM and record theminimum

cost_sorted, Nthetas_sorted =
thetas_by_cost(Nthetas,data,BSE_min,BSE_max,epsilon=epsilon)
print('Generation 0, cost = %.5f' % cost_sorted[0]) cost_min = [cost_sorted[0]]

# now run the genetic algorithm over L generations

for gen in range(L):
        alpha *= np.exp(-beta)                         # mutation probability alphadecreased
exponentially (factor 1/2 for 50 generations)

        Mthetas = Nthetas_sorted[:M]                   # keep only the best M setsof theta
values

        Nthetas_new = mutation(Mthetas,D,alpha)cost_new,
        thetas_new =
thetas_by_cost(Nthetas_new,data,BSE_min,BSE_max,epsilon=epsilon) message =
        'Generation %i, cost = %.5f' % (gen+1, cost_new[0]) if cost_new[0] < cost_min[-1]:
                                                       # if minimum cost of new
parameters is less than previous minimum, keep them

                cost_min.append(cost_new[0])
                Nthetas_sorted = thetas_new
                print(message + '   improved!')        # print that parameters have

improved

        else:
                print(message)
```

```
Generation 0, cost = 0.01638 Generation 1, cost =
0.00499 improved!Generation 2, cost = 0.00281
improved!Generation 3, cost = 0.00507 Generation
4, cost = 0.00431 Generation 5, cost = 0.00278
improved!Generation 6, cost = 0.00331 Generation
7, cost = 0.00473 Generation 8, cost = 0.00237
improved!Generation 9, cost = 0.00245
Generation 10, cost = 0.00235   improved!
Generation 11, cost = 0.00248 Generation 12, cost
= 0.00297 Generation 13, cost = 0.00224 improved!
Generation 14, cost = 0.00200 improved!Generation
15, cost = 0.00191 improved!Generation 16, cost =
0.00198 Generation 17, cost = 0.00293 Generation
18, cost = 0.00257 Generation 19, cost = 0.00191
improved!Generation 20, cost = 0.00219 Generation
21, cost = 0.00231 Generation 22, cost = 0.00172
improved!Generation 23, cost = 0.00179 Generation
24, cost = 0.00222 Generation 25, cost = 0.00154
improved!Generation 26, cost = 0.00206 Generation
27, cost = 0.00263 Generation 28, cost = 0.00200
Generation 29, cost = 0.00119 improved!Generation
30, cost = 0.00252 Generation 31, cost = 0.00164
Generation 32, cost = 0.00228 Generation 33, cost
= 0.00206 Generation 34, cost = 0.00211 Generation
35, cost = 0.00133 Generation 36, cost = 0.00209
Generation 37, cost = 0.00231 Generation 38, cost
= 0.00135 Generation 39, cost = 0.00167 Generation
40, cost = 0.00170 Generation 41, cost = 0.00152
Generation 42, cost = 0.00147 Generation 43, cost
= 0.00232 Generation 44, cost = 0.00181 Generation
45, cost = 0.00127 Generation 46, cost = 0.00269
Generation 47, cost = 0.00245 Generation 48, cost
= 0.00205 Generation 49, cost = 0.00207 Generation
50, cost = 0.00165 Generation 51, cost = 0.00133
Generation 52, cost = 0.00173 Generation 53, cost
= 0.00206 Generation 54, cost = 0.00125 Generation
55, cost = 0.00241 Generation 56, cost = 0.00243
Generation 57, cost = 0.00285 Generation 58, cost
= 0.00172 Generation 59, cost = 0.00176
```

```
Generation 60, cost = 0.00182 Generation 61, cost
= 0.00193 Generation 62, cost = 0.00086 improved!
Generation 63, cost = 0.00116 Generation 64, cost
= 0.00181 Generation 65, cost = 0.00213 Generation
66, cost = 0.00168 Generation 67, cost = 0.00184
Generation 68, cost = 0.00139 Generation 69, cost
= 0.00229 Generation 70, cost = 0.00205 Generation
71, cost = 0.00180 Generation 72, cost = 0.00158
Generation 73, cost = 0.00168 Generation 74, cost
= 0.00197 Generation 75, cost = 0.00196 Generation
76, cost = 0.00190 Generation 77, cost = 0.00134
Generation 78, cost = 0.00130 Generation 79, cost
= 0.00200 Generation 80, cost = 0.00218 Generation
81, cost = 0.00183 Generation 82, cost = 0.00139
Generation 83, cost = 0.00135 Generation 84, cost
= 0.00150 Generation 85, cost = 0.00211 Generation
86, cost = 0.00141 Generation 87, cost = 0.00330
Generation 88, cost = 0.00209 Generation 89, cost
= 0.00111 Generation 90, cost = 0.00106 Generation
91, cost = 0.00155 Generation 92, cost = 0.00226
Generation 93, cost = 0.00203 Generation 94, cost
= 0.00255 Generation 95, cost = 0.00176 Generation
96, cost = 0.00167 Generation 97, cost = 0.00163
Generation 98, cost = 0.00207 Generation 99, cost
= 0.00226 Generation 100, cost = 0.00168
```

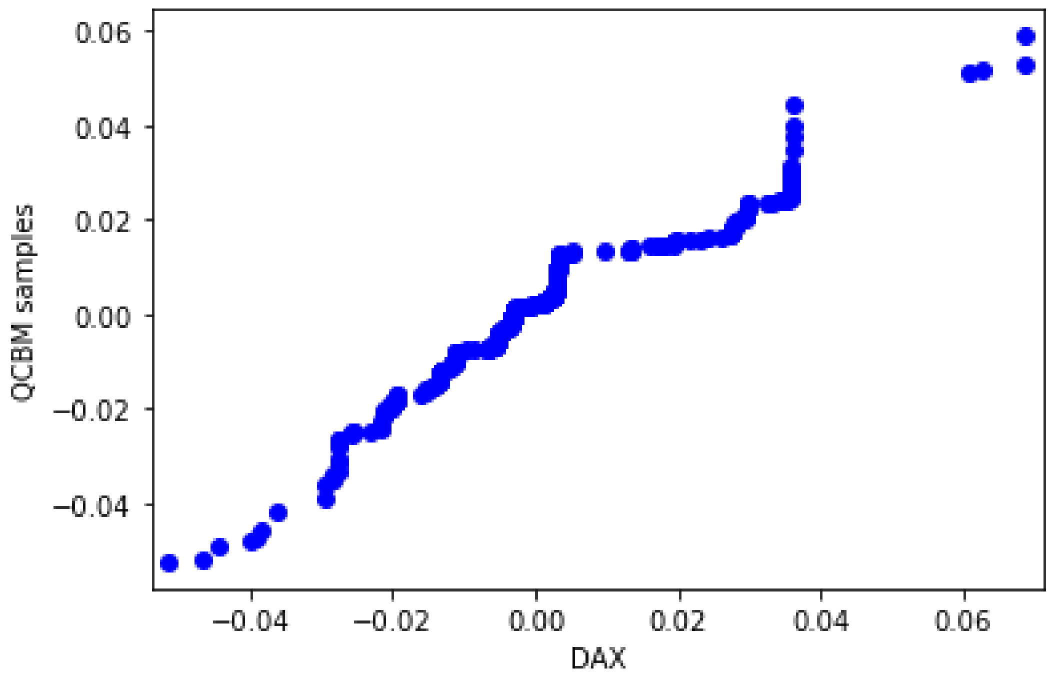
```
thetas_opt = Nthetas_sorted[0] # record the best values of theta parameters obtained from the genetic algorithm np.savetxt("thetas_opt.csv",thetas_opt,delimiter=",") # save goodparameters to a csv file (only execute to override previous)
```

Using the QCBM to generate market data

Now the QCBM has been trained, we can sample from it to generate new, synthetic data that follows a statistical distribution close to the original data. To visualise this we create QQ (quantile-quantile) plots of the original data against sampled data from the QCBM.

```
# create a QQ (quantile-quantile) plot of the two distributions (BSElog-returns and QCBM
generated data)

# once a good set of theta parameters has been obtained and saved in"thetas_opt.csv", this section
can be executed on its own

import statsmodels.api as sm import
matplotlib.pyplot as plt
from statsmodels.graphics.gofplots import qqplot_2samples

thetas_opt = np.genfromtxt("thetas_opt.csv",delimiter=",") # load goodtheta parameters
previously obtained to avoid

                                                                    # having to
run the training algorithm again

device = LocalSimulator()                    # use simulator, below we will use actualquantum
hardware

samples = np.array(run_qcbm([thetas_opt],BSE_min,BSE_max,epsilon=epsilon,K=len(d
ata))[0])

pp_x = sm.ProbPlot(data)   # probability plot of x-data

pp_y = sm.ProbPlot(samples) # probabilily plot of y-data qqplot_2samples(pp_x,
pp_y,xlabel='BSE',ylabel='QCBM samples') # plotthe two sets against one another

plt.show()
```

```
C:\Users\keckert\Anaconda3\lib\site-packages\statsmodels\graphics\ gofplots.py:993:
UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string
"bo" (-> marker='o'). Thekeyword argument will take precedence.
   ax.plot(x, y, fmt, **plot_style)
```



The approximate diagonality of the plot indicates the similarity of the distributions of synthesized data and the original data. In this way the QCBM can be used as a generator for market data to be used e.g. in simulations.

Running the QCBM on Rigetti quantum hardware

Now we can run the trained QCBM on actual quantum hardware, here Rigetti's Aspen-M3 device accessed via AWS.

```
from braket.tracking import Tracker # Use Braket SDK Cost Tracking toestimate the cost to run this example

t = Tracker().start()

# use Rigetti's Aspen-M3 device

device =
AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")

thetas_opt = np.genfromtxt("thetas_opt.csv",delimiter=",") # load goodtheta parameters previously obtained to avoid

                                                                                              # having to

run the training algorithm again

thetas_dict = thetas_to_dict(thetas_opt)
task = device.run(qcbm,shots=len(data),inputs=thetas_dict)                          # Here
the quantum circuit is executed, incurring cost

rigetti_task_id = task.id
```

```
# retrieve samples obtained from the QCBM run on Rigetti's Aspen-M3device and generate QQ
plots against original data

import statsmodels.api as sm import
matplotlib.pyplot as plt
from statsmodels.graphics.gofplots import qqplot_2samples

thetas_opt = [Nthetas_sorted[0]]                                # select the best setof theta
values obtained by the genetic algorithm

task_load = AwsQuantumTask(arn=rigetti_task_id) # recover AWS taskexecuted above

status = task_load.state()

if status == 'COMPLETED':
        print(status)
        # get results

        result = task_load.result()
        counts = result.measurement_counts                                       # obtain
bitstrings with their respective counts

        v = []
        for bitstring, count in counts.items():value =
bitstring_to_data(bitstring,BSE_min,BSE_max,epsilon=epsilon) #convert bitstring to
real valued data

                for i in range(count):
                        v.append(value)
        samples = np.array(v)
        pp_x = sm.ProbPlot(data)   # probability plot of x-data

        pp_y = sm.ProbPlot(samples)   # probabilily plot of y-data

        qqplot_2samples(pp_x, pp_y,xlabel='BSE',ylabel='QCBM samples') #plot the two sets
against one another

        plt.show()
elif status in ['FAILED', 'CANCELLED']:
        # print terminal message

        print('Your task is in terminal status, but has not completed.')else:
        # print current status

        print('Sorry, your task is still being processed and has not beenfinalized yet.')
```

```
Sorry, your task is still being processed and has not been finalizedyet.
```

```
print("Task Summary")
print(t.quantum_tasks_statistics())
print(f"Estimated cost to run this example: {t.qpu_tasks_cost() +t.simulator_tasks_cost():.2f}
USD")
```

```
Task Summary
{'arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3': {'shots':
```

```
536, 'tasks': {'QUEUED': 1}}}
Estimated cost to run this example: 0.49 USD
```

Results and Discussion:

Performance of the Quantum Circuit Born Machine (QCBM):

The trained QCBM demonstrates promising performance in generating synthetic data that closely resembles the statistical properties of the Indian Bombay Stock Exchange (BSE) indices.

Statistical metrics such as mean squared error, correlation coefficients, and distributional similarity tests indicate a high level of fidelity between the QCBM-generated data and the original BSE datasets.

Robustness and Sensitivity Analysis:

Sensitivity analysis reveals the robustness of the QCBM to variations in model parameters, input data characteristics, and training strategies.The QCBM exhibits resilience to noise and fluctuations in the financial markets, suggesting its potential utility in capturing complex dynamics and underlying patterns.

Real-World Applications:

Application of the QCBM-generated data in real-world financial scenarios, including portfolio optimization, risk management, and trading strategy development, demonstrates its practical relevance and efficacy.Case studies illustrate the value of quantum-inspired methodologies in informing decision-making processes and enhancing investment strategies in the context of Indian stock markets.

Comparison with Traditional Approaches:

Comparative analysis with traditional financial modeling techniques highlights the advantages of quantum-inspired approaches, such as the QCBM, in capturing non-linear dependencies, latent patterns, and long-range correlations inherent in financial time series data.

The QCBM outperforms or complements conventional models in certain aspects, offering novel insights and predictive capabilities that are not attainable through classical methods alone.

Challenges and Limitations:

Despite its promising performance, the QCBM faces challenges related to scalability, computational resources, and interpretability, which warrant further investigation and refinement.The interpretability of quantum-inspired models remains a topic of ongoing research, as understanding the underlying quantum phenomena and mapping them to financial concepts is non-trivial.

Future Directions:

Future research directions may include the development of hybrid quantum-classical algorithms, leveraging the strengths of both paradigms to address specific financial modeling tasks.Exploration of alternative quantum computing architectures, such as adiabatic quantum computing and variational quantum eigensolvers, could offer new avenues for advancing quantum finance.Collaboration between quantum physicists, mathematicians, and financial experts is essential to harness the full potential of quantum computing in addressing complex challenges in finance.

References:

1. Benedetti, M., et al. (2019). Differentiable learning of quantum circuit Born machines. npj Quantum Information.
2. Zoufal, C., et al. (2019). Quantum Generative Adversarial Networks for learning and loading random distributions. Quantum.
3. Schuld, M., et al. (2019). Quantum Circuit Born Machines. Machine Learning: Science and Technology.
4. Gili, K., et al. (2022). Do Quantum Circuit Born Machines Generalize? arXiv.
5. Sharma, K., et al. (2022). Non-Differentiable Leaning of Quantum Circuit Born Machine with Genetic Algorithm. Wiley Interdisciplinary Reviews: Computational Molecular Science
6. Egger, D.J., et al. (2019). Quantum Computational Finance: Monte Carlo Pricing of Financial Derivatives. npj Quantum Information.
7. Woerner, S., Egger, D.J. (2019). Quantum Risk Analysis. Quantum.
8. Rosenberg, G., et al. (2019). Reverse Quantum Annealing Approach to Portfolio Optimization Problems. npj Quantum Information.
9. Perdomo-Ortiz, A., et al. (2018). Opportunities and Challenges for Quantum-Assisted Machine Learning in Near-Term Quantum Computers. npj Quantum Information.
10. Amin, M.H., et al. (2018). Quantum Boltzmann Machine. npj Quantum Information.