A Project Report

on

Quantum Random Walk

by

Angad Bawa

2021B5A73171H

Saksham Attri

2021A7PS2950H

Aryaman Kushwaha

2021B5AA2412H

Under the supervision of

Prof. Tanay Nag

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS OF

PHY F376: DESIGN PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(December - 2024)

ACKNOWLEDGMENTS

We are grateful to our educational institution, BITS Pilani Hyderabad Campus, for allowing us to be part of this project. We sincerely thank our mentor, Prof. Tanay Nag, for agreeing to oversee this project and providing guidance whenever needed. Working under their mentorship has been a highly instructive experience.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "Quantum Random Walk and Resetting " submitted by Mr. Angad Bawa (ID No. 2021B5A73171H), Mr. Saksham Attri (ID No. 2021A7PS2950H) and Mr. Aryaman Kushwaha (ID No. 2021B5AA2412H) in fulfillment of the requirements of the course PHY F376, Design Project Course, embodies the work done by them under my supervision and guidance.

Date: (Prof. Tanay Nag)

BITS- Pilani, Hyderabad Campus

ABSTRACT

This project investigates the behaviour of quantum random walkers (QRWs) under resetting mechanisms. Using a model governed by space and spin degrees of freedom via unitary transformations, we analyse QRWs in one- and two-dimensional systems and compare their dynamics to classical random walks to evaluate improvements in search efficiency. Additionally, the study focuses on one-dimensional QRWs with Initial Position Resetting strategies, demonstrating their potential to enhance detection probabilities and reduce search times significantly.

Contents

1. Introduction to Quantum Random Walk

1.1 Defining the operator

The quantum random walk (QRW) model can be understood by first considering a particle moving along a one-dimensional line, as described in *Quantum Random Walks: An Introductory Overview* by J. Kempe (2024).[1] The particle's behaviour is governed by two spin degrees of freedom and its spatial position. The particle's quantum state is represented as a tensor product of its spatial state and spin state. The evolution of the particle incorporates a combination of randomization processes that influence its movement decisions. These randomizers place the particle's position and spin states into superposition, allowing it to simultaneously exist in multiple state-spin combinations. The evolution is achieved through the application of a unitary operator, constructed as the product of position and spin randomization operators. Consequently, the final superposition state encodes the probability of the particle being in each possible position and spin combination, forming the foundation of the quantum random walk dynamics.

The position of the particle is defined across a one-dimensional discrete lattice, represented by an (n,1) vector with a non-zero value at the origin. For the spin, the particle's state is described using two basis vectors corresponding to the spin-up (+½) and spin-down (-½) states. The spin state of the particle is then expressed as a normalized linear combination of these two basis states, capturing the quantum superposition of spin orientations.

$$|\uparrow\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |\downarrow\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

To introduce randomization in the spin state, we utilize a rotation matrix **R(θ),** which transforms the spin state based on a specified angle θ. This matrix acts as a spin randomizer, creating a probabilistic distribution over the spin states.

For position randomization, we define an operator **S**, designed to modify the particle's position on the lattice based on its current spin state. Specifically, **S** acts such that terms of the superposition state associated with spin-up (+½) move one step up the lattice, while terms with spin-down (-½) move one step down. This interaction couples the particle's spin state to its spatial dynamics, enabling the quantum walk to evolve as a function of both spin and position.

We can now define an operator U such that

$$U = S \cdot (R(\theta) \otimes I)$$

1.2 Evolving the random walk

Consider the initial state of the walker represented by the ket $|\psi_0>$ the dimensions of this will be (2n,1) as it is the tensor product of initial spin (2,1) and position states (n,1). Evolving the walker now is a matter of performing $|\psi_1> = U\ |\psi_0>$.

The rotation matrix **R(θ)** first places the particle's spin state into a superposition, assigning specific probability amplitudes to the spin-up (+½) and spin-down (-½) states. Following this, the position randomization operator S acts on the particle, moving components of the state with spin-up one step up the lattice and those with spin-down one step down.

This combined action of the **R(θ)** and **S** operators generates branching possibilities, where each iteration of the quantum walk introduces new superposition states. As these operations are repeated, the quantum random walk evolves, encoding a probabilistic distribution over both position and spin states across the lattice. This iterative process forms the foundation of the quantum walk's characteristic dynamics, differentiating it fundamentally from classical random walks.

1.3 Code Snippet

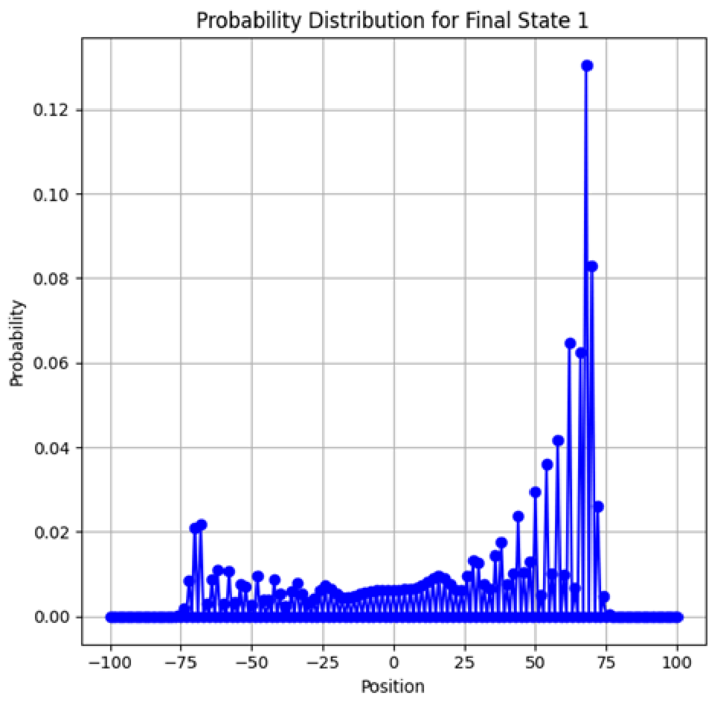
Code below goes through defining operators needed for 1D QRW with and without resetting.

```
import numpy as np

import matplotlib.pyplot as plt




# Number of steps and positions

n_steps = 100

n_positions = 2 * n_steps + 1

# Define the coin operator (Hadamard-like)

coin = np.array([[1/np.sqrt(2), 1/np.sqrt(2)],

                 [1/np.sqrt(2), -1/np.sqrt(2)]])
```

```
# Define the vectors
up = np.array([1, 0])
down = np.array([0, 1])

# Reshape up to be a column vector
up_t = up.reshape(-1, 1)

# Perform the outer product
up_matrix = up_t @ up.reshape(1, -1)

down_t = down.reshape(-1,1)
down_matrix = down_t @ down.reshape(1,-1)
### Writing the shift operator
#### - thus S = |↓><↓ | x Σ|i-1> < i|  + Σ|i+1> < i| x |↑ > <↑ |
n = n_positions
S_right = np.zeros((n,n))
for i in range(0,n-1):
   S_right[i+1][i] = 1
S_left = np.zeros((n,n))
for i in range(1,n):
   S_left[i-1][i] =1
n = n_positions
S_right = np.zeros((n,n))
for i in range(0,n-1):
   S_right[i+1][i] = 1

S_left = np.zeros((n,n))

for i in range(1,n):

   S_left[i-1][i] =1

S1 = np.kron(up_matrix,S_right)

S2 = np.kron(down_matrix,S_left)

S = S1 + S2
```

2. One-dimensional QRW with its randomization

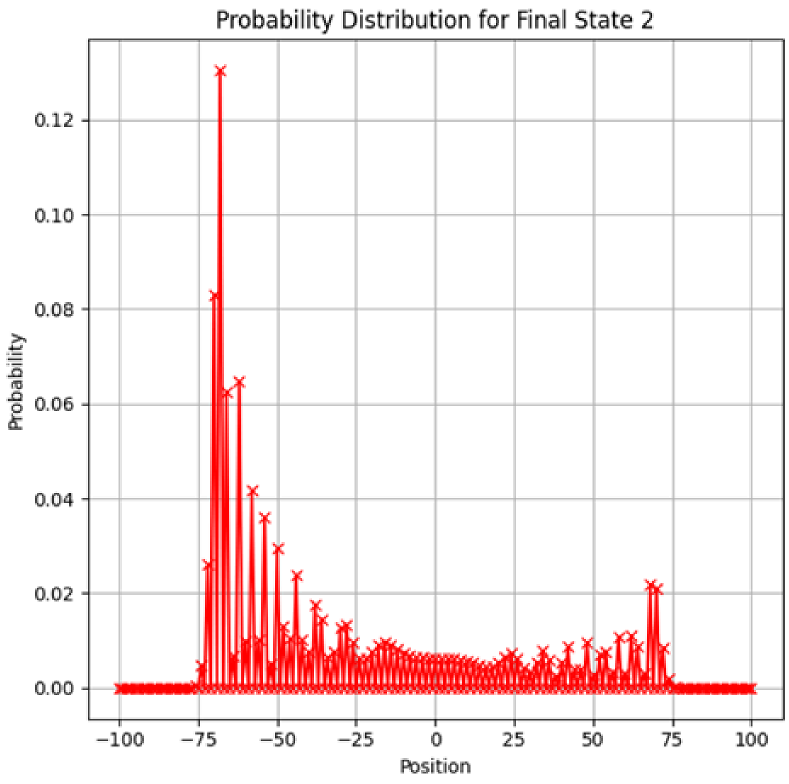
2.1 Localisation profile with the initial state $\psi \times |\uparrow>$



```
#positon space

zero_ket = np.zeros(n_positions)

zero_ket[n_steps] = 1

initial_state = np.kron(up,zero_ket)

state = initial_statecoin_adjusted = np.kron(coin,np.eye(n,n))

for _ in range(n_steps):

   state = coin_adjusted @ state

   state = S @ state
```

```
# Step 1: Extract the probability coefficients

probability_amplitudes = np.abs(state)**2  # Square of the modulus
gives the probability

# Step 2: Sum the probabilities for each position

probabilities = np.array([probability_amplitudes[i] +
probability_amplitudes[i + n] for i in range(n)])

# Step 3: Plot the histogram

positions = np.arange(n)

plt.bar(positions, probabilities, color='blue')

plt.xlabel('Position')

plt.ylabel('Probability')

plt.title('Quantum Random Walk Probability Distribution')

plt.show()
```

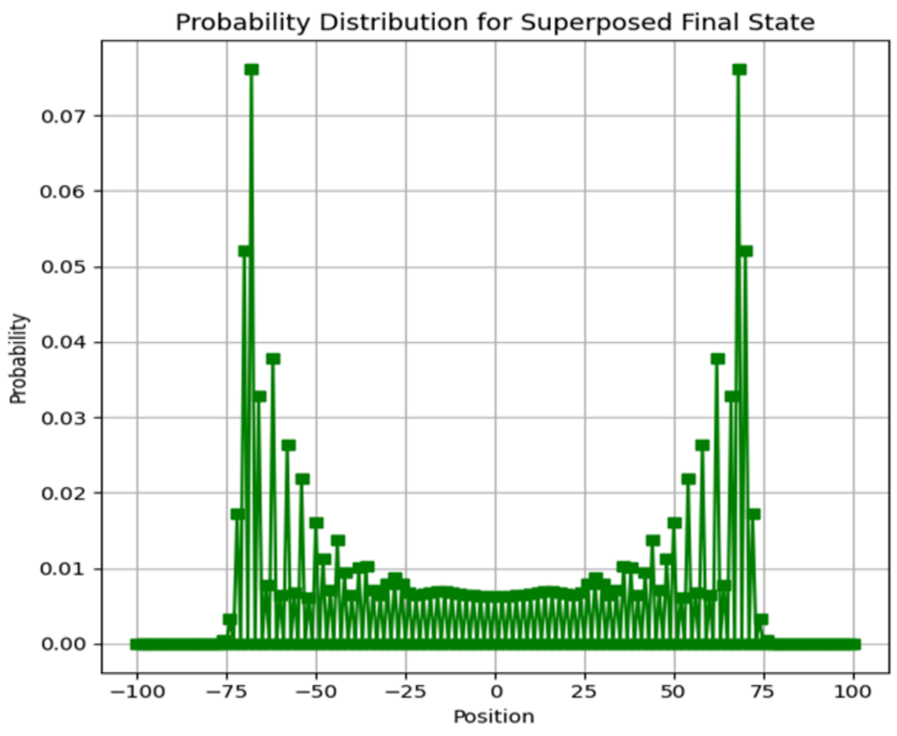
2.2 Localisation profile with the initial state ψ × | ↓ >



```
#positon space

zero_ket = np.zeros(n_positions)

zero_ket[n_steps] = 1

initial_state = np.kron(up,zero_ket)

state = initial_state

coin_adjusted = np.kron(coin,np.eye(n,n))

for _ in range(n_steps):

    state = coin_adjusted @ state

    state = S @ state

# print(state )

# Step 1: Extract the probability coefficients
```

```
probability_amplitudes = np.abs(state)**2  # Square of the modulus
gives the probability

# Step 2: Sum the probabilities for each position

probabilities = np.array([probability_amplitudes[i] +
probability_amplitudes[i + n] for i in range(n)])

# Step 3: Plot the histogram

positions = np.arange(n)

plt.bar(positions, probabilities, color='blue')

plt.xlabel('Position')

plt.ylabel('Probability')

plt.title('Quantum Random Walk Probability Distribution')

plt.show()
```

2.3 Localisation profile with initial state ψ $\times(\frac{1}{\sqrt{2}}|\uparrow> - \frac{i}{\sqrt{2}}|\downarrow>)$



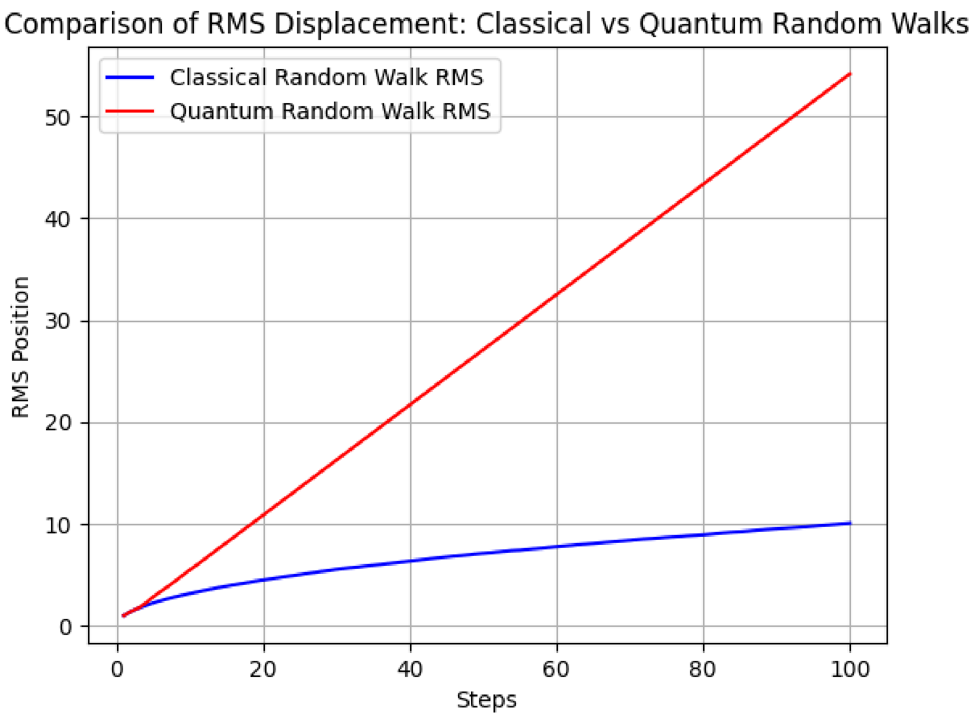
```
#write the initial to be a super Positon state

up_coef = 1/np.sqrt(2)

down_coef = 1j/np.sqrt(2)

spin_superpos = up_coef*up + down_coef*down

#positon space

zero_ket = np.zeros(n_positions)

zero_ket[n_steps] = 1

initial_state = np.kron(spin_superpos,zero_ket)

state = initial_state

coin_adjusted = np.kron(coin,np.eye(n,n))

for _ in range(n_steps):

   state = coin_adjusted @ state
```

```
    state = S @ state

# Step 1: Extract the probability coefficients

probability_amplitudes = np.abs(state)**2  # Square of the modulus
gives the probability

# Step 2: Sum the probabilities for each position

probabilities = np.array([probability_amplitudes[i] +
probability_amplitudes[i + n] for i in range(n)])

# Step 3: Plot the histogram

positions = np.arange(n)

plt.bar(positions, probabilities, color='blue')

plt.xlabel('Position')

plt.ylabel('Probability')

plt.title('Quantum Random Walk Probability Distribution')

plt.show()
```

2.4 Discussion and RMS Distance Plot

From the plots presented in sections 2.1, 2.2, and 2.3, it is evident that the quantum random walk (QRW) can exhibit a bias in a specific direction depending on the initial spin conditions. This observation suggests that the choice of the initial spin superposition, along with the applied rotation matrix, plays a crucial role in determining the resulting probability distribution. In other words, by altering the initial spin state and the unitary transformations (such as the rotation matrix) applied to it, we can influence the directionality and bias of the quantum walk's behavior. This flexibility allows for tailored probability distributions, which can be leveraged for more efficient quantum search algorithms or other quantum information tasks.

The quantum random walk demonstrates a fundamentally different propagation mechanism compared to the classical random walk. While the classical random walk exhibits a linear variance scaling, where the variance σ^2 after N steps grows proportionally with the number of steps, the quantum random walk presents a dramatically different behaviour. In the quantum walk, the variance scales quadratically, increasing as $\sigma^2 \sim N^2$, which results in a significantly faster spread of the walker. Consequently, the expected distance from the origin grows linearly with N, rather than the square root scaling observed in classical random walks.[1]



3. One-dimensional QRW with extended hopping and θ variation

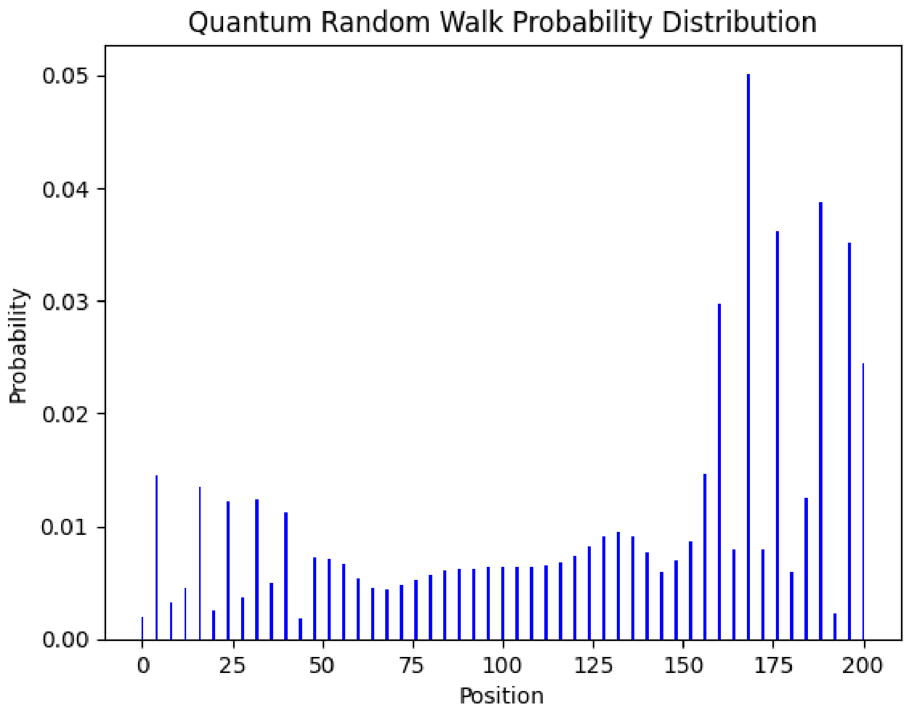
3.1 Extended Hopping

We plot the localisation profiles as well as the RMS distance plot with different initial states where the shift operator is as follows

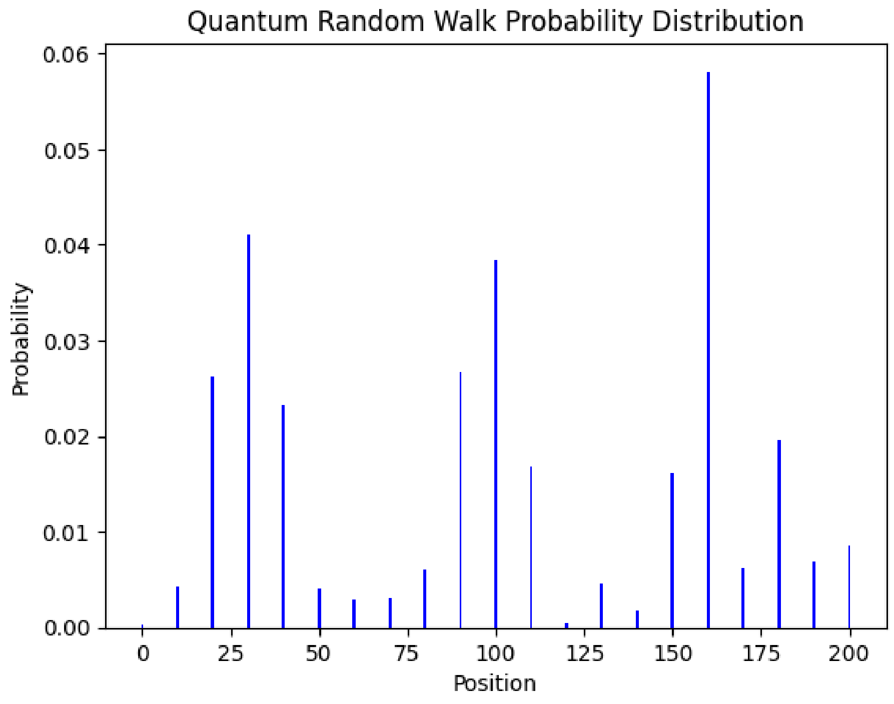
$$S = |\downarrow\rangle\langle\downarrow| \times \Sigma\, |i-r\rangle\langle i| \; + \Sigma\, |i+r\rangle\langle i| \times |\uparrow\rangle\langle\uparrow|$$

Changes have been made in the same code as given above, taking r = 2, 5 and 10 to show comparison among the plots for different hopping ranges.

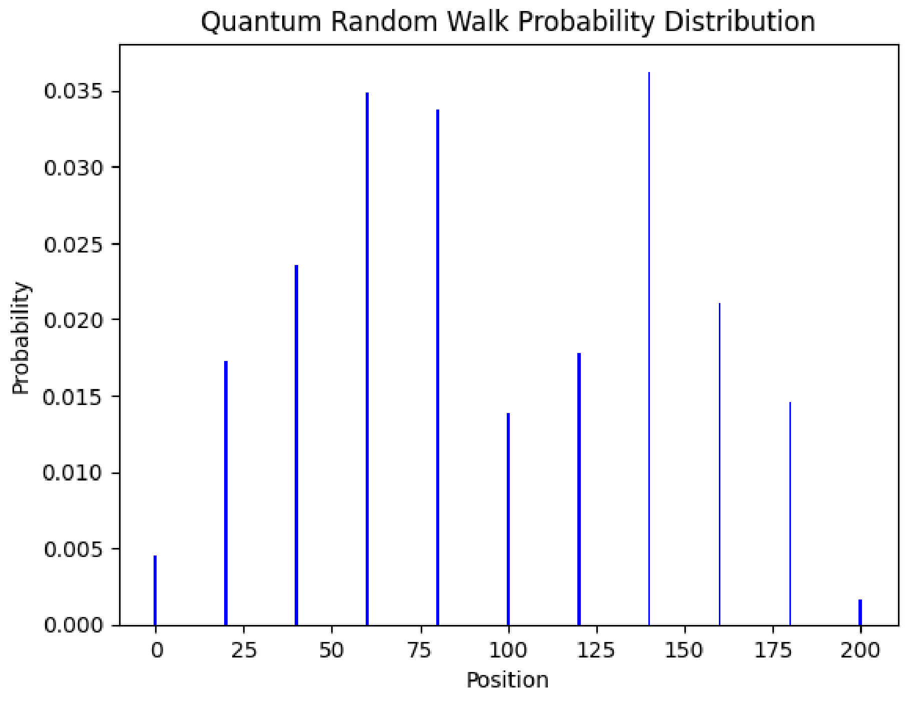
3.1.1.1 Localisation profile with the initial state ψ × | ↑ > for r = 2



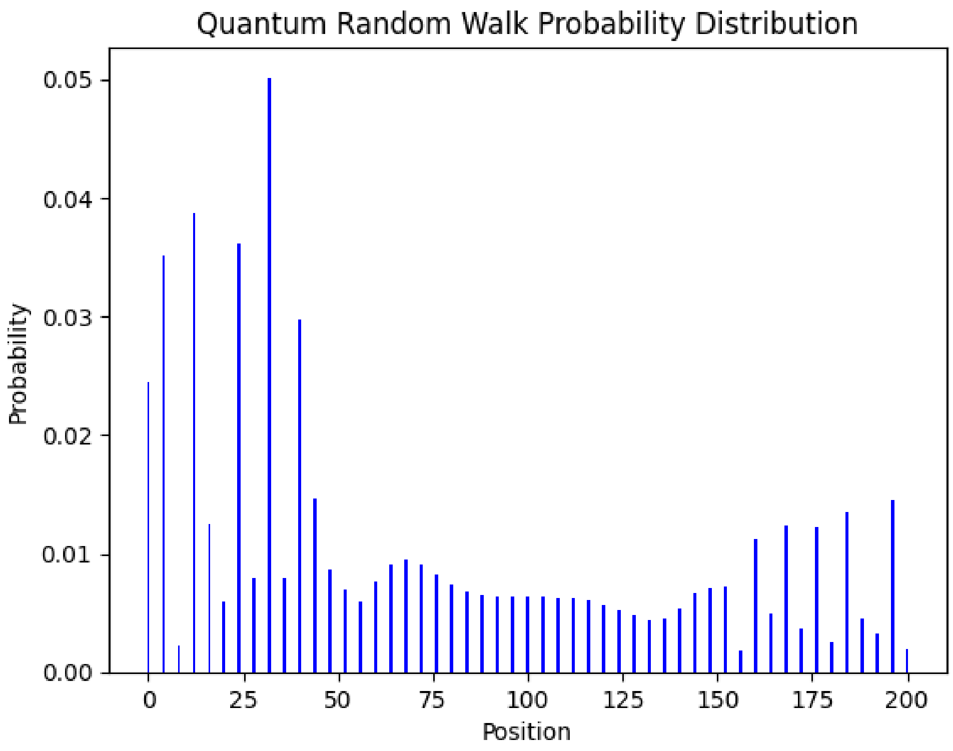
3.1.1.2 Localisation profile with the initial state ψ × |↑ > for r = 5



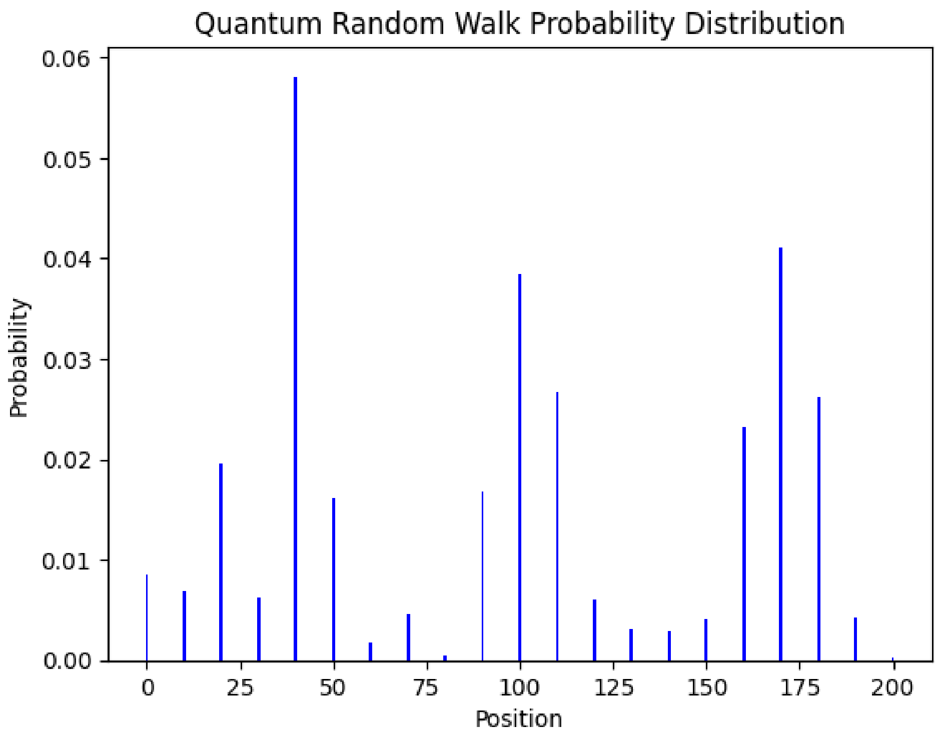
3.1.1.3 Localisation profile with the initial state ψ × |↑ > for r = 10



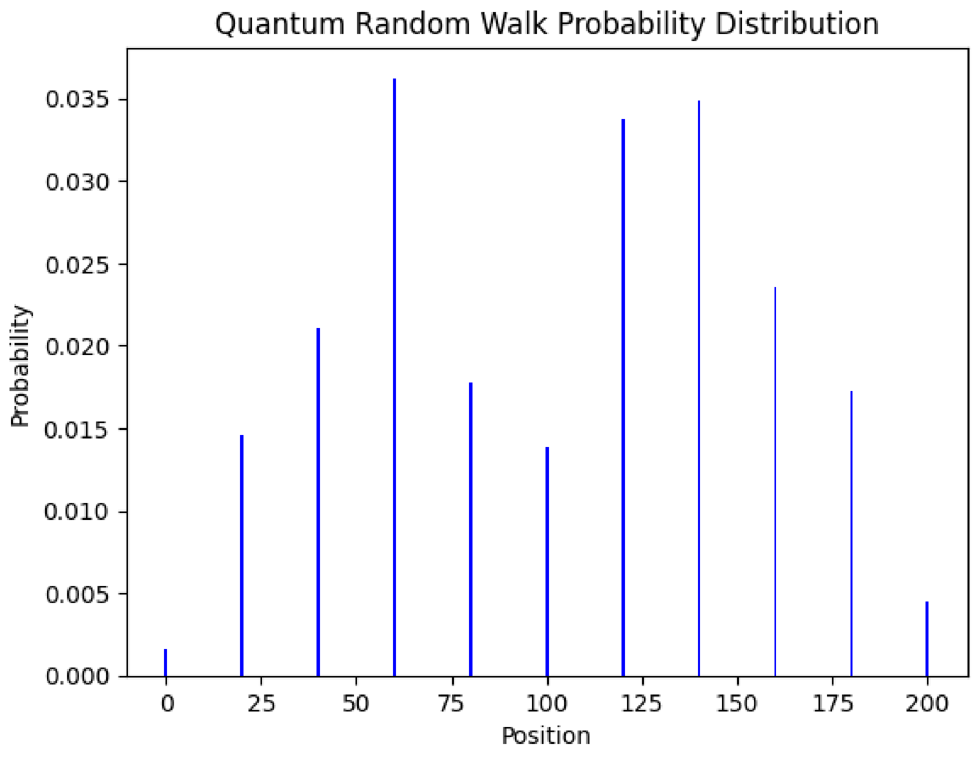
3.1.2.1 Localisation profile with the initial state ψ × |↓ >
for r = 2



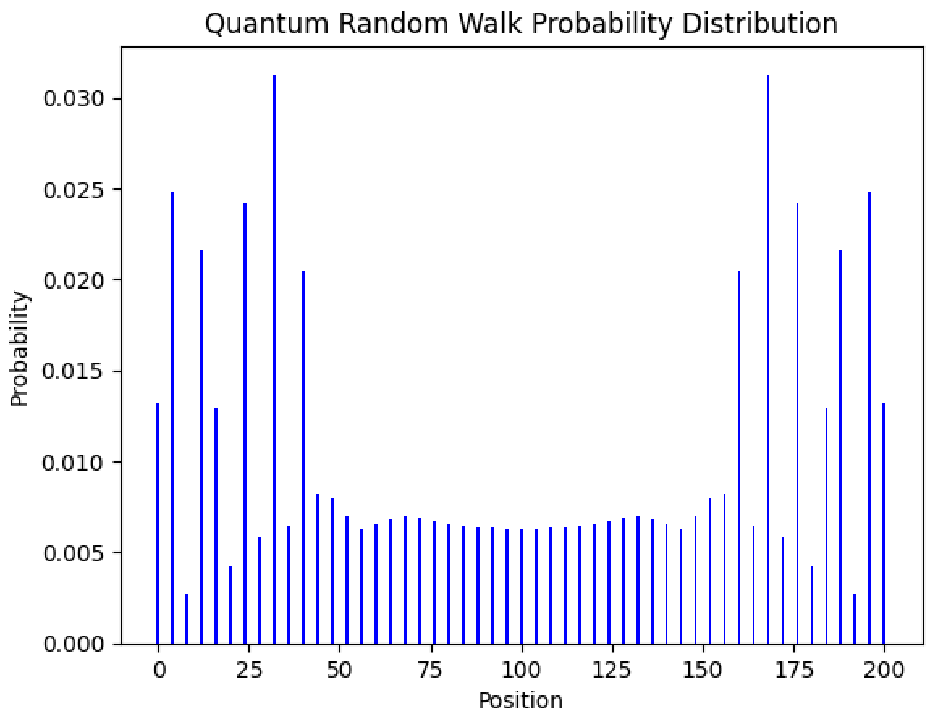
3.1.2.2 Localisation profile with the initial state ψ × |↓ >
for r = 5



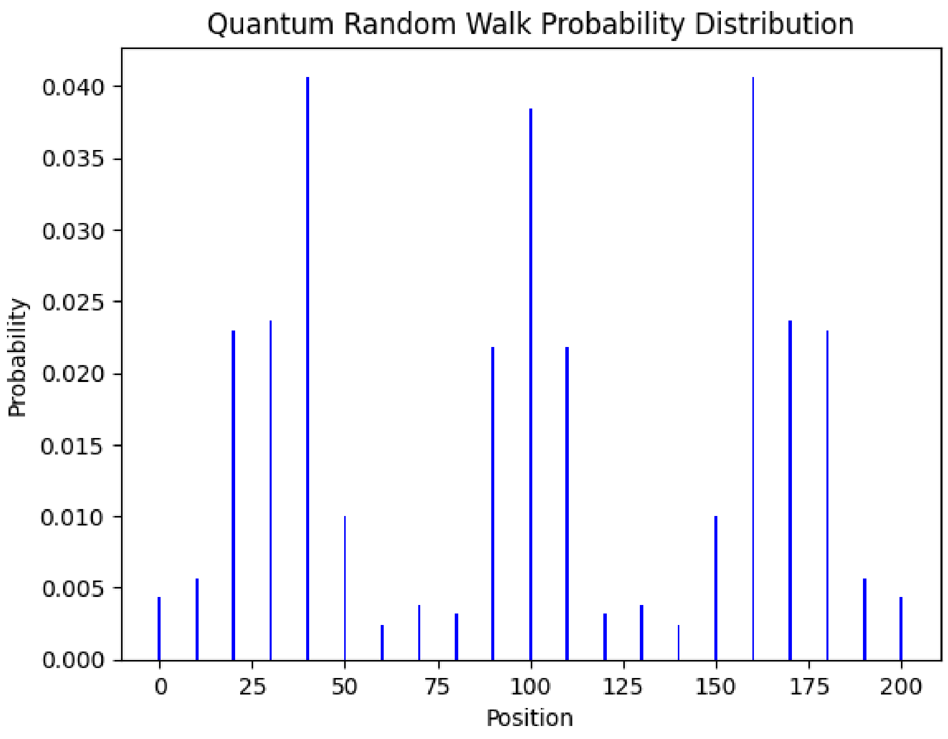
3.1.2.3 Localisation profile with the initial state $\psi \times |\downarrow>$ for r = 10



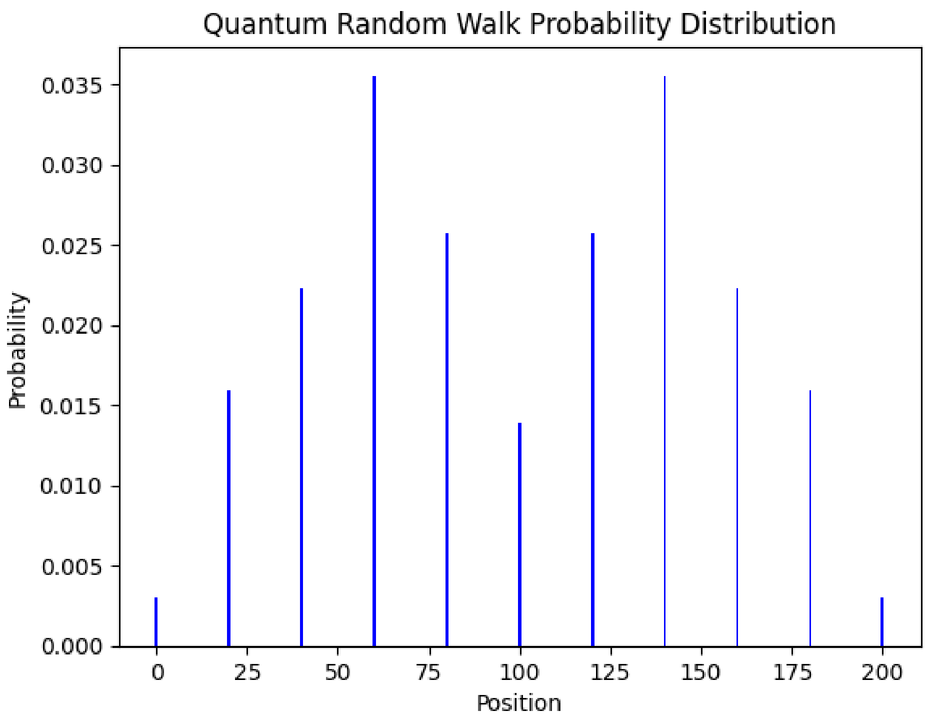
3.1.3.1 Localisation profile with initial state $\psi \times (\frac{1}{\sqrt{2}}|\uparrow> - \frac{i}{\sqrt{2}}|\downarrow>)$ for r = 2



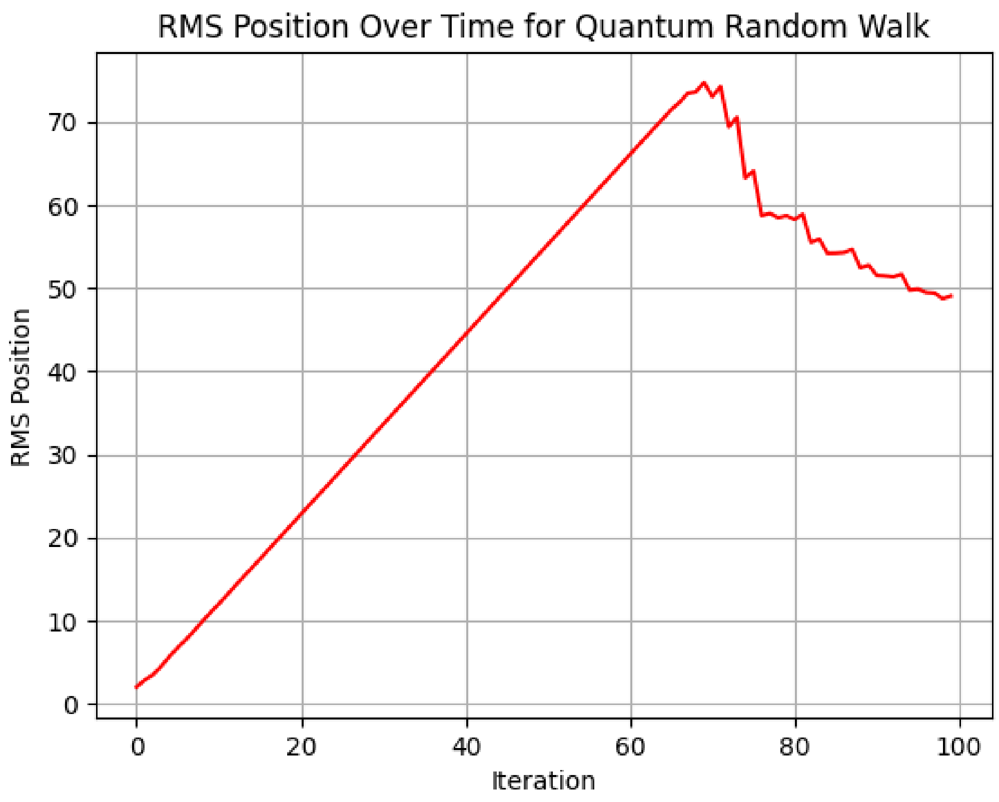
3.1.3.2 Localisation profile with initial state ψ $\times(\frac{1}{\sqrt{2}}|\uparrow> - \frac{i}{\sqrt{2}}|\downarrow>)$ for r = 5



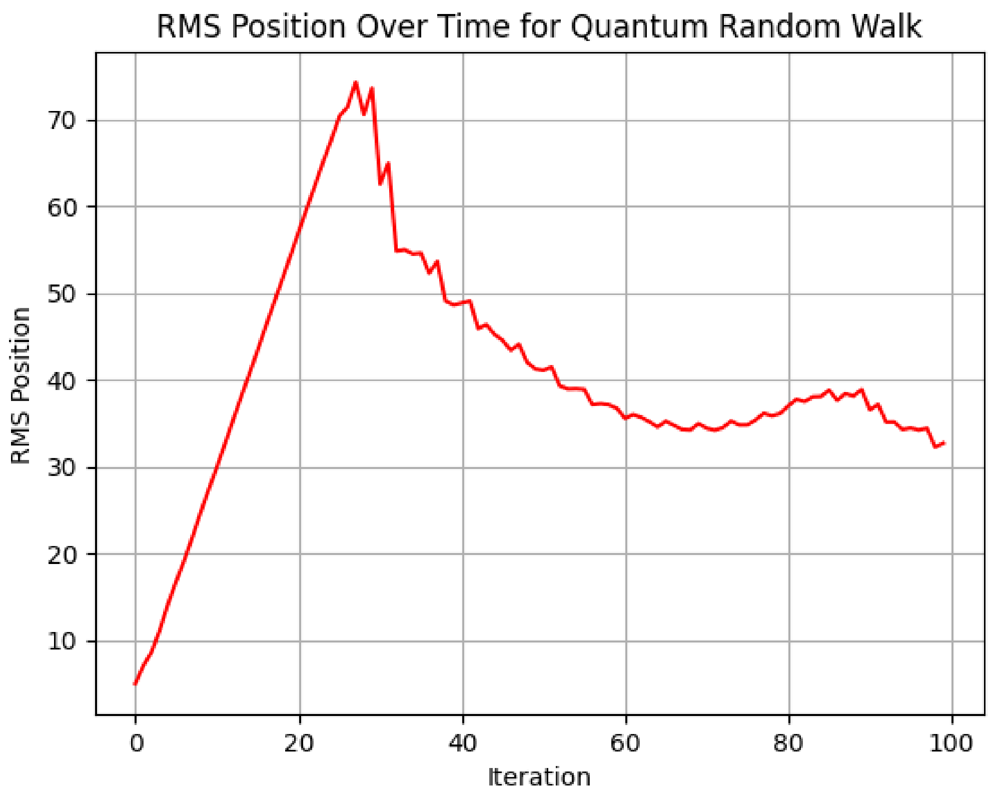
3.1.3.3 Localisation profile with initial state ψ $\times(\frac{1}{\sqrt{2}}|\uparrow> - \frac{i}{\sqrt{2}}|\downarrow>)$ for r = 10



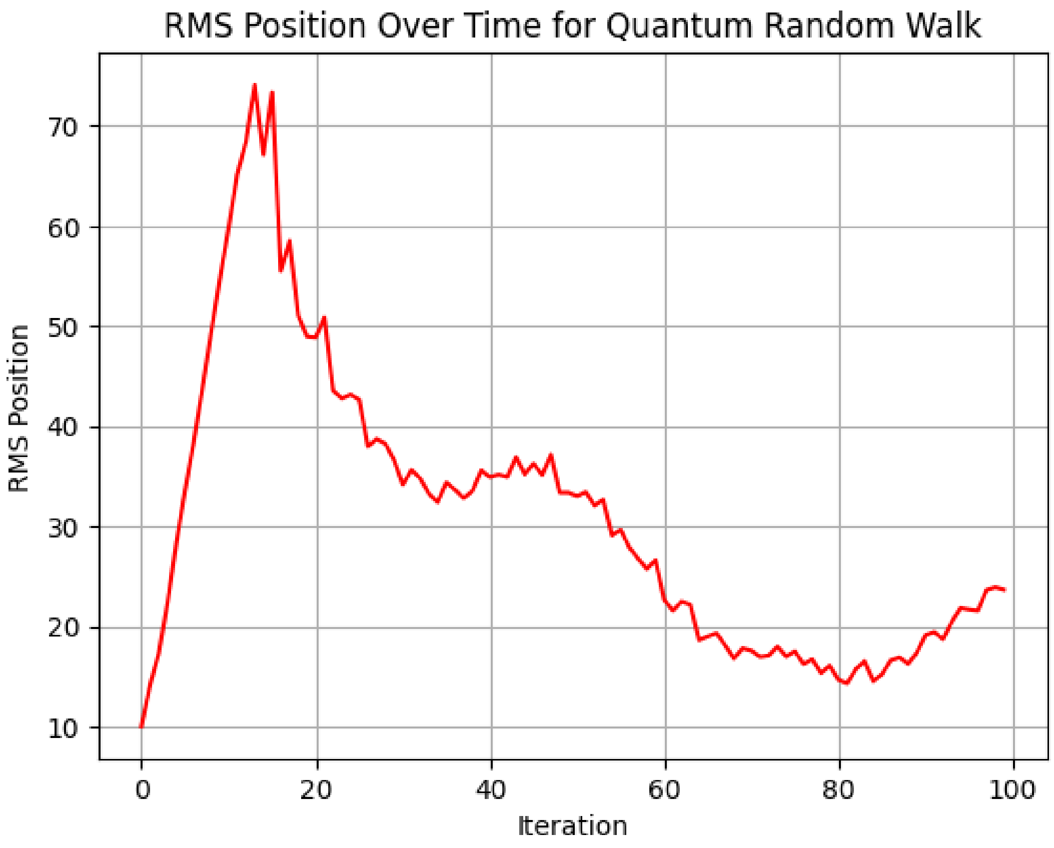
3.1.4.1 RMS Distance Plot for r = 2



3.1.4.2 RMS Distance Plot for r = 5



3.1.4.3 RMS Distance Plot for r = 10



3.1.5 Code snippet

```
# Adjusted S_right for i+10

S_right = np.zeros((n, n))

for i in range(0, n - 10):  # Ensure i+10 stays within bounds

    S_right[i + 10][i] = 1



# Adjusted S_left for i-10

S_left = np.zeros((n, n))

for i in range(10, n):  # Ensure i-10 stays within bounds

    S_left[i - 10][i] = 1
```

```
# Construct S1 and S2 using Kronecker products

S1 = np.kron(up_matrix, S_right)

S2 = np.kron(down_matrix, S_left)



# Combined S matrix

S = S1 + S2
```

3.1.6 Analysis of the different plots

As the hopping range increases, we observe a wider spread in the localization profiles across the different initial state cases. Additionally, the distinct peaks observed for r = 1 become shorter as r increases. This occurs due to the loss of probability when the quantum random walk crosses the boundary of the 200-position range under consideration. This phenomenon is also evident in the RMS plots for each case. With an increased hopping range, the slope of the RMS graph becomes steeper compared to the r = 1 case, indicating a faster spread. However, the spread diminishes after a certain number of iterations in each case, as the probability is lost when the walker exits the bounded region.

3.2 ‘θ’ step size variation

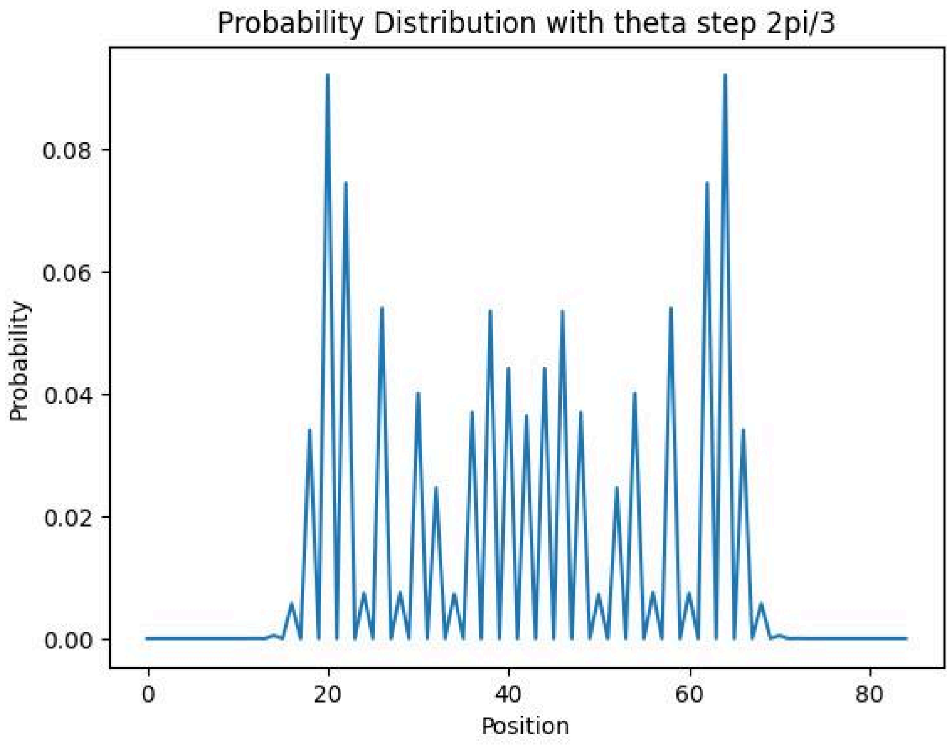
In this quantum random walk simulation, the theta step size dynamically modifies the coin operator, which governs the walker's evolution. The Hadamard coin, based on a parameter θ is defined as:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

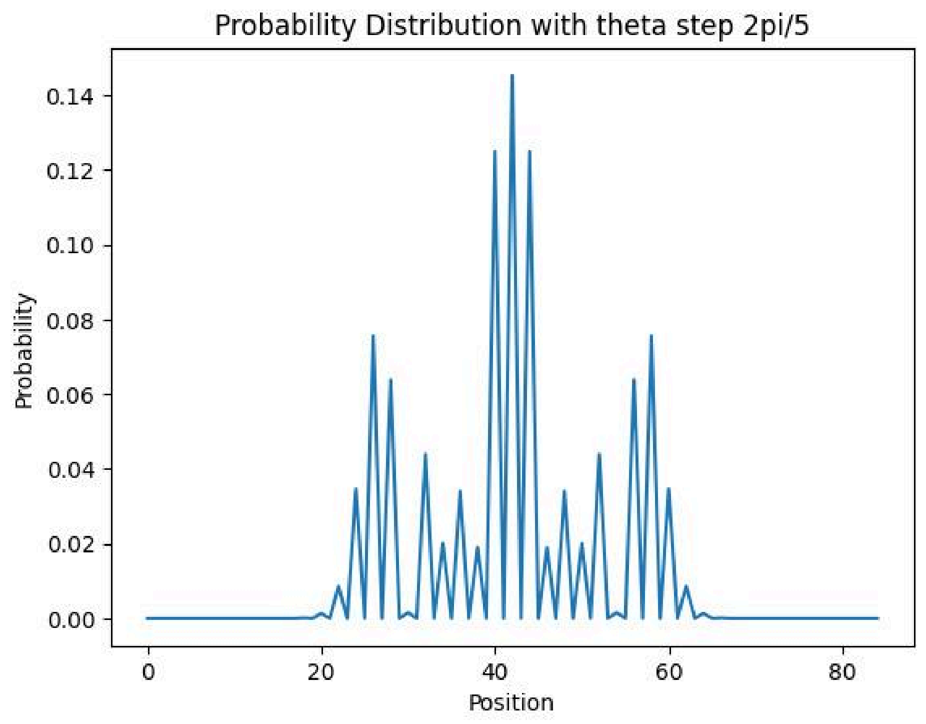
By updating θ incrementally with a step size of $\frac{2\pi}{x}$ (for $x = 3, 5, 7, 11, 121, 1331$), the coin evolves dynamically during the walk.

This variation introduces a time-dependent bias to the quantum coin, influencing the walker’s probability distribution. The resulting profiles for different step sizes demonstrate how controlled parameter changes impact the walker’s spread, providing insights into dynamic quantum systems.

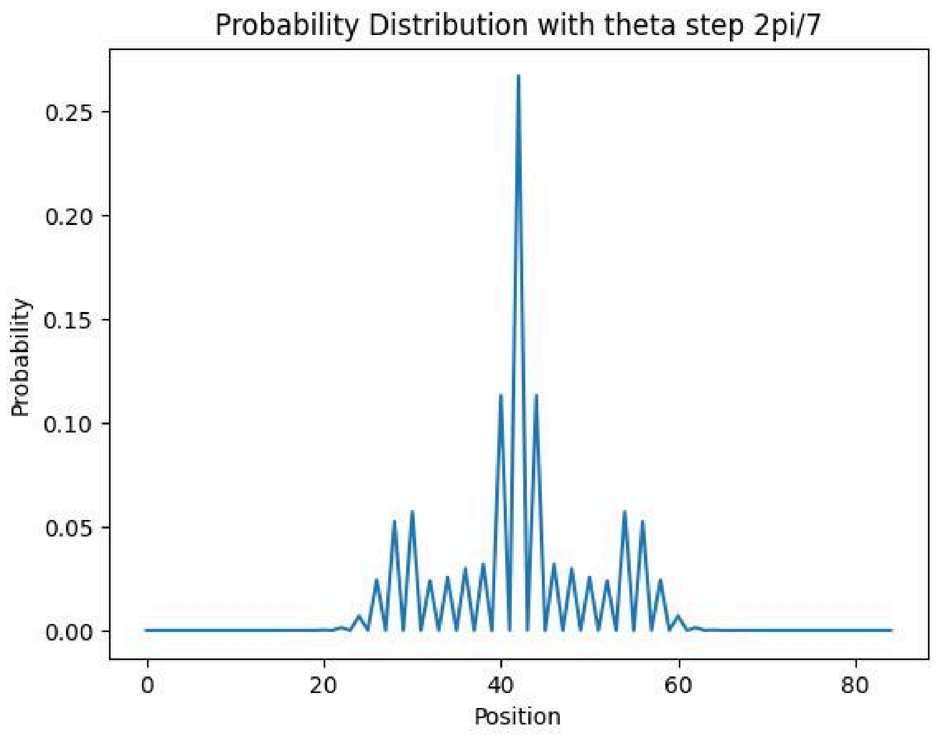
3.2.1.1 Localisation profile with θ step size = $\frac{2\pi}{3}$



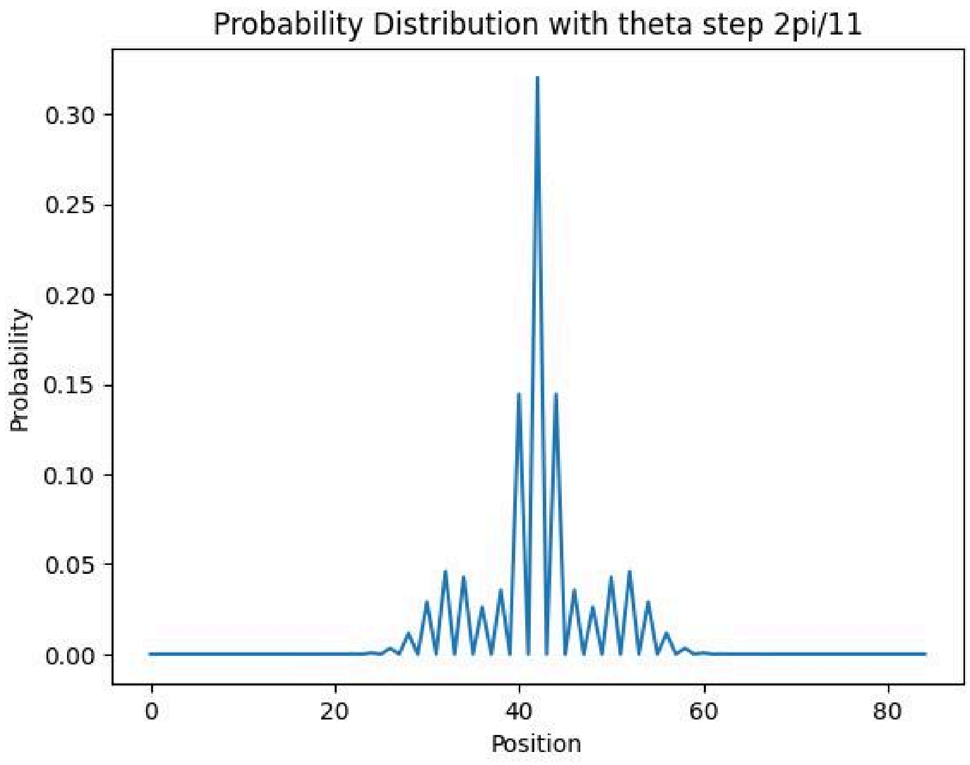
3.2.1.2 Localisation profile with θ step size = $\frac{2\pi}{5}$



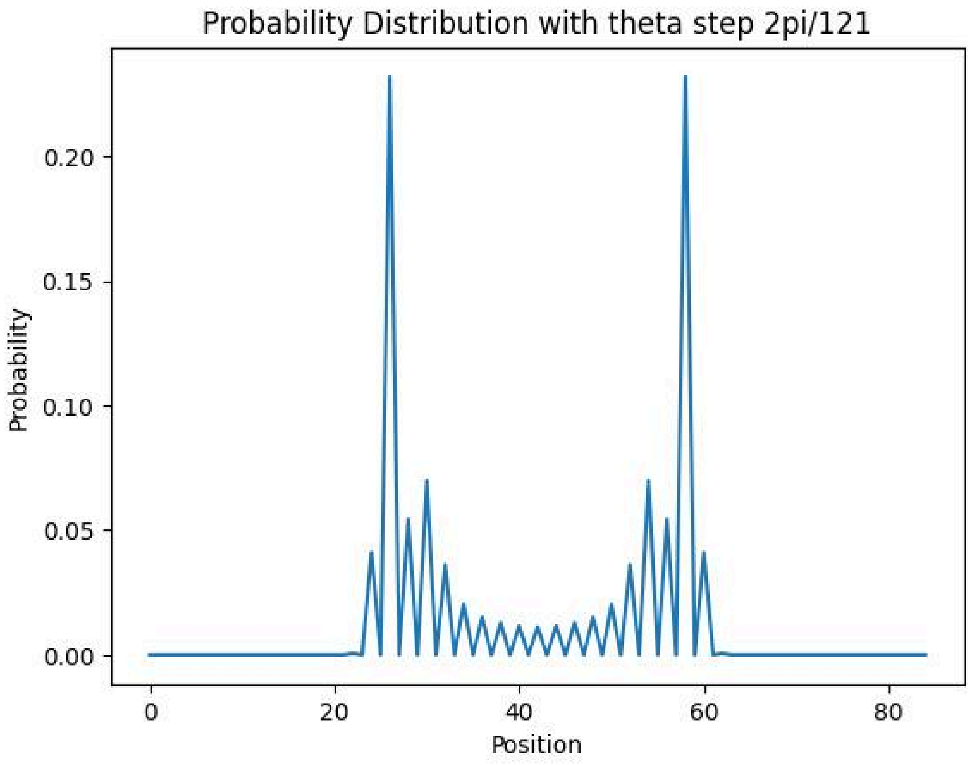
3.2.1.3 Localisation profile with θ step size = $\frac{2\pi}{7}$



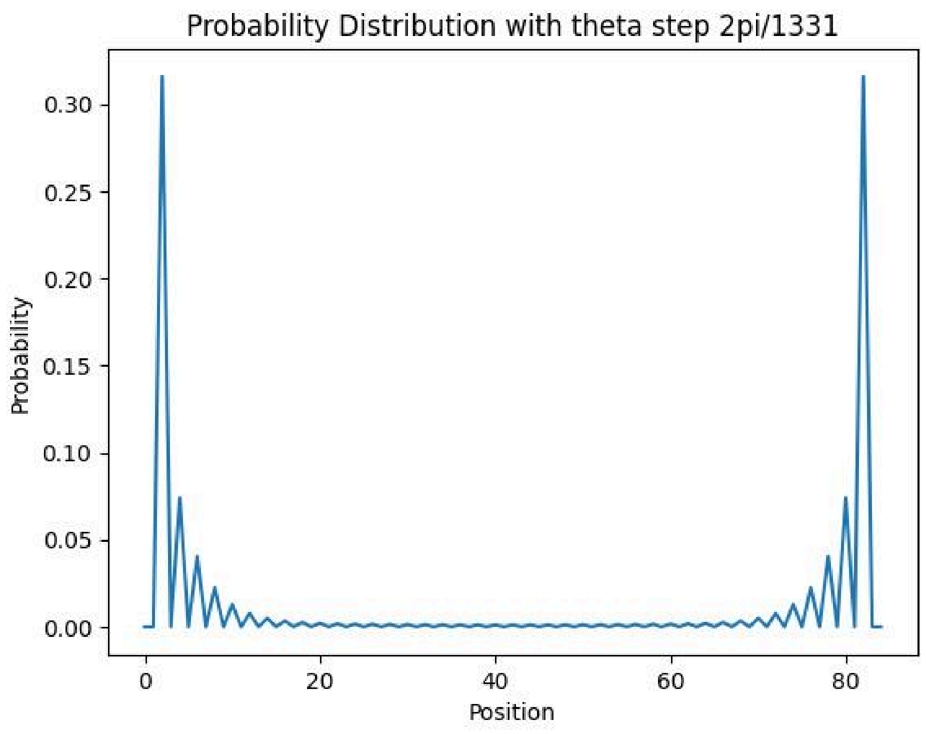
3.2.1.4 Localisation profile with θ step size = $\frac{2\pi}{11}$



3.2.1.5 Localisation profile with θ step size = $\frac{2\pi}{121}$



3.2.1.6 Localisation profile with θ step size = $\frac{2\pi}{1331}$



3.2.2 Code snippet

```python
def walker(steps, n, coin_instance, theta_step_size = 0, theta =
np.pi/4, plot = True, plot_title = 'QRW'):

    positions = [0 for i in range(n)]

    positions[n//2] = 1

    positions = np.array(positions).reshape(-1,1)



    S = shift_matrix(n)

    state = np.kron(coin_instance,positions)

    c = coin(theta, n)

    for i in range(steps):

        state = shift(flip(c,state), S)

        theta += theta_step_size
        if theta_step_size != 0:

            c = coin(theta, n)


    output = [(np.square(np.abs(state[i][0])) +
np.square(np.abs(state[i+n][0]))) for i in range(n)]


    if plot:

        x = [i - n//2 for i in range(n)]

        plt.plot(x, output)

        plt.xlabel('Position')

        plt.ylabel('Probability')

        plt.title(plot_title)

        plt.show()

    return state
```

4. Detection Probability for One-dimensional QRW without resetting

4.1 P_{det} Results

We now want to calculate the probability of detection of the particle at a target site δ within *n* attempts and we label it as $P_{det}(n)$. We utilize the formulas and protocols discussed in Chatterjee et al.[2] for $P_{det}(n)$. As they discussed a tight binding Hamiltonian model, we adapt these into our system with both spin and position states.[2]

Given a target site δ, we define the projection operator $\mathbf{D} = |\,\delta><\delta\,|$. Then we say the probability of detecting the particle at site δ at *n* steps conditioned on no previous detection in the previous $n-1$ attempts as

$$P_n = \frac{<\delta|U[(1-D)U]^{n-1}\ |\psi>}{(1-P_1)(1-P_2)\dots(1-P_{n-1})}$$

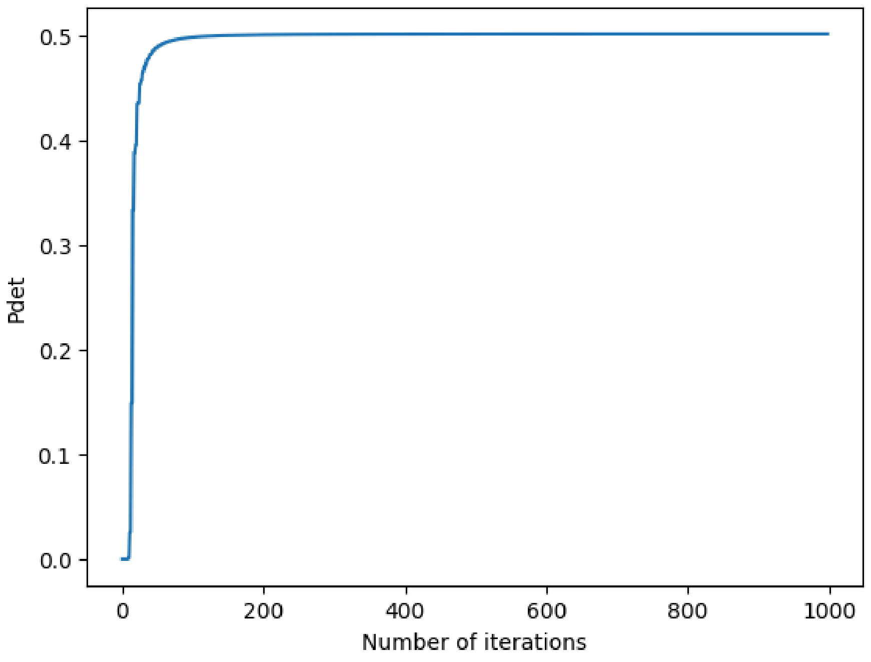
The first detection probability at the site δ now becomes

$$F_n = P_n \bullet \prod_{i=1}^{n-1} (1\text{-}P_i)$$

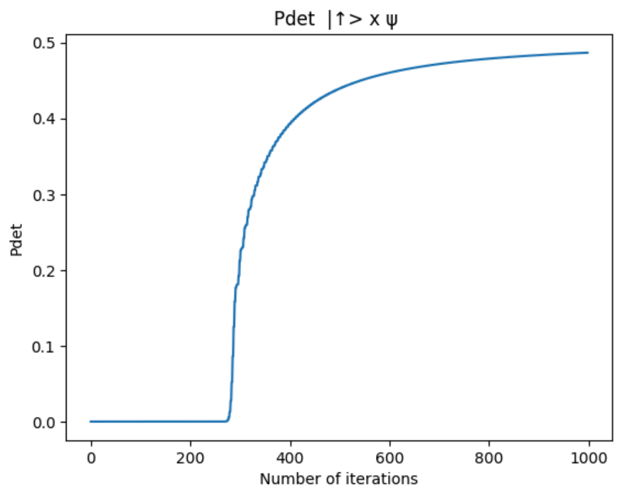
Thus we collect the total probability that we detect the particle at the site δ as

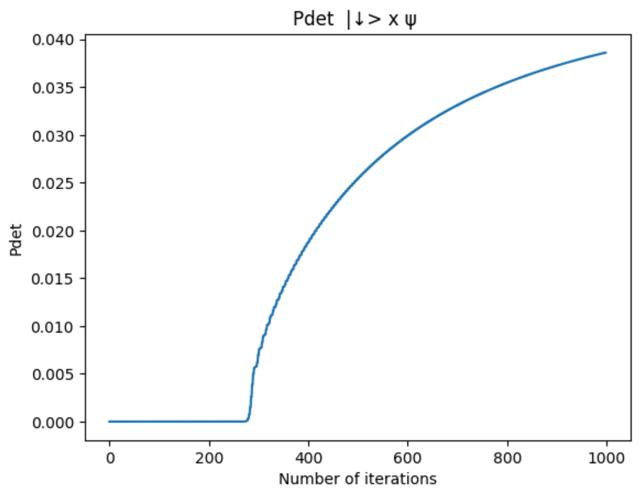
$$P_{det}(n) = \sum_{m=1}^{n} F_m$$

We observe from the plots given below that the value of $P_{det}(n)$ rises and then saturates around 0.5.



We can now consider a target site that would be affected by the choice of initial conditions, for our purposes we selected a point 200 to the right of the initial position. This position would be sufficiently far so that the bias of the initial position can affect the saturation value of $P_{\text{det}}(n)$.





4.2 Code snippet

```
# Creating the projection operator

init_space[500] = 0

init_space[target_site] = 1

target_ket = np.kron(up,init_space)

target_ket = target_ket.reshape(-1,1)

D = target_ket @ target_ket.T

D.shape

detection_op = (np.eye(2*n,2*n) - D)@U

detection_op.shape

# Fn calculation

P_list = []

F_list = []

p1 = np.abs(target_ket.T @ ( U @ inital_state))**2

p1 = p1.item()

print(p1)

P_list.append(p1)

evolved_ket = inital_state

F_list.append(p1)

print(F_list)

for i in range(2,total_itr):

  # evolved ket [ ( I − D^ ) U^ ( τ ) ] n − 1 | init ⟩

  evolved_ket = detection_op @ evolved_ket

  num = np.abs(target_ket.T @ (U @ evolved_ket))**2
  num = num.item()

  denom = 1

  for j in range(1,i):

      # print(i)

      # print(j)

      denom *= (1-P_list[j-1])

  Pn = num / denom

  P_list.append(Pn)

  Fn = 1

  for j in range(1,i):

      Fn *= (1-P_list[j-1])

  Fn*=P_list[i-1]

  F_list.append(Fn)

PdetList = []

PdetList.append(F_list[0])

print(PdetList)

Pdet = F_list[0]

for i in range(2,total_itr):

  Pdet += F_list[i-1]

  PdetList.append(Pdet)

plt.plot(PdetList)
```

5. Detection Probability for One-Dimensional QRW with resetting

5.1 P_{det} Results

As said in Chatterjee et al.[2] “Repeated projective measurement without resetting cannot guarantee detection in a quantum system, while resetting strategies can help ensure detection.”

The tight binding model results suggest that using an initial point resetting strategy ensured that

$$P_{det}(n \to \infty) \to 1$$

We aim to recreate these results using our QRW model. The protocol calls for selecting a resetting interval after which we return the state to the origin.

$$n = rR + \tilde{n} \qquad (r \text{ : resetting rate})$$

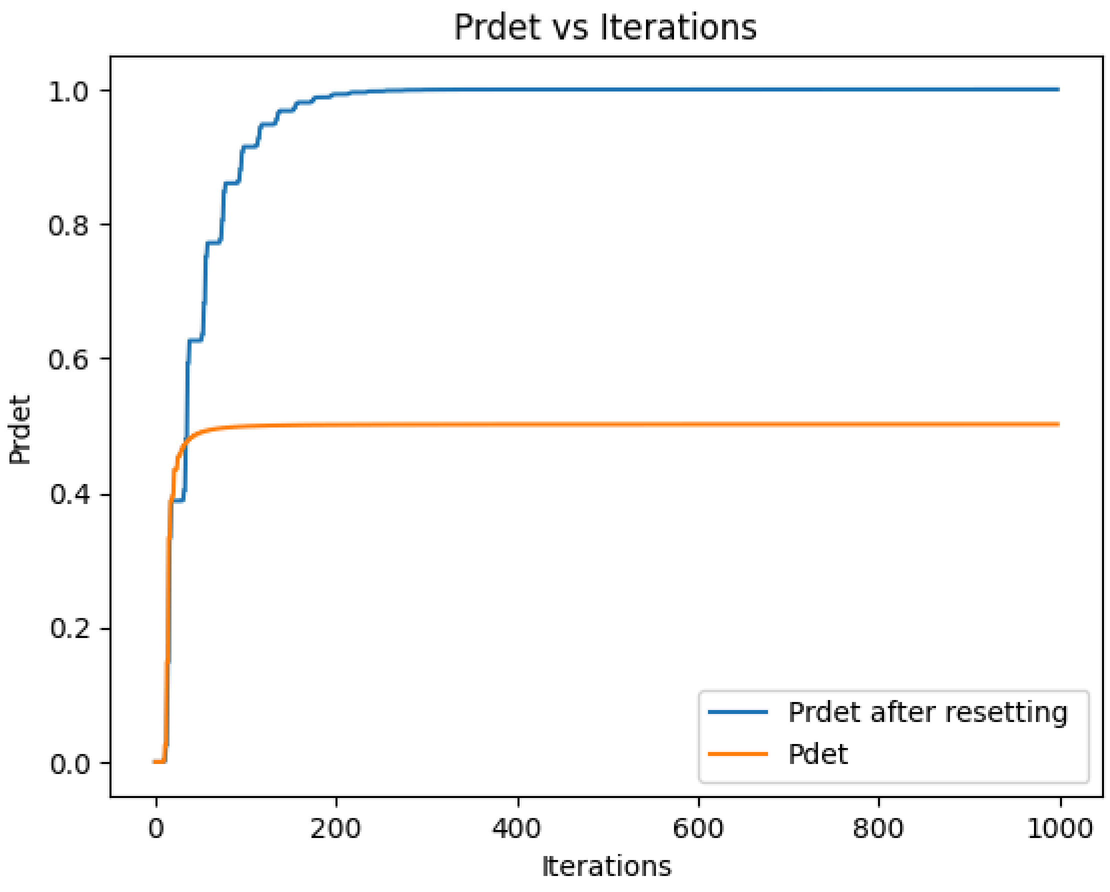
$$R = \text{integer}\left[\frac{(n-1)}{r}\right]$$

The first hitting probability in the presence of resetting is given as

$$F_n(r) = [1 - P_{det}^{x0}(r)]^R \, F_{\tilde{n}}^{x0}$$

This gives the total probability of detection up to time n in IPR as

$$P_{det}^{r}(n) = \sum_{m=1}^{n} F_m(r)$$



In our approach, we employed the quantum random walk (QRW) model and, similarly to the resetting protocols discussed in Chatterjee et al.[2], we achieved a detection probability $P_{det}(n \rightarrow \infty) \rightarrow 1$. This result confirms that our QRW-based model also guarantees detection as the number of steps increases, demonstrating the effectiveness of the quantum resetting approach in ensuring detection in the QRW framework.

5.2 Code snippet

```
def p_and_f_totaIterations_param(total_itr):
   # Fn calculation
   P_list = []
   F_list = []
   p1 = np.abs(target_ket.T @ ( U @ inital_state))**2
   p1 = p1.item()
   # print(p1)
   P_list.append(p1)
   evolved_ket = inital_state
   F_list.append(p1)
   # print(F_list)
   for i in range(2,total_itr):
       # evolved ket [ ( I − D^ ) U^ ( τ ) ] n − 1 | init ⟩
       evolved_ket = detection_op @ evolved_ket
       num = np.abs(target_ket.T @ (U @ evolved_ket))**2
       num = num.item()
       denom = 1
       for j in range(1,i):
           # print(i)
           # print(j)
           denom *= (1-P_list[j-1])
       Pn = num / denom
       P_list.append(Pn)
```

```
        Fn = 1

        for j in range(1,i):

            Fn *= (1-P_list[j-1])

        Fn*=P_list[i-1]

        F_list.append(Fn)

    return F_list

def fntilda_param(total_itr):

    # Fn calculation

    P_list = []

    F_list = []

    p1 = np.abs(target_ket.T @ ( U @ inital_state))**2

    p1 = p1.item()

    # print(p1)

    P_list.append(p1)

    evolved_ket = inital_state

    F_list.append(p1)

    # print(F_list)

    for i in range(2,total_itr):

        # evolved ket [ ( I − D^ ) U^ ( τ ) ] n − 1 | init ⟩

        evolved_ket = detection_op @ evolved_ket

        num = np.abs(target_ket.T @ (U @ evolved_ket))**2

        num = num.item()

        denom = 1

        for j in range(1,i):

            # print(i)
            # print(j)
            denom *= (1-P_list[j-1])
        Pn = num / denom
        P_list.append(Pn)
        Fn = 1
        for j in range(1,i):
            Fn *= (1-P_list[j-1])
        Fn*=P_list[i-1]
        F_list.append(Fn)
    # print(total_itr)
    return F_list[total_itr-2]
def Pdet(r):
    Pdet_r = 0
    for i in range(1,r):
        Pdet_r += fntilda_param(i)
    return Pdet_r
def Fn_r(n,r):
    R = n//r
    n_tilda = n - R*r
    coef1 = (1 - Pdet(r))**R
    if(n_tilda == 0):
        return 0
    coef2 = fntilda_param(n_tilda)
    return coef1*coef2
r = 20
```

```python
# calculate Prdet(n)

Prdet_list = []

Prdet_n = 0

for i in range(1,total_itr):

   Prdet_n += Fn_r(i,r)

   Prdet_list.append(Prdet_n)

plt.plot(Prdet_list)

plt.plot(PdetList)

plt.xlabel('Iterations')

plt.ylabel('Prdet')

plt.title('Prdet vs Iterations')

plt.legend(['Prdet after resetting ','Pdet'])

plt.show()
```

5.3 Mean First Detection Time

Building on our results, we extend our analysis to calculate the **first detection time** (FDT), adapting the formulas from Chatterjee et al. [2] to our quantum random walk (QRW) model.

$$\langle t_f \rangle_r = \langle n\tau \rangle = \tau \sum_{n=1}^{\infty} n F_n(r).$$

$$\langle t_f \rangle_r = \frac{r\tau\left[1 - P_{\text{det}}^{x_0}(r)\right]}{P_{\text{det}}^{x_0}(r)} + \sum_{\tilde{n}=1}^{r} \frac{\tilde{n}\tau F_{\tilde{n}}^{x_0}}{P_{\text{det}}^{x_0}(r)}.$$

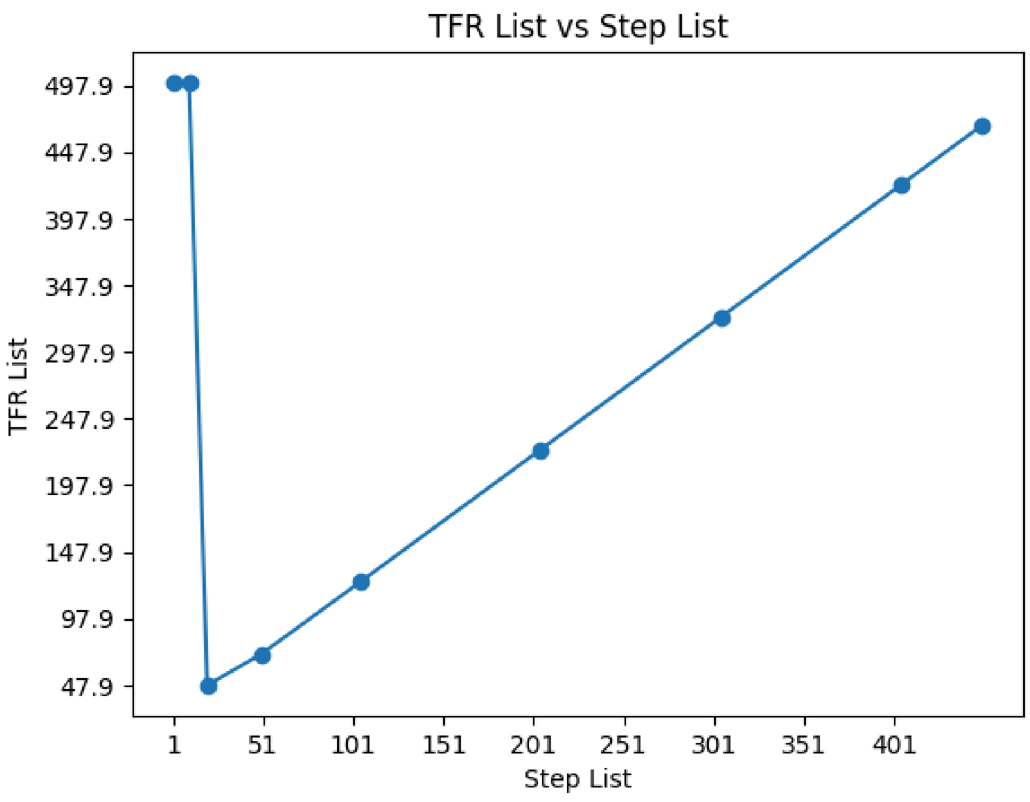
In our adaptation, we set τ=1, as the time parameter corresponds to a single step in the QRW iteration. To visualize the trends in FDT while accommodating computational constraints, we consider a discrete set of resetting rates, **r**, as defined by the following list:

```
step_list = [1,10,20,50,105,205,305,405,450]
```

It is important to note that the FDT values for r=1 and r=5 are set to the maximum of the graph's y-range for visualization purposes, as their true values approach infinity due to the inability to detect the target site within those reset intervals.

The resulting graph reveals a clear linear relationship between the FDT of the target site and the reset interval [3] . This demonstrates the significance of optimizing the reset rate to minimize the FDT. Future objectives include expanding this analysis to a continuous range of r values and exploring different target sites to deepen our understanding of the detection dynamics in the QRW model. These results seem to follow the same trends as can be observed in [3].

.



5.4 Code Snippet

```
def tfr(r):

   if(Pdet(r) == 0):

       return 0

   expr1 = (1-Pdet(r))*r/Pdet(r)

   expr2 = 1/Pdet(r)

   num2 = 0

   for i in range(1,r):

       num2+= i*fntilda_param(i)

   expr2 = num2*expr2

   return expr1 + expr2

tfr_list = []
```

```
step_list = [1,10,20,50,105,205,305,405,450]

for r in step_list:

   tfr_list.append(tfr(r))

tfr_list[0] = 500

tfr_list[1] = 500

plt.plot(step_list, tfr_list, marker='o')

plt.xlabel('Step List')

plt.ylabel('TFR List')

plt.title('TFR List vs Step List')

plt.xticks(np.arange(min(step_list), max(step_list)+1, 50))

plt.yticks(np.arange(min(tfr_list), 500, 50))

plt.show()
```

6. 2D Quantum Random Walk

6.1 Introduction

The 2D Quantum Random Walk extends the principles of the 1D Quantum Random Walk to two spatial dimensions, enabling exploration of this phenomena with greater complexity.
The walker evolves on a 2D grid of size $n \times n$. This is represented in a manner similar to a 1D walk, i.e., as an array A of n^2 positions on the grid. We enumerate cells in the grid sequentially, left to right, while moving upwards. So, when n is odd, $n^2/2$ is the position in the array that corresponds to the middle cell in the grid. With this in mind, the quantum state is represented as $Q = A^T \otimes H$ where H is the Hadamard Coin state, resulting in a state vector (Q) of dimensions $2n^2 \times 1$.
Initially, the quantum walk is configured such that the entire probability is concentrated at the grid's center position ($n^2/2$). This starting state is a tensor product of the initial positional state and a chosen initial coin state, as explained above.

The evolution of the walker is governed by the unitary transformation:

$$U = S_y R(\theta) S_x R(\theta)$$

Here:

- $R(\theta)$ is the rotation (coin) operator.
- S_x and S_y are the shift operators, which move the walker along the x and y directions.

At each step, U is applied to the quantum state, resulting in probability amplitudes propagating across the grid. This propagation involves interference patterns that arise from the quantum walk's superposition and unitary nature.

6.2 Dynamic Coin Variant

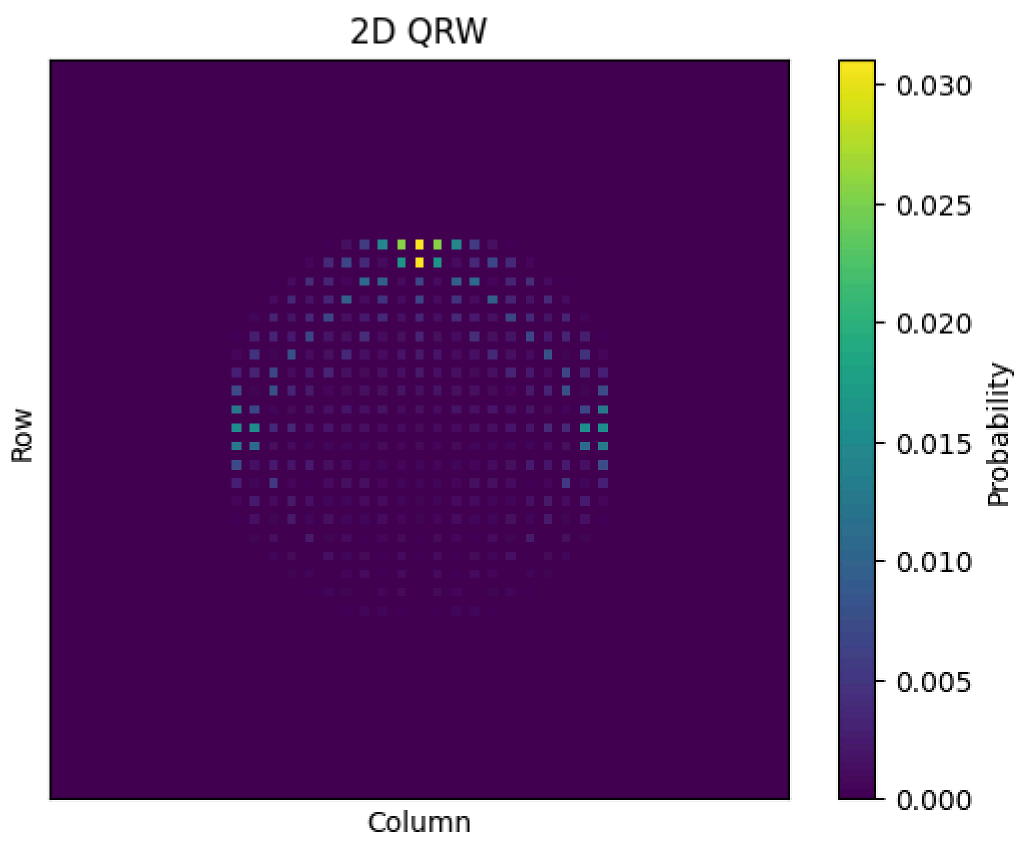
In a dynamic variant of the walk, the coin operator R(θ) is adjusted at each step by modifying the parameter θ. Specifically, θ evolves dynamically as:

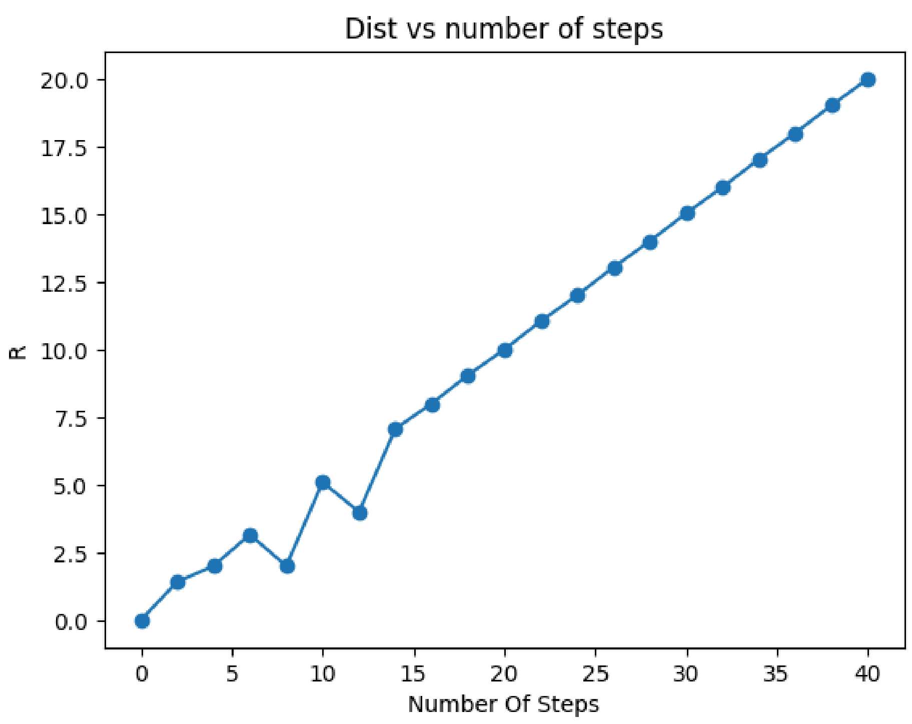
$$\theta = \frac{2\pi i}{(steps - 1)}$$

where *i* is the current step index. This modifies the Hadamard coin operation at each step. With the dynamically varying θ, the quantum walk evolves differently as the walker experiences a changing superposition at every step. This creates new interference patterns, which can influence the probability distribution significantly.

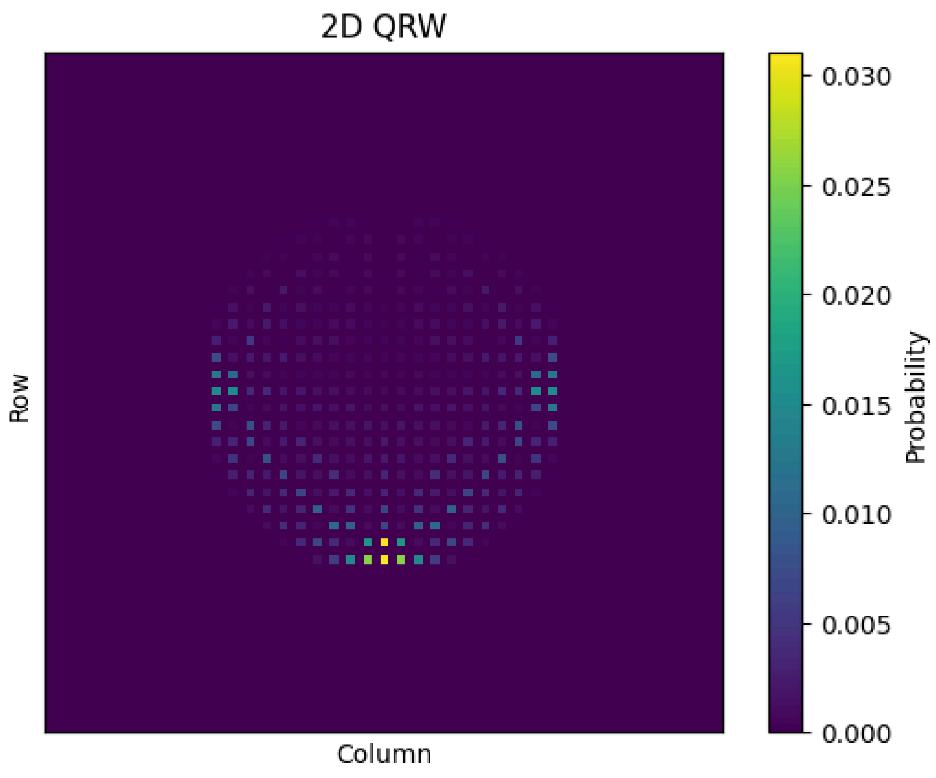
6.3 Results

6.3.1 For a 2D Quantum Random Walk on a 81 × 81 grid with 20 steps and initial coin/spin state $|\uparrow\rangle$:

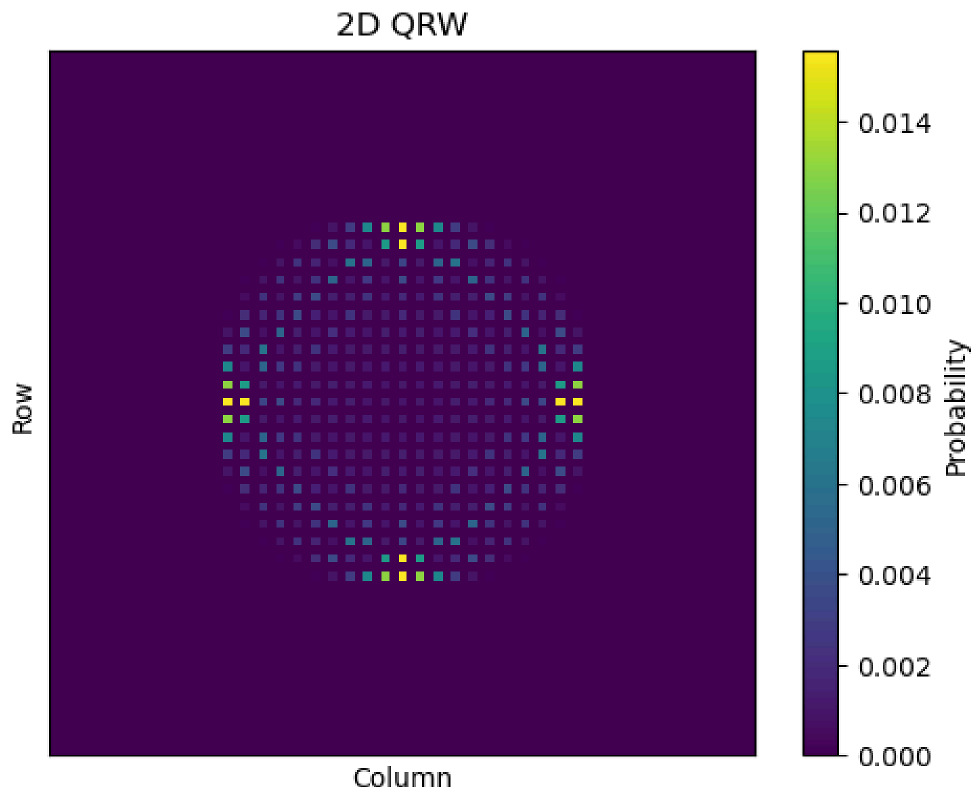




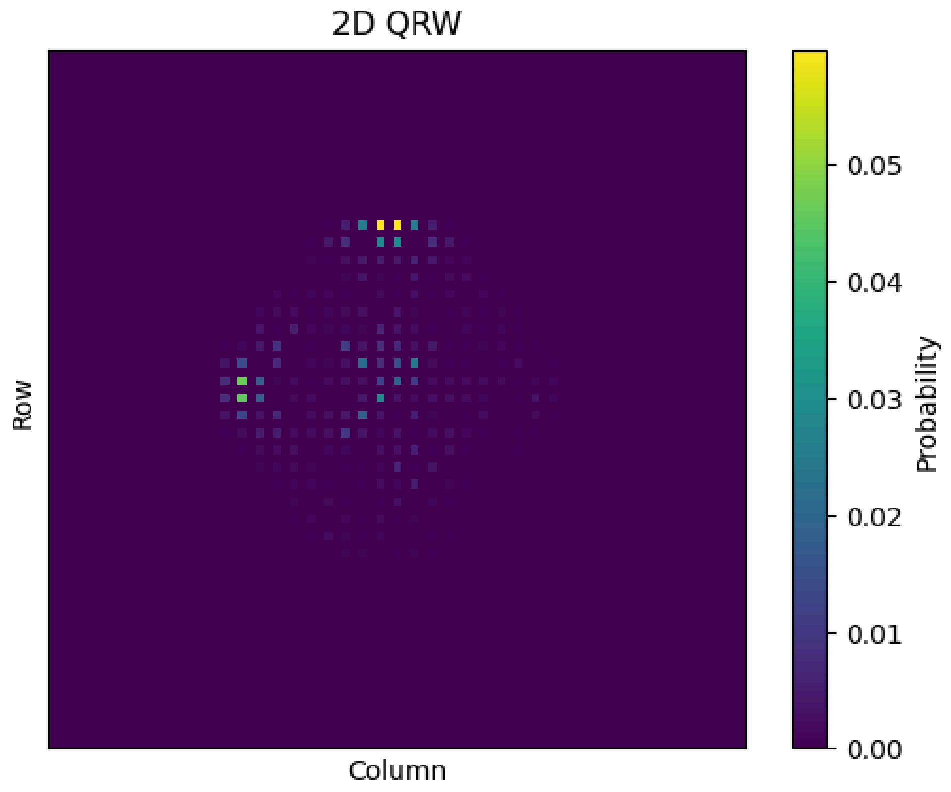
6.3.2 For a 2D Quantum Random Walk on a 81 × 81 grid with 20 steps and initial coin/spin state | ↓ > :



6.3.3 For a 2D Quantum Random Walk on a 81 × 81 grid with 20 steps and initial coin/spin state $(\frac{1}{\sqrt{2}}|\uparrow> - \frac{i}{\sqrt{2}}|\downarrow>)$:



6.3.4 For dynamically varying θ with initial coin/spin state $|\uparrow>$:



6.3.5 Analysis of the different plots

Starting with the $|\uparrow>$ state, the walker exhibits a higher probability distribution near the top. In the symmetric superposition state $(\frac{1}{\sqrt{2}}|\uparrow> - \frac{i}{\sqrt{2}}|\downarrow>)$, the probability forms four peaks—top, bottom, left, and right—highlighting the inherent symmetry in this setup. With the $|\downarrow>$ state, the distribution shifts to a pronounced bottom peak, mirroring the initial directional orientation. Interestingly, introducing a dynamically varying θ produces a markedly large peak near the top and a smaller peak near the left, disrupting symmetry and generating an asymmetrical but visually compelling pattern.

6.4 Code snippet

```python
# -*- coding: utf-8 -*-
"""QRW2D.ipynb

Automatically generated by Colab.

"""

# Imports
import numpy as np
import matplotlib.pyplot as plt

steps = 20
n = 4*steps+1

# Coin

def coin(theta):
  H = np.array([[np.cos(theta), np.sin(theta)], [np.sin(theta),
-np.cos(theta)]])
  I = np.eye(n**2)
  H_tilda = np.kron(H,I)
  return H_tilda

def flip(state, theta):
  return coin(theta) @ state

# Prerequisites
up_ket = np.array([[1],[0]])     # |0>
down_ket = np.array([[0],[1]])  # |1>
```

```
up_bra = np.array([[1, 0]])
down_bra = np.array([[0, 1]])

outer_product_left_term = np.dot(up_ket,up_bra)
outer_product_right_term = np.dot(down_ket,down_bra)

# S_y
T_lY = np.eye(n**2,k = n)
T_rY = np.eye(n**2,k = -n)

shift_left_termY = np.kron(outer_product_left_term,T_lY)
shift_right_termY = np.kron(outer_product_right_term,T_rY)
final_shift_termY = shift_left_termY + shift_right_termY

def shiftY(state):
  return final_shift_termY @ state

# S_x
T_lX = np.eye(n**2,k = 1)
for i in range(n-1,n**2,n):
  if i < n**2-1:
     T_lX[i][i+1] = 0

T_rX = np.eye(n**2,k = -1)
for i in range(n,n**2,n):
  if i > 0:
     T_rX[i][i-1] = 0

shift_left_termX = np.kron(outer_product_left_term,T_lX)
shift_right_termX = np.kron(outer_product_right_term,T_rX)
final_shift_termX = shift_left_termX + shift_right_termX

def shiftX(state):
  return final_shift_termX @ state

def walk(state, theta):
  intermediate = shiftX(flip(state,theta))
  return shiftY(flip(intermediate,theta))

def maxProbDist(state, plot, step):
  values = state.tolist()
  idk = []
  for i in range(0,len(values)//2):
      idk.append(values[i][0]**2)

  for i in range(len(idk)):
      idk[i] += values[i+len(values)//2][0]**2

  matrix = np.array(idk).reshape(n,n)
  max_index = np.unravel_index(np.argmax(matrix), matrix.shape)
  plot.append((step,np.sqrt((n//2 - max_index[0])**2+ (n//2 -
max_index[1])**2)))

def walker(init_coin, theta_variation = True, plot_title = '2D QRW'):
  # Simulate the walk
  initial_pos = [0 for i in range(n**2)]
  initial_pos[((n**2)//2)] = 1

  initial_pos = np.array(initial_pos).reshape(-1,1)

  init_state = np.kron(init_coin,initial_pos)
  st = init_state

  r_plot = []
  maxProbDist(st,r_plot,0)
  for i in range(steps):
      if theta_variation:
      st = walk(st,2*i*np.pi/(steps -1))
      else:
      st = walk(st,np.pi/4)
      maxProbDist(st,r_plot,2*(i+1))

  # Generate the plot matrix
  values = st.tolist()

  idk = []
  for i in range(0,len(values)//2):
      idk.append(np.square(np.abs(values[i][0])))

  for i in range(len(idk)):
      idk[i] += np.square(np.abs(values[i+len(values)//2][0]))

  plot_matrix = np.array(idk).reshape(n,n)

  plt.figure()
  plt.imshow(plot_matrix, cmap='viridis', interpolation='nearest')
```

```
  plt.xticks([])
  plt.yticks([])
  plt.colorbar(label='Probability')  # Add a colorbar to show the scale
of values
  plt.title(plot_title)
  plt.xlabel('Column')
  plt.ylabel('Row')
  plt.show()

  # Distance plot
  x, y = zip(*r_plot)
  plt.plot(x, y, marker='o')
  plt.xlabel('Number Of Steps')
  plt.ylabel('R')
  plt.title('Dist vs number of steps')
  plt.show()

init_coin = np.array([[1/np.sqrt(2)],[1j/np.sqrt(2)]])

walker(init_coin, theta_variation = False, plot_title = '2D QRW')
```

7. Conclusion

In conclusion,, we successfully adapted the Initial Position Resetting (IPR) results from Chatterjee et al. [2] into our quantum random walk (QRW) model. This adaptation highlights a significant outcome: systems where particles are governed by both spin and spatial degrees of freedom, we observe a substantial increase in detection probability and a reduction in mean first detection time.

Additionally, we implemented a computationally efficient simulation of a 2D QRW (2D lattice with 2 spin degrees of freedom), providing valuable insights into the behavior of quantum walkers and their variations across a two-dimensional lattice.

Future objectives include conducting continuous-range simulations for the results discussed in Section 5.3, further refining our understanding of first detection times. Beyond this, exploring the effects of resetting within 2D QRWs presents an exciting avenue for advancing this work.

8. References

1. Kempe, J. (2003). Quantum random walks: An introductory overview. *Contemporary Physics, 44*(4), 307–327. https://doi.org/10.1080/00107151031000110776
2. Chatterjee, P., Aravinda, S., & Modak, R. (2024). Quest for optimal quantum resetting: Protocols for a particle on a chain. *Physical Review E, 110*(3), 034132. https://doi.org/10.1103/PhysRevE.110.034132
3. Yin, R., & Barkai, E. (2023). Restart expedites quantum walk hitting times. *Physical Review Letters, 130*(5), 050802. https://doi.org/10.1103/PhysRevLett.130.050802