Quantum Mechanics: A Non-Local Theory

Nathan Smith and Christopher Jacobs

1 Overview

The goal of this project is to numerically simulate the classical and quantum mechanical CHSH correlator, and analyze the physical significance of each result. We do this by examining two different scenarios denoted as Problem 3.1 and Problem 3.2 in the following sections. The solution to both of these problems point to the same conclusion: quantum mechanics contains no hidden local variables that can account for the measurement correlation between two entangled particles measured at a distance, in different lab setups. Consequently, we are forced to conclude that quantum mechanics is fundamentally a non-local theory.

2 Problem 3.1

Problem 3.1 presents our beloved scientists, Alice and Bob, measuring the angular momentum of classical particles. They set up a micro explosion that sends two particles to opposite sides of the lab, Alice receives a fragment with angular momentum J, while Bob receives another fragment with angular momentum $-J$.

For each fragment, Alice will choose a direction α_i along which to measure her fragment. Bob will do the same, choosing a direction β_j. The outcome of each measurement will be $sign[\alpha_i \cdot J]$ and $-sign[\beta_j \cdot J]$ respectively. For the first part of this problem, we calculate the CHSH correlator for the classical case of Alice and Bob's micro-explosion experiment.

2.1 Formalism for the Quantum Case

For the second part of this problem, we assume that Alice and Bob are measuring a singlet state of spin-1/2 particles. We do this calculation by using the correlator:

$$\langle a_x b_y \rangle = \sum_{ab} ab \; \langle a_x b_y | \Pi_a^{\pm}(\theta_a) \otimes \Pi_b^{\pm}(\theta_b) | a_x b_y \rangle \tag{1}$$

Where the general projection operator Π is:

$$\hat{\Pi}_{\hat{n}}^{\pm}(\pm) = \frac{1}{2}(I \pm \vec{\sigma} \cdot \vec{n}) \tag{2}$$

Now the CHSH correlator is defined as:

$$S_{xyx'y'}(\theta) = \langle a_x b_y \rangle + \langle a_x b_{y'} \rangle + \langle a_{x'} b_y \rangle - \langle a_{x'} b_{y'} \rangle \tag{3}$$

Which is what we calculate for both the quantum case and the classical case, and use Equation (1) for the quantum case. We use the exact same method from the previous homework.

2.2 Formalism for the Classical Case

We also calculate the CHSH correlator in the classical situation, except we do this via numerical simulation.

The code iterates through a discreet set of angles θ and calculates $S_{xyx'y'}(\theta)$. Each experiment generates random momentum vectors in the $x - z$ plane, and performs a measurement along the unit vectors β_i and α_i discussed above.

3 Results

Figure 1 Shows the calculation of $S_{xyx'y'}(\theta)$ with parameters:

$x_1 \rightarrow \phi = 0 \quad x_2 \rightarrow \phi = \frac{3}{4}\theta \quad y_1 \rightarrow \phi = 0 \quad y_2 \rightarrow \phi = 3\theta$

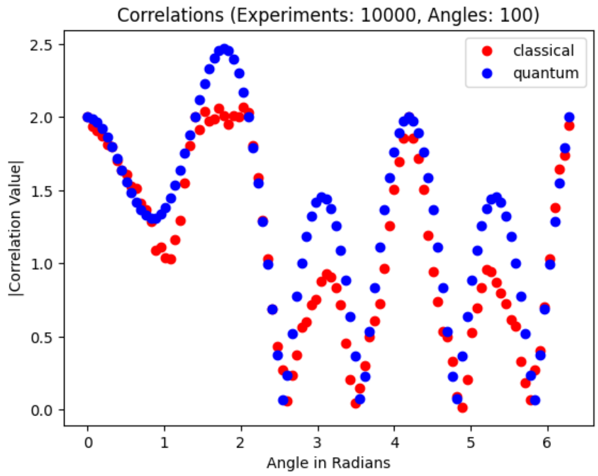

Figure 1:

For the classical case, we got a maximum value of $S \approx 2.04$. We expect a maximum of 2 for the classical correlator because classical theory is a local theory, but we get above 2 because we perform finite measurements. This can be observed by the hard capping behavior of the classical data between 1 and 2 radians in figure 1. For the quantum case, we get a maximum value of $S \approx 2.5$, see figure 1 at 1.57 radians.

We also explored the space of possible combinations of ratios between measurement settings θ and produced histogram plots of the maximum correlator value for the classical and quantum case. Notice again in Figure 2 that S exceeds 2 by less than 0.1. However, Figure 3 clearly shows a violation of the inequality $S \geq 2$ for the quantum case, showing quantum theory as non-local.

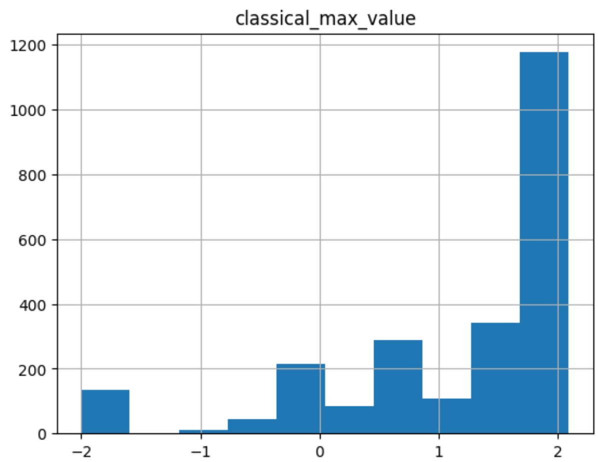

Figure 2: x-axis: Correlator Value S, y-axis: Count

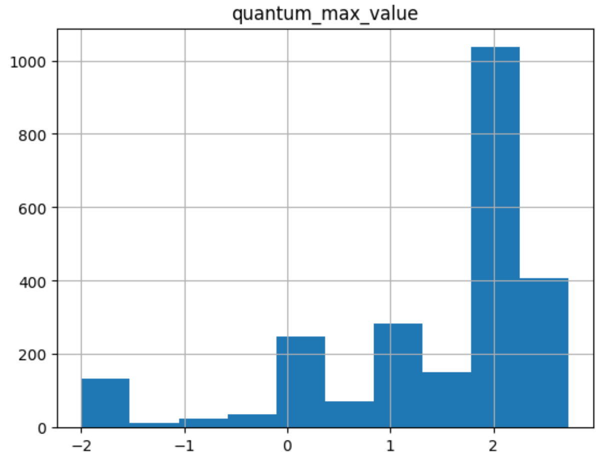

Figure 3: x-axis: Correlator Value S, y-axis: Count

4 Problem 3.2

The Bell-Mermin model for quantum mechanics is a local hidden variable model for quantum theory. Measurement in this model is very much the same as in problem 3.1, that is 2 scientists have two settings for their set up and both of them separately coin flip between them prior to each measurement. Preparation is different from problem 1 in the fact that in this model the measurable quantity (a singlet state) is described by 2 unit vectors. One of the describing vectors is related to a quantum state vector which is produced randomly (much like a random angular momentum vector is produced above) and represents a spin projection along an axis. Though out the experiment this random quantum state vector remains constant while the second describing vector is produced randomly (N times depending on how many measurements are to be taken). The outcome of the measurement is given by a probability distribution related to the dot product between the measurement vector and the sum of the two vectors describing the measurable quantity.

5 Motivation

While problem 3.2 required verifying that the probabilities given in the Bell-Mermin models were the same as in the quantum case, this was forgone as the main focus of this analysis was from problem 1. The main purpose of this further analysis was to gain an understanding of this other model and to concretely show that quantum mechanics cannot be explained by a local model as there is a maximum of the Bell-Mermin correlator at 2 (above 2 due to using a finite amount of measurements like in problem 3.1).

6 Results

Figure 4 Shows the data collected from the simulation of $S_{xyx'y'}(\theta)$ (both the Bell-Mermin and quantum correlator) with parameters:

x$_1 \rightarrow \phi = 0 \quad x_2 \rightarrow \phi = \frac{3}{4}\theta \quad y_1 \rightarrow \phi = 0 \quad y_2 \rightarrow \phi = 3\theta$

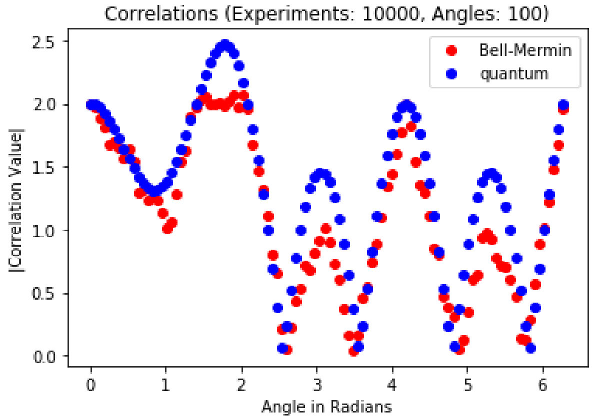

Figure 4: x-axis: Angle in Radians S, y-axis: Correlator Value

From the Bell-Mermin model we see that the correlation value for this set of parameters is exactly the same as the classical case from problem 3.1 (compare with figure 1). This is because the sum of the two description vectors is acting similarly to the angular momentum in problem 3.1. The graph also visually supports that the Bell-Mermin model is a local theory with the capping behavior from 1 to 2 radians.

7 Further Exploration

Effort was put into using Qiskit (a way to interact with one of IBM's quantum computers) to simulate data for the quantum experiment using a quantum computer to simulate singlet states. This could have been used to experimentally produce the quantum correlater curves.

Another interesting exploration could be to scan through a range of ratios between measurement angles (similarly to section 3) between the classical correlator and the Bell-Mermin correlator to explore if any of the ratios would produce significantly different graphs.

8 Codes

8.1 Problem 3.1

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import random
from itertools import permutations, combinations, combinations_with_replacement
fromcasesh.progress import track
from rich.progress import Progress
from joblib import Parallel, delayed


def gen_momentum_vectors(N):
    #generate random 2d, unit vectors ---> (x, z)
    x_vals = np.random.uniform(-1, 1, size = N)
    z_vals = np.random.uniform(-1, 1, size = N)

    J_plus = np.asarray([np.asarray([x, z]).reshape(2,1) for x, z in zip(x_vals, z_vals)
    ])
    J_minus = np.asarray([-1*Jp for Jp in J_plus])

    return [J_plus, J_minus]


def gen_measurement_vector(theta):
    #generate a measurement vector --> (x,z)
    #functions assume radians
    components = np.array(([np.sin(theta),np.cos(theta)]))
    magnitude = math.sqrt(sum(pow(element, 2) for element in components))
    unit_vector = components/magnitude
    return unit_vector


def check_thetas(thetas, val):
    mask = [theta == val for theta in thetas]
    return any(mask)



def experiment(number_of_momentum_vectors, theta_11 = None, theta_12 = None, theta_21 =
    None, theta_22 = None):
    thetas = [theta_11, theta_12, theta_21, theta_22]

    if check_thetas(thetas, None):
        raise ValueError(f"Thetas cannone be None. Thetas: {thetas}")

    #creates 4 experiement vectors from input theta
    alpha_1 = gen_measurement_vector(theta_11)
    alpha_2 = gen_measurement_vector(theta_12)
    beta_1 = gen_measurement_vector(theta_21)
    beta_2 = gen_measurement_vector(theta_22)

    #used to calcuate correlators
    alpha_1_beta_1 = []
    alpha_2_beta_1 = []
    alpha_1_beta_2 = []
    alpha_2_beta_2 = []

    #creates N momentum vectors
    momentum_vectors = gen_momentum_vectors(number_of_momentum_vectors)

    #does experiment
    for up_momentum_vectors in momentum_vectors[0]:
        #randomly chooses an alpha and a beta
        #print(np.random.uniform(0, 1, size = 1)[0])
        counter_alpha = 0
        counter_beta = 0
        if random.randint(1, 2)  == 1:
            measurement_1 = alpha_1
            counter_alpha += 1
```

```
        else:
            measurement_1 = alpha_2
            counter_alpha += 2

        if random.randint(1, 2) == 1:
            measurement_2 = beta_1
            counter_beta += 1
        else:
            measurement_2 = beta_2
            counter_beta += 2
        #normalize momentum vectors
        components = np.array(up_momentum_vectors)
        magnitude = math.sqrt(sum(pow(element, 2) for element in components))
        unit_vector = components/magnitude



        #measurements
        a_alpha_i = np.sign(np.dot(measurement_1,unit_vector))
        b_beta_i = -np.sign(np.dot(measurement_2,unit_vector))

        #print(a_alpha_i,b_beta_i)
        measurement = int(a_alpha_i[0])*int(b_beta_i[0])


        #sorts measruements into 4 correlators to later calculate CHSH correlator

        if counter_alpha == 1 and counter_beta == 1:
            alpha_1_beta_1.append(measurement)
        elif counter_alpha == 1 and counter_beta == 2:
            alpha_1_beta_2.append(measurement)
        elif counter_alpha == 2 and counter_beta == 1:
            alpha_2_beta_1.append(measurement)
        elif counter_alpha == 2 and counter_beta == 2:
            alpha_2_beta_2.append(measurement)

    #print(alpha_1_beta_1,alpha_1_beta_2,alpha_2_beta_1,alpha_2_beta_2)


    correlator_11 = np.sum(alpha_1_beta_1)/len(alpha_1_beta_1)
    correlator_12 = np.sum(alpha_1_beta_2)/len(alpha_1_beta_2)
    correlator_21 = np.sum(alpha_2_beta_1)/len(alpha_2_beta_1)
    correlator_22 = np.sum(alpha_2_beta_2)/len(alpha_2_beta_2)

    #print(correlator_11,correlator_12,correlator_21,correlator_22)
    S_classic = correlator_11 + correlator_12 + correlator_21 - correlator_22

    S_quantum = -np.cos(theta_11-theta_21)-np.cos(theta_11-theta_22)-np.cos(theta_12-theta_21)+np.cos(theta_12-theta_22)


    return [S_classic, S_quantum]

y_axis_classical = []
y_axis_quantum = []
x_axis = []
number_of_experiments = 4000 #has to be big,N->infinity
angle_divisions = 100

for angle in np.linspace(0,2*np.pi,angle_divisions):

    angles = {
    "theta_11" : 0*angle,
    "theta_12" : .75*angle,
    "theta_21" :  0 * angle,
    "theta_22" : 3*angle
    }

    """
    angles = {
    "theta_11" : 0*angle,
    "theta_12" : 2*angle,
    "theta_21" : 1 * angle,
```

```
    "theta_22" : 3*angle
    }
    """

    #print(angles)


    classic, quantum = experiment(number_of_experiments, **angles)
    #print(classic,quantum)
    #print("Correlator: ", experiment(angle,2000), "Angle(in radians): ", angle)
    x_axis.append(angle)
    y_axis_classical.append(classic)
    y_axis_quantum.append(quantum)
    #print(experiment(angle,number_of_experiments), angle)

#### Generating Plots

plt.xlabel('Angle in Radians')
plt.ylabel('|Correlation Value|')
plt.title(f'Correlations (Experiments: {number_of_experiments}, Angles: {angle_divisions})')
plt.plot(x_axis,np.abs(y_axis_classical),'ro')
plt.plot(x_axis,np.abs(y_axis_quantum),'bo')
plt.legend(["classical", "quantum"])
```

Listing 1: Simulation Code

8.2 Problem 3.2

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import random

def gen_hidden_vectors(N):
    #generate random 2d, unit vectors ---> (x, z)
    x_vals = np.random.uniform(-1, 1, size = N)
    z_vals = np.random.uniform(-1, 1, size = N)

    hidden_v = np.asarray([np.asarray([x, z]).reshape(2,1) for x, z in zip(x_vals, z_vals)])


    return hidden_v

def gen_measurement_vector(theta):
    #generate a measurement vector --> (x,z)
    #functions assume radians
    components = np.array(([np.sin(theta),np.cos(theta)]))
    magnitude = math.sqrt(sum(pow(element, 2) for element in components))
    unit_vector = components/magnitude
    return unit_vector

def gen_n_vector(N):
    #generate a measurement vector --> (x,z)
     #generate random 2d, unit vectors ---> (x, z)
    x_vals = np.random.uniform(-1, 1, size = N)
    z_vals = np.random.uniform(-1, 1, size = N)
    n = np.asarray([np.asarray([x, z]).reshape(2,1) for x, z in zip(x_vals, z_vals)])
    return n


def check_thetas(thetas, val):
    mask = [theta == val for theta in thetas]
    return any(mask)



def experiment(number_of_runs, theta_11 = None, theta_12 = None, theta_21 = None, theta_22 = None):
    thetas = [theta_11, theta_12, theta_21, theta_22]
```

```python

    if check_thetas(thetas, None):
        raise ValueError(f"Thetas cannone be None. Thetas: {thetas}")

    #creates 4 experiement vectors from input theta
    alpha_1 = gen_measurement_vector(theta_11)
    alpha_2 = gen_measurement_vector(theta_12)
    beta_1 = gen_measurement_vector(theta_21)
    beta_2 = gen_measurement_vector(theta_22)


    #used to calcuate correlators
    alpha_1_beta_1 = []
    alpha_2_beta_1 = []
    alpha_1_beta_2 = []
    alpha_2_beta_2 = []


    #creates +1 measurement vector and u vectors
    n_axis = gen_n_vector(1)[0]
    print(n_axis[0])
    components = np.array(n_axis)
    magnitude = math.sqrt(sum(pow(element, 2) for element in components))
    n_axis = components/magnitude
    u1_vector = -1*n_axis
    u2_vector = -1*u1_vector

    #does experiment
    for measurement in range(0,number_of_runs):
        #randomly chooses an alpha and a beta
        #print(np.random.uniform(0, 1, size = 1)[0])
        counter_alpha = 0
        counter_beta = 0
        if random.randint(1, 2)  == 1:
            measurement_1 = alpha_1
            counter_alpha += 1
        else:
            measurement_1 = alpha_2
            counter_alpha += 2
        if random.randint(1, 2) == 1:
            measurement_2 = beta_1
            counter_beta += 1
        else:
            measurement_2 = beta_2
            counter_beta += 2
        #generates random hidden variable
        v1_vector = gen_hidden_vectors(1)[0]
        #normalize v_vector
        components = np.array(v1_vector)
        magnitude = math.sqrt(sum(pow(element, 2) for element in components))
        v1_vector = components/magnitude
        v2_vector = -1*v1_vector




        #measurements
        a_alpha_i = np.sign(np.dot(measurement_1,u1_vector+v1_vector))
        b_beta_i = np.sign(np.dot(measurement_2,u2_vector+v2_vector))
        measurement = a_alpha_i[0]*b_beta_i[0]


        #sorts measruements into 4 correlators to later calculate CHSH correlator

        if counter_alpha == 1 and counter_beta == 1:
            alpha_1_beta_1.append(measurement)
        elif counter_alpha == 1 and counter_beta == 2:
            alpha_1_beta_2.append(measurement)
        elif counter_alpha == 2 and counter_beta == 1:
            alpha_2_beta_1.append(measurement)
        elif counter_alpha == 2 and counter_beta == 2:
            alpha_2_beta_2.append(measurement)

```

```
    #print(alpha_1_beta_1,alpha_1_beta_2,alpha_2_beta_1,alpha_2_beta_2)


    correlator_11 = np.sum(alpha_1_beta_1)/len(alpha_1_beta_1)
    correlator_12 = np.sum(alpha_1_beta_2)/len(alpha_1_beta_2)
    correlator_21 = np.sum(alpha_2_beta_1)/len(alpha_2_beta_1)
    correlator_22 = np.sum(alpha_2_beta_2)/len(alpha_2_beta_2)

    S_classic = correlator_11 + correlator_12 + correlator_21 - correlator_22

    S_quantum = -np.cos(theta_11-theta_21)+np.cos(theta_11-theta_22)-np.cos(theta_12-
    theta_21)-np.cos(theta_12-theta_22)


    return S_classic, S_quantum

y_axis_classical = []
y_axis_quantum = []
x_axis = []
number_of_experiments = 2000
angle_divisions = 100
for angle in np.linspace(0,2*np.pi,angle_divisions):

    angles = {
    "theta_11" : 0*angle,
    "theta_12" : .75*angle,
    "theta_21" :  angle,
    "theta_22" : 3*angle
    }

    #print(angles)


    classic, quantum= experiment(number_of_experiments, **angles)
    #print(classic,quantum)
    #print("Correlator: ", experiment(angle,2000), "Angle(in radians): ", angle)
    x_axis.append(angle)
    y_axis_classical.append(classic)
    y_axis_quantum.append(quantum)
    #print(experiment(angle,number_of_experiments), angle)

plt.xlabel('Angle in Radians')
plt.ylabel('|Correlation Value|')
plt.title(f'Correlations (Experiments: {number_of_experiments}, Angles: {angle_divisions
    })')
plt.plot(x_axis,np.abs(y_axis_classical),'ro')
plt.plot(x_axis,np.abs(y_axis_quantum),'bo')
plt.legend(["Bell-Mermin", "quantum"])
```

Listing 2: Simulation Code