A Study of Quantum Walks and Its Application

Chen Wu
PHYSICS 231 Final Project Report
Department of Applied Physics, Stanford University

(Dated: June 5, 2019)

This report reviews the concepts of Quantum Walks (QW) including both discrete-time Quantum Walks (DTQW) and continuous-time Quantum Walks (CTQW). Comparison between both of them and classical random walk is illustrated from the perspective of python simulation. The analytical derivation of DTQW with Discrete Fourier Transform (DFT) will be presented. As an application of Quantum Walk, a case of 1-D topological phase transition will be discussed. A simulated Quantum Walk will be used to reveal the bound state at the position where the topological transition occurs. Generalization to 2-D topological transition will be briefly introduced. A experimental realization of the Quantum Walks will be mentioned to show that this algorithm is physically meaningful for future research.

I. INTRODUCTION

Quantum Walk (QW) is usually regarded as a counterpart of Classical Random Walk in the quantum processing realm. At the beginning of the invention of QW, the terminology was Quantum Random Walks instead of simply Quantum Walks. It is not a big difference, but this disagreement of terminology implies that people realized that "Quantum Random Walks" is not random at all[6]. At least, from the perspective of simulation, Quantum Walks will be much less random than the Classical Random Walk. The reason is that, in Classical Random Walk, the changing parameter is the position of the particle. However, the changing parameter in Quantum Walk is the states. The inner product of the state after a number of steps is just the probability of the position that we can find the particle.

We will see later that the difference between the Classical Random Walk and Quantum Walks are enormous. At first glance, maybe they are not at all similar. It will be shown by simulation that the propagation speed of the state in Quantum Walk will be much faster than the Classical Random Walk, so QW has a great potential to be implemented in the searching algorithm. (There will be no ambiguity if we say the propagation of the probability density because the propagation of states will be the propagation of the density operator in Heisenberg's Picture, though we will stick to Schrodinger's Picture in this paper.) Apart from that, the protocol used in a QW can be insightful for experimentalists to apply optical implementation on condensed matter problems. In general, Quantum Walk will be a useful method for both theoretical and experimental research.

The structure of this paper will be as follows: In sections II, the Classical Random Walk will be briefly reviewed. From the propagating model of the Classical Random Walk, Markov Chain, the idea of the unitary transformation of the state in Quantum Mechanics will be explicitly mapped. We will see that QW is actually something that's entirely natural rather than some arbitrarily defined method. In section III, Quantum Walk will be introduced. Both of the discrete-time Quantum Walk (DTQW) and continuous-walk Quantum Walk will be discussed. The simulation results of both cases will be shown, including the comparison between the QW and Classical Random Walk. Section IV will be used to introduce the analytical solution of the DTQW and compare with the simulated results[1][6]. In section V, an application of the Quantum Walk, specifically DTQW, on a 1-D topological state will be introduced. Takuya Kitagawa's paper gives a comprehensive introduction to this approach[5]. We will briefly summarize it and show some simulation results. Apart from the theory, the motivation and the future research direction of applying DTQW on the study of topological transition will be discussed. For all the sections mentioned, the simulations are carried out by using either Python or Wolfram Mathematica with only basic calculation packages.

II. A HINT OF CLASSICAL RANDOM WALK

A. Basic setup

There is a close relationship between the Classical Random Walk and Quantum Walk. In many cases, a quantum mechanical description is an approach that quantizes the classical description. Therefore, it will be insightful to start the discussion of Quantum Walk by discussing the Classical Random Walk first.

We only talk about the simplest model of Classical Random Walk here, the one with equal probabilities of moving right and left for 1 unit length. Two parameters, t and n, need to be clarified. t is defined to be the number of steps to propagate the particle or the walker, which means $t \in \{0, 1, 2, 3 \dots\}$. n is defined to be the position of the walker. In terms of graph theory, n is equivalent to vertices' number of a graph[3]. Therefore,

as a convention, $n \in \mathrm{Z}$.

The procedure is to toss a 2-sided coin and decide to go to left or right by 1 unit during each step t. It is a well-known result that after t steps, assuming the walker is at $n = 0$ vertices at step t, the probability of the walker standing on the n^{th} position is

$$P(n,t) = \frac{1}{2^t}\left(\frac{t}{\frac{n+t}{2}}\right) \text{ if } n+t \text{ is even;}$$
$$P(n,t) = 0 \text{ if } n+t \text{ is odd.} \tag{1}$$

By using this expression of probability distribution, it is not hard to get the standard deviation with respect to t steps: $\sigma(t) = \sqrt{t}$.

According to this setup, a simulation is carried out. To get the probability distribution intuitively, 10000 particles are located in the origin $n = 0$ and later are propagated for 100 steps ($t = 100$). By reading the equation (1), the probability of n will be 0 when n is an odd number because t is chosen to be an even number. We can see that the distribution of this Random Walk is approximately a normal distribution as expected and the standard deviation of the propagation is approximately the trend of $\sqrt{t}$.

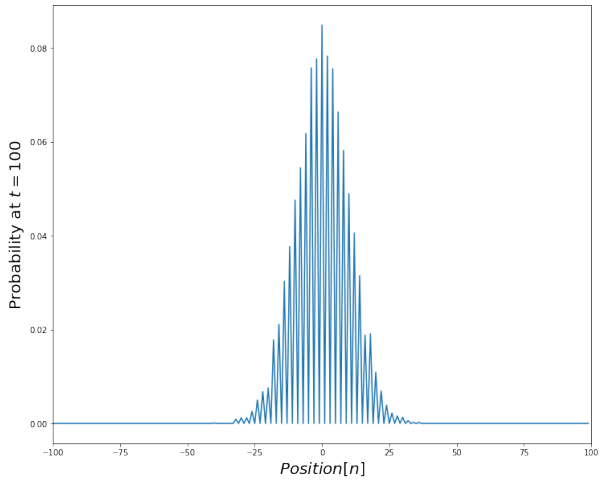


Figure 1: The probability distribution $P(t = 100)$ after 100 steps of propagation in Classical Random Walk. 10000 particles are used in this case.

B. Model of probability propagation

After setting up the probability of specific numbers of step at a specific vertex, a propagating method can be set up. By considering the principle of random walk in a simplified manner, the shifting at t^{th} step is only determined by the walker's position at $(t-1)^{th}$ step or $t - \epsilon$ time for discrete and continuous time, respectively. This is exactly the idea of the Markov Chain.

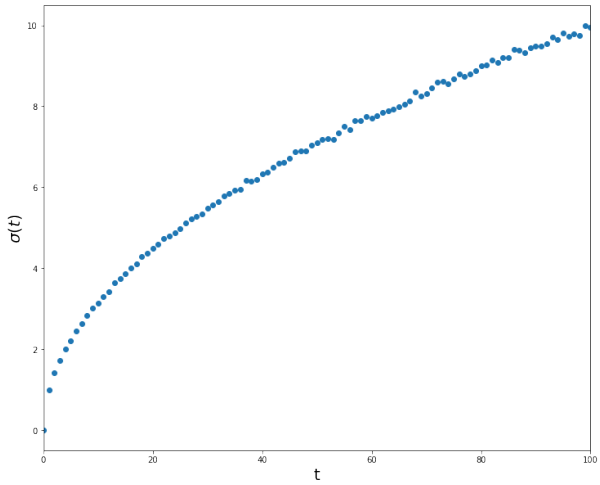


Figure 2: The standard deviation changes with the number of steps. It is roughly $\sqrt{t}$.

1. *Discrete-Time Markov Chain*

For a Classical Random Walk or say a undirected graph $G(V, E)$[3], all the propagation information is contained in a stochastic matrix M, in which $M_{ij} = \frac{1}{d_i}$. Here, d_i is the degree of the vertex i, or say how many lines leaving i. $\vec{p}(t)$ is defined to be the probability of n-sites at step t. Therefore, $\vec{p}(t)$ has n components with each component representing the probability of finding the walker at that correspondence site. The propagation of the probabilities of two consecutive step is $\vec{p}(t+1) = M\vec{p}(t)$. By using this recursive expression we can get the probability at any arbitrary step t:

$$\vec{p}(t) = M^t\vec{p}(0), \tag{2}$$

in which the $\vec{p}(0)$ in our case is that $p_0(0) = 1$ and 0 for other components.

2. *Continuous-Time Markov Chain*

Another time of model is called the Continuous-Time Markov Chain. The difference from the model above is that, instead of defining time step to be a natural number, we need to define the difference between two time steps to be a real number ϵ where ϵ can be arbitrarily small. A transition probability, γ, per unit time also needs to be defined. Therefore, at time t, a transition matrix M_{ij} and a generating matrix H_{ij} can be defined

as[6]:

$$M_{ij}(\epsilon) = \begin{cases} 1 - d_j\gamma\epsilon, & \text{if } i = j. \\ \gamma\epsilon, & \text{otherwise.} \end{cases}$$
$$H_{ij}(\epsilon) = \begin{cases} d_j\gamma\epsilon, & \text{if } i = j. \\ -\gamma, & i \neq j. \\ 0, & i \text{ and } j \text{ are non-adjacent.} \end{cases} \quad (3)$$

The definition of H_{ij} here may seem non-intuitive at the first the glance, but it has the exact same meaning as the generator in the time-dependent Schrödinger Equation, which is the amount of probability would like to be added to the previous probability vector. It can be shown that M and H are related by a first order differential equation and the solution to that differential equation is[6]:

$$M(t) = e^{-Ht}. \quad (4)$$

And the final propagation equation is:

$$\vec{p}(t) = M\vec{p}(0). \quad (5)$$

III. SIMULATION OF QUANTUM WALK

Now it is the time to put the classical intuition into the Quantum Walks. Just like the Markov Chain, Quantum Walks can also be classified as a Discrete-Time version and a Continuous-Time version. And the difference is awfully similar to the difference between the two versions of the Markov Chain. We briefly introduce them here and provide the simulation results. An analytical derivation of the Discrete-Time Quantum Walk will be given in the next section.

A. Discrete-Time Quantum Walk (DTQW)

Similar to the corresponding Discrete-Time Markov chain, DTWQ has a step that the walker tosses a dice to decide to go to the left or the right. Therefore, to automate the propagation, two Hilbert spaces need to be defined. The first one is the position space, H_P, of the states $|x\rangle$. For the implementation, the space state is usually defined as discrete, so $|n\rangle$ will be used to represent the site number and $n \in Z$. The second one is the coin space or spin space, H_C, with the states $\{|0\rangle, |1\rangle\}$ in which $|0\rangle$ is spin-up and $|1\rangle$ is spin-down. Since the position will be related to the spin, the overall state will be in the tensor product Hilbert space: $H = H_C \otimes H_P$, which will be $|\psi\rangle_0 = |0\rangle|n\rangle$ and $|\psi\rangle_1 = |1\rangle|n\rangle$.

Since the states are defined, the protocol of the propagation can be defined now. Firstly, we want to toss the dice. A frequently used "tossing" mechanism is called Hadamard Gate[3][6]:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (6)$$

Usually, this is called a balanced unitary coin[3] because it gives no bias by acting on either $|0\rangle$ or $|1\rangle$ except off by a minus sign. The outcomes will all be a superposition of spin-up and spin-down. However, during the propagation, this minus sign will contribute a lot for destructive interference. Because the Hadamard Gate does not affect the position space, the operator acting on the state when tossing the dice should be $H \otimes I$ where I is the identity matrix.

After the dice has been tossed, a conditional translation operator, S, needs to act on resulting states. S is defined to be [6]:

$$S = |0\rangle\langle 0| \otimes \sum_{n=-\infty}^{\infty} |n+1\rangle\langle n| + |1\rangle\langle 1| \otimes \sum_{n=-\infty}^{\infty} |n-1\rangle\langle n|. \quad (7)$$

So $|0\rangle$ component in the state will make the walker moving forward and $|1\rangle$ component in the state will make the walker moving backward.

Now the overall operator U is:

$$U = S(H \otimes I). \quad (8)$$

This is the operator for only one time step t. Considering the Discrete-Time Markov Chain, this U operator is equivalent to M in Eq.2. Therefore, after N steps, the state will be:

$$|\Psi(N)\rangle = U^N|\Psi(0)\rangle \quad (9)$$

After N steps, if we measure the final state, we will see a probability distribution. It's important to notice that if we measure the state after each step, then we will just get the Classical Random Walk.

The simulation can be achieved in Python. The initial state is defined as $\Psi(t = 0) = |1\rangle|0\rangle$. After using 100 time steps, we can get an asymmetric probability distribution Fig.3. There are several things that are similar to the Classical Random Walk and there are also a lot of things which are drastically different from the Classical Random Walk. Since the total steps used is 100 which is even, it is expected that only the even sites will give the non-zero probability, which can be seen in the Fig.3 and it is similar to the claim in Classical Random Walks. However, different from the Classical Walks, the distribution is skewed to the negative sites. This is one of the key features of the Quantum Random Walk. The skewed direction depends significantly on the initial state and the tossing/coin operator we used. Here, the spin-down initial spin state is used and acted by the Hadamard Gate. An intuitive understanding is that Hadamard gate will give a global phase of π

to the $|1\rangle$ state. The minus sign gives more destruction to the wave going to the right than to the left. Because of the interference property in the quantum realm, Quantum Walk will be significantly difference from the classical counterpart in which the distribution will be symmetric around the mean.

Based on the same gate, Hadamard Gate, one can also produce a symmetric probability distribution by defining the initial condition to be $\Psi(t = 0) = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle) \otimes |0\rangle$. (Just a reminder, the space before the $\otimes$ is the spin space. The one after the $\otimes$ is the position space). Based on this initial condition, after 100 steps, we can get a symmetric probability distribution in Fig.4. Of course, that is still extremely different from the Classical Random Walk. It seems that no matter symmetric or asymmetric, Quantum Walks tend to push the probability distribution to the sides rather than maintaining a Gaussian distribution which will be reached in Classical Random Walks.

From two probability distributions, it is not clear to see the speed of the spread. In fact, the speed of spreading in Quantum Walks will be much faster than in Classical Walk. The intuition is that the Classical Walks tends to be localized but the Quantum Walks tend to be spread out. In Fig.5, the yellow line is the $\sigma(t)$ of the DTQW and the blue line is the previous Classical Random Walk. The $\sigma(t)$ is given by the equation[6]:

$$\sigma(t) = \sqrt{\sum_{n=-\infty}^{\infty} n^2 p(t,n)}. \tag{10}$$

Here, the standard deviation is calculated for the symmetric case in which the average site position is 0. The asymmetric case will have the exact same standard deviation for each time step. The spreading speed is not related to the initial condition. In this case, $\sigma(t) = 0.54t$ approximately. For higher time steps, $\sigma(t)$ of DTQW will be much higher than that of the Classical Random Walk which is $\sigma(t) = \sqrt{t}$.

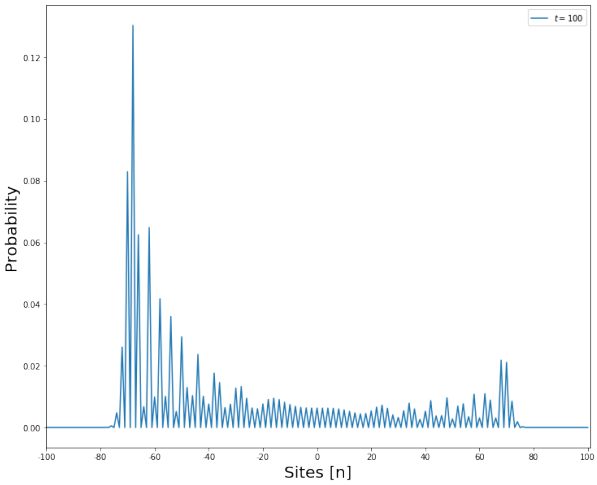


Figure 3: . The asymmetric probability distribution of the Hadamard Gate generated state. The initial state is $|1\rangle|0\rangle$. The time step used is 100. The odd sites have probability of 0.

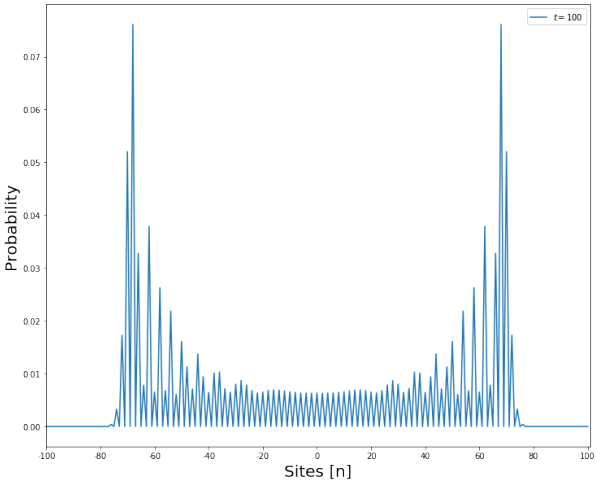


Figure 4: . The symmetric probability distribution of the Hadamard Gate generated state. The initial state is $\frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle) \otimes |0\rangle$. The time step used is 100. Just like the asymmetric case, odd sites have probability of 0.

B. Continuous-Time Quantum Walk (CTQW)

DTQW is not the only type of Quantum Walk. CTQW is the quantum counterpart of the Continuous-Time Markov Chain. Different from the DTQW, CTQW does not need to toss a coin, because the probability is inside the definition of the transition rate. Therefore, here what is propagated is the state of the spatial Hilbert space. Just like the Markov Chain case, a generator matrix will be needed to boost the propagation. Actually, this propagator will be even more intuitive than in the Markov Chain. After we defined $\hbar$ to be 1 for simplicity, the generator is just the Hamiltonian. Recalling the generator matrix in Markov Chain [11], the generator here becomes [6]:

$$H_{ij} = \begin{cases} 2\gamma & \text{if } i = j. \\ -\gamma, & i \neq j. \\ 0, & i \text{ and } j \text{ are non-adjacent.} \end{cases} \tag{11}$$

Since we already know where the walker needs to go, the Hamiltonian can be written down right away[6]:

$$H|n\rangle = -\gamma|n-1\rangle + 2\gamma|n\rangle - \gamma|n+1\rangle. \tag{12}$$

Because the continuous time needs to be dealt with, each time interval is an arbitrarily small

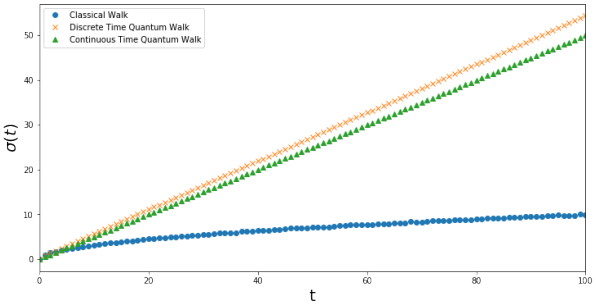


Figure 5: . The standard deviation of 3 types of random walk: Discrete-Time Quantum Walk; Continuous-Time Quantum Walk; Classical random Walk. The total time step is 100. The transition probability of the CTQW is $\frac{1}{2\sqrt{2}}$

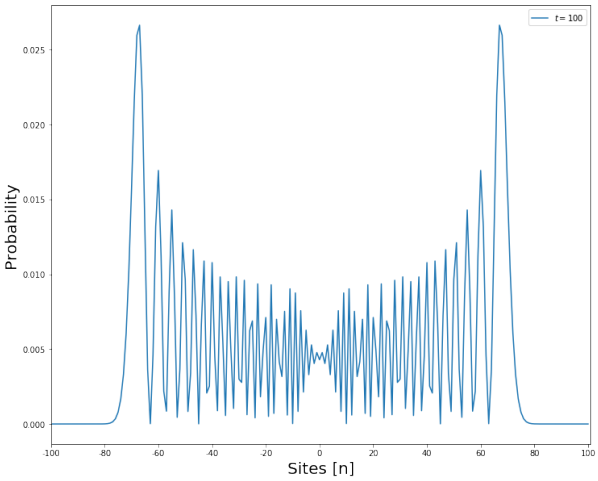


Figure 6: . The Continuous-Time Quantum Random Walk with 100 time steps, $\gamma = 1/2\sqrt{2}$. Initial condition is $|\Psi(0)\rangle = |0\rangle$

real number. A better way is to exponentiate the Hamiltonian just like the Continuous-Time Markov Chain case but with an extra $-i$:

$$U(t) = e^{-iHt}. \tag{13}$$

This will just be a time evolution of the state:

$$|\Psi(t)\rangle = e^{-iHt}|\Psi(0)\rangle \tag{14}$$

Unlike the DTQW, we need to define the transition rate, γ, for CTQW. In this simulation, it is defined that $\gamma = 1/(2\sqrt{2})$. Starting from the initial state, $|\Psi(0)\rangle = 0$, after 100 steps, a symmetric ballistic distribution will be produced.

The distribution of the CTQW is always symmetric because it has no spin component. What is only defined is adjacent transition probability and we defined it to be the same throughout all sites. The spreading speed is in Fig.5 and represented in the green line. This $\sigma(t)$) is not definite. By enlarging the value of γ, the slope can be significantly increased. Actually, by simply changing γ to $1/2$, the slope will be higher than the DTQW case. However, no matter what, just like DTQW, $\sigma(t)$ will be linear in t.

C. Summary for the simulated Quantum Walks

Several simulated results have been presented here. It will be better to summarize a little about what we just did.

Despite the large discrepancy between Classical Random Walks and Quantum Walks, the fundamental principle, Markov Chain, is the common base for both cases.

In Discrete-Time case, the propagation is defined in a single time step including the tossing/coin operation and the conditional selection operation. (Actually, a more general terminology for the tossing operator is the rotational operator. What will be needed is to rotate the initial spin state on the Bloch Sphere). Then within each time step, the whole package of operators will be used once. During this process, no measurement will be acted so that the probability can interfere itself without being demolished. After a sufficient amount of time steps, the measurement will be applied and the probability interference pattern will be established. The symmetry of the distribution is highly determined by the initial states and the gate used. Both asymmetric and symmetric distribution can be produced in DTQW.

In Continuous-Time case, a generator in continuous Hilbert Space (spatial space in this case) will be needed because the boost in the sites will be studied in an infinitesimal time interval. It is exactly the same as the definition of the position generator in the first Quantum Mechanics course. Then by exponentiating the Hamiltonian, the state can be directly propagated. In this case, no coin space will be needed. The probability is encoded in the parameter of transition probability between two adjacent sites, which later will be passed into the expression of Hamiltonian. Without the need of the coin, in the Hadamard Gate, it will always be symmetric about the site n=0.

In terms of spreading rate, both DTQW and CTQW will be faster than the Classical Random Walk by a power of $1/2$.

IV. ANALYTICAL SOLUTION FOR DISCRETE-TIME QUANTUM WALK

Simulation of Quantum Walks will be a quick way to implement and appreciate the dramatic difference from the Classical Random Walks. However, as a theory, an analytical basis will always be crucial. In this section, we will show the analytical results of the DTQW. CTQW's analytical solution

will not be introduced here. For a more detailed introduction of CTQW, [6] will be an appropriate source.

To be precise, the DTQW discussed here will be a two-way infinite timed Hadamard quantum walk [1]. Two-way infinite timed means that there will be no boundary on the way of propagating. So probability will not be absorbed in some specific sites along the propagation path. Hadmard is the Hadamard Gate that we have been used so far. It needs to be emphasized that Hadamard Gate is not a general gate for Quantum Walks. A general gate should be a linear combination of the representation of the $SU(2)$ rotation group. Later we will see an application of the Quantum Walks on topological transition in which an $SU(2)$ rotation matrix around y-axis of the Bloch sphere will be used. But now, in order to make an easier comparison to the simulation result, we will stick to the Hadamard gate.

Traditionally, there will be two approaches to analytically calculated the probability distribution. First one is called Schrödinger Approach in the discrete version in which the eigenvalues and eigenvectors are found in the Fourier space and inversely transform back to the get the states in spatial bases. Another approach will be using the discrete version of Path Integral [1]. In this paper, only the first one will be discussed. And the step on [6] and [1] will be followed.

A. Building blocks in Fourier basis

Recall that the unitary transformation matrix for Hadamard Gate is [6]. Suppose at time step t, the state, tensor product of spin and position, is $|\Psi(t)\rangle$, where:

$$|\Psi(t)\rangle = \sum_{x=-\infty}^{\infty} \psi_{0,x}(t)|0,x\rangle + \psi_{1,x}(t)|1,x\rangle \quad (15)$$

The probability at time step t will be $P_x(t) = |\psi_{0,x}(t)|^2 + |\psi_{1,x}(t)|$. Another constraint is the normalization condition in which the sum of x for this probability distribution will be 1 at any given time. Our goal now is to get the analytical expression for this probability distribution and compare with the simulated results presented in the last section.

At $t+1$ step, the state will be:

$$\begin{aligned}|\Psi(t+1)\rangle = \sum_{x=-\infty}^{\infty} \psi_{0,x}(t+1)|0,x+1\rangle \\ + \psi_{1,x}(t+1)|1,x-1\rangle,\end{aligned} \quad (16)$$

in which

$$\begin{aligned}\psi_{0,x}(t+1) &= \frac{\psi_{0,x}(t)+\psi_{1,x}(t)}{\sqrt{2}} \\ \psi_{1,x}(t+1) &= \frac{\psi_{0,x}(t)-\psi_{1,x}(t)}{\sqrt{2}}\end{aligned} \quad (17)$$

To calculate the probability distribution easily, we choose to work in the Fourier basis. Therefore, a Discrete Fourier Transform will be used where [1]

$$\tilde{\Psi}(k,t) = \sum_{x=-\infty}^{\infty} \Psi(x,t)e^{ikx}. \quad (18)$$

All $|x\rangle$ is transformed into $|\tilde{k}\rangle$ basis where k is the wave number physically. Now all the operators can be diagonalized in the new basis. Fourier transforming the basis of the conditional selection operator S can be expressed as:

$$\begin{aligned}S|j,\tilde{k}\rangle &= \sum_{x=-\infty}^{\infty} e^{ikx}|j,x+(-1)^j)\rangle \\ &= \sum_{x'=-\infty}^{\infty} e^{ik(x'-(-1)^j)}|j,x')\rangle \\ &= e^{-ik(-1)^j}|j,\tilde{k}\rangle.\end{aligned} \quad (19)$$

Then we need to get the expression of the Hadamard Gate in the Fourier basis. We consider the process from the perspective of the overall unitary transformation[8]:

$$\begin{aligned}U|j',\tilde{k}'\rangle &= S(\sum_{j"=0}^{1}\sum_{j'=0}^{1} H_{j"j'}|j',\tilde{k}')) \\ \langle j,\tilde{k}|H|j',\tilde{k}'\rangle &= \langle j,\tilde{k}|\sum_{j"=0}^{1} e^{-ik(-1)^{j"}} h_{j"j'}|j"\rangle|\tilde{k}'\rangle \\ &= e^{-ik(-1)^{j"}} h_{jj'}\delta_{jj"}\delta_{\tilde{k}\tilde{k}'} \\ &= \delta_{jj"}e^{-ik(-1)^{j"}} h_{jj'}\delta_{\tilde{k}\tilde{k}'}\end{aligned} \quad (20)$$

(This expression is a little different from the [6] where the derivation is a bit sloppy). Throughout this derivation, the lower case h_{ij} is the coefficient of the diagonalized Hadamard Gate. Based on the diagonalized result of U, a Fourier bases Hadamard Gate can be comfortably expressed as:

$$\tilde{h}_{jj'} = e^{-ik(-1)^{j"}}\delta_{jj"}h_{jj'}. \quad (21)$$

In terms of the matrix expression:

$$\tilde{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} e^{-ik} & e^{-ik} \\ e^{ik} & -e^{ik}\end{bmatrix}. \quad (22)$$

To be strict, this term is not exactly the Fourier Transform of the Hadamard Gate. Because the Fourier Transformed S only adds a phase to state[19]. From the above calculation, this phase is merged into the Fourier Transformed Hadamard Gate for simplicity. Therefore, in the Fourier bases, the unitary transformation of the states is

just the $\tilde{H}$. And it is not hard to see that, given $|\alpha_k\rangle$ is the eigenstates of $\tilde{H}$,

$$\begin{aligned} U|\alpha_k\rangle|\tilde{k}\rangle &= (\tilde{H}_k|\alpha_k\rangle)|\tilde{k}\rangle \\ &= \alpha_k|\alpha_k\rangle|\tilde{k}\rangle, \end{aligned} \tag{23}$$

where the eigenstates of U is known if the eigenstates of the $\tilde{H}$ is known. They even have the same eigenvalues. By simply using the traditional characteristic polynomial method, the eigenvalues and corresponding eigenvectors can be got:

$$\begin{aligned} \alpha_k &= e^{-i\omega_k}, \\ \beta_k &= -e^{i\omega_k}, \end{aligned} \tag{24}$$

and,

$$\begin{aligned} \tilde{H} &= \frac{1}{\sqrt{c^-}} \begin{bmatrix} e^{-ik} \\ \sqrt{2}e^{-i\omega_k} - e^{-ik} \end{bmatrix}. \\ \tilde{H} &= \frac{1}{\sqrt{c^+}} \begin{bmatrix} e^{-ik} \\ -\sqrt{2}e^{i\omega_k} - e^{-ik} \end{bmatrix}. \end{aligned} \tag{25}$$

by defining $\sin\omega_k = \frac{1}{\sqrt{2}}\sin k$ and $c^{\pm} = 2(1+\cos^2 k) \pm 2\cos k\sqrt{1+\cos^2 k}$.

Since k is a continuous space, we can write the unitary transformation in terms of continuous spectrum:

$$U^t = \int_{-\pi}^{\pi} e^{-i\omega_k t}|\alpha_k,\tilde{k}\rangle\langle\alpha_k,\tilde{k}| - e^{i\omega_k t}|\beta_k,\tilde{k}\rangle\langle\beta_k,\tilde{k}|\frac{dk}{2\pi} \tag{26}$$

To this stage, all the building blocks to get the analytical solution have been presented.

B. Analytical solution and comparison to the simulated results

As in the simulated case, we defined the initial state to be $|\Psi(0)\rangle = |j=0\rangle|x=0\rangle$ and the result should be expected to be a right-skewed probability distribution. By applying [26] on the initial state, the state at time step t should be:

$$\begin{aligned} |\Psi(t)\rangle = \int_{-\pi}^{\pi} & e^{-i\omega_k t}|\alpha_k,\tilde{k}\rangle \left\langle \alpha_k,\tilde{k}\middle|0,0\right\rangle \\ & - e^{i\omega_k t}|\beta_k,\tilde{k}\rangle \left\langle \beta_k,\tilde{k}\middle|0,0\right\rangle \frac{dk}{2\pi} \end{aligned} \tag{27}$$

By writing the initial state in Fourier bases, and then write it in terms of vector expression, the expression of $\left\langle \alpha_k,\tilde{k}\middle|0,0\right\rangle$ and $\left\langle \beta_k,\tilde{k}\middle|0,0\right\rangle$ can be calculated directly. And we can reach to the expression of the state:

$$\begin{aligned} |\Psi(t)\rangle = \int_{-\pi}^{\pi} & \frac{e^{-i(\omega_k t-k)}}{\sqrt{c^-}}|\alpha_k,\tilde{k}\rangle \\ & + \frac{e^{-i(\pi+\omega_k t+k)}}{\sqrt{c^+}}|\beta_k,\tilde{k}\rangle\frac{dk}{2\pi} \end{aligned} \tag{28}$$

From this expression, a version of [15] in Fourier basis can already be calculated with some algebra:

$$\begin{aligned} \tilde{\psi}_0(k,t) &= \frac{e^{-i\omega_k t}}{2}(1+\frac{\cos k}{\sqrt{1+\cos^2 k}}) \\ &\quad + \frac{(-1)^t e^{i\omega_k t}}{2}(1-\frac{\cos k}{\sqrt{1+\cos^2 k}}) \\ \tilde{\psi}_1(k,t) &= \frac{e^{ik}}{2\sqrt{1+\cos^2 k}}(e^{-i\omega_k t}-(-1)^t e^{i\omega_k t}) \end{aligned} \tag{29}$$

This is almost what we need. From this expression, we can easily get the state in the real space by Inverse Fourier Transform which, in this case, defined as

$$\psi(x) = \int_{-\pi}^{\pi} e^{ikx}\tilde{f}(k)\frac{dk}{2\pi}. \tag{30}$$

Therefore the final answer will be in the real space and in the integral form:

$$\begin{aligned} \psi_0(x,t) &= \int_{-\pi}^{\pi}(1+\frac{\cos k}{\sqrt{1+\cos^2 k}})e^{-i(\omega_k t-kx)}\frac{dk}{2\pi}; \\ \psi_1(x,t) &= \int_{-\pi}^{\pi}\frac{e^{ik}}{\sqrt{1+\cos^2 k}}e^{-i(\omega_k t-kx)}\frac{dk}{2\pi}. \end{aligned} \tag{31}$$

This expression only walks for the case when $n+t$ is even. When $n+t$ is odd, the state function will be 0 just like all the cases before.

This is the final solution of the analytical interpretation of the DTQW. With this expression, $P(n,t)$ is computed by Python Scipy Package. By choosing t=100, sites will be range from -100 to 100. It is computed and form an overall probability distribution in Fig.7. In this case, since t is even, the probabilities of even sites are calculated by using [31]. The probabilities of the odd sites are simply defined to be 0. By comparing with the simulated results described in the last section, it shows that they match perfectly including all zero- and, more importantly, non-zero-probability sites.

V. APPLICATION OF QUANTUM WALKS ON 1-D TOPOLOGICAL PHASE TRANSITION

Throughout the previous several sections, several aspects of Quantum Walks are introduced. Maybe it gives a sense that Quantum Walks might give more help to the searching methods in computer science algorithms than Physics. Here we will try to fix this kind of misunderstanding. In fact, Quantum Walks works quite well in studying real physical systems. Here an application of Quantum Walks, specifically the Discrete-Time

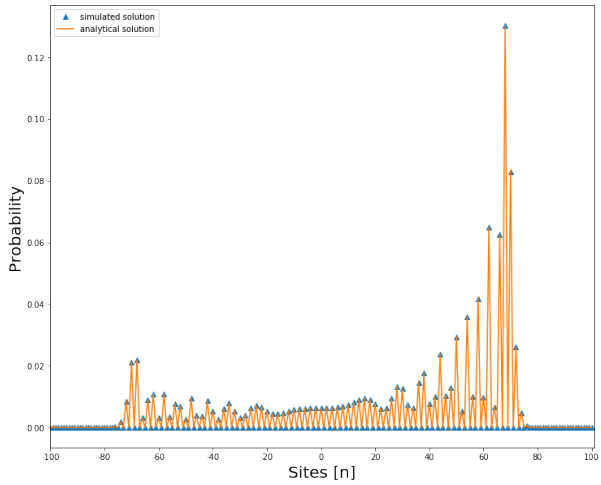


Figure 7: The comparison between the simulated solution and analytical solution. For the analytical solution, the probabilities are only defined for even sites. The probability is defined to be 0 for all odd sites. Therefore, it is exactly the same as the simulated results. To distinguish them clearly, the blue triangular dot is used for simulated results and the orange line is used for analytical solutions. The integral for the analytical solution is computed by the Scipy Package.

Quantum Walks, on a physical system will be introduced. The system we choose is the 1-D topological state of matter. A famous example will be Su-Schrieffer-Heeger Model. From the personal perspective, the reason to study this application is that it can actually be implemented without using any complicated and specific computational Packages but reveal the amazing truths of the topological state of matter.

A. Symmetry of DTQW

To understand this application, a deeper understanding of physics beneath the DTQW needs to be discussed. Recall the discussion of unitary transformation in the above sections. Now, following the terminology in [5], we call the rule in one time step the protocol of the walk. It is mentioned before that the Hadamard Gate we used is just a specific example. With the same protocol, the coin operator now becomes the y-direction $SU(2)$ rotation Matrix, R_y, instead of the Hadamard Gate. So now our unitary transformation for one time step protocol is:

$$U = S(R_y(\theta) \otimes I), \tag{32}$$

where

$$R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}. \tag{33}$$

From the time dependent Schrödinger transformation we know that $U = e^{-iH_{eff}\Delta t}$. The condition selection operator can be rewritten as:

$$S = \int_{-\pi}^{\pi} e^{ik\hat{\sigma}_z} \otimes |k\rangle\langle k| dk \tag{34}$$

In this case, we can write H_{eff} as:

$$H_{eff} = \int_{-\pi}^{\pi} (E(k)\vec{n}(k).\vec{\sigma}) \otimes |k\rangle\langle k|, \tag{35}$$

by using a trick that define E(k) to be: $\cos E(k) = \cos(\theta/2)\cos k$. Here $\vec{n}(k)$ is the Bloch vector of with the quasi-momentum of k. By saying quasi-momentum, it means k is defined through a periodic boundary with a length of 2π. In many literatures, the periodic interval is called the Brillouin Zone. From this trick, we can get the energy band structure with various θ and k.Fig.[8]. Clearly, the energy closes the energy gap when $k = 0$, where $E(k) = 0$. The gap also closed when $k = \pi$, where $E(k) = \pi$ because of the periodicity of k. The intuition up till now is that the Bloch vector is doing some spinning, or winding, around the center of the Bloch Sphere as k is scanning the Brillouin Zone. However, the situation is even better. In fact, the Block vector is winding the equator of the Bloch Sphere. $\vec{n}(k)$ is determined at the same time that our Hamiltonian is defined. That is, the protocol the DTQW determines the Bloch vector, and the expression of $\vec{n}$ is:

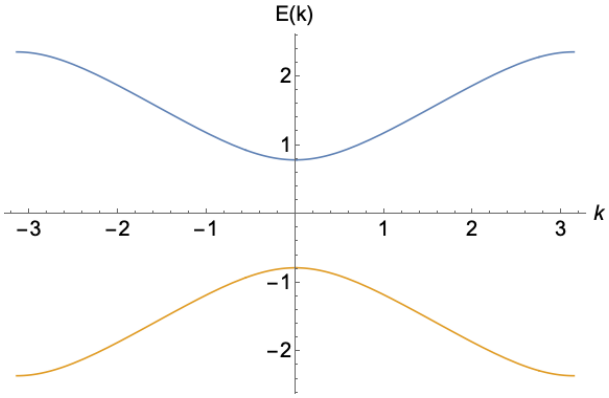
$$\vec{n}(k) = \frac{(\sin\frac{\theta}{2}\sin k, \sin\frac{\theta}{2}\cos k, -\cos\frac{\theta}{2}\sin k)}{\sin E(k)} \tag{36}$$

From this equation, it shows that $\vec{n}$ is not winding any other latitude but exactly the equator. But there is still a tricky point of the definition of $\vec{n}$. When k is 0 or π, the energy gap closes and thus creates degeneracies. Therefore, it is hard to say where $\vec{n}(k)$ is pointing on the Block Sphere surface when k is these two values. So $\vec{n}(k)$ is not defined for $k = 0, \pi$.

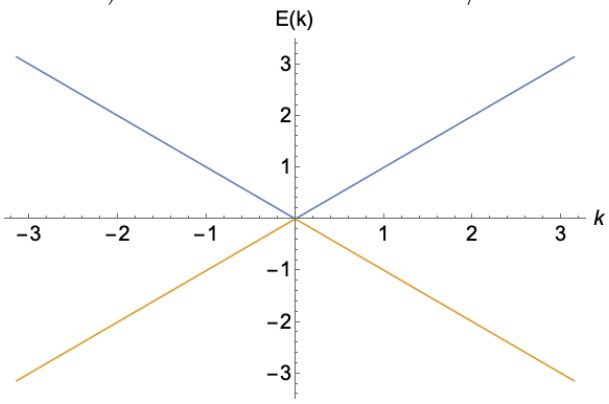
Now based on this intuition of a needle winding the equator of the Bloch Sphere, it is the time to discuss the symmetry in this model. Since the Bloch vector is lying on the equator, a normal vector, $\vec{A}(\theta)$ can be defined such that it is always perpendicular to the plane winding by the Bloch vector. $\vec{A}$ is defined to be:

$$\vec{A} = (\cos\frac{\theta}{2}, 0, \sin\frac{\theta}{2}) \tag{37}$$

. Recall that the representation of rotation around an arbitrary axis $\hat{A}$ on Bloch Sphere for angle of θ is $\exp\left(-i\theta\hat{A}.\vec{\sigma}\right)$. If we want to rotate $\vec{n}$ around $\vec{A}$ to reach the state of $-\vec{n}$, the general rotation representation becomes $\exp\left(-i\pi\vec{A}.\vec{\sigma}\right)$. This rotaion is defined to be Γ. Since H_{eff} is odd about



a) The band structure when $\theta = \pi/2$



b) The band structure when $\theta = 0$

Figure 8: The band structure of the Hamiltonian that defined in [35]. Plotted by Mathematica

$\vec{n}(k)$, under this rotation, H_{eff} becomes $-H_{eff}$. That is:

$$\Gamma^{-1} H_{eff} \Gamma = -H_{eff}. \tag{38}$$

This is a symmetry expression and it is called sublattice symmetry which will guarantee the eigenenergy appears in pairs, E and $-E$, with the corresponding eigenstates of $|\psi\rangle$ and $\Gamma|\psi\rangle$, respectively. It is also a known property of the Su-Schrieffer-Heeger Model (SSH), a type of 1-D topological model, mentioned above. But here, we derived solely from the DTWQ protocol without even touching the physical properties of the SSH model. This symmetry gives an appropriate reason why it could be possible to use DTQW to simulate the 1-D topological system. The topological number here is the winding number that the Bloch vector spinning around the equator. And in the legal region, $\theta \neq 0, 2\pi$, the winding number is 1 when k scans the Brillouin zone once. It is conventionally called Z=1 topological state.

Following the property of the topological state, the topological number will only change value when the energy gap, similar to Fig.[8], closes. (But this is not an "if and only if" statement. The reverse statement will not be necessarily true.) In the case discussed above, suppose θ is moving in the interval $(0, 2\pi]$. When θ encounters 2π, the energy closes gap as at $E = 0, \pi$. Fig.[8(b)]. Now the Bloch vector has winded around the equator once. After 2π or 0, because these are the same points, this process will proceed again and the Bloch vector will still wind around the equator once. Therefore, the winding number, W, is staying at 1. For this reason, this system has only one topological phase. Even if we regard the gap-closing as a bound case, the topological number keeps to be 1 again and again through the winding.

We must be aware of that this symmetry constraint comes from the definition of Hamiltonian or, equivalently, the DTQW protocol. Although we calculated the symmetry expression by thinking about Bloch vectors, it is the symmetry of the Hamiltonian. In other words, this sublattice symmetry of the Hamiltonian forces the Bloch vector to be on the equator. If we redefine a completely different protocol, a completely different Hamiltonian will be generated, which does not necessarily have such symmetry. The new Hamiltonian might not have a Bloch vector lying on the equator. Then the winding number of Bloch vector cannot be regarded as a proper topological number.

B. Simulation of a system with two topological phases with DTQW

In the previous subsection, the algorithm-like Quantum Walk concepts become physical by converting the DTQW protocol into the expression of Hamiltonian. But the single-phase topological system in the past section is not so interesting. In this subsection, the DTQW protocol will be again modified to possess two distinct topological phases without breaking the sublattice symmetry of the Hamiltonian.

From the last subsection, it is claimed that the energy gap will be closed at some specific value of θ and also claimed that, after passing the point where the energy gap closes, a new topological phase might appear. Therefore, changing the definition of θ in the DTQW protocol might be a great chance to create a new topological phase. On the other hand, to make such boundary of different topological states visible, it will be better to, in some way, connect the definition of θ to the sites (n), which is absolutely visible in the DTQW simulation.

Strictly following the paper [5], the new protocol will be defined as $U(\theta_1, \theta_2) = T_1 R_y(\theta_2) T_0 R_y(\theta_1)$, where $R_y(\theta)$ is the same as in the last subsection

and

$$T_0 = \sum_{n=-\infty}^{\infty} (|n+1\rangle\langle n| \otimes |0\rangle\langle 0| + I \otimes |1\rangle\langle 1|);$$
$$T_1 = \sum_{n=-\infty}^{\infty} (|n-1\rangle\langle n| \otimes |1\rangle\langle 1| + I \otimes |0\rangle\langle 0|). \tag{39}$$

(Note that in order to stay close to the article [5], the order of the tensor product of the Hilbert space is now changed to $H = H_P \otimes H_C$).

Following the procedure of the last subsection, the band structure can be calculated:

$$\cos E(k) = \cos\frac{\theta_1}{2}\cos\frac{\theta_2}{2}\cos k - \sin\frac{\theta_1}{2}\sin\frac{\theta_2}{2};$$
$$n_x(k) = \frac{\cos\frac{\theta_2}{2}\sin\frac{\theta_1}{2}\sin k}{\sin(E(k))},$$
$$n_y(k) = \frac{\cos\frac{\theta_1}{2}\sin\frac{\theta_2}{2} + \cos\frac{\theta_2}{2}\sin\frac{\theta_1}{2}\cos k}{\sin(E(k))},$$
$$n_z(k) = -\frac{\cos\frac{\theta_1}{2}\cos\frac{\theta_2}{2}\sin k}{\sin(E(k))}. \tag{40}$$

It looks ugly. But it is easy to check that by defining $\vec{A}$ vector to be $(\cos\theta_1/2, 0, \sin\theta_1/2)$ and the same symmetry operator, Γ, as the last subsection, the sublattice symmetry can be preserved and $\vec{A}$ will be perpendicular to the plane that $\vec{n}(k)$ is winding. It is defined in the articles [5][4] that there will be two winding numbers: $W = 0$ when $\left|\frac{\tan\theta_2/2}{\tan\theta_1/2}\right| > 1$ and $W = 1$ when $\left|\frac{\tan\theta_2/2}{\tan\theta_1/2}\right| < 1$. (Personally, I do not think that counting winding numbers here makes a lot of sense. However, after animating the energy band using various θ_2 by Mathematica or Python, it will be easier to see that there are two distinct topological phases indeed. It will be in the code). We will quote the Transition Phase Diagram provided by the article [5] in Fig.[9] which contains all the information of the relationship between phases and θ_1 and θ_2.

Thinking about the process of simulation, a good way will be fixing θ_1 and varying θ_2 or vice versa. We will use the former one which is also in Fig.[9]. According to this idea, the energy band structure can also be calculated. The energy band structures with different θ_2 are presented in Fig.[10] where $\theta_1 = -\pi/2$ as always. According to Fig.[9], when θ_2 is around $\pi/2$, the system should be in $W = 1$ phase. In the dynamical description, the positive and negative band are moving closer. It is not so clear from the screenshot of the Mathematica Animation, but reading the relatively large value of peak of two energy bands may give us the clue that two bands are moving close to each other from some larger value of energy. Then when $\theta_2 = \pi/2$, the energy gap closes. Further increasing the value of θ_2, if we saw that the energy bands are leaving apart from each other

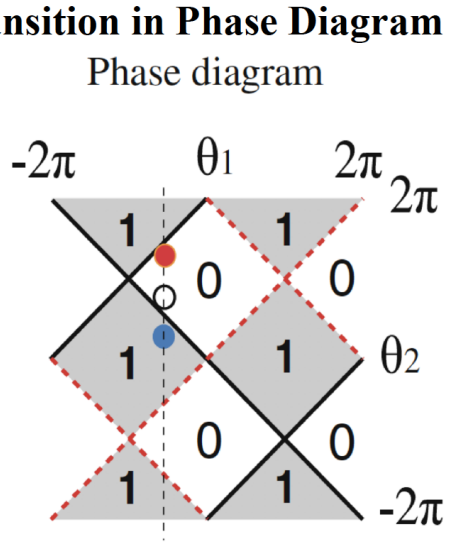


Figure 9: Transition Phase Diagram visualising that $W = 0$ when $\left|\frac{\tan\theta_2/2}{\tan\theta_1/2}\right| > 1$ and $W = 1$ when $\left|\frac{\tan\theta_2/2}{\tan\theta_1/2}\right| < 1$[5].

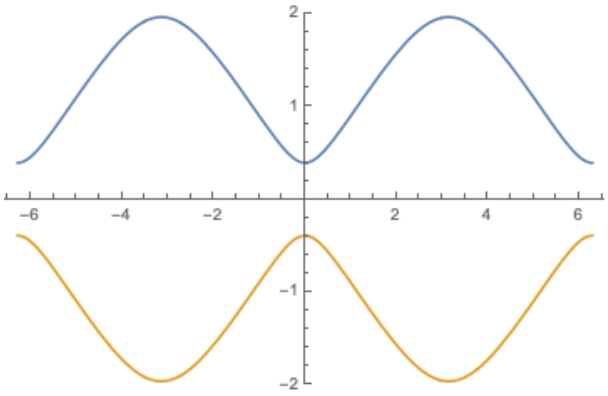
or the peaks of the energy bands are stretching back to be larger values following the opposite movement to the story happening in Fig.10.a, then it would probably be no boundary-crossing happened. However, the reality is that the two bands are not leaving each other. Peaks of two energy bands are squeezing and tend to be flat. In fact, when $\theta_2 = \pi$, two bands are almost flat with the absolute value that is less than ± 1. It means that there are two distinct phases on the two sides of $\theta_2 = \pi/2$.

To observe such effect in the simulation, it is also crucial to relate θ_2 to the sites number (n) which will be something visible in the probability distribution diagram. Still following the article [5], θ_2 is defined to be inhomogeneous along the displacement of the walker:

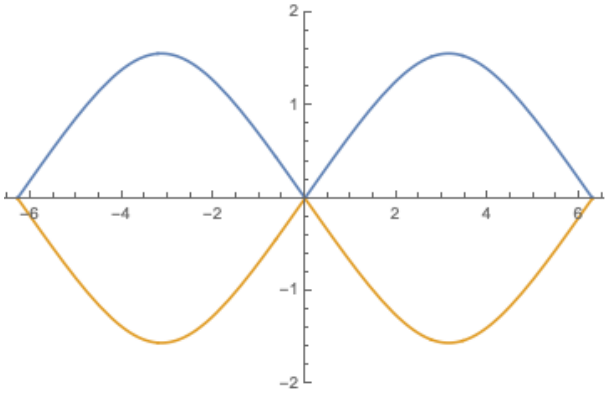
$$\theta_2 = \frac{1}{2}(\theta_{2-} + \theta_{2+}) + \frac{1}{2}(\theta_{2+} - \theta_{2-})\tanh x/3 \tag{41}$$

With this definition, when x is slightly positive, θ_2 will be approximately θ_{2+}. When x is slightly negative, θ_2 will be approximately θ_{2-}. So if θ_{2-} and θ_{2+} are defined on two sides of a phase boundary, $x = 0$ will be such phase boundary. Instead, if we defined θ_{2-} and θ_{2+} to be at the same phase, then throughout whole propagation, no boundary will be encountered.

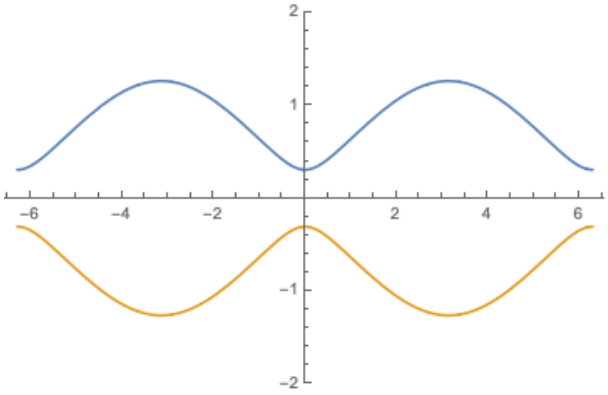
But what will happen if the walker is crossing the topological phase boundary? According to the theory of topological phase transition, there will be a bound state right on the boundary of two distinct topological phases. It is very robust and will not be removed by small perturbation of the Hamilto-



a) θ_2 in phase $W = 1$



b) θ_2 right on the topological boundary



c) θ_2 in phase $W = 0$

Figure 10: To read these three diagrams, always refer to [9]In this case, $\theta_1 = -\pi/2$ always. (a)θ_2 is around $pi/4$. According to [9], the system should be in the W=1 phase. (b)θ_2 is around $\pi/2$, which is about the boundary between two phases. (c)θ_2 is larger than $\pi/2$ but smalled than $3\pi/4$. It maybe not very clear from the static graph, but see carefully we will notice that (a) and (c) are actually too distinct phases.

nian. Intuitively the existence of the bound state is caused by the sublattice symmetry in which the energy always appear as pairs. Then it will be hard to split a zero energy state into a plus/minus pairs. The only way is to change the Hamiltonian so that to bring the so-called bulk states closer and then the bound states are possible to mix with the bulk states. This will bring a quarter of research about the topological state of matter so we just stop here and to simulate according to the protocol given in this subsection. The expectation is that a bound state will always be at $x = 0$ if the DTWQ pass through the topological phase boundary and some probability distribution will always be there to represent the bound state. If the transition carried out by the DTQW is in only one topological phase, then the probability around $x = 0$ will be pushed to side quickly just like a normal Quantum Walks ballistic pattern as the normal ballistic pattern shown by the previous simulations of Quantum Walks.

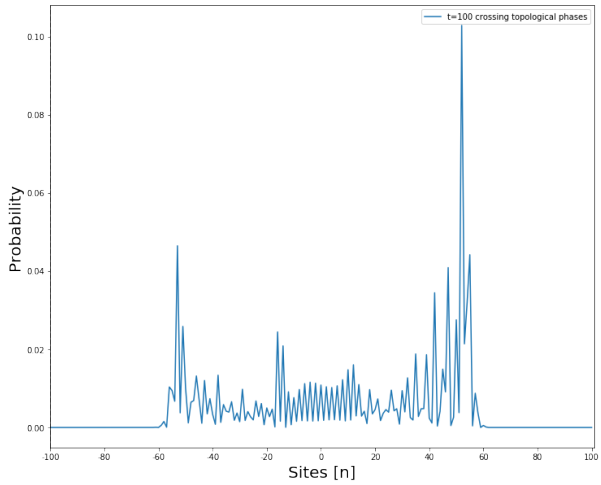
In the simulation, we choose the initial state to be $|\psi(0)\rangle = |n = 0\rangle|s = 0\rangle$. Then, 100 time steps are used. Compare to the result given by the article[5], this is a longer period. The reason is to check how persistent the bound state is. For the case of crossing boundary, $\theta_{2-} = \pi/4$ and $\theta_{2+} = 3\pi/4$ is chosen so, as we have seen earlier, that the system will cross the boundary at $\theta_2 = \pi/2$. For the case where no boundary has been crossed, $\theta_{2-} = 0$ and $\theta_{2+} = 0.99\pi/2$ is chosen so that the system is staying at the topological phase of W=1. The simulation is presented in Fig.[11]. In Fig.11.a, we can see a slight dense area around 0 but almost no probability appears for Fig.11.b. Actually, if less time is chosen, say $t = 30$ or $t = 60$, the difference will be more obvious. Numerically, the probability of Fig.11.a on $x = 0$ is around 0.0109 and the probability of Fig.11.b on $x = 0$ is around 0.00142. So numerically this is a nontrivial difference.

The author of the article[5] actually made an animation on Mathematica which is very easy to manipulate and see this effect. It will be in the code submitted.

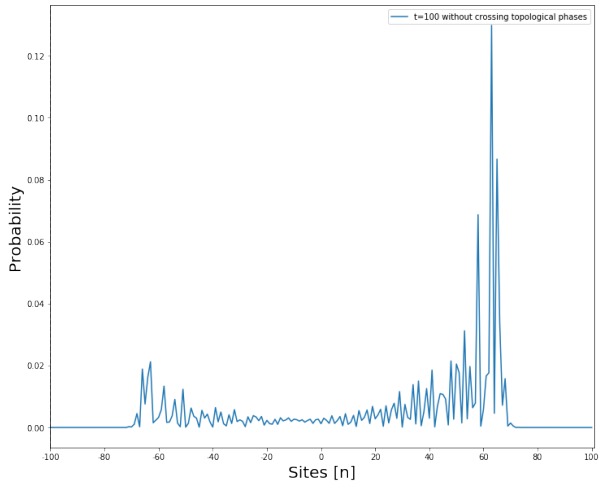
C. Extend to the 2D topological phase transition

From the previous section, we may get an intuition that the protocol can be easily modified and then more properties will pop out. Actually, this is just the intuition we need to use to modify the protocol so that a 2-D topological system can be realized. It includes three rotation operators and three conditional selection operators alternately acting on the initial state. The conditional selection will have two spatial dimensions. Therefore, the Hilbert space will be $H = H_x \otimes H_y \otimes H_c$. The conditional selections correspond to 3 boosts: only change x, only change y, and change x and y together. It will be a bit complicated, therefore it will not be implemented now but may be implemented in the future.

Recall that in the 1-D case in which the Bloch vector maps quasi-momentum k from the Brillouin



a) DTQW reached after crossing the topological phase transition. The time steps are 100. Initial state is $|0\rangle|0\rangle$. $\theta_1 = -\pi/2$, $\theta_{2+} = 3\pi/4$, $\theta_{2-} = \pi/4$



b) DTQW reached without crossing the topological phase transition. The time steps are 100. Initial state is $|0\rangle|0\rangle$. $\theta_1 = -\pi/2$, $\theta_{2+} = 0.99\pi/2$, $\theta_{2-} = 0$.

Figure 11: DTWQ for the case (a) that a topological boundary has been passed and (b) no topological boundary has encountered. Simulated by Python.

zone to the equator, the map is a 1D-to-1D map. However, in 2-D, the map is from the Brillouin zone to the surface of the Bloch sphere, which is a 1D-to-2D map. Therefore, the winding number is not a proper topological number anymore. The new topological number will be something like a wrapping number that counting how many times the Bloch vector has wrapped the whole surface of the Bloch sphere. There is a specific name for this wrapping number: Chern Number.

D. Physical implementation of DTWQ method

The reason why Quantum Walks will be a useful guideline for experimentalists is that it is usually possible to use cold atom or photonics way to implement those protocols and thus manipulate the topological state of matter in a more controllable way.

For example, in Karski group[2], the spin space was implemented by using two hyperfine energy level of Cs atom. The spin control or rotation can be achieved by the microwave cavity. The condition selection operation can be implemented by adiabatic translation of spin-dependent optical lattices[4].

Although the system they made cannot hold for large time steps right now[2][4], it is insightful that implementing different Quantum Walks protocols will be, in some way, doable. Since we have seen that Quantum Walks possess some symmetry properties, more symmetry properties might emerge by modifying the protocols. Physically implementing these protocols may provide us a new way to study the properties of matters. This is one of potentials of Quantum Walks in the field of physical implementation.

VI. CONCLUSION

In this section, several difficulties encountered when preparing this project and some topics that could be studied in greater depth will be concluded. The conclusions for the corresponding content are almost self-contained in the previous sections, which will not be restated here.

Throughout this report, two types of Quantum Walks, DTQW and CTQW, have been introduced. Compare to DTQW, CTQW is easier to implement in computational software or Python because it is simply the computation of the physical facts. It will be more complicated to implement DTQW. In the beginning, I was planning to use a Quantum Package of Mathematica to implemented DTQW, in which the notation is simply the Dirac Notation. Therefore the implementation will be straightforward. However, the speed of the code is much slower compared to Python even after significant optimization. It doable to use this specific Package to implement simple Hadamard Walk, but it will be even hard to just complicate the protocol for a little bit. Therefore, except plotting the energy band for the 1-D topological system, Python is used.

Python is surprisingly efficient to implement this algorithm. More importantly, the code can be implemented really similar to the mathematical formula of the Quantum Walks with the linear algebra functions of the most basic packages, Numpy and Scipy. The code

for basic DTQW is modified based on code posted Susan Stepney's Blog "https://susan-stepney.blogspot.com/2014/02/mathjax.html".

To be honest, the simulation of the 1-D topological system is not so satisfing. Actually, the simulation does not work well for some of the phase transitions. In addition, from Fig.11.b we can find that the bound state is not a Gaussian shape but it should be a Gaussian theoretically. In the Mathematica simulation provided along the article[5] in Fig.12, the shape of the bound state follows perfectly with the theory, which is even better than the figures given in the original paper[5]. In the original article, the peak value of the bound state is around 0.05. However, through this Mathematica simulation, the peak value is around 0.2. Recall that the peak value is around 0.01 in Python simulation, though it is already significantly higher than the peak value of the case where no boundary crossing occurs. It seems that by strictly following the protocol given in the paper, the simulation cannot be as good and as ideal as what they presented. Perhaps the definition of θ_2 will need to be tuned a little bit according to different simulation approaches.

Later, several sub-topics of this study will be accomplished. The first thing the 2-D topological system in which a 6-step protocol will be programmed. The second one is the application of CTQW. In this paper, CTQW was just touched a little bit, but no clue of applications is mentioned. Therefore. it will also be a good topic to study further.

[1] Andris Ambainis, Eric Bach, Ashwin Nayak, Ashvin Vishwanath, and John Watrous. One-dimensional quantum walks. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 37–49. ACM, 2001.

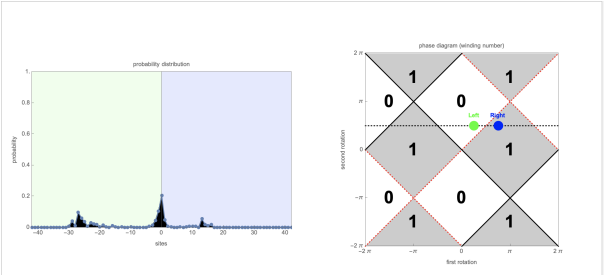
[2] Michal Karski, Leonid Förster, Jai-Min Choi, Andreas Steffen, Wolfgang Alt, Dieter Meschede, and Artur Widera. Quantum walk in position space with single optically trapped atoms. *Science*, 325(5937):174–177, 2009.

[3] Julia Kempe. Quantum random walks: an introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.

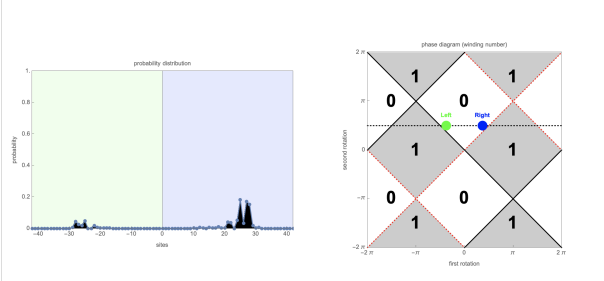
[4] Takuya Kitagawa. Topological phenomena in quantum walks: elementary introduction to the physics of topological phases. *Quantum Information Processing*, 11(5):1107–1148, 2012.

[5] Takuya Kitagawa, Mark S Rudner, Erez Berg, and Eugene Demler. Exploring topological phases with quantum walks. *Physical Review A*, 82(3):033429, 2010.

[6] Renato Portugal. *Quantum walks and search algorithms*. Springer, 2013.



a) DTQW reached after crossing the topological phase transition.



b) DTQW reached without crossing the topological phase transition.

Figure 12: DTWQ for the case (a) that a topological boundary has been passed and (b) that no topological boundary has encountered. It is simulated by Mathematica code written by Takuya Kitagawa, the author of [5], without using Quantum Package.

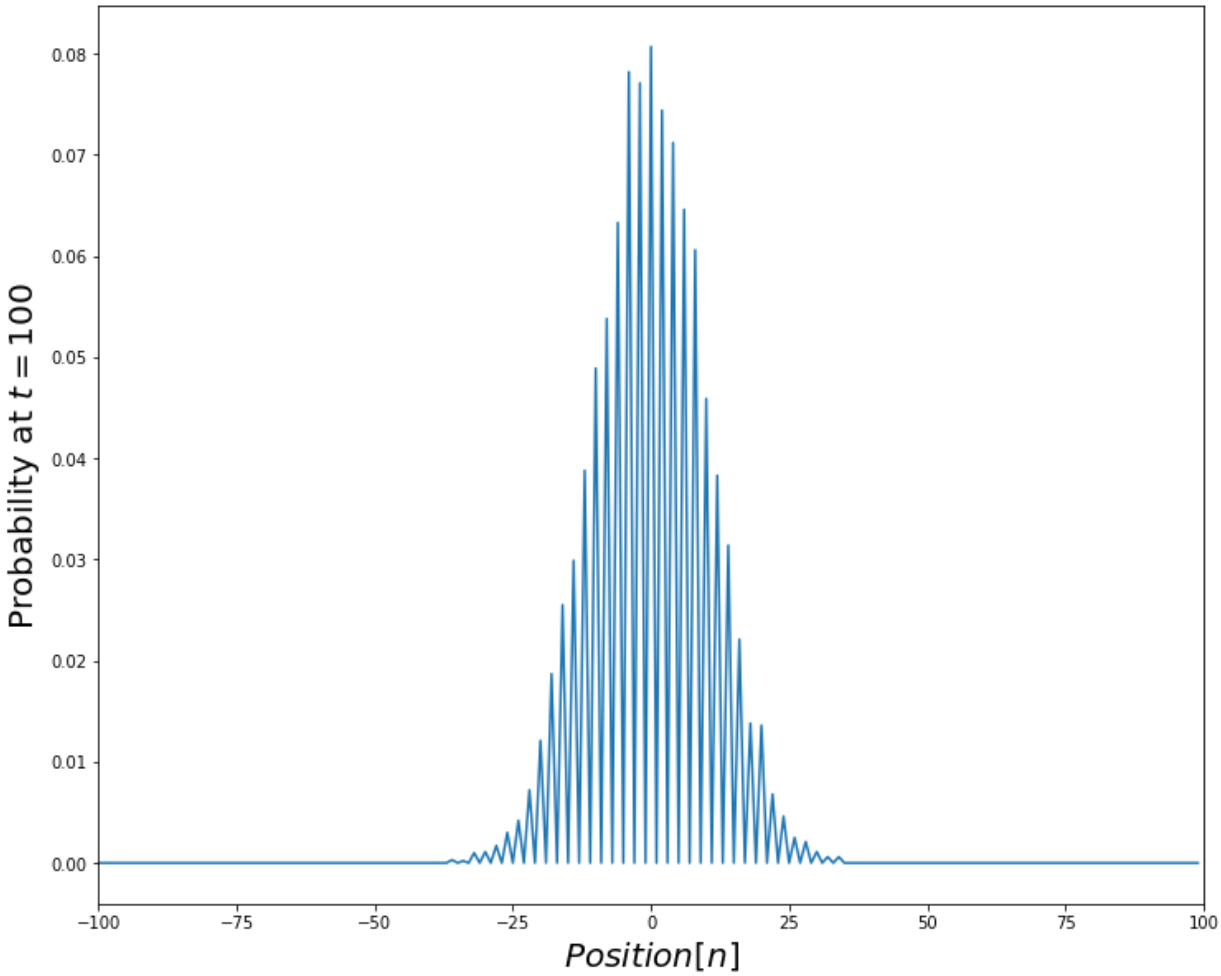
In [2]:
```
from numpy import *
from matplotlib.pyplot import *
import numpy as np
import random
from scipy import *
import scipy as sp
from scipy.linalg import expm, sinm, cosm
import scipy.integrate as integrate
```

Classical Random Walk

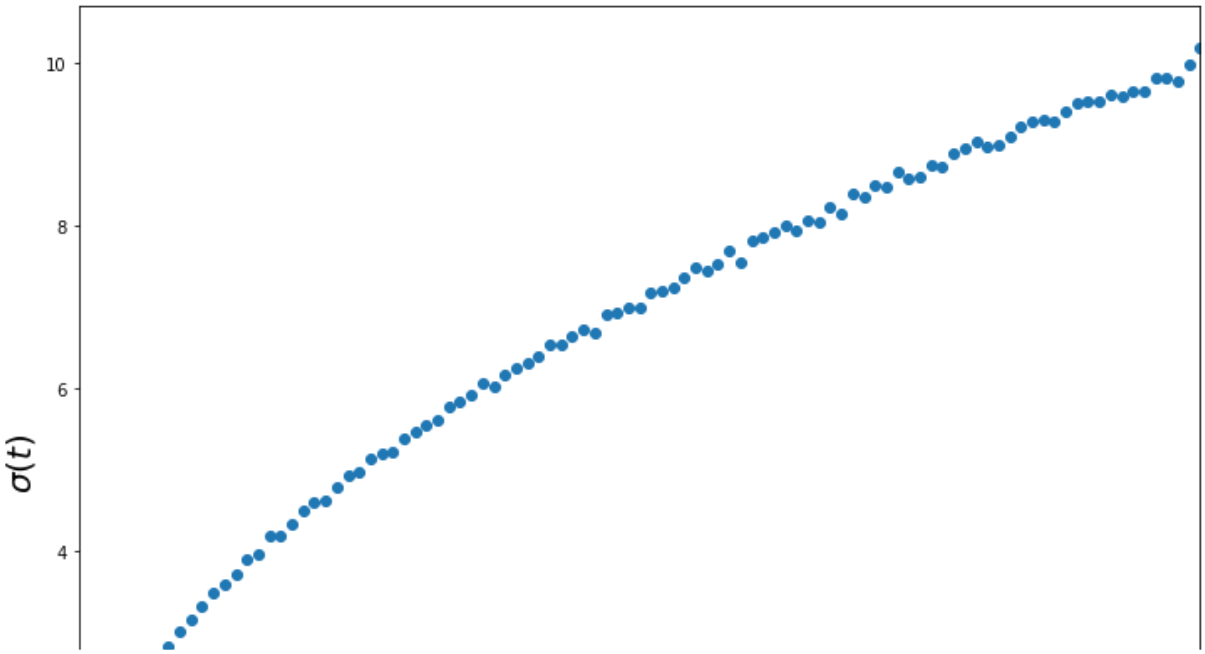
This code may be a bit slow. Wait for more than 1 mins please. I have not optimized it.

In [5]:
```
sigmaRW=[] #record the standart deviation of the Random Walk

particle_num=10000 #randomly propagate 10000 particles

for n in range(0,101):
    step = n
    particle=[0]*particle_num
    for i in range(step):
        for j in range(len(particle)):
            movement = [1,-1]
            particle[j]+=random.choice(movement) #randomly push the particle


    count = 0
    distribution = []
    for i in range(-100,100):
        for j in range(particle_num-1):
            if particle[j] == i:
                count+=1
        distribution.append(count)
        count = 0

    for i in range(len(distribution)):
        distribution[i]=distribution[i]/particle_num

    sigmaRW.append(std(particle))

x = []
y = distribution
for i in range(-100,100):
    x.append(i)
```

In [6]:
```
fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)
plot(x, y,'-')
#plot(x, y,'o')
xlim(-100, 100)
xlabel('$Position [n]$',size=20)
ylabel('Probability at $t = 100$',size=20)
show()
```



In [7]:
```
nstep =[]
for i in range(0,101):
    nstep.append(i)

fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)
#plot(nstep, sigma,'-')
plot(nstep, sigmaRW,'o')
xlim(0, 100)
xlabel('t',size=20)
ylabel('$\sigma(t)$',size=20)
#legend()
show()
```



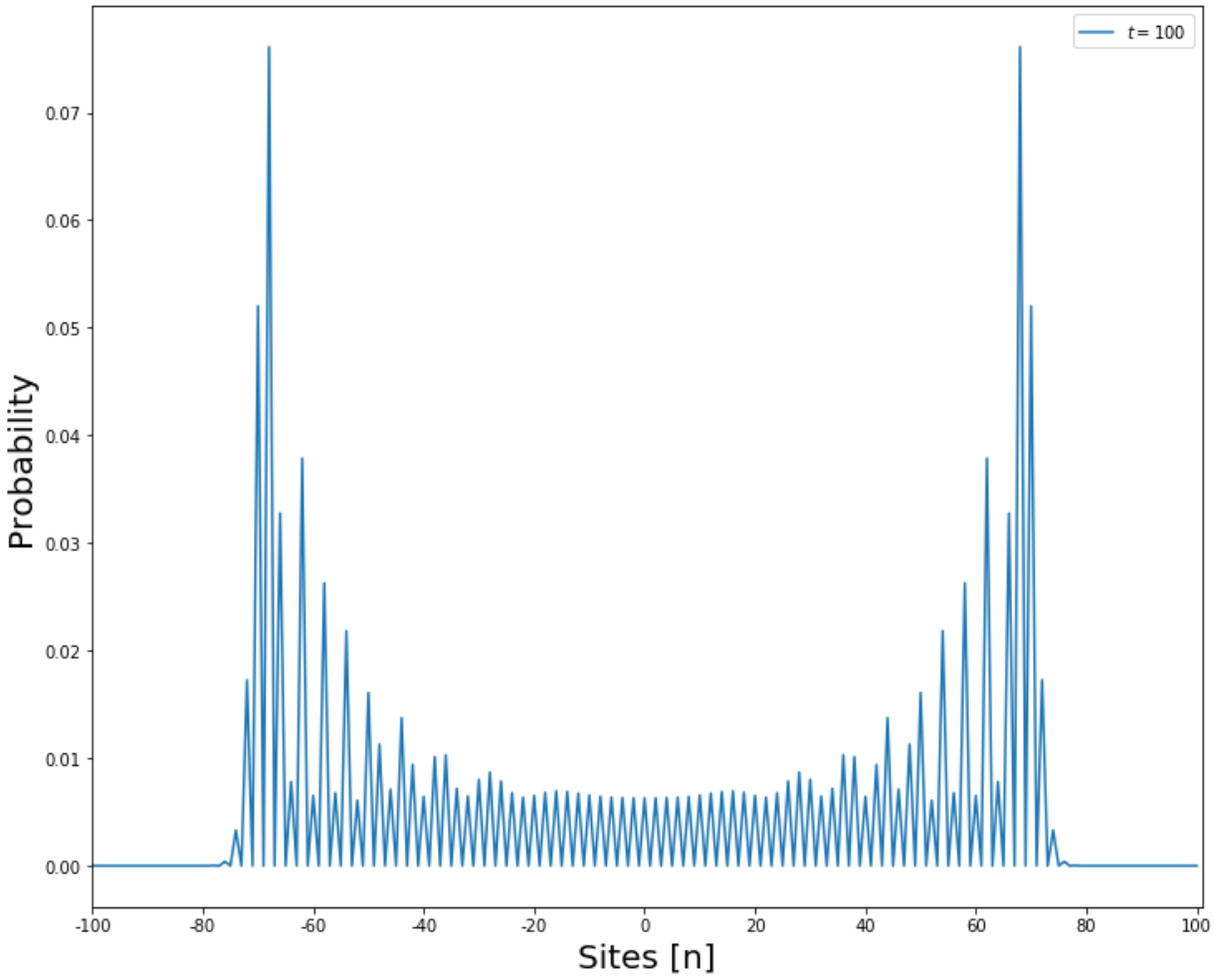
Quantum Random Walk

This code is modified based on the DTQW code on Susan Stepney's blog:https://susan-stepney.blogspot.com/2014/02/mathjax.html (https://susan-stepney.blogspot.com/2014/02/mathjax.html).

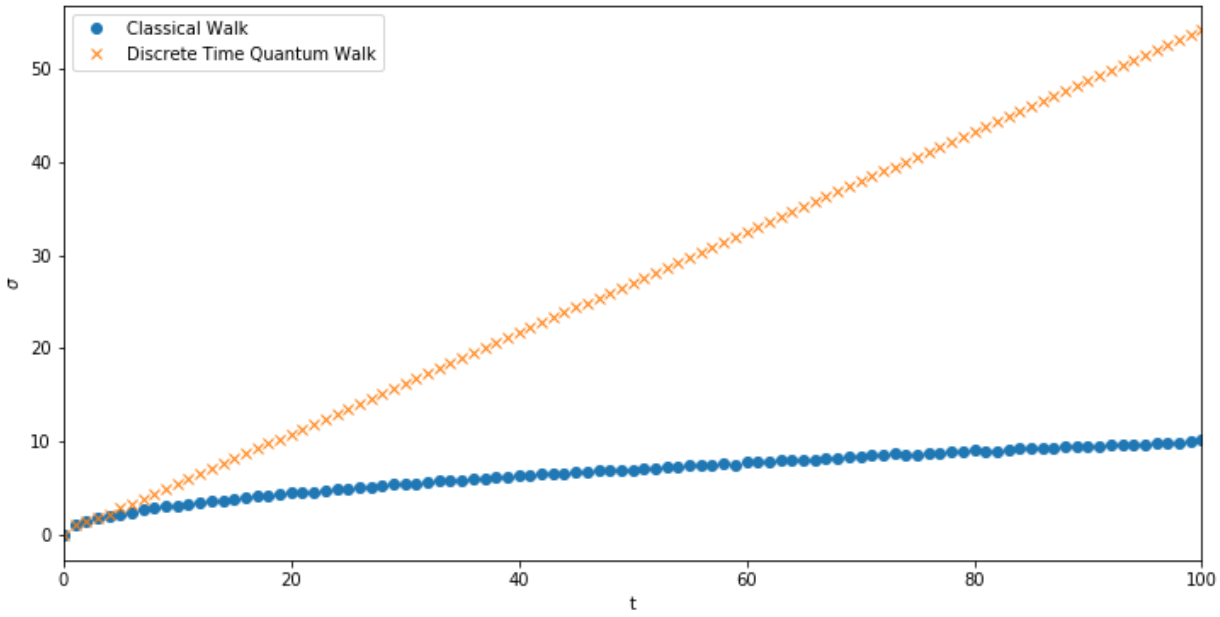
Symmetric DTQW

In [49]:
```
sigmaSQW=[] # will record the standard deviation of the symmetric DTQW latte

for n in range(0,101):

    N = n      # number of random steps
    P = 2*N+1    # number of positions

    spin0 = array([1, 0])  # |0>
    spin1 = array([0, 1])  # |1>

    C00 = outer(coin0, coin0)  # |0><0|
    C01 = outer(coin0, coin1)  # |0><1|
    C10 = outer(coin1, coin0)  # |1><0|
    C11 = outer(coin1, coin1)  # |1><1|


    C_hat = (C00 + C01 + C10 - C11)/sqrt(2.)

    ShiftPlus = roll(eye(P), 1, axis=0)#roll the matrix so that S(+)(1,0,0,(
    ShiftMinus = roll(eye(P), -1, axis=0)#roll the matrix so that S(-)(1,0,(
    S_hat = kron(ShiftPlus, C00) + kron(ShiftMinus, C11)#condition selectior

    U = S_hat.dot(kron(eye(P), C_hat))#Total unitary operator

    posn0 = zeros(P)
    posn0[N] = 1
    psi0 = kron(posn0,(coin0+coin1*1j)/sqrt(2.))

    psiN = linalg.matrix_power(U, N).dot(psi0)

    prob_sym_DTQW = empty(P)
    for k in range(P): #meaure the state after N step
        posn = zeros(P)
        posn[k] = 1
        M_hat_k = kron( outer(posn,posn), eye(2))
        proj = M_hat_k.dot(psiN)
        prob_sym_DTQW[k] = proj.dot(proj.conjugate()).real

    #Get the standard deviation of the Prob dist of the final state
    nlattice =[]
    for i in range(-n,n+1):
        nlattice.append(i)
    mean_nsquare = 0
    mean_n = 0
    for i in range(len(prob_sym_DTQW)):
        mean_nsquare += (nlattice[i]**2)*prob_sym_DTQW[i]
        mean_n += nlattice[i]*prob_sym_DTQW[i]

    sigmaSQW.append(sqrt(mean_nsquare-mean_n))
```

In [50]:
```
fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

plot(arange(P), prob_sym_DTQW, label='$t=100$')
loc = range (0, P, P//10) #Location of ticks
xticks(loc)
xlim(0, P)
ax.set_xticklabels(range (-N, N+1, int(P / 10)))
xlabel('Sites [n]', size=20)
ylabel('Probability', size=20)
legend()
show()
```



In [10]:
```
fig = figure(figsize=(12, 6))
ax = fig.add_subplot(111)
plot(nstep, sigmaRW,'o',label='Classical Walk')
plot(nstep, sigmaSQW,'x', label = 'Discrete Time Quantum Walk')
xlim(0, 100)
xlabel('t')
ylabel('$\sigma$')
legend()
show()
```

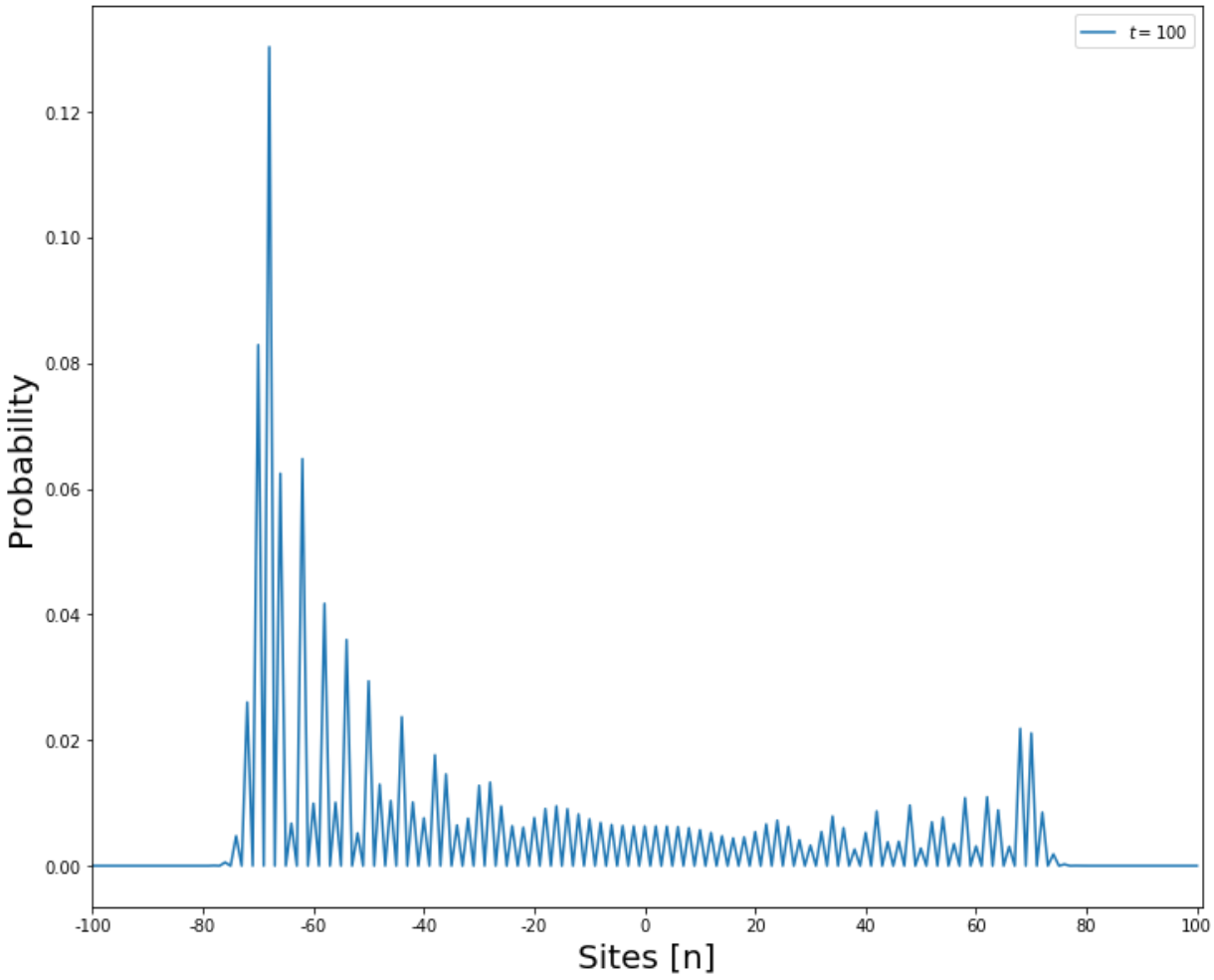


Asymmetric DTQW

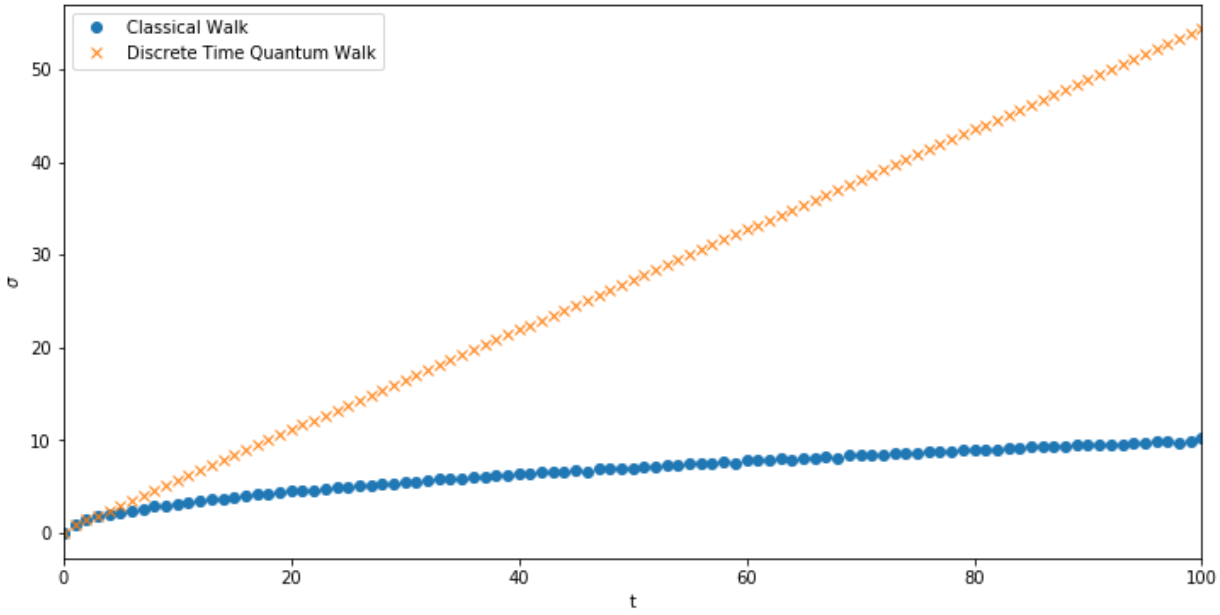
Left Skew

In [11]:
```
sigmaASQW=[]
for n in range(0,101):

    N = n      # number of random steps
    P = 2*N+1    # number of positions

    coin0 = array([1, 0])  # |0>
    coin1 = array([0, 1])  # |1>

    C00 = outer(coin0, coin0)  # |0><0|
    C01 = outer(coin0, coin1)  # |0><1|
    C10 = outer(coin1, coin0)  # |1><0|
    C11 = outer(coin1, coin1)  # |1><1|


    C_hat = (C00 + C01 + C10 - C11)/sqrt(2.)

    ShiftPlus = roll(eye(P), 1, axis=0)
    ShiftMinus = roll(eye(P), -1, axis=0)
    S_hat = kron(ShiftPlus, C00) + kron(ShiftMinus, C11)

    U = S_hat.dot(kron(eye(P), C_hat))

    posn0 = zeros(P)
    posn0[N] = 1
    psi0 = kron(posn0,coin1)

    psiN = linalg.matrix_power(U, N).dot(psi0)

    prob = empty(P)
    for k in range(P):
        posn = zeros(P)
        posn[k] = 1
        M_hat_k = kron( outer(posn,posn), eye(2))
        proj = M_hat_k.dot(psiN)
        prob[k] = proj.dot(proj.conjugate()).real

    nlattice =[]
    for i in range(-n,n+1):
        nlattice.append(i)
    mean_nsquare = 0
    mean_n = 0
    for i in range(len(prob)):
        mean_nsquare += (nlattice[i]**2)*prob[i]
        mean_n += nlattice[i]*prob[i]

    sigmaASQW.append(sqrt(mean_nsquare-mean_n))
```

In [12]:
```
fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

plot(arange(P), prob, label='$t=100$')
loc = range (0, P, P//10) #Location of ticks
xticks(loc)
xlim(0, P)
ax.set_xticklabels(range (-N, N+1, int(P / 10)))
xlabel('Sites [n]', size=20)
ylabel('Probability', size=20)
legend()
show()
```

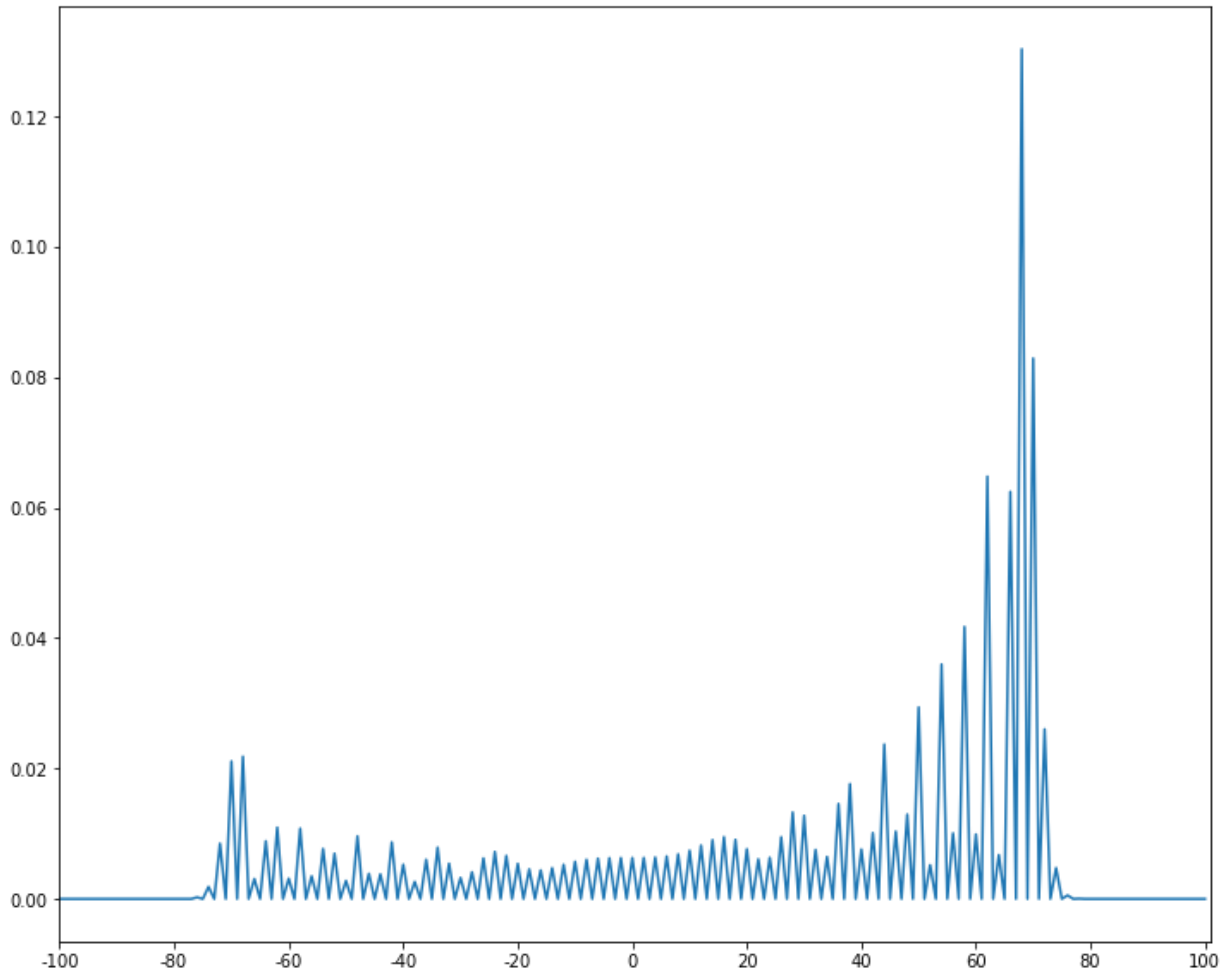


In [13]:
```
fig = figure(figsize=(12, 6))
ax = fig.add_subplot(111)
#plot(nstep, sigma,'-')
plot(nstep, sigmaRW,'o',label='Classical Walk')
plot(nstep, sigmaASQW,'x', label = 'Discrete Time Quantum Walk')
xlim(0, 100)
xlabel('t')
ylabel('$\sigma$')
legend()
show()
```



Right skew

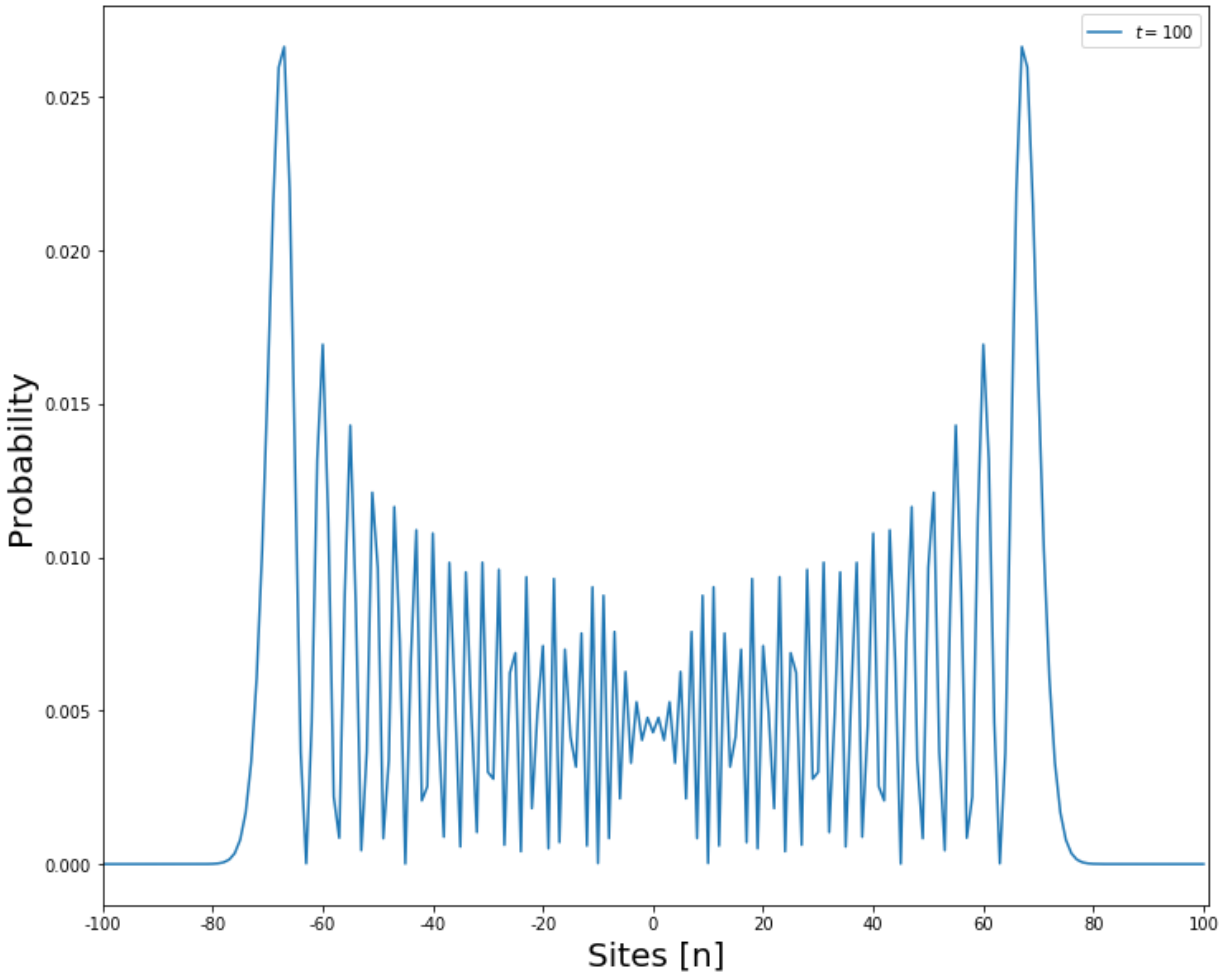
In [15]:
```
posn0 = zeros(P)
posn0[N] = 1     # array indexing starts from 0, so index N is the central p
psi0 = kron(posn0,coin0)

psiN = linalg.matrix_power(U, N).dot(psi0)

prob = empty(P)
for k in range(P):
    posn = zeros(P)
    posn[k] = 1
    M_hat_k = kron( outer(posn,posn), eye(2))
    proj = M_hat_k.dot(psiN)
    prob[k] = proj.dot(proj.conjugate()).real

fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

plot(arange(P), prob)
loc = range (0, P, int(P/10)) #Location of ticks
xticks(loc)
xlim(0, P)
ax.set_xticklabels(range (-N, N+1, int(P / 10)))

show()
```



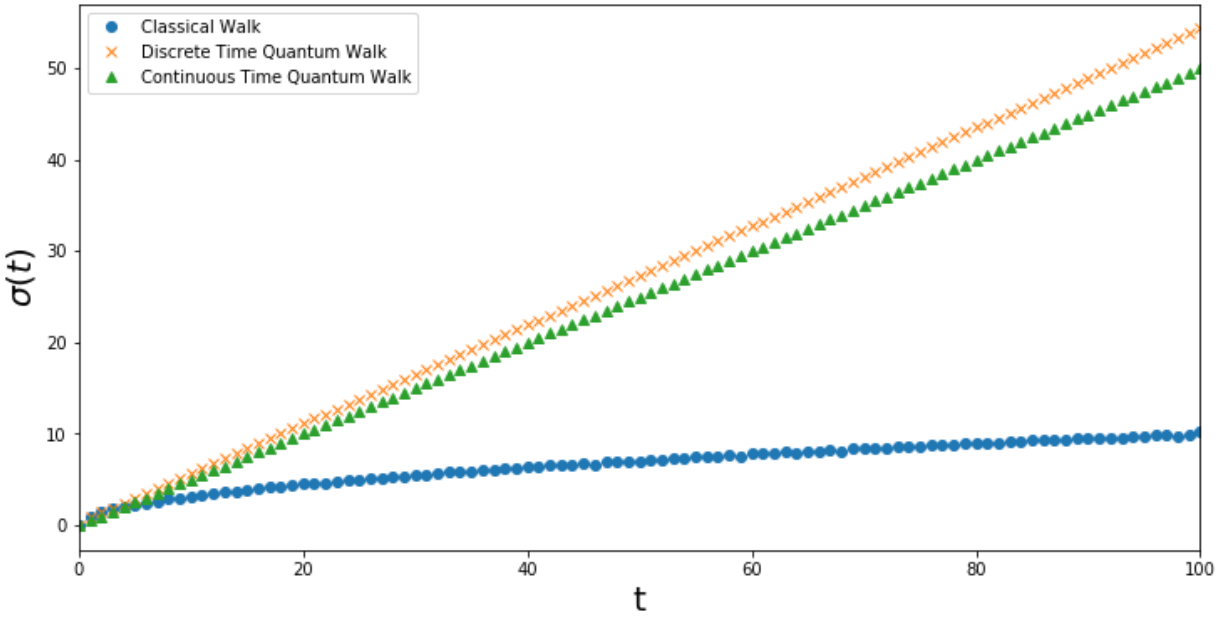
Continuous Time Quantum Walk

In [21]:
```
sigmaCTQW=[]
for n in range(0,101):
    t=n
    N = n
    P = 2*N+1
    gamma = 1/(2*np.sqrt(2))

    H = []

    for i in range(P):
        for j in range(P):
            if j==i:
                H.append(2*gamma)
            elif j==i+1 or j==i-1:
                H.append(-gamma)
            else:
                H.append(0)

    H=np.asarray(H)
    H=H.reshape(P,P)
    U = expm(-1.j*H*t)

    posn0 = zeros(P)
    posn0[N] = 1
    psi0=posn0

    psiN = U.dot(psi0)

    prob = empty(P)
    for k in range(P):
        posn = zeros(P)
        posn[k] = 1
        M_hat_k = outer(posn,posn)
        proj = M_hat_k.dot(psiN)
        prob[k] = proj.dot(proj.conjugate()).real

    nlattice =[]

    for i in range(-n,n+1):
        nlattice.append(i)
    mean_nsquare = 0
    mean_n = 0

    for i in range(len(prob)):
        mean_nsquare += (nlattice[i]**2)*prob[i]
        mean_n += nlattice[i]*prob[i]

    sigmaCTQW.append(sqrt(mean_nsquare-mean_n))
```

In [22]:
```
fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

plot(arange(P), prob, label='$t=100$')
#plot(arange(P), prob, 'o')
loc = range (0, P, P//10) #Location of ticks
xticks(loc)
xlim(0, P)
ax.set_xticklabels(range (-N, N+1, int(P / 10)))
xlabel('Sites [n]', size=20)
ylabel('Probability', size=20)
legend()
show()
```

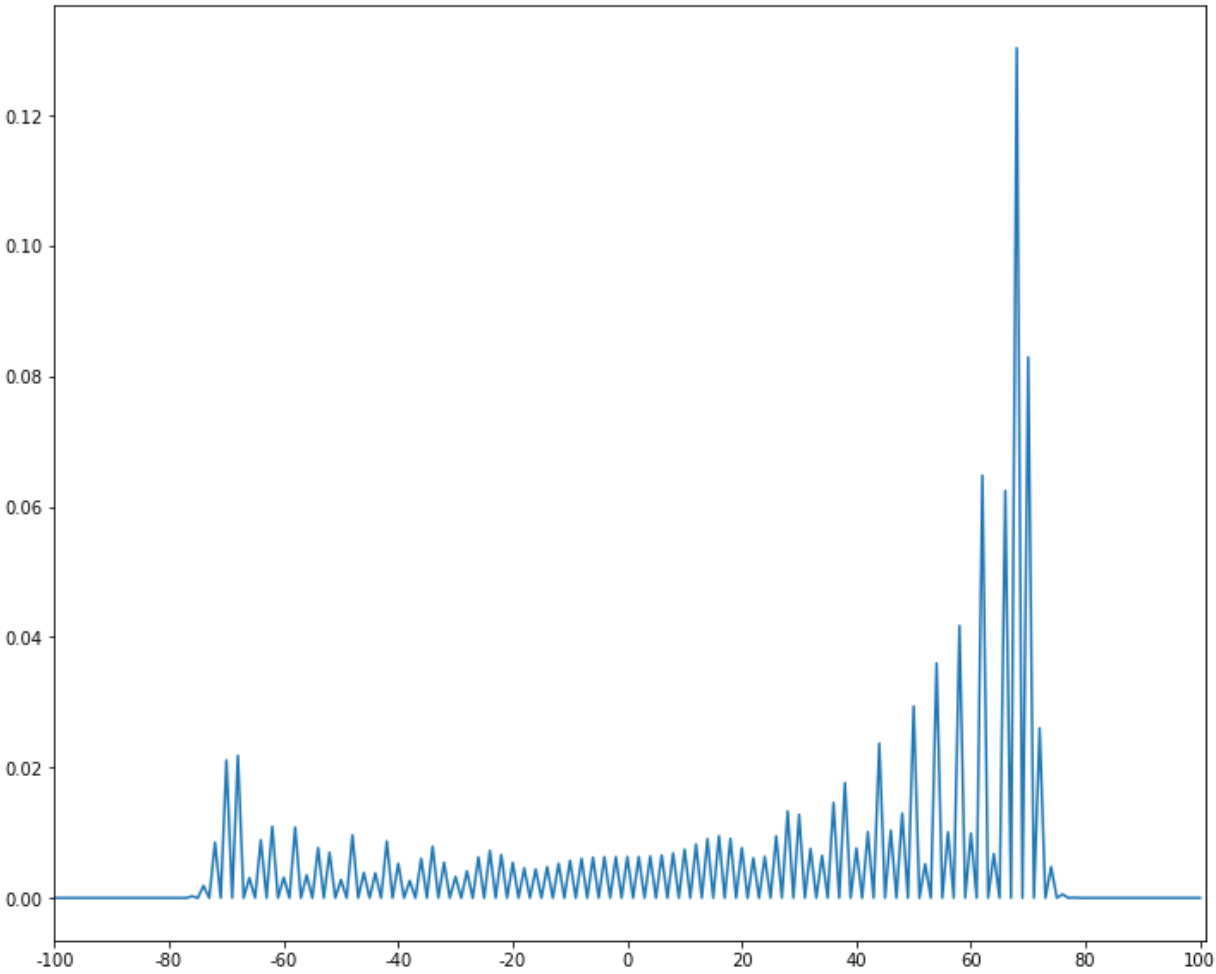


In [23]:
```
fig = figure(figsize=(12, 6))
ax = fig.add_subplot(111)
#plot(nstep, sigma,'-')
plot(nstep, sigmaRW,'o',label='Classical Walk')
plot(nstep, sigmaASQW,'x', label = 'Discrete Time Quantum Walk')
plot(nstep, sigmaCTQW,'^', label = 'Continuous Time Quantum Walk')
xlim(0, 100)
xlabel('t',size=20)
ylabel('$\sigma(t)$',size =20)
legend()
show()
```



Comparison between Analytical Solution and DTQW

In [31]:
```
for n in range(100,101):

    N = n       # number of random steps
    P = 2*N+1     # number of positions

    coin0 = array([1, 0])  # |0>
    coin1 = array([0, 1])  # |1>

    C00 = outer(coin0, coin0)  # |0><0|
    C01 = outer(coin0, coin1)  # |0><1|
    C10 = outer(coin1, coin0)  # |1><0|
    C11 = outer(coin1, coin1)  # |1><1|


    C_hat = (C00 + C01 + C10 - C11)/sqrt(2.)

    ShiftPlus = roll(eye(P), 1, axis=0)
    ShiftMinus = roll(eye(P), -1, axis=0)
    S_hat = kron(ShiftPlus, C00) + kron(ShiftMinus, C11)

    U = S_hat.dot(kron(eye(P), C_hat))

    posn0 = zeros(P)
    posn0[N] = 1
    psi0 = kron(posn0,coin0)

    psiN = linalg.matrix_power(U, N).dot(psi0)

    prob = empty(P)
    for k in range(P):
        posn = zeros(P)
        posn[k] = 1
        M_hat_k = kron( outer(posn,posn), eye(2))
        proj = M_hat_k.dot(psiN)
        prob[k] = proj.dot(proj.conjugate()).real


fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

plot(arange(P), prob)
loc = range (0, P, P//10)
xticks(loc)
xlim(0, P)
ax.set_xticklabels(range (-N, N+1, int(P / 10)))

show()
```



In [32]:
```
def psi0_integrand(k, t, x):
    wk = arcsin(sin(k)/sqrt(2))
    return (1+cos(k)/sqrt(1+cos(k)**2))*exp(-1j*(wk*t-k*x))/(2*pi)

def psi1_integrand(k, t, x):
    wk = arcsin(sin(k)/sqrt(2))
    return (exp(1j*k)/sqrt(1+cos(k)**2))*exp(-1j*(wk*t-k*x))/(2*pi)
```

In [33]:
```
def complex_quadrature(func, a, b, **kwargs):
    def real_func(k,t,x):
        return sp.real(func(k,t,x))
    def imag_func(k,t,x):
        return sp.imag(func(k,t,x))
    real_integral = integrate.quad(real_func, a, b, limit=200, **kwargs)
    imag_integral = integrate.quad(imag_func, a, b, limit=200, **kwargs)
    return real_integral[0] + 1j*imag_integral[0]# real_integral[1:], imag_i
```

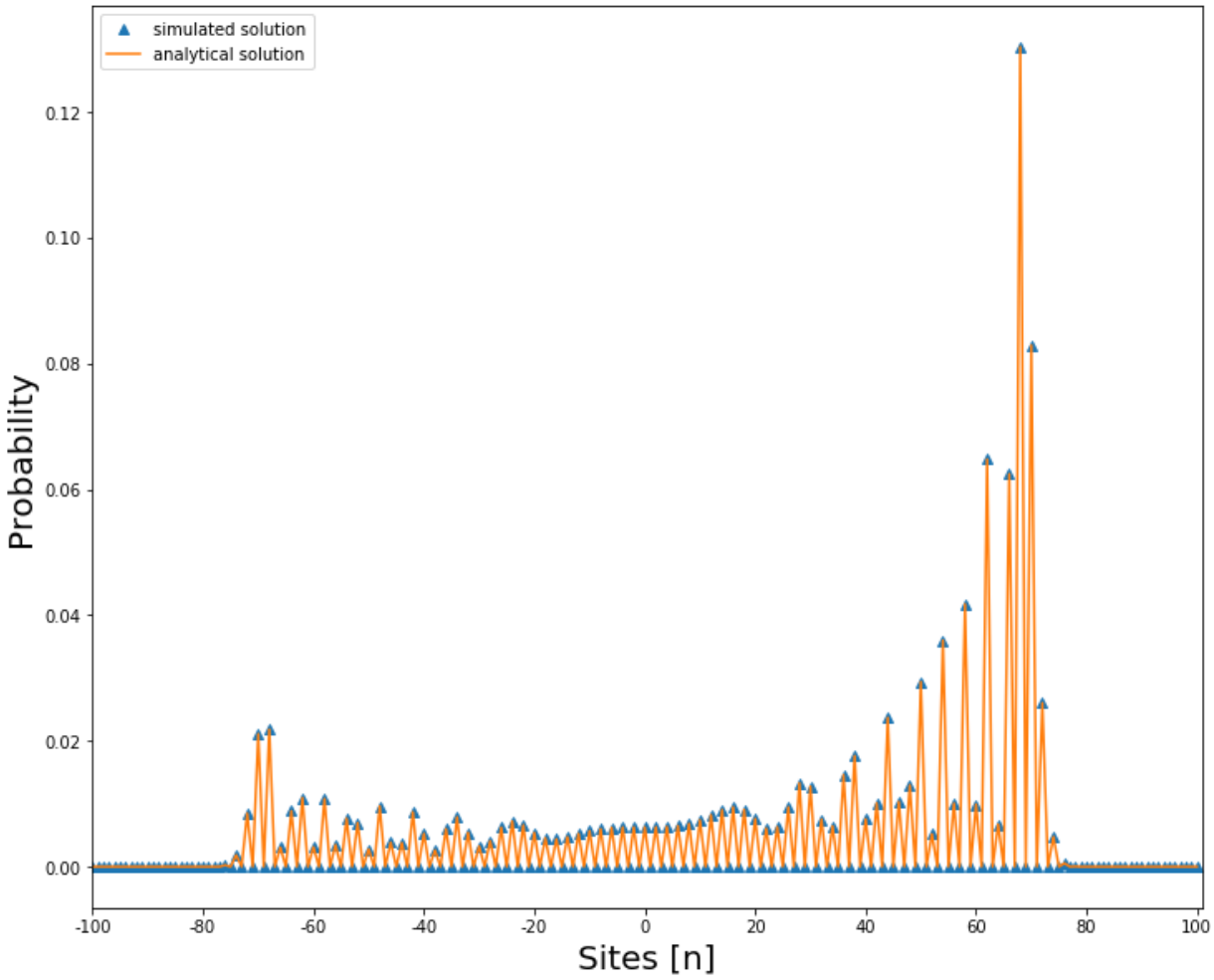
In [34]:
```
nlattice =[]
for i in range(-100,100+1):
    nlattice.append(i)
```

In [35]:
```
psi0=[]
psi1=[]
even = -100
for i in nlattice:
    t=100
    x=i
    if x%2 ==0:
        psi0.append(complex_quadrature(psi0_integrand, -pi,pi, args=(t, x)))
        psi1.append(complex_quadrature(psi1_integrand, -pi,pi, args=(t, x)))
    else:
        psi0.append(0)
        psi1.append(0)
```

In [36]:
```
psi_Analystical=[]
for i in range(len(psi0)):
    psi_Analystical.append(psi0[i].conjugate() *psi0[i] +psi1[i].conjugate()
```

In [37]:
```
fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

N=100
plot(arange(P), prob,'^',label='simulated solution',)
plot(arange(P), psi_Analystical,label='analytical solution')
loc = range (0, P, P//10) #Location of ticks
xticks(loc)
xlim(0, P)
ax.set_xticklabels(range (-N, N+1, P // 10))
xlabel('Sites [n]', size=20)
ylabel('Probability', size=20)
legend()
show()
```

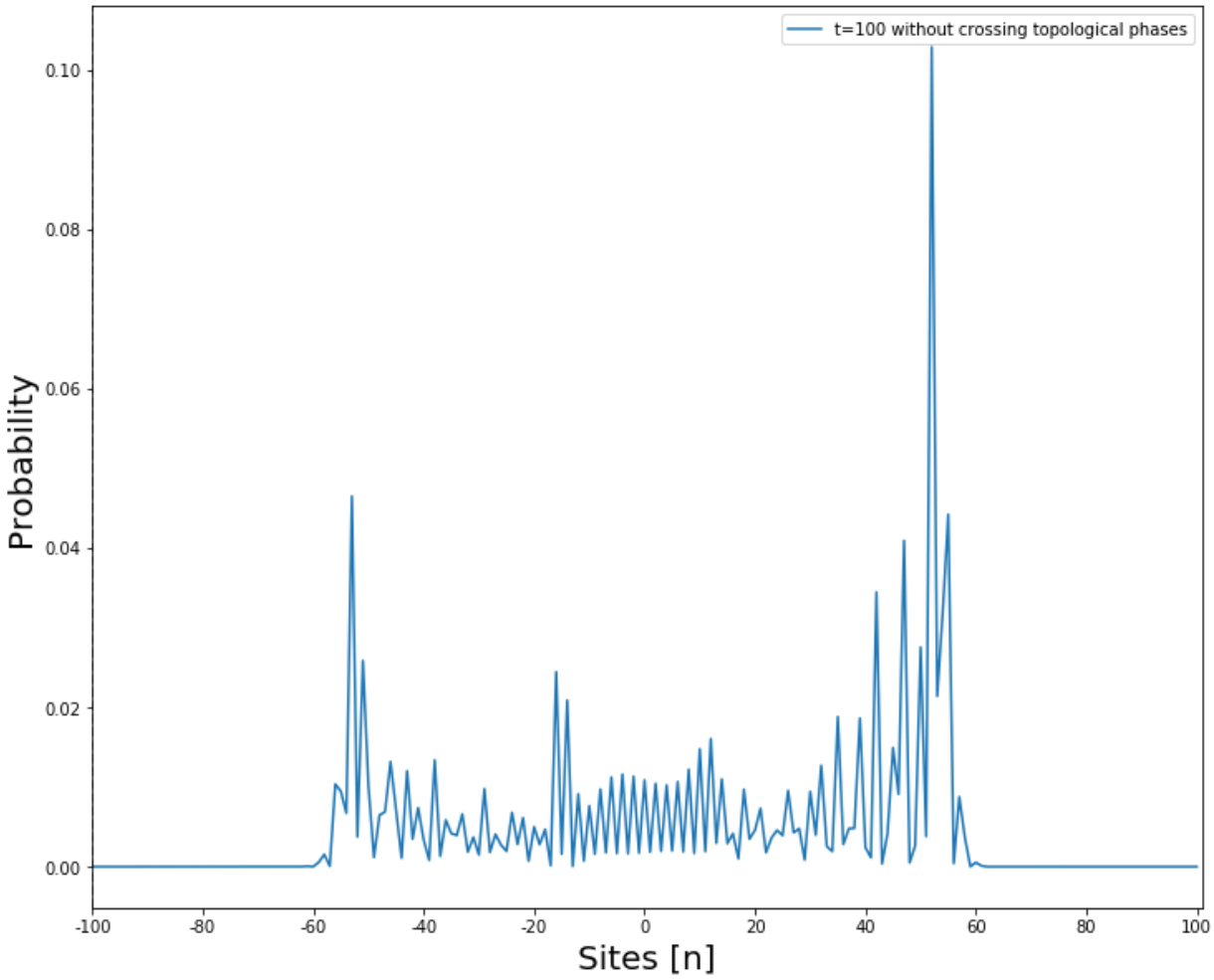
```
/Users/chenwu/anaconda3/lib/python3.6/site-packages/numpy/core/numeric.p
y:492: ComplexWarning: Casting complex values to real discards the imagin
ary part
  return array(a, dtype, copy=False, order=order)
```



Quantum Walk on 1D Topological Phase Transition

In [132]:
```
N = 100
P = 2*N+1

coin0 = array([1, 0])
coin1 = array([0, 1])

C00 = outer(coin0, coin0)
C01 = outer(coin0, coin1)
C10 = outer(coin1, coin0)
C11 = outer(coin1, coin1)

def Ry(theta):
    y_SU2 = cos(theta/2)*C00+sin(theta/2)*C10-sin(theta/2)*C01+cos(theta/2)*
    #array([[cos(theta/2), -sin(theta/2)],[sin(theta/2), cos(theta/2)]])
    return kron(eye(P),y_SU2)

ShiftPlus = roll(eye(P), 1, axis=0)
ShiftMinus = roll(eye(P), -1, axis=0)
T_up = kron(ShiftPlus, C00) + kron(eye(P), C11)
T_down = kron(ShiftMinus, C11) + kron(eye(P), C00)

def theta2(x,theta2Plus,theta2Minus):
    return (1/2)*(theta2Plus+theta2Minus)+(1/2)*(theta2Plus-theta2Minus)*tar

def U(x,theta2Plus,theta2Minus, theta1):
    return T_down.dot(Ry(theta2(x,theta2Plus,theta2Minus)).dot(T_up.dot(Ry(t

posn0 = zeros(P)
posn0[N] = 1
psi0 = kron(posn0, coin0)
psiN = psi0

x = -N
for i in range(N):
    x+=2
    psiN = U(x,1*pi/4,  3*pi/4, -pi/2).dot(psiN)#0.99*pi/2,  0

prob = empty(P)
for k in range(P):
    posn = zeros(P)
    posn[k] = 1
    M_hat_k = kron( outer(posn,posn), eye(2))
    proj = M_hat_k.dot(psiN)
    prob[k] = proj.dot(proj.conjugate()).real
```

In [133]:
```
fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

plot(arange(P), prob, label='t=100 without crossing topological phases')
loc = range (0, P, P//10)
xticks(loc)
xlim(0, P)
axvline(x = 0,color='black',ls='dashed')
ax.set_xticklabels(range (-N, N+1, int(P / 10)))
xlabel('Sites [n]', size=20)
ylabel('Probability', size=20)
legend()
show()
```

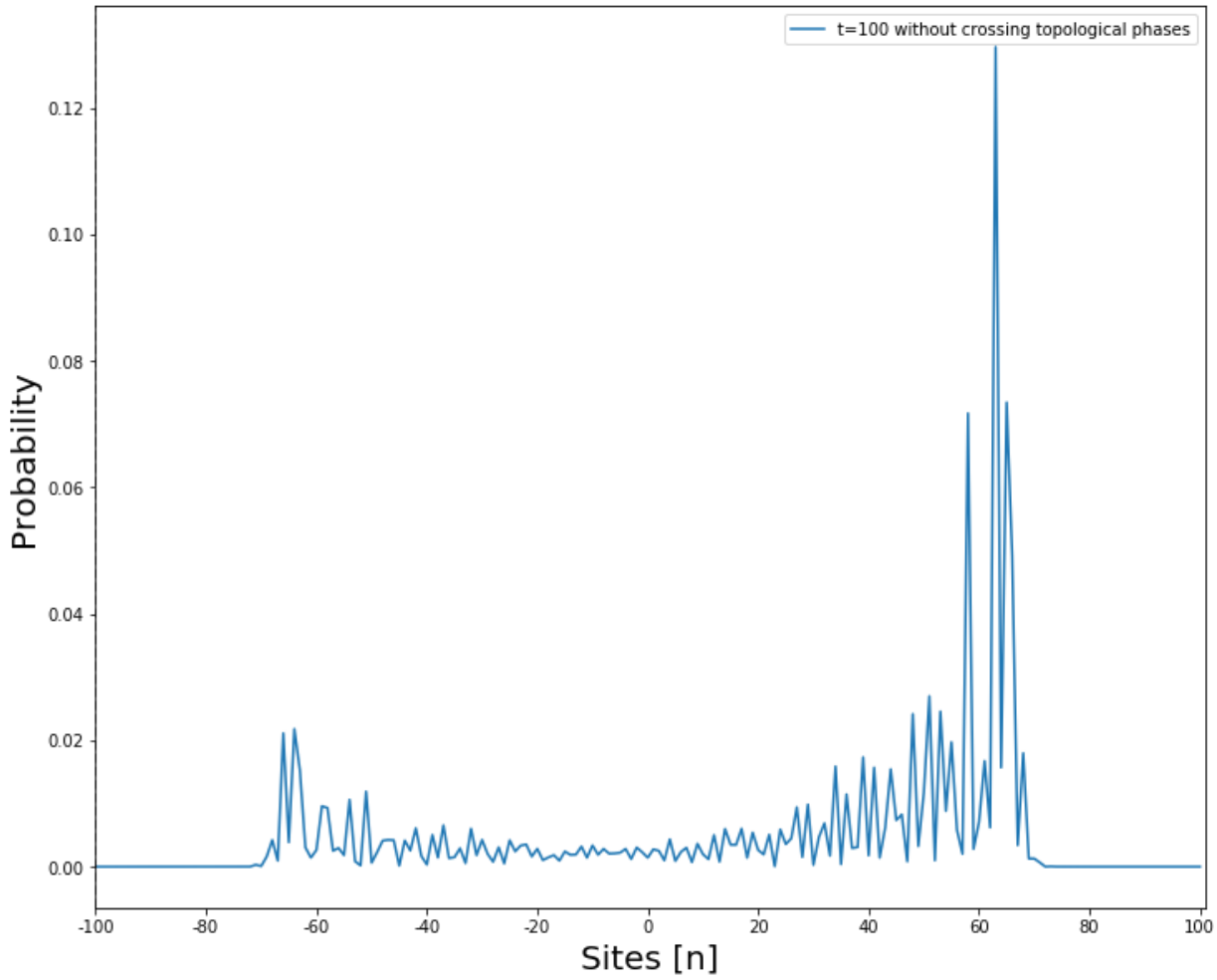


In [117]:
```
prob[100]
```

Out[117]:
```
2.8518530201539177e-13
```

In [136]:
```
N = 100
P = 2*N+1

coin0 = array([1, 0])
coin1 = array([0, 1])

C00 = outer(coin0, coin0)
C01 = outer(coin0, coin1)
C10 = outer(coin1, coin0)
C11 = outer(coin1, coin1)

def Ry(theta):
    y_SU2 = cos(theta/2)*C00+sin(theta/2)*C10-sin(theta/2)*C01+cos(theta/2)*
    #array([[cos(theta/2), -sin(theta/2)],[sin(theta/2), cos(theta/2)]])
    return kron(eye(P),y_SU2)

ShiftPlus = roll(eye(P), 1, axis=0)
ShiftMinus = roll(eye(P), -1, axis=0)
T_up = kron(ShiftPlus, C00) + kron(eye(P), C11)
T_down = kron(ShiftMinus, C11) + kron(eye(P), C00)

def theta2(x,theta2Plus,theta2Minus):
    return (1/2)*(theta2Plus+theta2Minus)+(1/2)*(theta2Plus-theta2Minus)*tar

def U(x,theta2Plus,theta2Minus, theta1):
    return T_down.dot(Ry(theta2(x,theta2Plus,theta2Minus)).dot(T_up.dot(Ry(t

posn0 = zeros(P)
posn0[N] = 1
psi0 = kron(posn0, coin0)
psiN = psi0

x = -N
for i in range(N):
    x+=2
    psiN = U(x,0,.99*pi/2, -pi/2).dot(psiN)

prob = empty(P)
for k in range(P):
    posn = zeros(P)
    posn[k] = 1
    M_hat_k = kron( outer(posn,posn), eye(2))
    proj = M_hat_k.dot(psiN)
    prob[k] = proj.dot(proj.conjugate()).real
```

In [137]:
```
fig = figure(figsize=(12, 10))
ax = fig.add_subplot(111)

plot(arange(P), prob, label='t=100 without crossing topological phases')
loc = range (0, P, P//10)
xticks(loc)
xlim(0, P)
axvline(x = 0,color='black',ls='dashed')
ax.set_xticklabels(range (-N, N+1, int(P / 10)))
xlabel('Sites [n]', size=20)
ylabel('Probability', size=20)
legend()
show()
```



In [138]:
```
prob[100]
```

Out[138]:
```
0.0014239696869836505
```

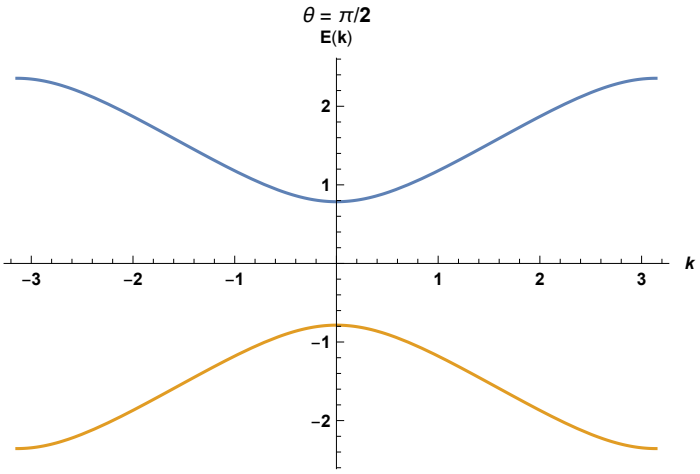
In [ ]:
```

```

In[150]:=
```
Clear["Global`*"]
```

In[151]:=
```
Ep[k_, θ_] = ArcCos[Cos[θ/2] Cos[k]];
Em[k_, θ_] = -ArcCos[Cos[θ/2] Cos[k]];
```

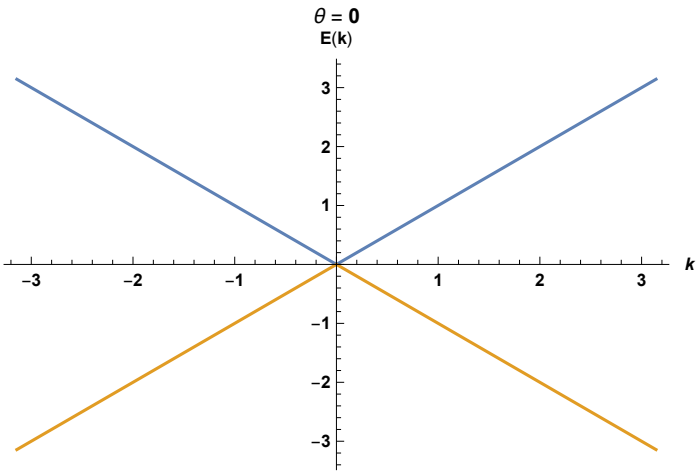
In[153]:=
```
Plot[{Ep[k, Pi/2], Em[k, Pi/2]}, {k, -Pi, Pi},
 AxesLabel → {k, "E(k)"}, PlotLabel → "θ = π/2"]
```

Out[153]=



In[154]:=
```
Plot[{Ep[k, 0], Em[k, 0]}, {k, -Pi, Pi}, AxesLabel → {k, "E(k)"}, PlotLabel → "θ = 0"]
```

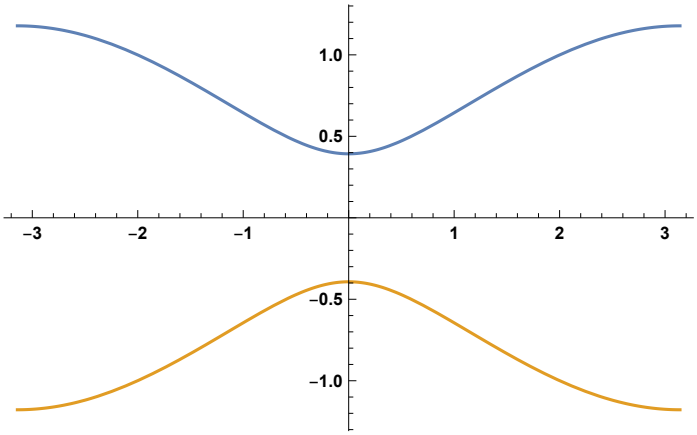
Out[154]=



In[155]:=
```
EE1[k_, θ1_, θ2_] = ArcCos[Cos[θ2/2] Cos[θ1/2] Cos[k] - Sin[θ1/2] Sin[θ2/2]];
EE2[k_, θ1_, θ2_] = -ArcCos[Cos[θ2/2] Cos[θ1/2] Cos[k] - Sin[θ1/2] Sin[θ2/2]];
```

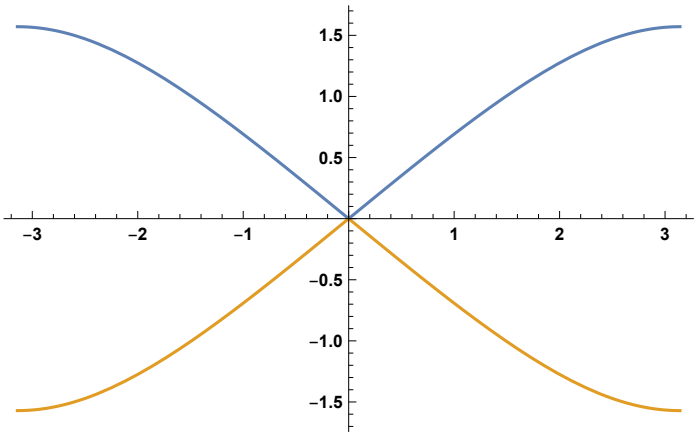
In[157]:=
```
Plot[{EE1[k, -Pi/2, 3 Pi/4], EE2[k, -Pi/2, 3 Pi/4]}, {k, -Pi, Pi}]
```

Out[157]=



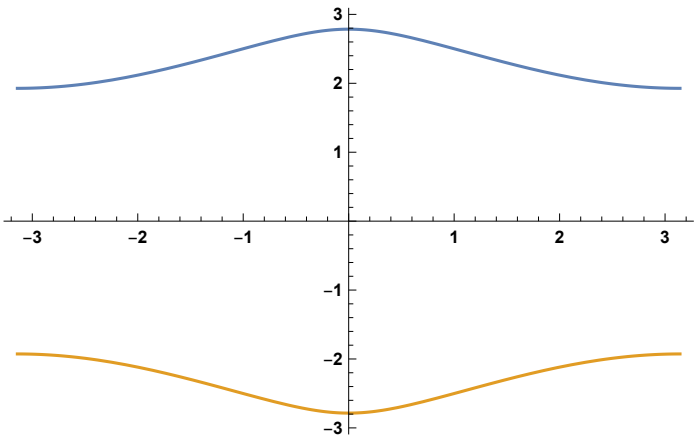
In[158]:=
```
Plot[{EE1[k, -Pi/2, 2 Pi/4], EE2[k, -Pi/2, 2 Pi/4]}, {k, -Pi, Pi}]
```

Out[158]=



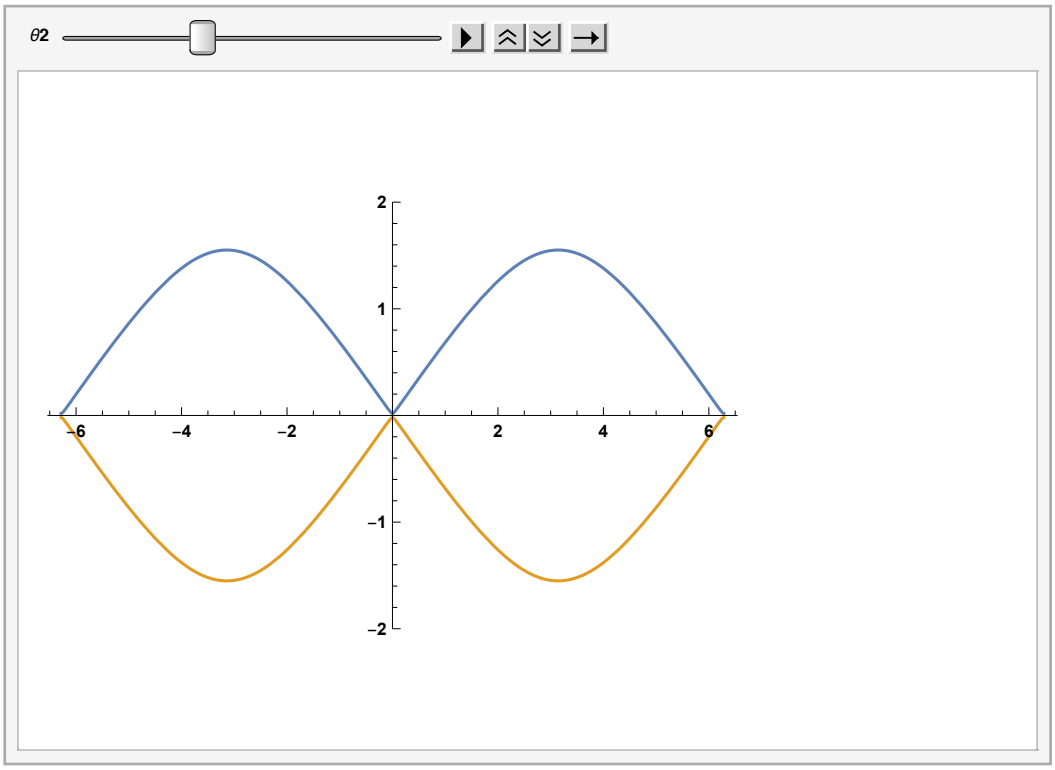
In[159]:=
```
Plot[{EE1[k, -Pi/2, -4], EE2[k, -Pi/2, -4]}, {k, -Pi, Pi}]
```

Out[159]=



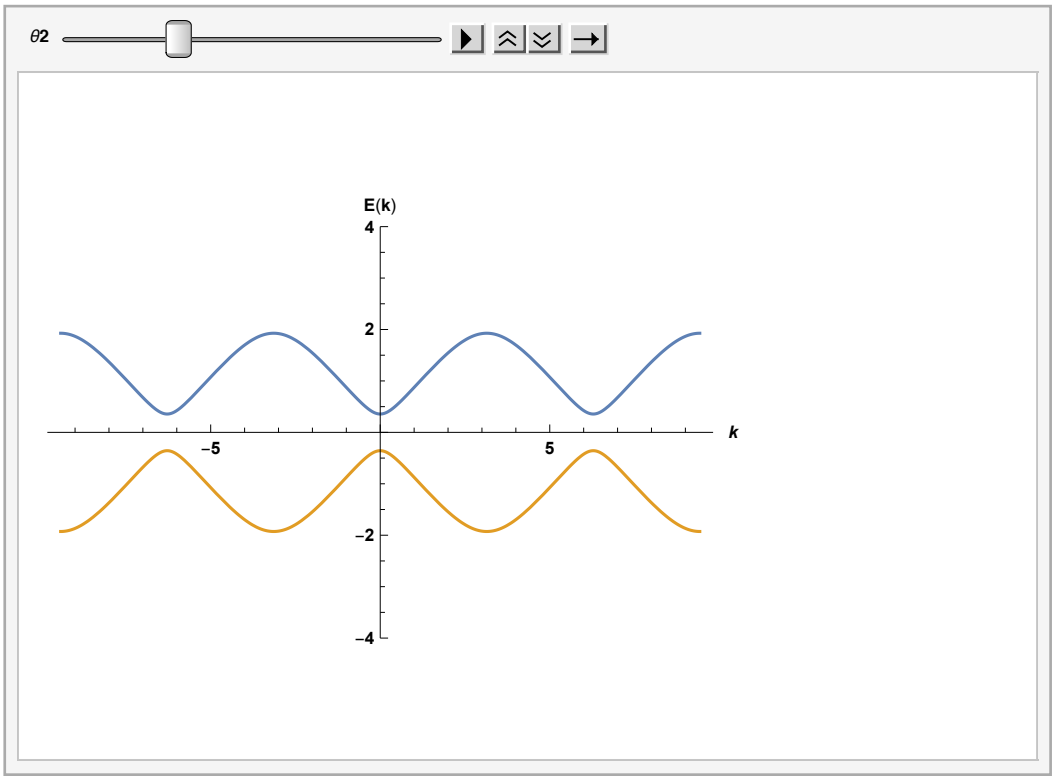
In[54]:=
```
Animate[Plot[{EE1[k, -Pi/2, θ2], EE2[k, -Pi/2, θ2]},
  {k, -2 Pi, 2 Pi}, PlotRange → {-2, 2}], {θ2, Pi/4, 4 Pi/4}]
```

Out[54]=



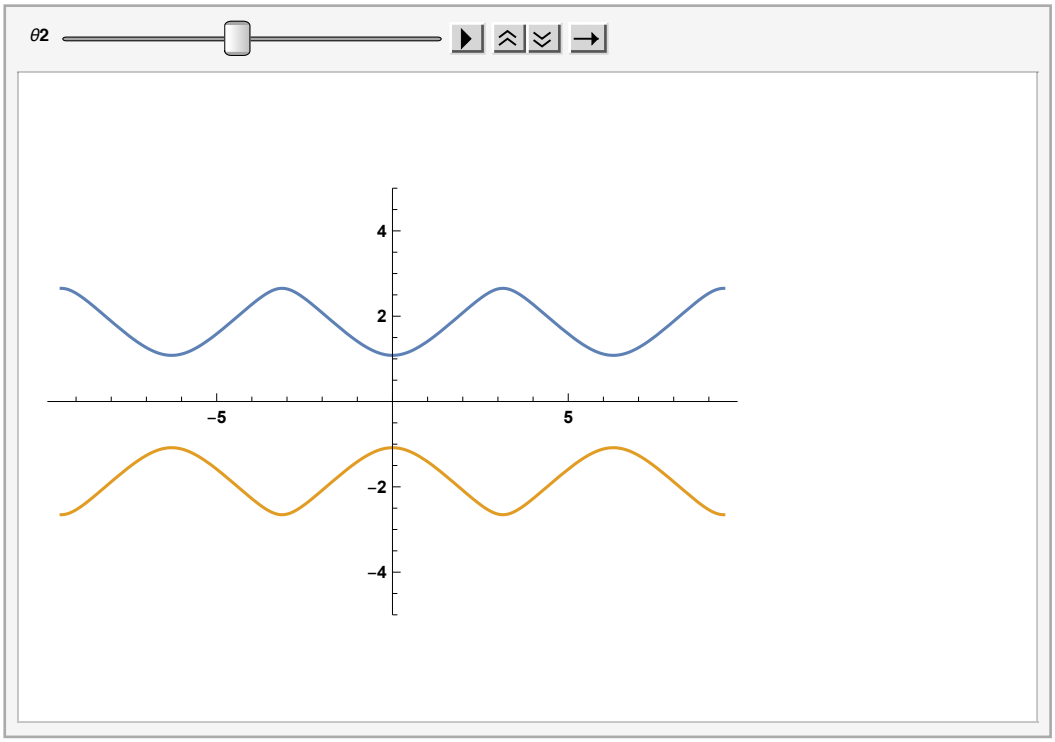
In[53]:=
```
Animate[Plot[{EE1[k, -Pi/2, θ2], EE2[k, -Pi/2, θ2]},
  {k, -3 Pi, 3 Pi}, PlotRange → {-4, 4}, AxesLabel → {k, "E(k)"}], {θ2, 0, Pi}]
```

Out[53]=



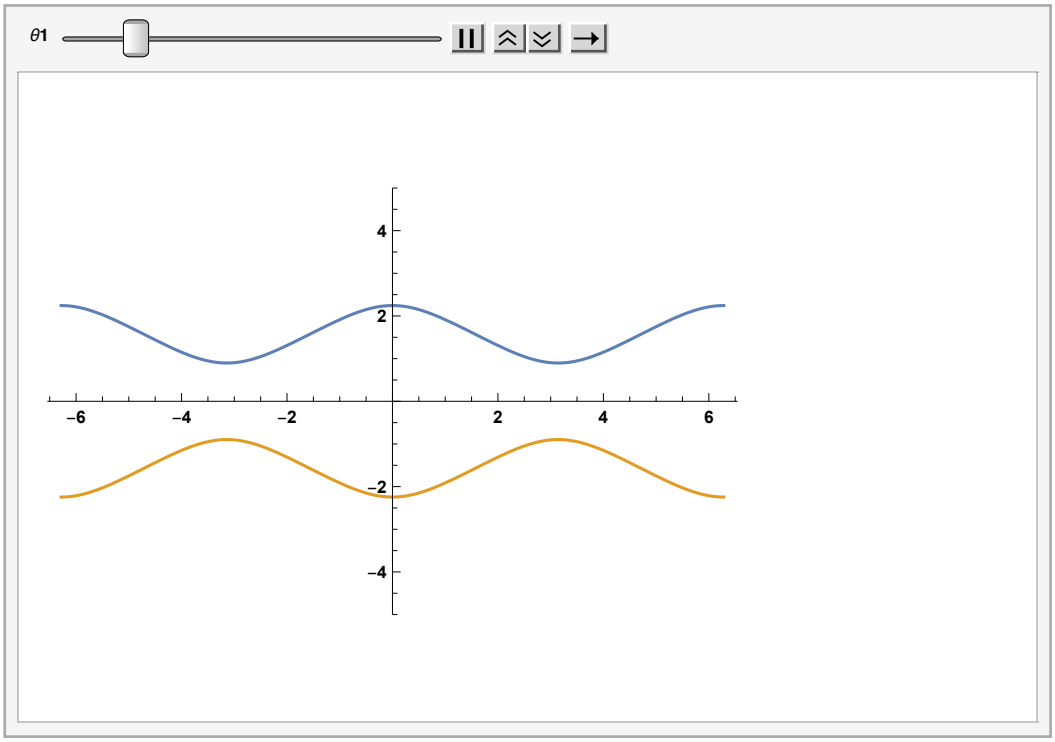
In[49]:=
```
Animate[Plot[{EE1[k, -π/2, θ2], EE2[k, -π/2, θ2]}, {k, -3 π, 3 π}, PlotRange → {-5, 5}],
 {θ2, -2 π, 2 π}, AnimationRunning → False, DisplayAllSteps → True]
```

Out[49]=



In[45]:=
```
Animate[Plot[{Ep[k, θ1], Em[k, θ1]}, {k, -2 Pi, 2 Pi}, PlotRange → {-5, 5}],
 {θ1, -2 Pi, 2 Pi}]
```

Out[45]=



1-D Topological simulation using split-step DTQW

The the code is provided by the author of Topological phenomena in quantum walks: elementary introduction to the physics of topological phases: Takuya Kitagawa

In[160]:=
```
distribution[plotmax_, initialangle_, theta1m_, theta1p_, theta2_] :=
  Module[{n = plotmax, iangle = initialangle, thetam = theta1m, thetap = theta1p,
    theta2angle = theta2, t, i, boundarylength, Initialup, Initialdown,
    theta1, a, temp, d, rotation1, rotation2}, boundarylength = 0.01;
   Initialup = N[Cos[iangle]]; (* Not Needed Normalised*)
   Initialdown = N[ Sin[iangle]]; theta1 = Table[(thetam + thetap) / 2 +
      (thetap - thetam) / 2 * Tanh[(i - 2 n + 1 / 2) / boundarylength], {i, 4 n + 1}];
   (* at "l" step and "i" position, coin up "j" + down "k" *)
   a = Table[0, {l, n}, {i, 4 n + 1}, {k, 2}];
   temp = Table[0, {i, 4 n + 1}, {k, 2}]; d = Table[0, {i, 4 n + 1}, {k, 2}];
   rotation1 = N[Table[MatrixExp[-I PauliMatrix[2] theta1[[i]] / 2], {i, 4 n + 1}]];
   rotation2 = N[MatrixExp[-I PauliMatrix[2] theta2angle / 2]];
   (* Normalised Initial Coin Condition at 2n, zero step is t=1,
   edge is n+1*) a[[1, 2 n, 1]] = Initialup; a[[1, 2 n, 2]] = Initialdown;
 (* Time Evolution Step *) For[t = 1, t <= n - 1, t++,
    For[i = 1 + 2 n - 2 t, i <= 2 n + 2 t + 1, i++, (* Coin Flip *)
     d[[i, All]] = rotation1[[i, All, All]].a[[t, i, All]];];
    (* Shift Process with the normalization *) For[i = 1 + 2 n - 2 t,
     i <= 2 n + 2 t + 1, i++, temp[[i + 1, 1]] = d[[i, 1]]; temp[[i, 2]] = d[[i, 2]];];
    For[i = 1 + 2 n - 2 t, i <= 2 n + 2 t + 1, i++, (* Coin Flip *)
     d[[i, All]] = rotation2.temp[[i, All]];];
    (* Shift Process with the normalization *) For[i = 1 + 2 n - 2 t, i <= 2 n + 2 t + 1,
     i++, a[[t + 1, i, 1]] = d[[i, 1]]; a[[t + 1, i - 1, 2]] = d[[i, 2]];]; ];
   Table[{i - 2 n, Abs[a[[t, i, 1]]]^2 + Abs[a[[t, i, 2]]]^2},
    {t, 1, n - 1, 1}, {i, n, 3 n, 1}] ];
```

In[161]:=
```
phasediagram[theta1m_, theta1p_, theta2_] :=
 Module[{thetam = theta1m, thetap = theta1p, theta2angle = theta2,
   pline1, pline2, pline3, pline4, pline5, pline6, theta2line, tx, dots},
  pline1 = Plot[x, {x, -2 π, 2 π}, PlotStyle -> {Red, Dotted, Thickness[.005]}];
  pline2 = Plot[2 π - x, {x, 0, 2 π}, PlotStyle -> {Red, Dotted, Thickness[.005]}];
  pline3 = Plot[-2 π - x, {x, -2 π, 0}, PlotStyle -> {Red, Dotted, Thickness[.005]}];
  pline4 = Plot[-x, {x, -2 π, 2 π}, PlotStyle -> {Black, Thickness[.005]}];
  pline5 = Plot[2 π + x, {x, -2 π, 0}, PlotStyle -> {Black, Thickness[.005]}]; pline6 =
   Plot[-2 π + x, {x, 0, 2 π}, PlotStyle -> {Black, Thickness[.005]}]; theta2line =
   Plot[theta2, {x, -2 π, 2 π}, PlotStyle -> {Black, Dotted, Thickness[.005]}];
  tx = Graphics[{GrayLevel[.8], Rotate[Rectangle[{π - π/Sqrt[2], -π/Sqrt[2]},
       {π + π/Sqrt[2], π/Sqrt[2]}], 45 Degree, {π, 0}], GrayLevel[.8], Rotate[
     Rectangle[{-π - π/Sqrt[2], -π/Sqrt[2]}, {-π + π/Sqrt[2], π/Sqrt[2]}], 45
      Degree, {-π, 0}], GrayLevel[.8], Polygon[{{-π, -π}, {-2 π, -2 π}, {0, -2 π}}],
    GrayLevel[.8], Polygon[{{π, -π}, {2 π, -2 π}, {0, -2 π}}], GrayLevel[.8],
    Polygon[{{π, π}, {2 π, 2 π}, {0, 2 π}}], GrayLevel[.8], Polygon[
     {{-π, π}, {-2 π, 2 π}, {0, 2 π}}], Text[Style["1", 30, Bold, Black], {π, 0}],
    Text[Style["1", 30, Bold, Black], {-π, 0}], Text[Style["1", 30, Bold, Black],
     {π, 3 π/2}], Text[Style["1", 30, Bold, Black], {-π, 3 π/2}],
    Text[Style["1", 30, Bold, Black], {π, -3 π/2}], Text[Style["1", 30,
      Bold, Black], {-π, -3 π/2}], Text[Style["0", 30, Bold, Black], {0, π}],
    Text[Style["0", 30, Bold, Black], {0, -π}], (*Text[Style["0",30,Bold,Black],
     {3π/2,π}],*)Text[Style["0", 30, Bold, Black], {-3 π/2, π}],
    Text[Style["0", 30, Bold, Black], {-3 π/2, -π}] }];
  dots = Graphics[{Green, Disk[{thetam, theta2}, π/10], Blue, Disk[{thetap, theta2},
      π/10], Text[Style["Left", Bold, Green], {thetam, theta2 + π/5}],
    Text[Style["Right", Bold, Blue], {thetap, theta2 + π/5}]}];
  Show[tx, pline1, pline2, pline3, pline4, pline5, pline6, theta2line,
   dots, PlotRange -> {{-2 π, 2 π}, {-2 π, 2 π}},
   PlotRangePadding -> 0, Axes -> False, Frame -> True,
   FrameTicks -> {{{-2 π, -π, 0, π, 2 π}, None}, {{-2 π, -π, 0, π, 2 π}, None}},
   AspectRatio -> 1, FrameLabel -> {{"second rotation", None},
     {"first rotation", "phase diagram (winding number)"}}] ];
```

In[162]:=
```
Manipulate[GraphicsRow[
  {Show[Graphics[{Opacity[0.1, Green], Rectangle[{-plotmax, 0}, {0, 1.0}],
      Opacity[0.1, Blue], Rectangle[{0, 0}, {plotmax, 1.0}]}],
    ListPlot[distribution[plotmax, iniangle, thetam, thetap, theta2][[t + 1, All]],
     Filling -> Axis, FillingStyle -> Directive[Black, Thick],
     PlotRange -> {{-plotmax, plotmax}, {0, 1}}, PlotStyle -> PointSize[Medium],
     Joined -> True, Mesh -> All], AspectRatio -> 0.6,
    PlotRange -> {{-plotmax, plotmax}, {0, 1}}, PlotRangePadding -> 0,
    Axes -> True, Frame -> True, FrameTicks -> {{{0, 0.2, 0.4, 0.6, 0.8, 1.0}, None},
      {Table[(plotmax - 2) / 4 i - (plotmax - 2), {i, 0, 8}], None}},
    FrameLabel -> {{"probability", None}, {"sites", "probability distribution"}}],
   phasediagram[thetam, thetap, theta2]}, ImageSize -> {1000, 500}],
 {{t, 0, "steps"}, 0, plotmax - 2, 1, Appearance -> "Labeled"},
 {{plotmax, 20, "maximum number of steps"},
  {100 -> "20", 42 -> "40"}, ControlType -> RadioButton},
 {{iniangle, 0, "initial spin"}, {0 -> "up", π / 2 -> "down"},
  ControlType -> RadioButton}, Delimiter,
 {{thetam, -3 * π / 8, "first rotation θ1: left bulk"}, -2 π, 2 π, π / 8},
 {{thetap, 9 * π / 8, "first rotation θ1: right bulk"}, -2 π, 2 π, π / 8},
 Delimiter,
 {{theta2, π / 2, "second rotation θ2"}, -2 π, 2 π, π / 8},
 AutorunSequencing -> {1},
 SaveDefinitions -> True]
```

Out[162]=

steps 18

maximum number of steps 20 40

initial spin up down

first rotation θ_1: left bulk

first rotation θ_1: right bulk

second rotation θ_2