Monte Carlo study of disorder in electronic tight-binding models

Author
Carl Pettersson

Supervisor
Erik van Loon

Thesis submitted for the degree of Bachelor of Physics
Project duration: II months

LUNDS UNIVERSITET

Department of Physics
Division of Mathematical Physics
Date: May 2023

Contents

Acknowledgements

First of all, I would like to thank my supervisor Erik for all of his great feedback and dedication during this entire project. Without him none of this would have been possible and I would not have been able to achieve, and even surpass, the goals that I had set for myself. Our weekly meetings have always been nice and have provided me with clear goals until the next meeting. Calculations for this thesis were performed on the v3-n2 cluster node, which was funded by Gyllenstiernska Krapperupsstiftelsen. Special thanks to my good friend Vilhelm Conradi for being there to discuss problems that arose as well as allowing me to borrow his computer, but mainly for supporting me as a friend. I would also like to thank my parents. Without their support this whole semester would have been a lot more stressful. Finally, I would like to thank all of my other friends who have supported me by making my time while studying just that more pleasant.

Abstract

Oftentimes solids are described by uniform, periodic lattices. In reality, however, there is often some disorder on some of the sites in the lattice. This disorder may come from, for example, there being a different type of atom or tightly bound electrons resulting in a larger on-site potential, affecting the electronic properties of the lattice. In this thesis the electronic properties of square, periodic lattices with a number of these impurity sites are studied. The main electronic properties that are studied are the electronic energies of the lattice, and the density of states. Several properties of the impurity sites are also studied. The Hamiltonians that describe the different lattices were created with the tight-binding model. Two models for the placements of impurities were considered. The binary mixture model which considers random placements of disorders and the Falicov-Kimball model which considers thermodynamic placements of disorder. The results from these models were discussed and compared. For the Falicov-Kimball model a Monte Carlo algorithm was developed to examine the system and provide accurate results. Some papers that had previously looked at approximate solutions to the Falicov-Kimball model were examined and some of the results were recreated in the hope that the developed program could be used as a benchmark for approximate solutions. Recreations of works with similar algorithms were very accurate and for the more approximate solutions my simulations shared all of the important characteristics. The efficiency of the Monte Carlo algorithm was evaluated by first studying a benchmark smaller system that could be calculated and verified by hand and then using a benchmark of a previous paper that solved the systems in a similar way. It was determined that the Monte Carlo algorithm worked as expected.

List of abbreviations

DOS - Density of states
BM - Binary mixture
FK - Falicov-Kimball
$\mathbf{N_D}$ - The number of impurity sites
$\mathbf{N_{DD}}$ - The number of diagonally connected impurity sites
$\mathbf{N_{AD}}$ - The number of horizontally and vertically connected impurity sites

1 Introduction and background

Within the field of solid-state physics, a common way of describing the behaviours of atoms and their electrons in a solid is with a uniform, periodic lattice. In reality, this description is not completely accurate and there are usually some sites on the lattice which have some types of impurities on them. This disorder may come from, for example, there being a different type of atom compared to the rest of the lattice or electrons that are tightly bound to an atom resulting in a larger on-site potential, affecting the electronic properties of the lattice.

The aim of this thesis is to study properties of electrons in square periodic lattices in the presence of disorder. This is done on two different models. One model is examined using a Monte Carlo algorithm [1], or more specifically a Metropolis-Hastings algorithm [2], and the other model is studied by a more random sampling. Although the Monte Carlo algorithm is a Metropolis-Hastings algorithm, it will still be referred to as a Monte Carlo algorithm. Where and how the impurity sites are placed depends upon the model.

The first model, the binary mixture model, or BM model, does not consider any thermodynamic properties nor the previously generated lattices when generating disorder configurations. Instead, each iteration generates a completely random disorder configuration.

The second model, the Falicov-Kimball model [3], or FK model, utilizes a Monte Carlo algorithm to generate and propose new lattice configurations which can be accepted or rejected based on the energetic properties of the configuration and the temperature. The FK model is widely used for investigating a variety of different phenomena such as crystallization [4], metal-insulator transitions [5][6][7], and many more. Finding solutions to the FK model is therefore of great interest. Many of these papers use different approximations to solve the FK model, often leading to some information being lost, mostly regarding finer details. The FK model has however been solved exactly for infinite dimensions by multiple methods such as using an equation of motion technique [8] and by using mean-field theory [9], which has been shown to be an effective solution in other studies like Ref. [10].

This thesis is focused on solving the FK model exactly for cases that were discussed in some of the aforementioned papers. In this way the developed Monte Carlo algorithm could be used as a benchmark for other approximate solutions, especially studies in two dimensions. The efficiency of the developed Monte Carlo algorithm was also evaluated as well as several properties of the studied lattices for certain important cases.

2 Theory

2.1 The tight-binding model

The tight-binding model is a model that describes the motion of electrons in a solid [11]. It simplifies their motions to be highly localised, either being on the locations of the atoms in the solid or having a chance to tunnel to an adjacent site. This model can be generalized for real-world lattices in all three dimensions, which makes it quite useful for studying both simple and more realistic cases. The tight-binding model is used for describing single-electron systems but can be used for multi-electron systems if it is assumed that there is no interaction between the electrons which is the case for this thesis.

2.1.1 The tight-binding Hamiltonian

The tight-binding Hamiltonian takes the form of a matrix with dimensions related to the number of sites on the lattice that is being studied. The energy eigenvalues of the Hamiltonian, ϵ_i, are the possible energies that the electrons in the system can have. Plotting them as a function of energy gives the density of states, DOS, of the system.

We first consider the one-dimensional case which is simply a straight line of atoms. When an electron sits on the n^{th} atom we denote the quantum state as $|n\rangle$. These states are orthogonal to each other which means that

$$\langle n|m\rangle = \delta_{nm}. \tag{1}$$

In a case where the electrons are considered to be completely localized the Hamiltonian would be very simple.

$$H = E_0 \sum_n |n\rangle \langle n| . \tag{2}$$

The Hamiltonian in Eq. 2 describes the electrons being stuck on their sites and only consists of a main diagonal of the on-site energy, E_0. As mentioned earlier, the tight-binding model also considers a probability of tunnelling between two different states. We add a term to the Hamiltonian that consists of $|m\rangle \langle n|$ terms where m and n once again are sites on the lattice. The notation $|m\rangle \langle n|$ signifies an electron moving from site n to m. For the tight-binding model, these sites have to be neighbours or $m = n \pm 1$. As one can see, the direction of tunnelling does not matter. We also add a hopping parameter, t, which is related to the tunnelling probability. The tunnelling probability is given by $\frac{t^2}{E_0^2}$. Adding the hopping parameter to the Hamiltonian one gets the final Hamiltonian, to describe one-dimensional systems. [12]

$$H = E_0 \sum_n |n\rangle \langle n| - t \sum_n \left(|n\rangle \langle n+1| + |n+1\rangle \langle n| \right). \tag{3}$$

As a matrix, this Hamiltonian still has the main diagonal consisting of E_0, but it now also has hopping parameters on other places. The lattice is considered to be periodic,

meaning that the first and last atoms are neighbours and that tunnelling between them is allowed.

The Hamiltonians for two and three-dimensional lattices are described in the same way and have an obvious similarity to the one-dimensional formulation as well. Instead of the states being just an integer, $|n\rangle$, we have them as vectors, $|\mathbf{r}\rangle$: [13]

$$H = E_0 \sum_{\mathbf{r}} |\mathbf{r}\rangle \langle \mathbf{r}| - t \sum_{\langle \mathbf{r}\mathbf{r}' \rangle} \left(|\mathbf{r}\rangle \langle \mathbf{r}'| + |\mathbf{r}'\rangle \langle \mathbf{r}| \right). \tag{4}$$

The sum index of the tunnelling term indicates that we only sum over neighbouring sites.

For a two-dimensional lattice, tunnelling does not only happen between atoms n and $n \pm 1$ but also to $n \pm N$ where N is the length of a side in the real-world lattice. A figure showing an example of a two-dimensional lattice with a side length $N = 4$ can be seen below in Fig. 1.

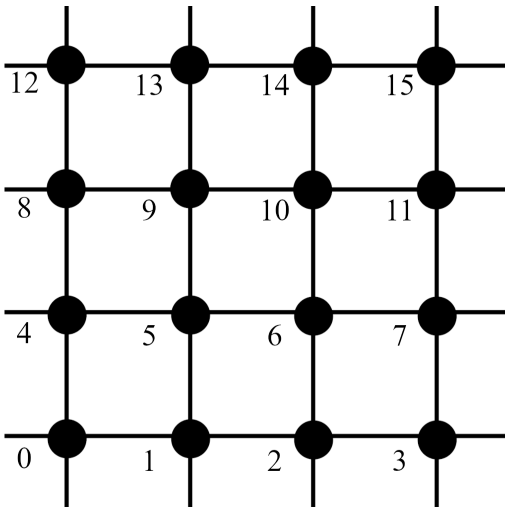


Figure 1: A two-dimensional lattice with a side length of $N = 4$.

It is clear that for example two of the neighbours to site 4 are sites 0 and 8, satisfying the condition of n and $n \pm N$ being neighbours.

2.1.2 Adding impurity sites to the tight-binding model

When creating the regular tight-binding Hamiltonians, as detailed in section 2.1.1, the sites were considered to have on-site energies of E_0. This on-site energy comes from properties of the atoms in the lattice. It does not come as a surprise that adding impurities to the lattice, in the form of either other types of atoms, other stationary electrons, or some sort of reshaping of the lattice, would affect the on-site energy. For simplicity's sake the

impurity sites are chosen to have on-site energies of $u = E_0 + w$ where w is the added energy from the disorder. These impurity energies of u simply replace E_0 on the main diagonal of the Hamiltonians for the sites that are disordered. It is important to note that certain papers that are referred to in this thesis follow a different naming convention, and refer to impurities as f-electrons and what here are called electrons as c-electrons.

2.2 The statistical properties of the systems

2.2.1 The Fermi-Dirac distribution and its uses

The Fermi-Dirac distribution is a probability distribution detailing the probability of fermions, such as electrons, occupying a certain energy eigenstate, ϵ_i. It is given by $f(\epsilon_i, \mu_e, \beta) = \frac{1}{e^{(\epsilon_i - \mu_e)\beta}+1}$, with μ_e being the electronic chemical potential, and where β is introduced as $\beta \equiv \frac{1}{k_B T}$. The units are chosen such that k_B is set to 1, essentially making β the inverse temperature. An important use for the Fermi-Dirac distribution in this thesis is calculating the total energies of the different disorder configurations. The total electronic energy of a system is given by the sum of all ϵ_i of the Hamiltonian multiplied with the Fermi-Dirac distribution to determine whether they are occupied or not. The total energy is also adjusted for the number and filling of the disorders.

$$E = \sum_{\epsilon_i} f(\epsilon_i, \mu_e, \beta) \cdot (\epsilon_i - \mu_e) - \mu_d N_D. \tag{5}$$

Here N_D is the number of impurity sites and μ_d fills a similar purpose to μ_e, but instead determines the number of the impurities. The electronic system is half-filled.

2.2.2 The partition function

When considering a system with M possible microstates all with respective energies E_i for $i = 1, 2, ..., M$, the possibility of each microstate appearing depends on its respective energy and the partition function. Here, these microstates are represented by different disorder configurations. The canonical partition function is described by a sum over all possible disorder configurations with a dependence on their respective energies and the β of the system. [14]

$$Z = \sum_i e^{-\beta E_i}. \tag{6}$$

For a disorder configuration i with energy E_i, the probability of that configuration appearing is given by

$$P = \frac{e^{-\beta E_i}}{Z}. \tag{7}$$

3 Method

For this thesis I wrote three main pieces of code, one for simulating the BM and FK model each, and one for reading the data to create plots. The entire code was created from scratch. For both models, the corresponding program was set to run over a set number of iterations with each iteration generating a new disorder configuration. The data from the runs were then saved so that they could be read and plotted in a separate file.

All of the calculations were done in Python and the finished code for a general run of the FK model can be seen in Appendix B. The runs were performed on multiple different devices, including v3-n2, a computer node at the Division of Mathematical Physics.

3.1 The observables

The main electronic property that was chosen to be studied was the DOS. From each Hamiltonian, the eigenvalues were calculated and added to histograms to create the DOS, of the lattice. For plotting the DOS, a smooth and continuous function is wanted. To get this instead of just a list of discrete numbers, the ϵ_i were added to small bins. This also reduced the impact that rounding errors had. The bins were chosen to be $\approx 0.01t$ wide. For the FK, model the eigenvalues also played another role. They were also used to calculate the total energy of the systems, which in turn was used to determine whether proposed Monte Carlo moves would be accepted or not.

At every iteration, information about the number and relative placements of the impurity sites were collected. The configurations that "survived" the most proposed changes along with the last standing configuration were saved. Finally, histograms of N_D, as well as the number of pairs of impurity sites that were diagonal neighbours to each other, which we call N_{DD}, were plotted. The number of pairs of either horizontal, or vertical neighbours, which we denote N_{ADD}, were plotted together with N_{DD}.

3.2 Constructing the Hamiltonians

The Hamiltonian for a one-dimensional lattice is quite trivial. Tunnelling between sites can only happen left to right or right to left leading to the tunnelling term of the Hamiltonian only being below and to the right of the main diagonal. For a lattice consisting of four atoms the resulting matrix can be seen below in Fig. 2.

$$\begin{pmatrix} E_0 & -t & 0 & -t \\ -t & E_0 & -t & 0 \\ 0 & -t & E_0 & -t \\ -t & 0 & -t & E_0 \end{pmatrix}$$

Figure 2: The tight-binding Hamiltonian for a one-dimensional lattice with $N = 4$.

When translating to the two and three-dimensional cases, what counts as a neighbour becomes a bit more complicated. The one-dimensional case is as previously mentioned a straight line with equidistant sites. For example, the neighbours of site 3 are naturally 2 and 4. When considering a two-dimensional lattice such as the one in Fig. 1, the neighbours of site 3 are 2, 0, 15, and 7. Translating this to the Hamiltonian this means that we would have $-t$ at matrix indices [3,2], [3,0], [3,15], and [3,7] as well as their conjugates. The size of the Hamiltonians also depend on the number of dimensions, with the one-dimensional case having the shape N x N and the two-dimensional case N^2 x N^2 where N is the linear size of the lattice. As described earlier the impurity sites had on-site energies of $u = E_0 + w$ instead of simply E_0.

3.3 Binary mixture and Falicov-Kimball

Two different models were examined, the BM model and the FK model. The base Hamiltonians of both systems as well as the parameters being held constant were the same in both models. Most of these parameters are given in units of t i.e. $t = 1$. For each simulation, the lattice side length, $N = 2$ or $N = 16$, and the non-disordered on-site energy, $E_0 = 0t$ were all held constant. The probability of each site having an impurity, $p = 0.5$ was also held constant. The BM model simply had a parameter adjusting the probability of each site having an impurity, but for the FK model μ_d had to be set to $u/2$ to get the same result. Both β and u were varied to examine different situations.

3.3.1 Binary mixture

The BM model was the simpler of the two models. For each iteration a new, randomly generated disorder configuration with no relation to the previous one was created and the eigenvalues that arose from this new Hamiltonian were saved to the DOS histogram.

3.3.2 Falicov-Kimball

The FK model, first proposed in Ref. [3], is one of the simpler models for describing interactions between immobile disorders on the atomic sites and mobile electrons. The mobile electrons are sometimes referred to as c-electrons with the added disorders being called f-electrons. Even though it is quite simple, this does not mean that the solutions are easy to find.

The method used in this thesis to solve the FK model employs a Monte Carlo algorithm, or more precisely a Metropolis-Hastings algorithm, to find disorder configurations that are more energetically favourable or configurations to which there can be a thermal excitation. This means that there is a strong dependence on the temperature when choosing whether to accept or reject proposed new configurations. For each iteration, a new disorder configuration was proposed and the energies for both the current and new Hamiltonian were calculated using Eq. 5 and compared. If the proposed disorder configuration resulted in a Hamiltonian with a lower energy, it immediately got accepted as the starting point for the

next iteration. If the new configuration instead resulted in a higher energy, the probability of this new configuration being accepted was given by the ratio of the probabilities given by Eq. 7. The resulting probability of the proposed change being accepted, $P_{\text{excitation}}$ was therefore given by

$$P_{\text{excitation}} = e^{-\beta(E_{\text{new}} - E_{\text{old}})}. \tag{8}$$

For each iteration the current Hamiltonian's ϵ_i were added to the DOS histogram. N_D, N_{AD}, and N_{DD} were also saved.

The Monte Carlo updates used for this method was quite simple, only consisting of one type of move, flipping. This move chose a rectangular area of varying size somewhere on the lattice, changing all of the impurity sites within the area to non-impurity sites and the opposite for the non-impurity sites. The different sizes of flipping areas all had probabilities associated with them, with the small flipping areas generally having larger probabilities than the larger ones.

When choosing the moves to add to a Monte Carlo algorithm it is important to consider the concept of detailed balance, nicely described in [15] with the quote "Corresponding to every individual process there is a reverse process, and in a state of equilibrium the average rate of every process is equal to the average rate of its reverse process". In other terms, at equilibrium the probability of any given move should be the same as its opposite [16]. The flipping move automatically considers detailed balance by being its own opposite move. For $N_D = \frac{N^2}{2}$ there is an equally large chance for a impurity site to become a non-impurity site as the opposite since there are equally many impurity and non-impurity sites. If $N_D > \frac{N^2}{2}$ there is a larger probability for N_D to decrease than increase due to there being more impurity sites than non-impurity sites and the flipping occurring at a random position would consequently be more likely to decrease N_D. This way the flipping move always works towards an equilibrium position.

A list of the used moves along with their respective probabilities is detailed in Appendix A.

4 Results and discussion

Both of the models follow the same general routine for each iteration. First a new disorder configuration is generated. As previously discussed, the way that these configurations are generated depends on the model being examined. Secondly, the configurations ϵ_i and total energy are calculated. Thirdly, all of the relevant observables are saved for later plotting. Finally, a new configuration is generated and in the case of the FK model, the new configurations energy is used to determine whether the new configuration will be used for the next iteration. The process is then repeated.

4.1 The Falicov-Kimball model

4.1.1 Results for the 2×2 lattice

Although considering a 2×2 lattice is a bit limiting in regards to the DOS, it is a useful benchmark since it can be easily calculated and verified by hand. Calculating the distributions of N_D is especially easy using Eq. 7. The results of these distribution for different β are seen below in Fig. 3.

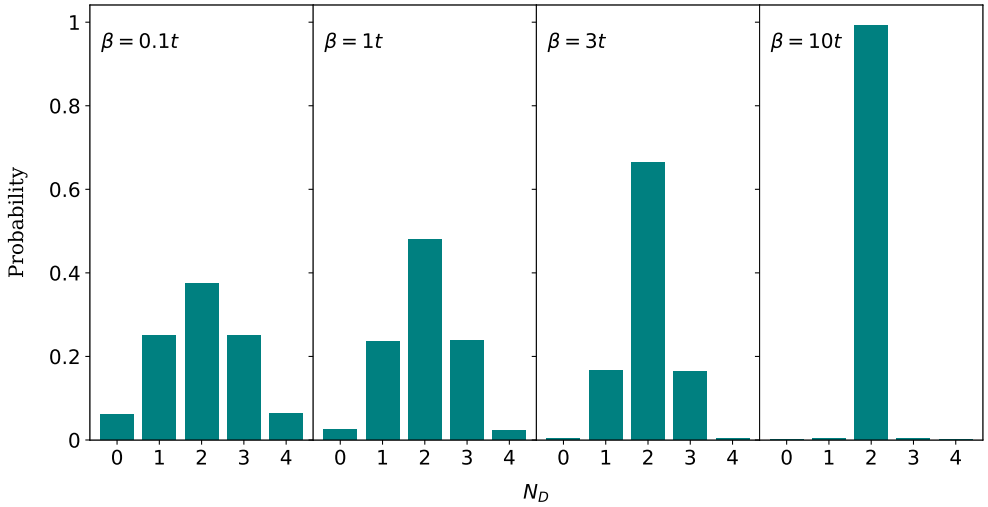


Figure 3: The respective probabilities for all N_D in the 2×2 lattice for several different values of β.

The theoretical probability distributions are also calculated with Eq. 7 and can be seen along values obtained from the Python program for $\beta = 1$ and $\beta = 10$. Here, and for the rest of the thesis, $\mu_d = \frac{u}{2}$. This means that $\frac{N^2}{2}$ is the most probable value for N_D. Due to the probabilistic nature of the simulation, it takes quite a few iterations for the values from the program to align with the theoretical values. For a 2×2 lattice it takes around 50 000 iterations before the theoretical values are reached.

Table 1: The theoretical probabilities from Eq. 7 and the probabilities obtained from the program of the different N_D appearing in a configuration for $\beta = 1$.

N_D	**Probability calculation with Eq. 7**	**Probability from program**
0	0.024	0.024
1	0.24	0.24
2	0.48	0.48
3	0.24	0.24
4	0.024	0.024

Table 2: The theoretical probabilities from Eq. 7 and the probabilities obtained from the program of the different N_D appearing in a configuration for $\beta = 10$.

N_D	**Probability calculation with Eq. 7**	**Probability from program**
0	0	0
1	0.005	0.005
2	0.99	0.99
3	0.005	0.005
4	0	0

The probability for different N_D as a function of β can also be plotted as shown below in Fig. 4. Note that the probabilities for $N_D = 1$ and $N_D = 3$ as well as $N_D = 0$ and $N_D = 4$ are the same and they are therefore plotted as one. This gives us a better idea of how N_D evolves as the temperature changes and more clearly shows when we go from N_D distributions having other values than $N_D = 2$ to only consisting of $N_D = 2$.

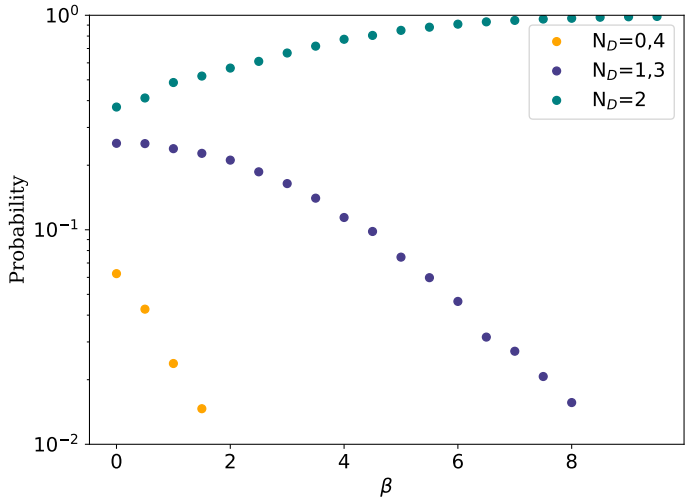


Figure 4: The probabilities for different N_D as functions of β.

The energies of the different configurations can also be studied. All configurations with either 0, 1, 3, or 4 only have one unique value of their energies. This is due to the sites themselves having no intrinsic properties, so it does not matter where the impurity sites are placed. For the $N_D = 2$ case, there are two different types of configurations. Four of the six possible microstates have the two impurity sites being placed on adjacent sites. One can consider a "impurity site vector", **d**, which connects two neighbouring impurity sites to be either **d**=(1,0) or **d**=(0,1).

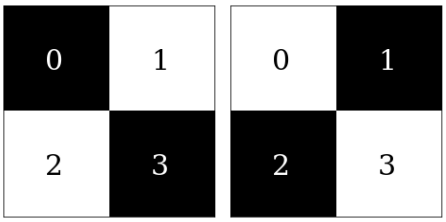


Figure 5: The two most energetically favourable cases in the 2×2 lattice. The black squares represent impurity sites and white sites represent non-impurity sites.

The other possible configuration has the two impurity sites being diagonally connected or connected by $\mathbf{d} = (1, 1)$. The configurations containing a **d**=(1,1) vector had considerably lower energies than the ones containing **d**=(1,0) or $\mathbf{d} = (0, 1)$. Figures of how the two different cases for **d**=(1,1) look can be seen in Figure 5, with the black squares representing impurity sites and white sites represent non-impurity sites.

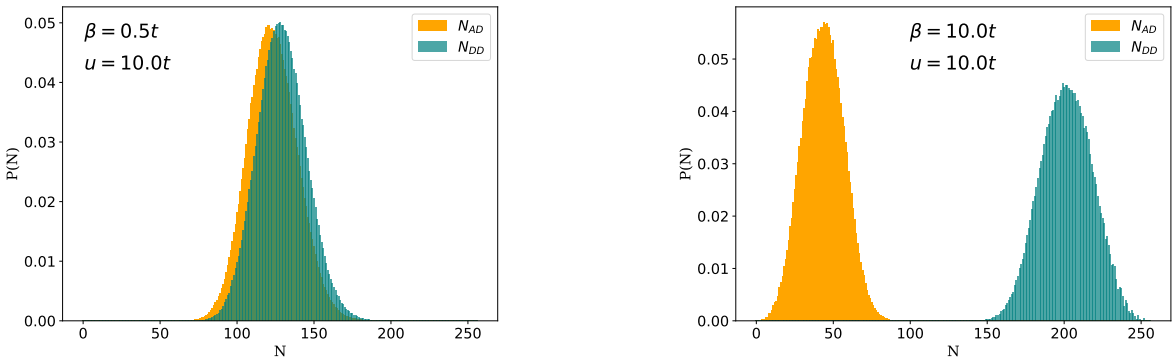
4.1.2 Discussion for the 2×2 lattice

From tables 1 and 2 one can quite clearly see that the values from the program align very nicely with the theoretical values. From Fig. 4, there is a clear correlation between decreasing temperatures and increasing probability of $N_D = 2$. All of these observations combined support the idea that the Monte Carlo works as intended.

The 16×2 lattice also shows why only flipping either a single, or a small number of sites could be problematic. From either of the two most common configurations in Fig. 5, the other could not be reached with a single small Monte Carlo move. For the other configuration to be reached it had to go through multiple less energetically favourable moves. For larger β it would basically be impossible for both of the cases in Fig. 5 to be studied in the same run. To avoid similar cases in the 16×16 lattice, moves that flipped large parts of the lattice and even moves that flipped the entire lattice were added.

4.1.3 Results from the 16×16 lattice

From studying the 16×2 lattice, it is clear that N_{DD} and N_{AD} play a large role in determining the likelihood of certain configurations appearing. The distributions of N_{DD} and N_{AD} are shown for two different values of β in Fig. 6.

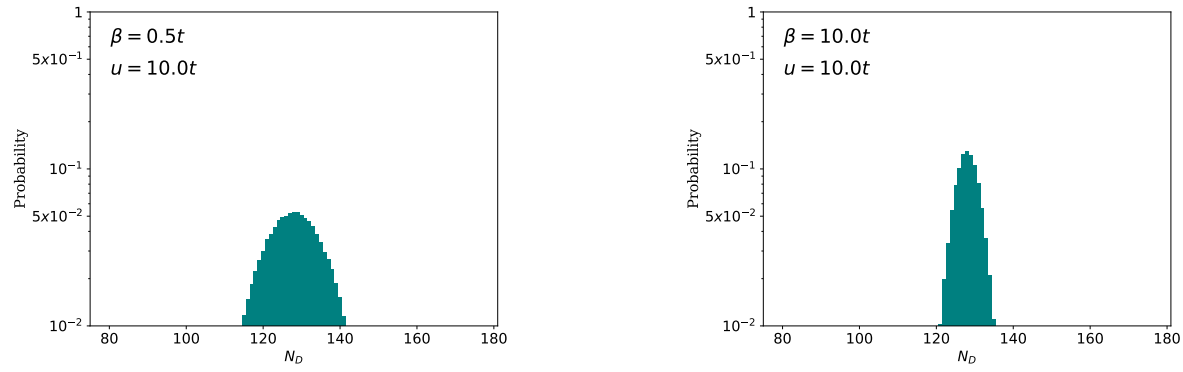


(a) **(b)**

Figure 6: Distributions of the probabilities for N_{DD} and N_{AD} for $\beta = 0.5$ and $\beta = 10$. Note that probabilities of N_{DD} and N_{AD} individually add up to 1.

When plotting N_{DD} and N_{AD} for different values of β as seen in Fig. 6, one sees a clear splitting up of the two distributions, with N_{DD} becoming much larger for increasing β whilst N_{AD} tends to be much smaller. The N_{AD} distribution never seems to overtake the N_{DD} distribution regardless of β and for smaller β, both values start moving towards $\frac{N^2}{2}$.

Something else that can be examined is the distribution of N_D for different values of β. Two of these distributions can be seen below in Fig. 7.



(a) **(b)**

Figure 7: The distribution of N_D for $\beta = 0.5$ and $\beta = 10$. Note the logarithmic scale.

Unsurprisingly much fewer N_D are represented for larger β. Similarly to with the 16×2 lattice it could also be interesting to plot the evolution of N_D as a function of β for a few different N_D. These were chosen to be at and close to $N_D = \frac{N^2}{2}$. As expected, the probability of $N_D = 128$ appearing greatly increases as β increases. The probabilities for N_D that are further away from $\frac{N^2}{2}$ reach zero much faster than those closer to $\frac{N^2}{2}$. Interestingly for $N_D = 123$ it seems to be relatively constant until it starts decreasing at a $\beta \approx 8$. This aligns nicely with Fig. 7 where $N_D \approx 123$ seems to be essentially the same for $\beta = 0.5$ and $\beta = 10$. One can also see how further increasing β from Fig. 7b would result in the probability for $N_D \approx 123$ to start decreasing.

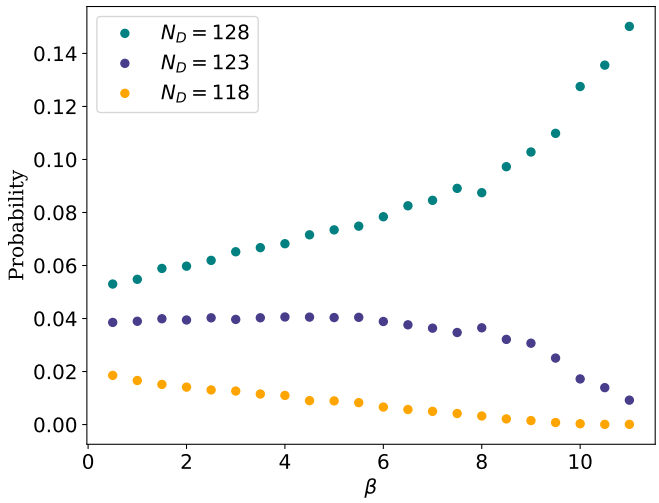


Figure 8: The probabilities for different N_D as functions of β. N_D were chosen at, and close to the equilibrium of $\frac{N^2}{2} = 128$.

The DOS is very important for determining electronic properties of lattices and is therefore also studied. One aspect that can be studied from the DOS is the conducting ability of the lattice. Insulators can be recognized by their DOS being separated into two different peaks. Metals, on the other hand, have their DOS as a single peak. Typical DOS for metals and insulators at two different β are plotted.

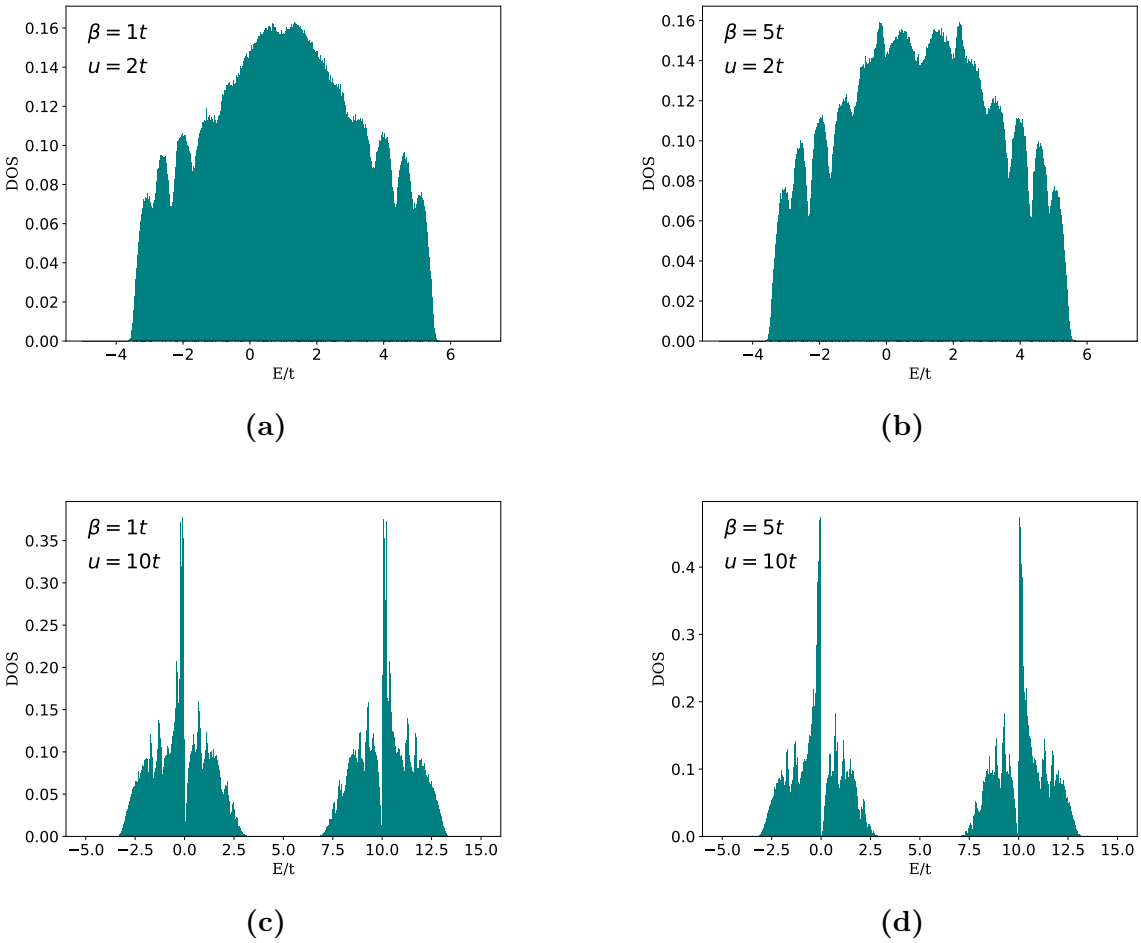


Figure 9: a-b) Typical DOS for metals for two different values of β. c-d) Typical DOS for insulators for two different values of β.

4.1.4 Discussion for the 16×16 lattice

As was noticed from the 16×2 lattice, it was clear that a **d**=(1,1) led to much lower E compared to configurations with **d**=(1,0) or **d**=(0,1). This was also the case for 16×16 lattice where there was a very strong correlation between larger N_{DD} and smaller E. This connection is quite clearly seen in Fig. 6 where regardless of temperature $N_{DD} > N_{AD}$.

Along with the energy of each configuration, β also plays a large role in determining whether new configurations with larger energies than the previous one are to be accepted or not. The equation determining the probability of these configurations being accepted, Eq. 8, is important to analyse. For small β, the difference in energy matters less as opposed to for large β where the energy difference is amplified by β. As mentioned earlier, there is a strong correlation between N_{DD} and lower energies which in turn naturally means that more

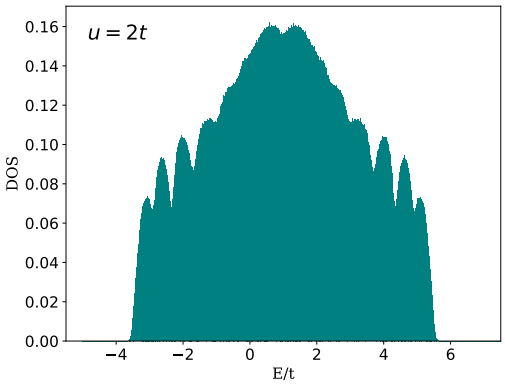
probable disorder configurations should have larger N_{DD}. This correlation does hold, but we once again see a strong temperature dependence with the distribution of N_{DD} and N_{AD}. For small β, the N_{DD} and N_{AD} distributions move closer towards each other and there is a large overlap. For increasing β, N_{DD} also increases while N_{AD} decreases. For sufficiently large $\beta \gtrapprox 3$, the energy difference is the largest contributor to whether configurations are accepted or not which eventually leads to the most energetically favourable pattern, with $N_{AD} \approx 0$, being a checkerboard pattern. The speed at which this checkerboard pattern is reached and the rarity of deviations from it are also strongly related to β. On the other hand, smaller values of β lead to almost every proposed change being accepted, making it so that many more N_D are represented and the checkerboard pattern that was seen for larger β never really appears.

The DOS in Figures 9a and 9b are for typical metals while 9c and 9d are for typical insulators. The determining factor whether the DOS are for metals or insulators was u, while β played the role of affecting the number of different configurations being examined. Comparing two DOS with equal u, one can see that increasing β mostly added towards the already most commonly occurring energies while having fewer of the values in between the two largest peaks. This comes as a quite natural consequence from the previously mentioned fact that larger β meant that a smaller number of configurations are studied, with the most energetically favourable configurations occurring even more often.

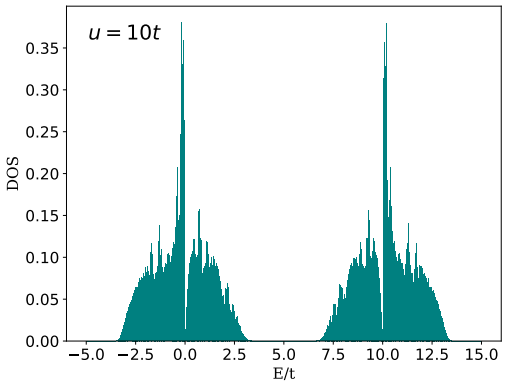
4.2 The binary mixture model

Due to the previously discussed lack of dependence of temperature for the BM model, the probability distribution of N_D simply becomes the same as a normalized binomial distribution of the number of microstates for each N_D.

The DOS is much more interesting, especially when comparing with the FK model. Figure 10 shows the DOS for two different values of u. In Fig. 10a the DOS is representative of a typical metal DOS while Fig. 10b shows the DOS typical for insulators. At each iteration, the eigenvalues of the current Hamiltonian were added to the DOS histograms. After all iterations the DOS was normalised.



(a) The DOS for $u = 2$ in the BM model.



(b) The DOS for $u = 10$ in the BM model.

Figure 10: The DOS from the BM model for two values of u. They bear a strong resemblance to the DOS in Figures 9a and 9c.

4.2.1 Discussion from the binary mixture model

Since DOS for metals and insulators were created for both the BM model and FK model it could be interesting to compare them. The DOS in Fig. 10 are therefore compared to those Fig. 9 for their respective values of u. It is clear that the BM DOS bears a larger resemblance to the FK DOS for smaller β as opposed to larger β. Due to the fact that for each iteration a new configuration is generated and accepted, the binary mixture model is similar to what is done with in the FK model for an infinite temperature, where every proposed change would be accepted. There are naturally some differences, mainly with how the new configurations are generated. The Monte Carlo algorithm has a few different moves it can propose with different probabilities assigned to them with, in general, smaller changes being more probable to be proposed than larger changes. This is very different to the BM that always generates completely random configurations. For a completely different configuration to be generated in the Monte Carlo algorithm, it has to run through a multitude of different configurations before reaching one that is completely different from the first one. Since the binary mixture generates a completely random configuration for each iteration, there is no problem with having to go through all the intermediate steps before reaching the completely new configuration. Since each configuration was generated without any consideration of previous ones the computing time each iteration took was quite short.

As discussed previously, there are clearly some merits to using the simpler binary mixture model in comparison to the Monte Carlo algorithm in the FK model. Two aspects make the BM model much faster compared to the FK model. Firstly, that a wider range of different configurations can be examined in a smaller number of iterations, and secondly, that each

iteration in itself takes a shorter time. The BM model is, however, less realistic due to its lack of dependence on temperature. As discussed earlier, there are some clear similarities between the DOS for the BM model and the DOS for FK model with an infinitely large temperature. The main difference between the two models is that the BM sacrifices a bit of realism and flexibility in what systems can be examined in exchange for faster computing while the FK does the opposite.

4.3 Comparison to literature and evaluation of the program

For this section the FK model was used since all of the examined literature also considers the FK model.

In this thesis, the DOS for a multitude of different lattices with different disorder configurations has been studied. This is something that has been done before, both with methods similar to the ones employed in this thesis and more approximate solutions. A paper that has solved the FK model in the same way as this thesis is Ref. [17], just with $N = 20$ instead of $N = 16$. The reason for why my recreation used $N = 16$ is simply that the rest of the thesis used that lattice size. This paper could prove a useful benchmark. Three of the DOS from Ref. [17] are recreated to see whether the results generated from the program are accurate.

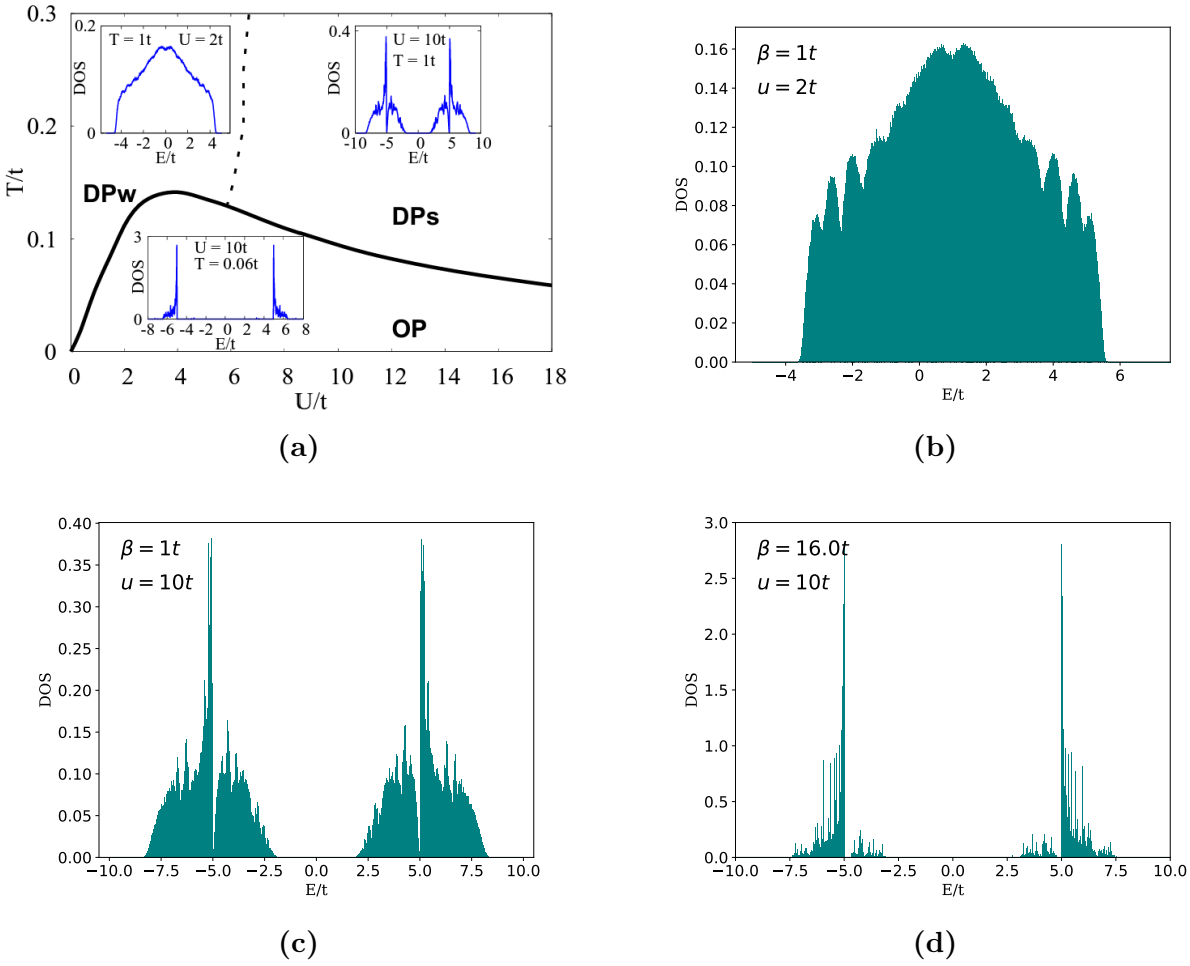
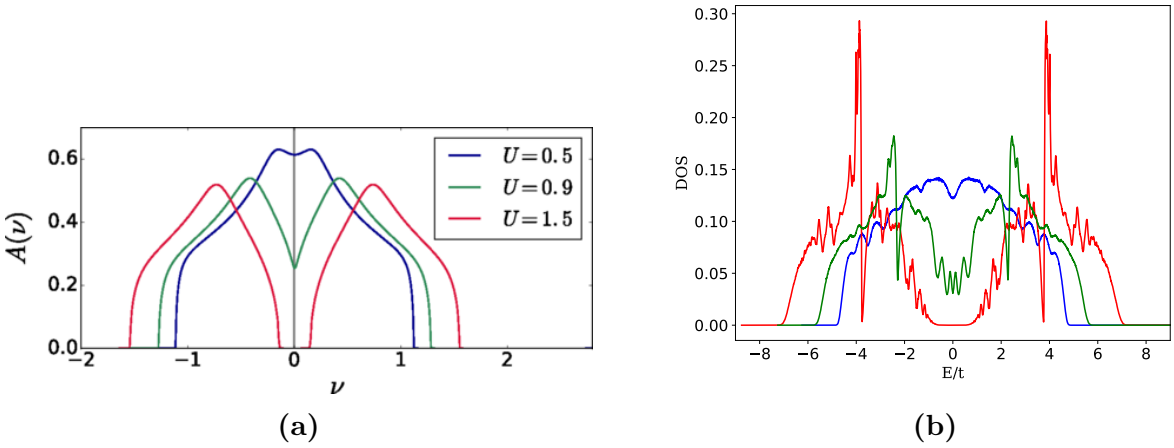


Figure 11: A useful benchmark from a paper that found solutions to the FK model in a similar way to what was done in this thesis. a) DOS from M. Thoss, M. Žonda, J. Okamoto. *Physical Review B*,100,075124, 2019 [17] for a lattice with $N = 20$. Used with permission from the American Physical Society. b-d) Recreations of the DOS in a) by using the program written for this thesis.

This figure contains three different cases for different values of T and u. We note the use of T instead of β i.e $T = 0.06t$ gives $\beta = 16t$. Comparing Figures 11b, 11c, and 11d to the three DOS in Figure 11a one can see that all of the main characteristics were nicely recreated. It is important to note that there is a difference in the sizes of the lattices that were examined. Ref. [17] considered a 20×20 lattice while my DOS were 16×16. This is definitely something that could have affected the results. Each Hamiltonian has N^2 eigenvalues, so even a small change in lattice size adds quite a large number of eigenvalues. Adding more eigenvalues, which is the case for larger lattices, consequently leads to smoother DOS with a smaller amount of peaks. Furthermore, there could also be some minor differences due to how the bins were created or other programming factors. In

general, the recreations align very nicely with the benchmark DOS.

Finding the DOS has also been done by other studies using different approximate methods which makes it useful to compare the more exact results from the program when using the same starting parameters. One of these studies that solve the system in a more approximate fashion is Ref. [18]. Here a parquet dual fermion approach is taken, which is an extension to dynamical mean field theory. Dynamical mean field theory has been used to exactly solve infinite-dimensional systems, but when used for finite-dimensional systems, such as in Ref. [18] it yields a more approximate solution. Similarly to [17] a figure detailing the obtained DOS is given in the paper, which can be seen in Fig. 12a.



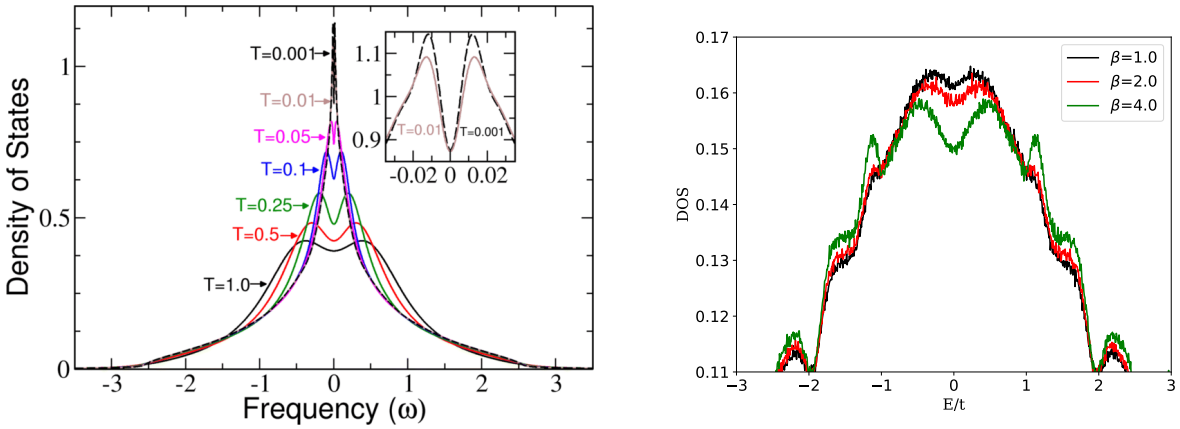
(a) **(b)**

Figure 12: A spectral function from a previous paper that solved the FK model in a more approximate way was recreated. a) A figure from K. Astleithner, A. Kauch, T. Ribic and K. Held. Physical Review B, 101:165101, 2020. [18] showing the spectral function obtained from using dual parquet equations to solve the FK model. Used with permission from the American Physical Society. b) Recreation of the three spectral functions seen in a).

Here there are some important remarks to be made. It is quite obvious that the amplitude of the DOS, especially for larger u does not match that in a). Right next to the peaks, there is a significant dip in the DOS with roughly the same magnitude as that of the peak. If one were to average these out, one would obtain something similar to the approximate solution in a). Disregarding the amplitudes, the shapes of the DOS, with their steeper edges towards the middle, as well as their behaviours at the very edges align nicely with the those in a). Due to the use of frequencies in s) the x-axis values are different. As a result of normalisation, this also affects the general amplitudes of the DOS, and explains why the amplitude in b) is different from a).

Another situation where using a Monte Carlo algorithm to solve the FK model could be

useful is when examining Mott transitions. Mott transitions are metal-insulator transitions that occur due to changes of potentials, such as u in this thesis. The transition between metal and insulator can be seen in the DOS, where the transition is noted by a separation of the DOS from one peak to two [19]. A paper regarding this transition is [20], although it considers an infinite dimensional system instead of the two-dimensional which is studied in this thesis. This paper mainly studies the central kink that appears within the DOS for decreasing temperatures. Although there are many other differences in the DOS, the central kink is successfully reproduced by my code. Note that only the DOS for $T = 0.25, T = 0.5$, and $T = 1$ are recreated.



(a) (b)

Figure 13: The DOS from a previous paper that studied the central kink that arises for decreasing temperatures was recreated with my code. a) A figure from R.D. Nesselrodt, J. Canfield, and J.K. Freericks. Journal of Superconductivity and Novel Magnetism, 33:2419, 2020. [20] showing how the central kink of the DOS changes for varying temperatures. b) Recreations of the DOS seen in a) for three different temperatures.

Note that due to the values of the DOS outside of the relevant area, the DOS with the largest values in Fig. 13a becomes the one with the lowest values in Fig. 13b.

Another paper that examines a very similar system is Ref. [21], which uses yet another way of getting solutions to the FK model. Typical DOS for both metals and insulators have already been plotted in Fig. 9.

Even though the developed Monte Carlo algorithm seems to give satisfactory results, it can still be useful to discuss the possible downfalls and what was done to lessen the possible errors. Due to there being no intrinsic properties of each site, this also means that mirrored or rotated versions of a certain disorder configuration would be equivalent in all respects. If a very energetically favourable configuration would arise, it would take multiple less

energetically favourable moves to reach a mirrored version of the configuration, or really any other energetically favourable configuration that is different from the first. This would be highly unlikely. This problem was previously discussed when looking at the 16×2 lattice. It was decided that to make it possible for both of these energetically favourable configurations to be reached during a single run by introducing moves that flip a large number of sites at the same time.

The number of sites that remain the same over a number of iterations is also something that could be of interest when studying the effectiveness and efficiency of the Monte Carlo algorithm. Every 50th iteration, the disorder configuration was saved and compared to the previously saved configuration. The number of sites that had changed between these two points was determined and eventually plotted as a function of the total number of iterations which can be seen in Fig. 14 for different values of β.

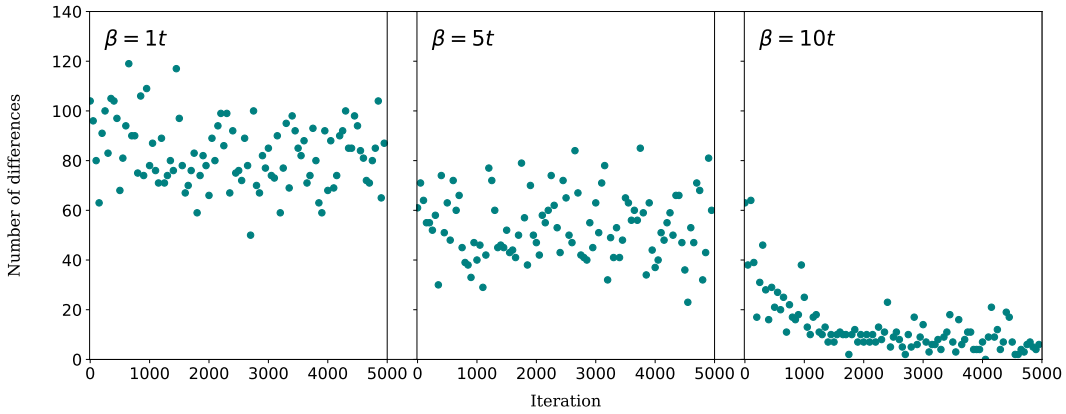


Figure 14: The number of differences between the first and last disorder configuration in 50 iteration intervals as a function of the total number of iterations for three different values of β.

As seen in Fig. 14 the autocorrelation depends heavily on β. For smaller β, there is a higher probability for less energetically favourable configurations to be accepted in accordance with Eq. 8. In the opposite case, when β is larger, changes essentially stop happening when an energetically favourable configuration has been reached. The Monte Carlo simulation essentially gets stuck in a local minimum. Due to this, plots for larger β converge towards smaller values while smaller β have larger values.

These plots suggest some improvements to the program. As seen in the plot for $\beta = 10t$ it is clear that roughly the first 1000 values are simply a warm up for the Monte Carlo algorithm. It is therefore reasonable to skip over the first 1000 configurations and to not use them for the DOS, N_D, etc. This change should improve the results for larger β while not really affecting runs for smaller β and was therefore added.

The autocorrelation is especially impactful for large β. Here the energy landscape is very steep, and the simulation tends to get stuck in local minima. Very few moves would take the system out of this minimum, and even then it is very likely that the next move would put the system back into the previous configuration. For small β the system never really reaches such a point. Even if it does, there is still a large probability of the system getting out of it.

For the purpose of creating the plots in Fig. 14 the flipping of the entire lattice was omitted. As mentioned previously, the resulting energy is the same and simply represents the same lattice from a different point of view. This leads to almost all 256 lattice sites being different from the previous measurements resulting in a plot entirely consisting of values of around 256.

5 Conclusion and Outlook

This thesis studied several properties of disordered lattices in the tight-binding model for two different models: the binary mixture model and the Falicov-Kimball model. The Hamiltonians used to describe the systems were created from the tight-binding model. Python programs were written to study both the BM model and FK model with a Monte Carlo algorithm being developed for studying the FK model.

In general, the program succeeded very well with recreating the DOS from the examined papers. Due to the wide variety of topics for which the DOS were recreated there were different conventions in how variables were presented, and what aspects of the DOS were relevant, which led to some minor differences between the original and recreated DOS. There is also the matters of binning the DOS. The width of the used bins was never specified in any of the papers, making it a bit more difficult to precisely recreate the DOS.

For the BM model, a 16×16 lattice was considered with each site having a probability of 0.5 to be a impurity site with an added potential to it. For each iteration, the eigenvalues of the Hamiltonian were calculated and added to the DOS. After the DOS was updated, a completely new disorder configuration was generated, and the process was repeated for a set number of iterations

The BM model resulted in much faster calculations but with fewer physical parameters being taken into account. The distribution of different N_D is simply a binomial distribution, following the same distribution as the number of microstates per N_D. Due to the completely random generation of each disorder configuration, there was no real reason to study anything regarding the properties of the placements of the impurity sites. The DOS closely resembled the DOS obtained from the FK model for small β. This is because small values of β mean that essentially every proposed move was accepted, leading to a somewhat similar process to that of the BM model.

The FK was studied for two different lattice sizes: one 16×2, and one 16×16. The 16×2 was mainly used to benchmark the accuracy of the program, since it is also easily calculated by hand. It was also used to see locally what effects lead to the most energetically favourable situations. The 16×16 case had many more variables that were studied. Here the quantity and relative locations of the impurity sites, the change of N_D as a function of β, and the DOS were all studied.

As previously discussed, the FK model is used to study a multitude of different phenomena and many papers have tried finding solutions for the model. The DOS presented in some of these papers were recreated. The accuracy of these recreations varied with the main differences in starting parameters being differences in the number of dimensions, sizes of the lattices, and naturally what approximate methods were used in the papers.

The hope with this thesis is that it can be used as a tool to benchmark approximate solutions. In that way it could help develop more accurate tools for solving the FK model

in two-dimensional systems. By providing these exact solutions improvements could be done to approximate methods with the eventual hope of being able to exactly solve the two-dimensional system in the same way that has been done for the infinite-dimensional system.

For the future of this project, there are still some things that could be upgraded with most of them focused on improving the efficiency of the Monte Carlo algorithm. The first thing that could be done, is to study the efficiency of different types of proposed moves could be studied such that a more efficient set of moves could be chosen to yield satisfactory results in a smaller number of iterations. In a similar matter, the probabilities of the different moves could also be continuously updated to yield results faster. This does however entail a problem, namely the question of when the actual calculations should be done and when the data should be used to update the probabilities of different moves. Finally, the thing that takes the most time when running the program is the calculation of the eigenvalues and the energies of each configuration that arise from these. The Hermitian nature of the Hamiltonians is already used to reduce the calculation time but if anything could be done to further increase the speed of which the eigenvalues and energies are found it would significantly reduce the time the program takes to run.

References

[1] A. Stevens. *Monte-Carlo simulation an introduction for engineers and scientists*. Taylor and Francis EBA 2022, Florida, 1 edition, 2023.

[2] S. Chib and E. Greenberg. Understanding the Metropolis-Hastings Algorithm. *The American Statistician*, 49:327, 1995.

[3] L.M. Falicov and J.C. Kimball. Simple Model For Semiconductor-Metal Transitions: SmB_6 And Transition-Metal Oxides. *Physical Review B*, 22:997, 1969.

[4] J. Jędrzejewski, J. Lach, and R. Ływa. Ground State Properties of the Spinless Falicov-Kimball Model: Crystallization and Metal-Insulator Transitions off the Hole-Particle Symmetry Point. *Physica A*, 154:529, 1989.

[5] M. Plischke. Coherent-Potential-Approximation Calculation on the Falicov-Kimball Model of the Metal-Insulator Transition. *Physical Review Letters*, 28:361, 1972.

[6] K. Michielsen. Bond-charge-site-charge interaction and metal-insulator transitions. *Physical Review B*, 50:4283, 1994.

[7] P. Farkasovsky. Pressure induced insulator-metal transitions in the spinless Falicov-Kimball model. *Physical Review B*, 52, 1995.

[8] V. Zlatić, J. K. Freericks, R. Lemanski, and G. Czycholl. Exact solution of the multicomponent Falicov-Kimball model in infinite dimensions. *Philosophical Magazine B: Physics of Condensed Matter*, 81:1443, 2001.

[9] J. K. Freericks and V. Zlatić. Exact dynamical mean-field theory of the Falicov-Kimball model. *Reviews of Modern Physics*, 75:1333, 2003.

[10] J. K. Freericks and A.M. Shvaika. Exact solution of a variety of X-ray probes in the Falicov-Kimball model with dynamical mean-field theory. *Condensed Matter Physics*, 15:43701, 2012.

[11] A.T. Paxton. Introduction to the tight binding approximation - implementation by diagonalisation. 2009.

[12] D. Tong. Solid State Physics University of Cambridge Part II Mathematical Tripos, 2017.

[13] D. Tong. Solid State Physics University of Cambridge Part II Mathematical Tripos, 2017.

[14] I. Ford. *Statistical Physics An Entropic Approach*. John Wiley Sons, London, 2013.

[15] G.N. Lewis. A new principle of equilibrium. *Proceedings of the National Academy of Sciences*, 11:179, 1925.

[16] D. Ter Haar. Foundations of Statistical Mechanics. *Reviews of Modern Physics*, 27:289, 1955.

[17] M. Žonda, J. Okamoto, and M. Thoss. Gapless regime in the charge density wave phase of the finite dimensional Falicov-Kimball model. *Physical Review B*, 100:075124, 2019.

[18] K. Astleithner, A. Kauch, T. Ribic, and K. Held. Parquet dual fermion approach for the Falicov-Kimball model. *Physical Review B*, 101:165101, 2020.

[19] M. Cyrot. Theory of Mott Transition: Applications to Transition Metal Oxides. *Le Journal de Physique*, 33:125, 1972.

[20] R.D. Nesselrodt, J. Canfield, and J.K. Freericks. Comparison Between the f-Electron and Conduction-Electron Density of States in the Falicov-Kimball Model at Low Temperature. *Journal of Superconductivity and Novel Magnetism*, 33:2419, 2020.

[21] A.M. Shvaika and J.K. Freericks. F-electron spectral function of the Falicov-Kimball model and the Wiener-Hopf sum equation approach. *Condens. Matter Phys*, 11:425, 2008.

Appendix A

Table 3: Table showing the respective probabilities for different sizes of flipped areas that were used in the Monte Carlo algorithm for the Falicov-Kimball model.

Size of flipped area	Probability
1x1	0.49
2x2	0.15
2x1	0.1
1x2	0.1
4x4	0.05
4x1	0.025
1x4	0.025
6x6	0.02
10x10	0.01
16x16	0.03

The different areas as well as their respective probabilities were arbitrarily chosen. That smaller areas were chosen to have higher probabilities but the specific probabilities were randomly chosen.

Appendix B

```
"""
Created on Thu Mar  2 12:22:00 2023

@author: carlp
"""
import numpy as np
import h5py as h5py
import time
import matplotlib.pyplot as plt
import random as random
import sys

hfont = {'fontname':'serif'}
csfont = {'fontname':'Comic Sans MS'}
plt.rcParams['font.size'] = 15

flipnumber=0
movenumber=0

st = time.time()
N=16
E0=0
```

```
t=1
Iterations= 2000000
NoBins=2633
binLowerE=-5
binUpperE=15
betaList=[]
wList=[]


prob128=[]
prob123=[]
prob118=[]

flipProb=1
moveProb=0
np.set_printoptions(threshold=sys.maxsize)

def neighbors(x,y):
    yield(x+1,y)
    yield(x-1,y)
    yield(x,y+1)
    yield(x,y-1)
def nearestNeighbors(x,y):
    yield(x+1,y)
    yield(x,y+1)
def diagnNeigbors(x,y):
    yield(x+1,y+1)
    yield(x-1,y+1)
def BaseHamiltonian(E0,t,N):
    DiagonalArray=np.zeros(N**2)
    for i in range(len(DiagonalArray)):
        DiagonalArray[i]=E0
    MarkHamill=np.diag(DiagonalArray)
    for y in range(N):
        for x in range(N):
            nFrom=y*N + x
            adjacent=neighbors(x,y)
            for neighborIndex in adjacent:
                if neighborIndex[0]==-1:
                    nToNegative=neighborIndex[1]*N + neighborIndex[0]
                    nTo=N+nToNegative
                elif neighborIndex[1]==-1:
                    nToNegative=neighborIndex[1]*N +  neighborIndex[0]
                    nTo=N**2+nToNegative
                elif neighborIndex[0]==N:
                    nToNegative=neighborIndex[1]*N + neighborIndex[0]
                    nTo=nToNegative - N
                elif neighborIndex[1]==N:
                    nToNegative=neighborIndex[1]*N +  neighborIndex[0]
                    nTo=nToNegative - N**2
                else:
```

```
                    nTo=N*neighborIndex[1]+neighborIndex[0]
                MarkHamill[(nFrom,nTo)]+=-t
    return(MarkHamill)

def FermiFunc(Beta,Eig,ChemE):
    return(1/(np.exp((Eig-ChemE)*Beta)+1))


def PotentialMatrix(w,N):
    ProbabilityArray=np.random.choice(2,N**2,p=pArtay)
    VMatrix=np.diag(ProbabilityArray)
    VMatrix=w*VMatrix
    DisorderCount=np.count_nonzero(ProbabilityArray)
    return(VMatrix,DisorderCount,ProbabilityArray)



def NumberOfElectrons(Beta,ChemE,DOS):
    return sum(DOS_E*FermiFunc(Beta, E, ChemE) for DOS_E,E in zip(DOS,
   IntegrationArray))*width

def Bins(EigenValues):
    hist, bin_edges = np.histogram(EigenValues, NoBins,(binLowerE,
   binUpperE))
    return((hist,bin_edges))

def Energy(ChemE, Beta,EigenValues,Disorders,ChemU):
    return sum(FermiFunc(Beta,EigenValues[i],ChemE)*(EigenValues[i]-ChemE
   ) for i in range(len(EigenValues))) - ChemU*Disorders

def move(w,Array):
    NewDisorderArray=Array.copy()
    nonzero=np.nonzero(NewDisorderArray)[0]
    listOfZeroNeighbours=[]
    while len(listOfZeroNeighbours)==0:
        d=random.choice(nonzero)
        listOfZeroNeighbours=[]
        moveNeighbors=[item for item in nearestNeighborList if d in item]
        for i in range(len(moveNeighbors)):
            if NewDisorderArray[moveNeighbors[i][0]]!=NewDisorderArray[
   moveNeighbors[i][1]]:
                listOfZeroNeighbours.append(moveNeighbors[i][0])
                listOfZeroNeighbours.append(moveNeighbors[i][1])
    listOfZeroNeighbours=list(filter(lambda a: a != d,
   listOfZeroNeighbours))
    moveTo=random.choice(listOfZeroNeighbours)
    NewDisorderArray[moveTo]=1
    NewDisorderArray[d]=0
    VMatrix=np.diag(NewDisorderArray)
    VMatrix=w*VMatrix
    DisorderCount=np.count_nonzero(NewDisorderArray)
```

```
    return(VMatrix,DisorderCount,NewDisorderArray)

flipProbabilities=[0.49,0.15,0.1,0.1,0.05,0.025,0.025,0.02,0.01,0.03]
flipMoves=[(1,1),(2,2),(2,1),(1,2),(4,4),(4,1),(1,4),(6,6),(10,10)
    ,(16,16)]
def flipBlock(w,Array,move):
    NewDisorderArray=Array.copy()
    row=random.randint(0, N-1)
    column=random.randint(0,N-1)
    for i in range(column,column+move[0]):
        for j in range(row,row+move[1]):
            if i>N-1:
                i = i - N
            if j>N-1:
                j=j-N
            realIndex=j*N+i
            if NewDisorderArray[realIndex]==0:
                NewDisorderArray[realIndex]=1
            else:
                NewDisorderArray[realIndex]=0
    VMatrix=np.diag(NewDisorderArray)
    VMatrix=w*VMatrix
    DisorderCount=np.count_nonzero(NewDisorderArray)
    return(VMatrix,DisorderCount,NewDisorderArray)


def ListOfNeighbours(N):
    verHorNeighborList=[]
    diagNeighborList=[]
    #First part for Horizontal and Vertical Neighbours
    for y in range(N):
        for x in range(N):
            nFrom=y*N + x
            adjacent=nearestNeighbors(x,y)
            for neighborIndex in adjacent:
                if neighborIndex[0]==-1:
                    nToNegative=neighborIndex[1]*N + neighborIndex[0]
                    nTo=N+nToNegative
                elif neighborIndex[1]==-1:
                    nToNegative=neighborIndex[1]*N +  neighborIndex[0]
                    nTo=N**2+nToNegative
                elif neighborIndex[0]==N:
                    nToNegative=neighborIndex[1]*N + neighborIndex[0]
                    nTo=nToNegative - N
                elif neighborIndex[1]==N:
                    nToNegative=neighborIndex[1]*N +  neighborIndex[0]
                    nTo=nToNegative - N**2
                else:
                    nTo=N*neighborIndex[1]+neighborIndex[0]
                verHorNeighborList.append((nFrom,nTo))
            #Diagonal elements
```

```
            diagonal=diagnNeigbors(x, y)
            for neighborIndex in diagonal:
                if neighborIndex[1]==N:
                    if neighborIndex[0]==N:
                        nTo=0
                    elif neighborIndex[0]==-1:
                        nTo=N-1
                    else:
                        nTo=N*neighborIndex[1]+neighborIndex[0]-N**2
                else:
                    if neighborIndex[0]==N:
                        nTo=N*neighborIndex[1]+neighborIndex[0]-N
                    elif neighborIndex[0]==-1:
                        nTo=N*neighborIndex[1]+neighborIndex[0]+N
                    else:
                        nTo=N*neighborIndex[1]+neighborIndex[0]
                diagNeighborList.append((nFrom,nTo))
    return(verHorNeighborList, diagNeighborList)


nearestNeighborList, diagNeighborList=ListOfNeighbours(N)
IntegrationArray = np.linspace(binLowerE,binUpperE,NoBins)
pArtay=np.array([0.5,0.5])
H0=BaseHamiltonian(E0, t, N)
noNearestNeighborList=[]
noDiagNeighborList=[]
for w,Beta in zip(wList,betaList):
    u=E0+w
    ChemU=u/2
    ChemE=u/2
    HistAmplitude=np.zeros(NoBins)
    excitation,other=0,0
    DisorderList=[]

    V,Disorders,DisorderArray=PotentialMatrix(w, N)
    Hamiltonian=V+H0
    Eig=np.linalg.eigh(Hamiltonian)
    EigVal=Eig[0]
    CurrentE=Energy(ChemE,Beta,EigVal,Disorders, ChemU)

    noChangesList=[]

    for CastleItter in range(Iterations):
        P=random.uniform(0,1)
        if P>flipProb: #Move
            NewV,NewDisorders,NewDisorderArray = move(w,DisorderArray)
            movenumber+=1
        else: #Flipping of block
            typeOfFlip=np.random.choice(len(flipProbabilities),p=
   flipProbabilities)
```

```
            NewV,NewDisorders,NewDisorderArray = flipBlock(w,
    DisorderArray,flipMoves[typeOfFlip])
            flipnumber+=1
        NewHamiltonian=NewV+H0
        NewEig=np.linalg.eigh(NewHamiltonian)
        NewEigVal=NewEig[0]
        NewE=Energy(ChemE,Beta,NewEigVal,NewDisorders,ChemU)
        ExcitedOrNot=random.uniform(0,1)


        if NewE<=CurrentE or ExcitedOrNot<=np.exp(-Beta*(NewE-CurrentE)):
            CurrentE=NewE
            EigVal=NewEigVal
            Disorders=NewDisorders
            DisorderArray=NewDisorderArray.copy()
            excitation+=1
        else:
            other+=1


        #Calculate the number of diagonal disorder neighbors
        #and vertical/horizontal disorder neighbors
        noNearest=0
        noDiagonal=0
        for i in range(len(nearestNeighborList)):
            if DisorderArray[nearestNeighborList[i][0]]==1 and
    DisorderArray[nearestNeighborList[i][1]]==1:
                noNearest+=1
        for i in range(len(diagNeighborList)):
            if DisorderArray[diagNeighborList[i][0]]==1 and DisorderArray
    [diagNeighborList[i][1]]==1:
                noDiagonal+=1
        noNearestNeighborList.append(noNearest)
        noDiagNeighborList.append(noDiagonal)



        if CastleItter % 50 == 0:
            A=DisorderArray.copy()
        if CastleItter % 50 == 49:
            B=DisorderArray.copy()
            noChanges=len(DisorderArray)-np.count_nonzero(A==B)
            noChangesList.append(noChanges)
            noChangesArray=np.array(noChangesList)


        #Saves # of disorders and adds to DOS histogram
        if CastleItter>=1000:
            if CastleItter % 10 == 0:
                DisorderList.append(Disorders)
                HistCurrent = Bins(EigVal)[0]
```

```
                HistAmplitude=np.add(HistAmplitude,HistCurrent)

            #Calculate the number of diagonal disorder neighbors
            #and vertical/horizontal disorder neighbors
            noNearest=0
            noDiagonal=0
            for i in range(len(nearestNeighborList)):
                if DisorderArray[nearestNeighborList[i][0]]==1 and
DisorderArray[nearestNeighborList[i][1]]==1:
                    noNearest+=1
            for i in range(len(diagNeighborList)):
                if DisorderArray[diagNeighborList[i][0]]==1 and
DisorderArray[diagNeighborList[i][1]]==1:
                    noDiagonal+=1
            noNearestNeighborList.append(noNearest)
            noDiagNeighborList.append(noDiagonal)


    #Take the middle between every set of two bin edges
    xListForPlot=[]
    for i in range(NoBins):
        xListForPlot.append((Bins(EigVal)[1][i]+Bins(EigVal)[1][i+1])/2)

    width=(((Bins(EigVal)[1][1]-Bins(EigVal)[1][0])))
    #Normalization see 06/02 in notebook
    HistAmplitudeNormalized = HistAmplitude/(N**2*((Iterations-1001)/10)*
width)

    #Check if normalization is 1
    Sum=sum(HistAmplitudeNormalized[i]*width for i in range(len(
HistAmplitudeNormalized)))
    print('Integral over everything',Sum)

    differentDisorders=[]
    for i in range(N**2+2):
        differentDisorders.append(i)

    #Histograms of number of disorders
    disAmplitude=np.histogram(DisorderList,bins=differentDisorders)[0]
    disAmplitudeNormalized=disAmplitude/((Iterations-1001)/10)
    #Histogram of number of neighbors
    nearestAmplitude=np.histogram(noNearestNeighborList,bins=
differentDisorders)[0]
    diagonalAmplitude=np.histogram(noDiagNeighborList,bins=
differentDisorders)[0]


    del differentDisorders[-1]
```

```
312
313     #Beta probability scatterplot
314     prob128.append(disAmplitudeNormalized[127])
315     prob123.append(disAmplitudeNormalized[122])
316     prob118.append(disAmplitudeNormalized[117])
317
318
319     #ChangeOverTime
320     changeOverTimeX=np.arange(0,Iterations,50)
321     figDetails=plt.figure(figsize=(8,6),dpi=600)
322     ax=plt.gca()
323     plt.gcf().subplots_adjust(bottom=0.12,left=0.11)
324     ax.plot(changeOverTimeX,noChangesArray,color='teal')
325     plt.xlabel('Iteration [eV]',**hfont)
326     plt.ylabel("DOS",**hfont)
327     # plt.bar(xListForPlot,HistAmplitudeNormalized,width=width,color='
    teal')#,edgecolor='black',facecolor='blue')
328     plt.savefig(f'ChangeOverTimeu{u}beta{Beta}.pdf',format="pdf",dpi=600)
329     plt.text(0.05,0.9,rf'$\beta={Beta}t$',fontsize=20,**hfont,transform=
    ax.transAxes)
330     plt.show()
331
332     hf=h5py.File(f'ChangeOverTime{w}{Beta}.h5', 'w')
333     RunningVariables=[N,E0,t,Iterations,Beta,u,width]
334     hf.create_dataset('noChangesArray',data=noChangesArray)
335     hf.create_dataset('changeOverTimeX',data=changeOverTimeX)
336     hf.create_dataset('RunningVariables',data=np.array(RunningVariables))
337     hf.close()
338
339
340     #DOS plotting.
341     figDetails=plt.figure(figsize=(8,6),dpi=600)
342     ax=plt.gca()
343     plt.gcf().subplots_adjust(bottom=0.12,left=0.16)
344     ax.bar(xListForPlot,HistAmplitudeNormalized,width=width*1,color='teal
    ')
345     plt.xlabel('E/t',**hfont)
346     plt.ylabel("DOS",**hfont)
347     # plt.bar(xListForPlot,HistAmplitudeNormalized,width=width,color='
    teal')#,edgecolor='black',facecolor='blue')
348     plt.text(0.05,0.9,rf'$\beta={Beta}t$',fontsize=20,**hfont,transform=
    ax.transAxes)
349     plt.text(0.05,0.8,rf'$u={u}t$',fontsize=20,**hfont,transform=ax.
    transAxes)
350     plt.savefig(f'DOSu{u}beta{Beta}.pdf',format="pdf",dpi=600)
351     plt.show()
352
353     hf=h5py.File(f'DOS{w}{Beta}.h5', 'w')
354     RunningVariables=[N,E0,t,Iterations,Beta,u,width]
355     hf.create_dataset('HistAmplitude',data=HistAmplitude)
```

```
    hf.create_dataset('HistAmplitudeNormalized',data=
HistAmplitudeNormalized)
    hf.create_dataset('xListForPlot',data=xListForPlot)
    hf.create_dataset('RunningVariables',data=np.array(RunningVariables))
    hf.close()
    
    
    
    
    
    #Disorder plotting.
    figDetails=plt.figure(figsize=(8,6),dpi=600)
    ax=plt.gca()
    plt.gcf().subplots_adjust(bottom=0.12,left=0.16)
    ax.bar(differentDisorders,disAmplitudeNormalized,width=0.8,color='
teal')
    ax.set_xlim(75,181)
    ax.set_yscale('log')
    plt.xlabel(r"$N_{D}$",**hfont)
    plt.ylabel("Probability",**hfont)
    plt.yticks([0.1,0.5,1,0.05,0.01], [r'$10^{-1}$', r'$5 x 10^{-1}$', r'
$1$', r'$5 x 10^{-2}$', r'$10^{-2}$'])
    plt.text(0.05,0.9,rf'$\beta={Beta}t$',fontsize=20,**hfont,transform=
ax.transAxes)
    plt.text(0.05,0.8,rf'$u={u}t$',fontsize=20,**hfont,transform=ax.
transAxes)
    plt.savefig(f'Disu{u}beta{Beta}.pdf',format="pdf",dpi=600)
    plt.show()

    hf=h5py.File(f'Dis{w}{Beta}.h5', 'w')
    RunningVariables=[N,E0,t,Iterations,Beta,u,width]
    hf.create_dataset('differentDisorders',data=differentDisorders)
    hf.create_dataset('disAmplitudeNormalized',data=
disAmplitudeNormalized)
    hf.create_dataset('RunningVariables',data=np.array(RunningVariables))
    hf.close()


    #Plotting number of diagonal neighbors and closest neighbors
    nearestAmplitudeNormalized=nearestAmplitude/((Iterations-1001))
    diagonalAmplitudeNormalized=diagonalAmplitude/((Iterations-1001))
    figDetails=plt.figure(figsize=(8,6),dpi=900)
    ax=plt.gca()
    plt.gcf().subplots_adjust(bottom=0.12,left=0.11)
    ax.bar(differentDisorders,nearestAmplitudeNormalized,width=0.8,color=
'orange', alpha=1,label=r'$N_{AD}$')
    ax.bar(differentDisorders,diagonalAmplitudeNormalized,width=0.8,color
='teal',alpha=0.7,label=r'$N_{DD}$')
    plt.xlabel("Number of type of neighbour",**hfont)
    plt.ylabel("Probability",**hfont)
```

```
    plt.text(0.05,0.9,rf'$\beta={Beta}t$',fontsize=20,**hfont,transform=ax.transAxes)
    plt.text(0.05,0.8,rf'$u={u}t$',fontsize=20,**hfont,transform=ax.transAxes)
    plt.legend()
    plt.savefig(f'Neighbour{u}beta{Beta}.pdf',format="pdf",dpi=900)
    plt.show()

    hf=h5py.File(f'Neighbor{w}{Beta}.h5', 'w')
    RunningVariables=[N,E0,t,Iterations,Beta,u,width]
    hf.create_dataset('differentDisorders',data=differentDisorders)
    hf.create_dataset('nearestAmplitude',data=nearestAmplitudeNormalized)
    hf.create_dataset('diagonalAmplitude',data=diagonalAmplitudeNormalized)
    hf.create_dataset('RunningVariables',data=np.array(RunningVariables))
    hf.close()


#ScatterPlot prob vs beta
figDetails=plt.figure(figsize=(8,6),dpi=600)
ax=plt.gca()
plt.gcf().subplots_adjust(bottom=0.12,left=0.16)
ax.plot(betaList,prob128,'o',color='teal')
ax.plot(betaList,prob123,'o',color='darkslateblue')
ax.plot(betaList,prob118,'o',color='orange')
ax.set_xlim(0,0.7)
plt.xlabel(r'$\beta$', **hfont)
plt.ylabel('Probability', **hfont)
# plt.bar(xListForPlot,HistAmplitudeNormalized,width=width,color='teal')
   #,edgecolor='black',facecolor='blue')
plt.savefig(f'ScatterPlotBetaProb{u}beta{Beta}.pdf',format="pdf",dpi=600)
plt.show()


hf=h5py.File(f'ChangeOverTime{w}{Beta}.h5', 'w')
RunningVariables=[N,E0,t,Iterations,Beta,u,width]
hf.create_dataset('betaList',data=betaList)
hf.create_dataset('prob118',data=prob118)
hf.create_dataset('prob123',data=prob123)
hf.create_dataset('prob128',data=prob128)
hf.create_dataset('RunningVariables',data=np.array(RunningVariables))
hf.close()

et = time.time()
print(f' Time {et-st}')
```