UTRECHT UNIVERSITY

BACHELOR THESIS

Example of all one can do with LaTeX

Authors:
Laurens STOOP
Studentnumber 3986209
Utrecht University
i.s.m.
TeXniCie A–Eskwadraat

Supervisors:
Andr'e MISCHKE
Utrecht University
and
Alessandro GRELLI
Utrecht University

15 June 2016

Abstract

In this numerical experiment the phase change behaviour of the two dimensional Ising model is simulated using the Monte Carlo algorithm, the critical temperature for this system is determent. To find the critical temperature the simulation is run for a number of temperatures around the theoretical value of $T_{c,theoretical} = 2.27 J/k_b$ from an initial randomised state. At the critical temperature the second derivative of the energy as function of temperature is zero. From this we found a value of $T_c = 2.40 J/k_b$.

Contents

Chapter 1

This is the first shite

1.1 Introduction

In this numerical experiment the behaviour of the Ising model is studied by carrying out Monte-Carlo simulations on a two state system in two dimensions.

To obtain insight in the behaviour of the Ising model a two dimensional square lattice is made. In this model every lattice site can be in one of two states. The lattice is modelled to be in thermal contact with a heat bath and has no external magnetic field applied to it. In order to implement the behaviour of the Ising model, the lattice sites interact with their nearest neighbours via the Ising model Hamiltonian. The phase behaviour is investigated by running Monte-Carlo simulations on the system. The underlying theory is further explained in section 1.2. The numerical model used is further explained in section 1.3 and a flow chart of the program is shown in figure 1.1. The results of our measurements are shown and discussed in section 1.4. The main conclusions are given in section 1.5.

Key insights in the Ising model and it's phase behaviour are expected to be obtained. As the 2D Ising model can exhibit a phase change at a certain temperature the main goal of the experiment is to find this critical temperature.

1.2 Theory

1.2.1 The Ising Model

For this experiment the Ising model in 2 dimensions is used. So all theory discussed will be for a 2D Ising model.

The Ising model can be imagined as array with on each site a two state particle which is in one of two states. Every particle on this lattice interacts with its nearest neighbours[1] in such a way that it costs less energy to be in the same state as its nearest neighbours. The system will thus save energy if a particle has the same state as its neighbours. The nearest neighbour interaction in the Ising model can be described by the Hamiltonian H in equation (1.2.1). In equation (1.2.1) the $S_i = \pm 1$ is the state of the particle at site i and $\sum_{\langle i,j \rangle}$ is the summation over the nearest neighbours of i.

$$\hat{H} = -J \sum_{\langle i,j \rangle} S_i S_j \tag{1.2.1}$$

A system in equilibrium will however not be necessarily be in one of the two ground states[2]. In the Ising model equilibrium is determined by minimising the the Helmholtz function $F = U - TS$, as the system is rigid. For low temperatures, the ordered states, the Helmholtz function is dominated by U and will thus minimise U. For high temperatures, disordered state of the system, the Helmholtz function is dominated by S and will thus maximise S.

Between the disordered and the ordered state the temperature is the mediator and will ultimately determine whether the minimisation of the energy or the maximisation of the entropy will be more dominant. The crossover between a ordered system and a disordered system will happen at a certain critical temperature T_c, as defined in equation (1.2.2), as derived by Onsager in 1944[3].

$$\frac{k_b T_c}{J} = \frac{2}{\ln(1+\sqrt{2})} \tag{1.2.2}$$

In equation (1.2.2) k_b is the Boltzmann factor and J the coupling constant.

The critical temperature can be numerically obtained in two different ways, both methods rely on Onsager's analytical solutions of the Ising model[4]. The first method relies on the dependence of the heat capacity C on the temperature T. Near the critical temperature the heat capacity per particle has the form of equation (1.2.3), with α a constant.

$$C \sim |T - T_c|^{-\alpha} \tag{1.2.3}$$

The second method relies on the dependence of the average magnetisation on the temperature. Near the critical temperature the average magnetisation $\langle M \rangle$ takes the form of equation (1.2.4), with β a constant. It is important to note that the average magnetisation for $T > T_c$ is zero as the system is then disordered.

$$\langle M \rangle \sim (T - T_c)^{\beta}, \quad T < T_c \tag{1.2.4}$$

Further reading concerning the theory of this experiment can be found in *Advanced Statistical Physics* [3], *Concepts of thermal physics* [1], *Introduction to Monte-Carlo methods for an Ising Model of a Ferromagnet* [4] and *Monte-Carlo simulations of Spin Systems* *wolfhard*.

[1]The nearest neighbour interaction at the borders can be achieved by viewing the two dimensional array as a three dimensional topological shape of a torus.

[2]A system in its ground state is a system in which every particle has the same state (in this case; all the spins are the same).

[3]His deviation can be found in *A Two-Dimensional Model with an Order-Disorder Transition* [2]

[4]The derivation of equation (1.2.3) and (1.2.4) is long and beyond the scope of this paper.

1.2.2 Monte-Carlo simulations

To implement the Ising model in a numerical program one has to consider the fact that the distinct number of configurations for the Ising model on a $N \times N$ lattice is 2^{N^2}, this number grows very fast with N. A way around this is to use the Monte-Carlo method, this method can be used as the real system takes a particular configuration and will then fluctuate within the configuration space as defined by the 2^{N^2} states. The Monte-Carlo method relies on repeated random sampling to go through the configuration space and wiil indirectly obtain in this manner the probability distribution of the system.

From the Monte-Carlo method the Metropolis algorithm is used as it can be easily implemented for the Ising model. For the Ising model this algorithm takes a random particle and flips it. If this lowers the energy of the system then you accept the new state of the system. Otherwise you leave the state flipped with the probability $1 - e^{-\beta \Delta E}$. This process is repeated until equilibrium.

Two things are important to notice about the Metropolis algorithm. Firstly for very low temperatures the probability to flip a spin while that flip costs energy will become zero as the $e^{-\beta \Delta E}$ factor will rise to 1, the system will thus always change towards the ground state. Secondly, at high temperature the $e^{-\beta \Delta E}$ factor will fall to 0, thus the system will always accept the flipped spin state and the algorithm will drive the system to a high entropy state.

Further reading concerning the algorithms of this experiment can be found in *Concepts of thermal physics* [1], *Introduction to Monte-Carlo methods for an Ising Model of a Ferromagnet* [4] and *Monte-Carlo simulations of Spin Systems wolfhard.*

1.3 Numerical model

To research the behaviour of the Ising model we have implemented a Monte-Carlo simulation for the Ising model with the use of the Metropolis algorithm in `Python`. The processes of the program are shown schematically in figure 1.1 as a flow chart. The full program is included in appendix .1 as listing 1 and is made under the GNU General Public License version 3.

The program was developed with flexibility in mind and takes a whole range of input when it is executed. The keyboard input that the program requests is

`n`	the number of rows of the array of the system
`m`	the number of columns of the array
`imax`	the number of iterations for this simulation run
`iequil`	the expected number of iterations needed to reach equilibrium
`T`	the temperature of the heat bath to which the system is connected
`initial system`	the initial system can be a random system (0), all up (1) or all down(-1)
`imageseq`	to better track the change in the system, several images can be made during the iterations

After having obtained the input the program will do the initialisation off the the system and its borders. In this initialisation the a $n + 2 \times m + 2$ array is made full of zero's. All the points that do not lie on the border[5] are then replaced by a ± 1 depending on the choice for a ground state or a disordered system. The initialisation of the system is finished by calculating the energy, average energy, magnetisation and average magnetisation and writing the state of the system to a file.

At this point in the program the Monte Carlo simulation starts by correcting the periodic boundaries or border as the last edge on the other side of the system and by changing a one random point of the array by flipping its state. The change in energy is then calculated and the system checks if the change is allowed. At this point a plot is made of the state of the system if the iteration number is a one-hundredth of the maximum number of iterations.

If the change is not allowed, the point if flipped back into it's original state. If the change is allowed the new state of the point is set in the array. If the number of iterations is lower then the expected iteration for a equilibrium state of the system the next iteration of the Monte Carlo simulation is run. If the number of iterations is higher then the expected iteration for equilibrium state, then the heat capacity, the energy, average energy, magnetisation and average magnetisation are calculated and written to a file. If the number of iterations used is lower then the maximum number of iterations then the Monte Carlo loop is run again, if it is lower the final state written to a file and the program is done.

In the numerical model and program natural unites are used and the coupling constant is kept at 1.

[5]The border is defined as the collection of all the particles that are the first or the last of any row or column.

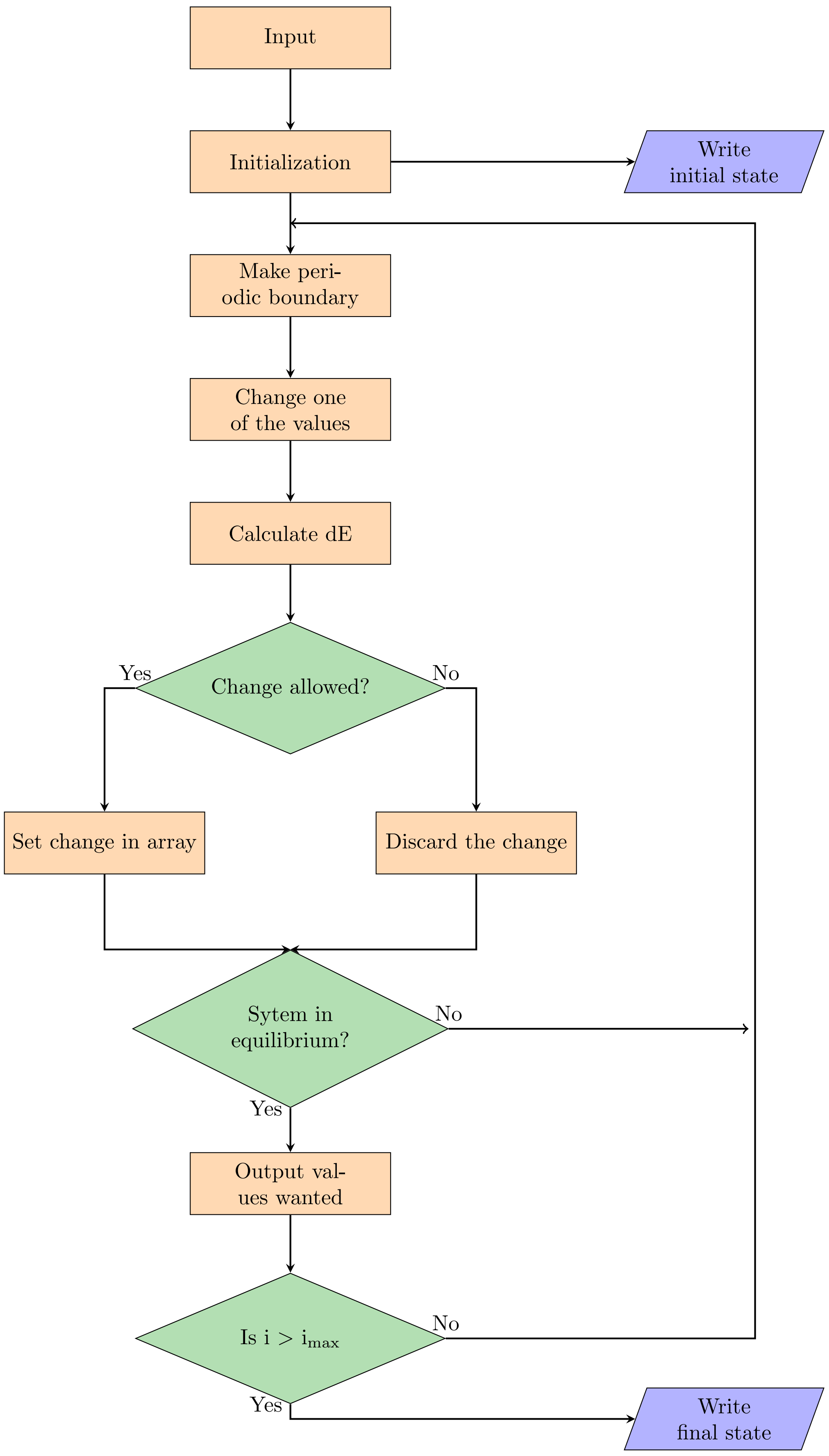

Figure 1.1 – In this flow chart the processes of the program are shown, for clarity the process that creates pictures of the array is omitted.

1.4 Results and discussion

1.4.1 Results

The program that was developed was analysed for reduced temperatures in the range from 0 to 5 in steps of 0.1 reduced temperature. At every temperature a system of 256 by 256 locations was subjected to the program with a required number of iterations of 10^7. For the last 10^6 iterations the system was taken to be in equilibrium and was thus sampled. All initial systems where taken to be in a random state at the start and no image sequences where made.

From the raw data figures where made after the last iterations of the program to show the state in which the system is. These were used to allow a quick check of the data before processing commenced. Three of the final states are shown in figure 1.2 to show the difference in the final states of the system.

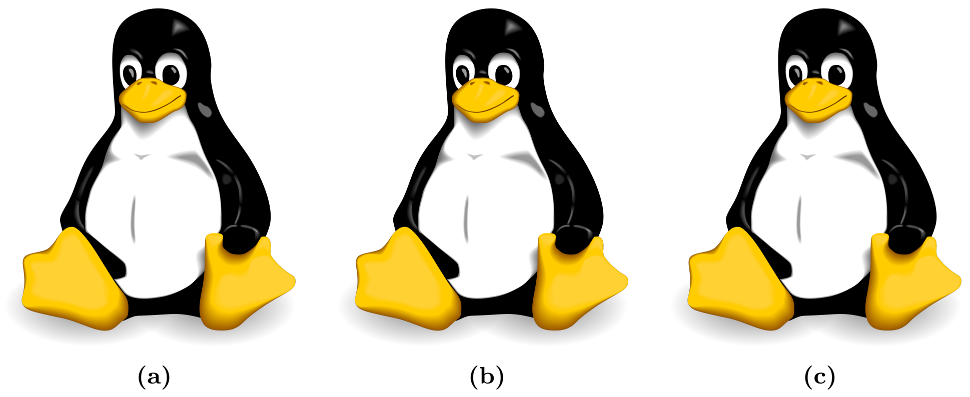

Figure 1.2 – The final states of different systems after many iterations. For a $T < T_c$ (1.2a) the systems goes to the ground state where all elements have the same value. At $T \approx T_c$ (1.2b) the system shows clustering or small range correlations between the values. For $T \gg T_c$ (1.2c) the systems stays randomized.

For each of the 51 temperature samples the obtained data was analysed with the help of a Mathematica notebook and the theoretical models, from section 1.2, where fitted to the data. The found model is plotted together with the data points for each temperature in figure 1.3.

1.4.2 Discussion

The value numerically found for the critical temperature ($T_{c,numerical} = 2.40\ J/k_b$) is very close to the theoretical value ($T_{c,theoretical} = 2.27\ J/k_b$). Due to limited usability of the Mathematica fit program no error's could be obtained for the found critical value, limited time restrained us in further attempts on the error estimation.

The difference in these values can be explained as an numerical error. As with all numerical experiments a limiting factor is the precision the model can achieve. This error is in the calculation of the values of energy and magnetisation, but more so in the randomisation algorithm used. The model is written in Python and for the randomisation a algorithm from the numpy package was used [6]. This algorithm uses the normal distribution for determining random values but as every random algorithm it has a limited randomisation. This means that the permutations that not give a lower energy state to the system don't have a completely random change of being allowed.

As seen in 1.3b there is a large difference between the theoretical and measured form of the heat capacity for $T \leq T_c$. At $T \approx T_c$ there are to few data point to give a clear view of this difference and more measurements are needed to give a conclusive difference. For $T < 0.75$ the values of C_v are all nearly zero. As all simulations went to a complete system in the ground state this is to be expected. But for $0.75 < T < T_c$ the values

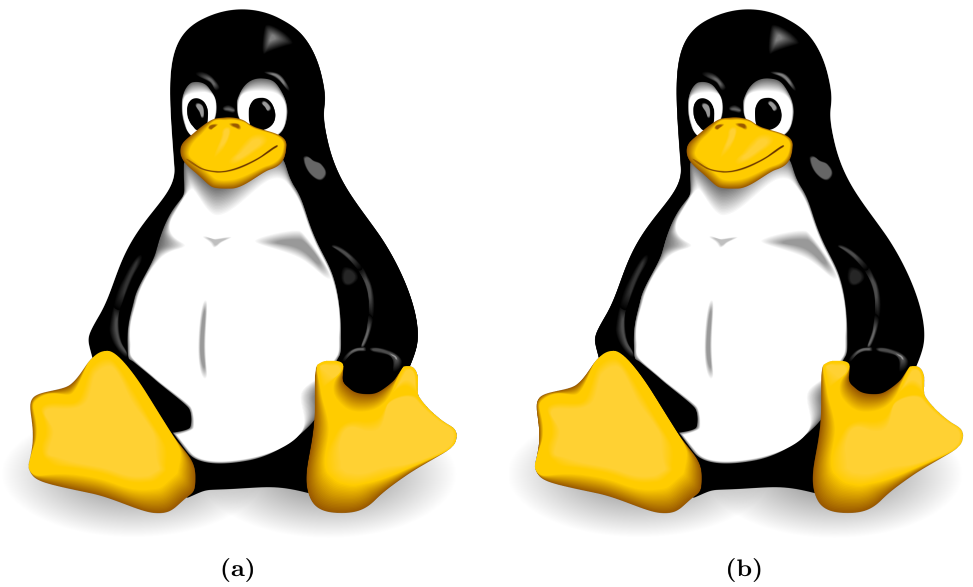

Figure 1.3 – The average energy (figure 1.3a) and the average heat capacity (figure 1.3b) of the system at every temperature step is shown. Through this the theoretical models, shown in red, are fitted as they are given in section 1.2. From the average energy model the found critical temperature is $T_{c,numerical} = 2.40\ J/k_b$.

for C_v show a more chaotic behaviour. This indicates the system was not in equilibrium or the sampling was biased. So a longer calculation time is needed to more accurately determine C_v. For our experiment the computational power and time where limited so this was not an option.

Due to the excessive computation time needed to load and analyse the 10^6 data points of every temperature, only the last 10^5 data points of every temperature where used. This should not have an impact in our obtained models as the system is in equilibrium before the system is sampled, as the system should be in equilibrium and should not change on average anymore.

1.5 Conclusions

From our results it can be seen that the Ising model in two dimensions has a very specific critical temperature where a phase change happens. The value determent by a numerical computation using the Monte Carlo simulation algorithm gives us a value of $T_{c,MonteCarlo} = 2.40J/k_b$ where the theoretical value determent using a.o. renormalisation group theory is $T_{c,theoretical} = 2.27J/k_b$ [3].

The difference in the found values and can be explained by numerical errors and limitations on computational power and time.

From this experiment it is clear that a Monte Carlo simulation is a good way to simulate simple physical systems. But the amount of calculations scales with the size and complexity of the system. Making it a not ideal candidate for more complicated system. It's however a good starting point for developing more sophisticated models. And can be expended upon really easily.

.1 The Dries Guide

According to Dries you can write any thesis by following these steps:

1. Make a Table of Contents
    - With the sections titles

   - With the bullit point conclusions

2. Propose supporting graphs
   - Place these in the sections
   - Correlate to the supporting data sets

3. Make these figures
   - Adopts your section conclusions to reflect the thruth of the figures

4. Whack text around it all
   - Maybe add some schematics of stuff

And that's all folks!

Bibliography

[1] S.J. Blundell and K.M. Blundell, *Concepts in Thermal Physics*, Oxford University Press, Oxford, 2006.

[2] L. Onsager, *A Two-Dimensional Model with an Order-Disorder Transition*, Phys.Rev 65 117, 1944. http://dx.doi.org/10.1103/PhysRev.65.117

[3] L. Filion and R. van Roij, *Advanced statistical physics*, Lecture notes of the "Voortgezette statistische fysica" course of 2015 at Utrecht University.

[4] J. Kotze, *Introduction to Monte Carlo methods for an Ising Model of a Ferro Magnet*, March 2008.
http://arxiv.org/abs/0803.0217v1

[5] W.Janke, *Monte Carlo Simulations of Spin Systems*, Computational Physics, 1996 -Springer
http://www.physik.uni-leipzig.de/~janke/Paper/spinmc.pdf

[6] NumPy Developers. *Numpy*
http://www.numpy.org/

Python

```
#################################################################################################
#                                               #
# A model simulating a 2D Ising Model                              #
#                                               #
# This software is made under the GNU General Public License, version 3 (GPL-3.0)        #
#                                               #
#################################################################################################

# import the packages needed
import numpy as np
import random as random
import math as math

# Seed the random generator to make sure we can compare the system during building and testsing
random.seed(123456789)

# For now ask for keyboard input, can be rewritten to use a file
def Input():
  # Some variables are used by multiple functions, instead of returning global can be used
  global n
  global m
  global imax
  global beta
  # the number of particles can be set and is not bound by restrictions. So two different values are taken
  print("Give␣n")
  n = int(raw_input())
  print("Give␣m")
  m = int(raw_input())
  # there is a maximal number of itterations
  print("Give␣imax")
  imax = int(raw_input())
  # for the boltzmann distribution T is needed
  print("Give␣T")
  T = float(raw_input())
  # put in the minus here, because not sure if the if takes it
  beta = -1/T

# a function to make the initial system A. For now the initial system is random with 50/50 distribution
# The array has no values on the edges, so the system can be ...
def Initialize():
  global A
  # makes a array full of A's
  A = np.zeros((n+2, m+2))
  # replace the zero's by +- 1
  for i in range(1, n+1):
    for j in range(1, m+1):
      if random.random() > 0.5:
        A[i][j] = 1
      else: A[i][j] = -1

# A function to set the borders. These are the same as the last edge on the other side
def CorrectBorders():
  for i in range(1, n+1):
    A[i][0] = A[i][m]
  for i in range(1, n+1):
    A[i][m+1] = A[i][1]
  for i in range(1, m+1):
    A[0][i] = A[n][i]
  for i in range(1, m+1):
    A[n+1][i] = A[1][i]

# The monte carlo function. Is choses one of the particles and changes it.
def ChangeRandom():
  global nchange
  global mchange
  nchange = random.randint(1, n)
  mchange = random.randint(1, m)
  A[nchange,mchange] = -A[nchange,mchange]

# For now the output is given by printing the array to the screen
def PrintA():
  for value in A:
    print(value)

# A function to calculate the change in Energy. So the energy is +1 if the new value is different from it's neighbour
def CalculatedE():
  dE = - A[nchange][mchange] * (A[nchange-1][mchange]+A[nchange+1][mchange]+A[nchange][mchange-1]+A[nchange][mchange+1])
  return dE

# See if the change is allowed by the the boltzmann distribution. No sure if it's correct. Some pass some don't
def CheckChange(dE):
  # the minus is in beta
  return beta*dE > math.log(random.random())


# Get Input from the user\a file
Input()
# Based on the input make the initial state of the system
Initialize()
# Make the borders correct
CorrectBorders()
# Write the initial system to an output file
PrintA()
#Start the itteration
for i in range(imax):
  # Change on of the values in the sample
  ChangeRandom()
  # If the changed particle is on the edge of the sample, the borders have to be redrawn.
  if nchange == 1 or nchange == n or mchange == 1 or mchange == m:
  # Checked to work with nchange and mchange in the corners
    CorrectBorders()
  print(nchange, mchange)
  PrintA()
  # Calculate the energy change due to the change
  dE = CalculatedE()
  print("dE␣=", dE)
  # Check if the change is allowed if not discard the change and do the next itteration
  if not CheckChange(dE):
    print("Nope")
  else:
    print("Yeah")
  print("\n")
```

```




#To do
  # Check if the system is in equillibrium if not do next itteration
  # Write out the wanted output for this itteration
  # See if done with the itteration, else do next itteration
  # Write away the output
```

Listing 1 – The python program used

Math in depth

.2 What you will find

In the coming few pages you will find quite a few examples of very nice and fancy math. The included math is taken as part from Aldo de Witte's bachelor thesis and showcases what is possible within LaTeX.

.3 Cochain complexes

Cohomology arises naturally from cochain complexes. In this section we will discuss cochain complexes of K-modules and discuss the K-modules which arise from them. We will call these modules cohomology modules associated to the cochain complex. These modules will often coincide with the cohomology modules of a cohomology theory as we will define in the next section.

Definition .3.1. A *cochain complex* U^* is a sequence of K-modules and homomorphisms

$$\cdots \to U^{q-1} \to U^q \to U^{q+1} \to \cdots$$

defined for all integers q such that at each stage the image of a given homomorphism is contained in the kernel of the next. We will refer to the homomorphisms $U^q \to U^{q+1}$ as d^q, or d if this does not cause any confusion. The fact that the image of a homomorphism is contained in the kernel of the next can be written as $d^{q+1} \circ d^q = 0$, or more concisely $d^2 = 0$. We call d^q the q-th *coboundary operator*. We will denote the kernel of d^q by $Z^q(U^*)$ and call its elements the q-th degree *cocycles* of the cochain complex U^*. We will denote the image of d^{q-1} by $B^q(U^*)$ and call its elements the q-th degree *coboundaries*. ♢

Using the fact that $d^2 = 0$, $B^q(U^*)$ is a subset of $Z^q(U^*)$ for all q.

Definition .3.2. The q-th *cohomology module* $H^q(U^*)$ associated to the cochain complex U^* is defined as the quotient module

$$H^q(U^*) = \frac{Z^q(U^*)}{B^q(U^*)}$$ ♢

Proposition .3.3. *Given a short exact sequence of cochain maps*

$$0 \to U^* \to V^* \to W^* \to 0,$$

consisting of short exact sequences

$$0 \longrightarrow U^q \xrightarrow{f^q} V^q \xrightarrow{g^q} W^q \longrightarrow 0,$$

there are homomorphisms

$$H^q(W^*) \xrightarrow{\delta} H^{q+1}(U^*)$$

for each q such that the following sequence is exact.

$$\cdots \longrightarrow H^{q-1}(W^*) \xrightarrow{\delta} H^q(U^*) \xrightarrow{f^*} H^q(V^*) \xrightarrow{g^*} H^q(W^*) \xrightarrow{\delta} H^{q+1}(U^*) \to \dots \quad (.3.1)$$

$$\begin{tikzcd}
 & \vdots & \vdots & \vdots & \\
0 \arrow[r] & U^{q+1} \arrow[u] \arrow[r, "f^{q+1}"] & V^{q+1} \arrow[u] \arrow[r] & W^{q+1} \arrow[u] \arrow[r] & 0 \\
0 \arrow[r] & U^q \arrow[u] \arrow[r, "f^q"] & V^q \arrow[u] \arrow[r, "g^q"] & W^q \arrow[u] \arrow[r] & 0 \\
0 \arrow[r] & U^{q-1} \arrow[u] \arrow[r] & V^{q-1} \arrow[u] \arrow[r, "g^{q-1}"] & W^{q-1} \arrow[u] \arrow[r] & 0 \\
 & \vdots \arrow[u] & \vdots \arrow[u] & \vdots \arrow[u] &
\end{tikzcd} \quad (.3.2)$$

Proof. We will define $\delta : H^q(W^*) \to H^{q+1}(U^*)$ by chasing over the above diagram. Let $w \in W^q$ be a cocycle. Since g is surjective, let $v \in V^q$ be such that $g(v) = w$. Then $0 = \partial_W w = d_W g(v) = g(d_V v)$, that is, $d_V v \in \ker(g) = \mathrm{im}(f)$. Hence there is a $u \in U^{q+1}$ such that $f(u) = d_V v$. Because $f(u) = d_V v$ we see that $d_V \circ f(u) = d_V^2 v = 0$. Using the fact that f is a cochain map we conclude that $f(d_U u) = 0$. Because f is injective $d_U u = 0$, hence u represents an equivalence class in $H^{q+1}(U^*)$ and we define $\delta[w] = [u]$. We will show that δ is well-defined. Take $w, w' \in W^q$ such that $[w] = [w']$. Now with the same reasoning as before choose $v, v' \in V^q$ such that $g(v) = w$ and $g(v') = w'$. Now consider $d_V v, d_V v' \in V^{k+1}$. We have $g(d_V v) = d_W g(v) = 0$ and the same for v', thus both are in the kernel of g and thus in the image of f. Hence we can find $u, u' \in U^{k+1}$ such that $f(u) = d_V v$ and $f(u') = d_V v'$. We conclude that $\delta[w - w'] = [u - u']$. To show that δ is well-defined we are left to show that $u - u'$ is a coboundary.
Now consider $w - w' = g(v) - g(v')$. By assumption this in the image of d_W and thus there is a $\underline{w} \in W^{q-1}$ such that $d_W \underline{w} = g(v) - g(v')$. By surjectivity of g there is a $\underline{v} \in V^{q-1}$ such that $g(\underline{v}) = \underline{w}$. Now we consider $v - v' - d_V \underline{v}$, then

$$g(v - v' - d_V \underline{v}) = g(v) - g(v') - g(v) + g(v') = 0.$$

Therefore $v - v' - d_V \underline{v}$ is in the kernel of g hence also in the image of f. Therefore there exists a $\tilde{u} \in U^k$ such that $f(\tilde{u}) = v - v' - d_V \underline{v}$. Now

$$d_V f(\tilde{u}) = d_V(f(v) - (v') - d_V f(\underline{v})) = f(u - u').$$

Therefore $f(d_U \tilde{u}) = f(u - u')$, and since f is injective we have that $d_U \tilde{u} = u - u'$. Hence $u - u' \in \mathrm{im} d_U$ consequently $[u - u'] = 0$. We therefore conclude that δ is well-defined.
We will now show that with this definition of δ the sequence (.3.1) is indeed exact. We denote by f^* and g^* maps in cohomology classes induced form f and g.
Proof of $\ker(g^*) = \mathrm{im}(f^*)$: Let $[h] \in \mathrm{Im}(f^*)$ hence there is a $[\tilde{h}]$ such that $f^*[\tilde{h}] = h$. Now $g^*[h] = g^* \circ f^*[\tilde{h}] = [(g \circ f)h] = [0]$, hence we have that $[h] \in \ker(g^*)$.
Let $[h] \in \ker(g^*)$, thus $[g(h)] = [0]$. This implies that $g(h) \in \mathrm{im}(d_W)$, which implies that there exists a $w \in W^{q-1}$ such that $d_W w = g(h)$. Because g is surjective there exists a $v \in V^{k-1}$ such that $g(v) = w$. Then $d_V v$ is again an element in V^q which is in $\mathrm{im}(d_V)$ hence we have that $[h - d_V v] = [h]$. By applying g on $h - d_V v$ we get that

$$\begin{aligned} g(h - d_V v) &= g(h) - g(d_V v) \\ &= g(h) - d_W g(v) = g(h) - d_W w = g(h) - g(h) = 0. \end{aligned}$$

We thus conclude that $h - d_V v$ is in the kernel of g, from which we in turn conclude that $h - d_V v$ is in the image of f. Hence there exists a $u \in U^q$ such that $f(u) = h - d_V v$. Then $f^*[u] = [f(u)] = [h - d_V v]$. But as $d_V v$ is in the image of d_V it follows that $f^*[u] = [h]$. We thus conclude that $[h] \in \mathrm{im}(f)$.
Proof of $\mathrm{im}(\delta) = \ker(f^*)$: Let $[h] \in \ker(f^*)$ since f is injective it has trivial kernel and it follows that $f(h) \in \mathrm{im}(d_V)$. Therefore there is a $v \in V^{q-1}$ such that $d_V v = f(h)$. Now we consider $g(v) \in W^{q-1}$. Following the definition of δ we have that $g(d_V v) = d_W g(v) = g(f(h)) = 0$, and thus that $d_V v \in \ker(g)$ and hence we conclude that $d_V v \in \mathrm{im}(f)$. By construction $f(h) = d_V v$, hence $[h] = \delta[g(v)]$ and thus we conclude that $[h] \in \mathrm{im}(\delta)$.
Let $[h] \in \mathrm{im}(\delta)$. Then there exists a $w \in W^{q-1}$, such that $\delta[w] = [h]$. Again we will follow the definition of δ. Because g is surjective there exists a $v \in V^{q-1}$ such that $g(v) = w$. Now $d_V v$ is in the kernel of g, which implies that $g(d_V v) = d_W g(v) = d_W w = 0$. Hence there is an h in U^q such that $f(h) = d_V v$. It follows directly that $f(h)$ is in the image of d_V which than implies that $[h]$ is in the kernel of f^*.
Proof of $\mathrm{im}(g^*) = \ker(\delta)$:
Let $[w] \in \mathrm{im}(g^*)$, hence there is a $[v] \in H^q(V)$ such that $g^*[v] = [w]$. Because v is a cocycle we have that $d_V v = 0$. Then as before we find u such that $f(u) = d_V v$. But as $d_V v = 0$ and f is injective it follows that $u = 0$ and thus that $\delta[g(v)] = [0]$. Which implies that $[w] \in \ker(\delta)$.
Let $[w] \in \ker(\delta)$, hence $\delta[w] = [u] = [0]$, and thus there exists a $\tilde{u} \in U^k$ such that $d_U \tilde{u} = u$. As before $f(u) = d_V v$, with $g(v) = w$.
For $x = v - f(\tilde{u})$,

$$d_V x = d_V v - d_V f(\tilde{u}) = f(u) - f(u) = 0.$$

Therefore x is a cocycle and thus represents an equivalence class in $H^q(V)$. Now $g^*[x] = [g(v) - g(f(\tilde{u}))] = [w]$. Hence $[w] \in \text{Im } g^*$, which concludes the proof that sequence (.3.1) is exact. $\square$

This definition of δ also gives rise to another commutative diagram which will be key to showing the existence of a cohomology theory.

Proposition .3.4. *The homomorphisms (??) of short exact sequences of cochain complexes together with the map δ give rise to a commutative diagram:*

$$\begin{tikzcd}
H^q(W^*) \arrow[r, "\delta"] \arrow[d] & H^{q+1}(U^*) \arrow[d] \\
H^q(\bar{W}^*) \arrow[r, "\bar{\delta}"] & H^q(\bar{U}^*)
\end{tikzcd} \tag{.3.3}$$

Proof. Denote by $\bar{f}^q$ the cochain map $U^q \to V^q$ and by $\bar{g}^q$ the cochain map $V^q \to W^q$. To show that diagram (.3.3) commutes we first construct a commutative lattice, that is, a three-dimensional commutative diagram for which all the sides commute:

$$\begin{tikzcd}
& U^{q+1} \arrow[rr] \arrow[dl] & & V^{q+1} \arrow[rr] \arrow[dl] & & W^{q+1} \arrow[dl] \\
\overline{U}^{q+1} \arrow[rr] & & \overline{V}^{q+1} \arrow[rr] & & \overline{W}^{q+1} & \\
& U^q \arrow[uu, dotted] \arrow[rr, dotted] \arrow[dl, dotted] & & V^q \arrow[uu, dotted] \arrow[rr, dotted] \arrow[dl, dotted] & & W^q \arrow[uu] \arrow[dl] \\
\overline{U}^q \arrow[uu] \arrow[rr] & & \overline{V}^q \arrow[uu] \arrow[rr] & & \overline{W}^q &
\end{tikzcd}$$

The horizontal segments of this diagram commute by the fact that we have a homomorphism between short exact sequences, and the vertical segments by the fact that the maps $U^* \to \bar{U}^*$,$V^* \to \bar{V}^*$ and $W^* \to \bar{W}^*$ are cochain maps.
We first describe the composition $H^q(W^*) \to H^{q+1}(U^*) \to H^{q+1}(\overline{U}^*)$ which sends $w \in W^q$ to some $\bar{u}$ in $\bar{U}^{q+1}$. Let $w \in W^q$. By surjectivity of g^q there exists $v \in V^q$ such that $g(v) = w$, we then lift v to $d_V v$. As we have shown in the definition of δ, there exists a $u \in U^{q+1}$ such that $g(u) = d_V v$, hence $\delta(w) = u$, and finally we map u to $\overline{u} \in \overline{U}^{q+1}$.
We will now describe the composition $H^q(W^*) \to H^q(\overline{W}^*) \to H^{q+1}(\overline{U}^*)$ which sends $w \in W^q$ to some $\bar{u}'$ in $\bar{U}^{q+1}$. Let $\bar{w}'$ be the image of w under $W^q \to \overline{W}^q$. Then by surjectivity of $\bar{g}^q$ there exists a $\overline{v}' \in V^q$ such that $\bar{g}(\overline{v}') = \bar{w}'$. We then lift $\overline{v}'$ to $d_{\overline{V}} \overline{v}'$, and as we have shown in the definition of $\bar{\delta}$ there exists $\bar{u}' \in U^{q+1}$ such that $\bar{g}(\bar{u}') = d_{\overline{V}} \overline{v}'$, and we had that $\delta(\bar{w}) = \bar{u}'$.
To show that diagram (.3.3) commutes we are left to show that $\bar{u}' - \bar{u}$ is a coboundary. If we map v to $\overline{v} \in \overline{V}^q$ then by commutativity of the diagram we see that $\overline{v}$ and $\overline{v}'$ both get mapped into $\overline{w}'$, i.e. $\bar{g}(\overline{v} - \overline{v}') = 0$. Thus there exists a $\overline{\bar{u}} \in \overline{U}^q$ such that $\bar{f}(\overline{\bar{u}}) = \overline{v} - \overline{v}'$. We will show that $d_U \overline{\bar{u}} = \overline{u} - \overline{u}'$. To do this we need that $d_{\overline{V}} \overline{v} = \bar{f}(\bar{u})$ which follows directly form the definition of $\bar{\delta}$. We then have that

$$\bar{f}(d_{\overline{U}} \overline{\bar{u}}) = d_{\overline{V}} \bar{f}(\overline{\bar{u}}) = d_{\overline{V}} \overline{v} - d_{\overline{V}} \overline{v}' = \bar{f}(\bar{u}) - \bar{f}(\bar{u}') = \bar{f}(\bar{u} - \bar{u}'),$$

and thus by injectivity of $\bar{f}$ we conclude that $d_U \overline{\bar{u}} = \overline{u} - \overline{u}'$. This completes the proof that diagram (.3.3) commutes. $\square$