Abstract

Fractals seem to appear everywhere around us. They often appear in nature and are seen in things like treetops, river networks and even animal colouration patterns.

The goal for this project was to explore fractals in a specific mathematical model, namely the Ising model.

We created a simulation of the Ising model, estimated the critical temperature, examined the characteristics of the fractal that appears at the critical temperature, and determined the dimensionality of this fractal along with a fractal from a third party.

We found that the Ising model has clear fractal characteristics at the critical temperature, which we estimated to be between 2.2 and 2.5.

We calculated the fractal dimensionality in the Ising model to be $D = 1.87 \pm 0.02$ and $D = 1.66 \pm 0.03$ in the third party fractal.
1 Introduction

For centuries fractals have made people wonder, and they are found all around us. In nature they can appear as romanesco broccoli, coastlines or snowflakes. Fractals have been found in laboratories and in mathematical models as well. This report will look at fractals as they appear in the mathematical model, the Ising model. We wish to find the fractal dimension for a simulated fractal when the Ising model reach the critical temperature $T_c$. We will in the report use an experimentally made fractals from a third party, calculate fractal dimension and compare with their result.

2 Theory

2.1 Fractals

2.1.1 Definition

Fractals are complex geometric shapes with certain characteristics: [1]

1 Scale invariance. Magnifying a fractal reveals new levels of complexity, we can as such never reach the "end" of a fractal, but will be able to explore the infinite depths revealed by magnification. Scale invariance is only a characteristic of mathematical fractals, and not of experimental or simulated fractals.

2 Self similarity. Self similarity means that it’s impossible to distinguish the level of magnification, one is observing. The different levels of magnification will appear similar, and in fact completely identical for the most simple fractals.

3 Non integer dimensionality. This simply means we get a non integer result, when we calculate the dimensionality of a fractal. Exactly how fractal dimensionality can be calculated will be explored in 2.1.2.

An example of a mathematical fractal is the Koch Curve

![Koch Curve](image)

(a) First Iteration  (b) Second Iteration

(c) Third Iteration  (d) Sixth Iteration

Figure 1: The evolution of the Koch curve, starts with a flat line. In each iteration the middle third of every straight line is transformed into a triangle. As we started with a flat line, the first iteration transforms one line (a), the second iteration starts with four straight lines from the first iteration and transforms the middle third of these (b). The Koch curve continues like this and from the curve’s self similarity and scale invariance we see a fractal.

The Koch curve is generated recursively, by starting with a flat line segment, and transforming
the middle third of every straight line into a triangle for each iteration, see Figure 1.
If we perform an infinite number of recursive iterations on the Koch curve, we can easily imagine
how the resulting shape, would display the first characteristic. When we magnify the infinite Koch
curve, we will see the same amount of complexity on all levels, since we have raised an infinite
number of middle thirds.
We see how the Koch curve displays the second characteristic, since we are performing the same
action, for all iterations, on all levels of magnification, resulting in a shape that is exactly similar.
Determining the third characteristic is however not so easy. We could look up the dimensionality
of the Koch curve and see that it is \( \frac{\log(4)}{\log(3)} \approx 1.26 \) [1], but it would be far more interesting to explore
how we can calculate the dimensionality of a geometric figure.

2.1.2 Similarity Dimensionality
Before delving into the calculation of dimensionality for fractals, we will explore the meaning of
the term dimensionality for some simple geometric shapes.
We start by looking at how we can measure the length of a straight line of length 1.
If we use copies of the original line scaled by a factor of 2, we will use 2 copies, to cover the original
area. If we use copies scaled by a factor of 3 we will use 3 copies. In general we see that we need
\( m \) copies of scale \( r \) to cover our original line, and in the case of the line \( m = r \).
If we add a dimension and look at the square in the plane. We will need 4 copies to cover the
original square, if we scale by a factor of 2, and 9 copies if we scale the side length by a factor of
3. We see that \( m = r^2 \).
So what does this mean for dimensionality? We know that lines are one dimensional, and that
squares are two dimensional, and from this we could propose that \( m = r^d \), where \( d \) is the dimen-
sionality of our figure. With some mathematical operations we get.

\[
d = \frac{\ln(m)}{\ln(r)} \quad (1)
\]
[1].
A given iteration of the Koch curve is made of 4 identical (albeit rotated) sections, each of these
being scaled down by a factor of three compared to the original. We get \( m = 4 \) and \( r = 3 \) \( \Rightarrow 
\)
\( d = \frac{\ln(4)}{\ln(3)} \approx 1.26 \) [1].
While this method for calculation of dimensionality is rather elegant, it unfortunately only applies
to fractals with complete self similarity. We will therefore need a different method.

2.1.3 Box Count
To define an alternate method for calculating dimensionality we will still make use of equation 1,
we however now wish to measure our geometric figure \( F \) at a certain scale \( \delta \). The measurement of
\( F \) at scale \( \delta \) is denoted as \( M_\delta(F) \). We then have the relation:

\[
M_\delta(F) \sim c\delta^{-d} \quad (2)
\]
With \( d \) being the dimensionality of \( F \) and \( c \) being the "length" of \( F \) calculated in \( d \) dimensions.
This equation, which is very similar to equation 1, is called the power law [2]. We apply the
logarithm function and get:

\[
\log(M_\delta(F)) \sim \log(c) - \log(\delta)d \quad (3)
\]
If we let \( \delta \to 0 \) we get:

\[
d = \lim_{\delta \to 0} \frac{\log(M_\delta(F))}{-\log(\delta)} \quad (4)
\]
A property of the power law, is that the dimensionality can be calculated as the slope of the plot \( \log(\delta), \log(M_\delta(F)) \) \[^2\] and it is this property we use to calculate dimensionality using box count.

Falconer mentions five different ways of performing the boxcount\[^2\], but we shall only look at one of them, namely the mesh box method.

To perform this boxcount, we overlay our figure \( F \), with a mesh of boxes with side length \( \delta \). We then count the number of boxes which contain parts of our figure \( M_\delta(F) \). And perform a fit on the plot of \( \log(\delta), \log(M_\delta(F)) \), to get our dimensionality.

### 2.2 The Ising Model

The Ising model is a mathematical model that can be of ferromagnetism consisting variables that represent magnetic dipole moments of atomic spins, all put into a lattice, where the atomic spins can be in one of two states \( +1 \) or \( -1 \), called spin up and spin down, as shown in Figure 2.

The Ising model is a way of showing phase transition.

#### 2.2.1 Phase Transition

A phase transition can be of different orders. The phase transitions of \( H_2O \) is something that is easily related to. When \( H_2O \) goes from solid to liquid and liquid to gas, phase transitions are happening. Another thing that is easily related to are metals. In metals the phase transition can be at the temperature where the metal goes from being conductive to superconductive. This specific temperature is called the critical temperature \( T_c \).

First-order phase transition is when there is a jump in the entropy and a second-order phase transition is when the entropy is continuous. The phase transitions of \( H_2O \) are of first order and the phase transition of the metal is of second order. Our system of atomic spins will make a second-order phase transition at the critical temperature.

![Ising Model Lattice](image)

Figure 2: An example of a lattice with spin up and spin down. Each spin can only interact with its neighbours. How the spins interact with their neighbours depends on the Hamiltonian (5). Image courtesy: \[8\]

#### 2.2.2 Spin State and Thermodynamics in the Ising Model

When needed, the atomic spins will be pictured as their state of spin (up or down) and put into lattices. This can be 1 dimensional, 2 dimensional, ..., n dimensional. If it is 1 dimensional, the atomic spins will be set as a string, in a 2 dimensional system the atomic spin will be set as a lattice, just as an ordinary graph and so on. The chosen number of dimensions will make a difference in the calculation for the atomic spins, when each look at their neighbours during each timestep.
The number of neighbours for each atomic spin will always be $2n$, where $n$ is the dimension. Each atomic spin in a 1 dimensional lattice will have 2 neighbours, 4 neighbours for a 2 dimensional lattice, 6 neighbours for a 3 dimensional lattice and so on, as shown in Figure 3. For each spin it is energetically favorable to be of the same state as their neighbours. At each timestep each atomic spin look at their neighbours, and depending on the neighbours states of spin and the energy of the atomic spin, the atomic spin might flip (change state of spin).

Usually the Hamiltonian for the spin, $s$, in the Ising model is written

$$H = -J \sum_{<ij>} \sigma_i \sigma_j - h \sum_i \sigma_i$$

where $h$ is the external field, $J$ is the interaction, $\sigma$ is the spin ($\pm 1$) and $i,j \in L$ (with $L$ as the lattice side length).

The calculation to decide whether to flip or not is by the Boltzmann distribution

$$P_c = \frac{e^{-E_c/kT}}{\sum_c e^{-E_c/kT}}$$

Where $c$ is the configuration, $k$ is Boltzmann’s constant and $E_c$ is the sum of the Hamiltonians. The Boltzmann distribution finds the probability for being in a certain state with the configuration $c$.

This means that if $T \to 0$, then $P_c \to \frac{1}{2}$, so the probability is same for the configuration where all spins are up or all spins are down, when the temperature is 0.

If $T \to \infty$, then $P_c$ is the same for all configurations.

We use a script without any units which means, that there is no reason in using Boltzmann’s constant. Instead we use the Boltzmann factor only

$$P_c \propto e^{-E_c/T}$$

The Boltzmann distribution is calculated for all atomic spins in the system at the same time. As the system gets closer to $T_c$, the system gets closer to equilibrium.
2.2.3 Entropy

The entropy of the system is calculated as following [5]

\[ S = - \sum_c P_c \log(P_c) \]  

(8)

This way of calculating the entropy is better known as Shannon Entropy. As the Shannon entropy depends on the configuration, it is hard to calculate, and we decided not to calculate the entropy of the system. All the way through the model there will be clusters of the atomic spins and at the critical temperature a fractal is created. It is made by the largest cluster and has the volume \( L^D \), where \( D \) is the fractal dimensionality.

3 Model

3.1 Structure

The model consists of seven different functions:

- **GenLattice** Helper function, that generates the lattice of spin up and spin down for the simulation.
- **IsingFunc** Responsible for calculating the Hamiltonian (5), and flipping the cells in the lattice, according to the Boltzmann distribution (7).
- **CreateNeighs** Helper function, that finds the neighbours for a lattice with periodic boundary conditions, and creates the corresponding neighbour matrix.
- **Cluster Func** Finds the clusters of spin up and spin down on the lattice.
- **Boxcount** Performs box count on the largest cluster on a given lattice.
- **CalcFracDim** Function, that performs box count over a given range of box sidelengths.
- **GenerateBoxes** Helper function, that generates the boxes used for boxcount.

For a detailed description of the most complicated functions see 3.2.

The model is run with four parameters:

- **tempRange** The temperature range for the simulation.
- **L** The side length of the lattice.
- **numberOfTimeSteps** The number of timesteps the simulation runs for.
- **sideRange** Range of box sidelengths.

See figure 4 for how these parameters are used in the model.
Figure 4: The structure of the finished model. For a run with parameters: $L$, $tempRange$, $numOfTimeSteps$ and $sideRange$, a lattice of size $L \times L$ is created by GenLattice, the sites in this lattice are updated according to the theory in 2.2 by IsingFunc for $numOfTimeSteps$, using CreateNeighs to find the neighbours for each cell, and clustering on the resulting lattice is then done by ClusterFunc. This is repeated for each increment in $tempRange$, with CalcFracDim using BoxCount, on the final cluster for each side length in $sideRange$, using GenerateBoxes for box generation.

Figure 5: An example of neighbours with periodic boundary condition, on a 4x4 lattice. The neighbours to the green square, marked in red, can easily be found by picking the squares immediately to the right, left, top and bottom of the green square. For the orange square, the right and bottom neighbour, are found normally, while the top and left neighbor are found using periodic boundary conditions wrapping around the lattice to find the neighbours.

### 3.2 Functions

#### 3.2.1 CreateNeighs

The most important feature of the createNeighs method is the boundary condition used for creating the neighbour matrix. Namely the periodic boundary condition. Periodic boundary conditions determines how we handle neighbours, which would be outside a normal lattice. Figure 5 shows an example of periodic boundary conditions, with periodic boundary
conditions. We handle neighbours that would fall outside the lattice by letting the lattice wrap around and meet itself, creating a donut shape (torus). Another important detail of CreateNeighs, is that neighbours are stored with their linear index. This means that cell (3,3) in a 4x4 lattice would be stored as cell 11, and more generally

\[ \text{linIndex}(i, j) = i + (j - 1)L \]  

where \( L \) is the side length of the lattice.

### 3.2.2 ClusterFunc

For clustering the cells in the lattice there are several methods. Our method utilises a queue structure to find the appropriate clustering for each cell. This queue has two pointers, a read pointer and a write pointer.

For a given cell in the lattice, we first assign it a cluster number and then add its linear index to our queue. We then find all of its neighbours with the same spin, assigning them the same cluster number as the first cell, write them to the queue, and increment our write pointer. We then increment our read pointer, and start the process over from the next linear index in the queue. This process terminates when the read pointer encounters an empty place in the queue.

We then repeat this for the next cell in the lattice, that has not been assigned a cluster number. An example of this can be seen in Figure 6.

![Figure 6: An example of the queue with read and write pointer. The queue has read the cell at linear index 1, and added its same spin neighbors to the queue, and the read pointer is now pointing the first one of these, while the write pointer is pointing at the next empty space in the queue.](image)

### 3.2.3 GenerateBoxes

An important detail of the box count method is to consider the side lengths, which is possible to use for the overlaying grid of boxes. If we do not use a side length which is devisable with the lattice side length, we will either have coverage of areas which are not in the lattice, or we will have areas of the lattice that are not covered by the box count grid.

A solution to this is to use a lattice sidelength, which can be factorized into a series of prime numbers. The usable box sidelengths for box count, would then be the possible combinations of these prime factors.

### 4 Experimental Fractal

As mentioned we have been allowed to use a picture of an experimentally made fractal. This experiment was conducted in 2012 by a group making a first year project alike our.

The group experimentally made viscous fingering in a Hele-Shaw cell, where they mixed glass
beads and glycerol which they spread on the lower glass plate. They injected air from the centre of the upper glass plate which created the wanted fractal[7].

The group ended with four useful results. We use one of the created fractals and wish to find and compare the fractal dimension. The fractal we use have been measured by the group using box count to a fractal dimension of $D = 1.72 \pm 0.02[7]$.

5 Results

We have run our simulation on four different lattices with sizes $L = 90$, $L = 180$, $L = 512$ and $L = 5400$, and the results of the three first mentioned are in the following shown as plots with temperatures, size of max clusters and differences in spin up and spin down.

5.1 Fractal Dimensionality

Fractal dimensionality was calculated by performing box count on two different clusters. A $5400 \times 5400$ lattice, created by running our Ising simulation for 953 time steps at the table value for the critical temperature $T_c = 2.269$, and the fractal described in 4, see Figure 7. Box count was performed on both lattices, with side lengths computed by using the prime factors of the lattice side lengths, resulting in 10 different side lengths for the third party lattice ranging from 2 to 1106, and 36 different side lengths for the $5400 \times 5400$ lattice ranging from 2 to 2700. See Figure 8 plot of these results.

5.2 Critical Temperature

We determine the critical temperature, by running our model for a sufficiently number of time steps, at least $L$, over each increment in a temperature range, and plotting the development in difference between spin up and down and size of the largest cluster.

The critical temperature is where we see a sudden increase in both of these parameters, and we get the value by reading the value of the temperature for the sudden increase.

The critical temperature will be determined by looking at the results in Figures [9, 10, 11].
Figure 8: Double logarithmic plot of box count results in third party fractal and the 5400 * 5400 simulation. See Figure 12 a) and b), for fit performed on these results

- $L=512$ This simulation ran for 500 timesteps for each temperature increment, which was not sufficient to reach equilibrium, since the number of timesteps is smaller than the lattice size.

- $L=180$ This simulation was run for 2000 timesteps for each temperature increment, which is sufficient to reach equilibrium at $T_c$. This simulation however started to develop a solid horizontal stripe of spin down, meaning that spin difference and max cluster size decreased after the critical temperature.

- $L=90$ This simulation ran for 2000 timesteps for each temperature increment, and has the most distinct increase in both spin difference and max cluster size.

Figure 9: Results from simulating a 512*512 lattice, with 500 time steps per temperature increment. This is not quite enough for the system to reach equilibrium, which means that the critical temperature found in this simulation is probably not correct. See Table 2 for the critical temperature.
6 Data Analysis

6.1 Fractal Dimensionality

Using a linear fit $-ax + b = y$ on the double logarithmic plots in Figure 8, with MATLAB’s curve fitting tool, we get the fits pictured in Figure 12. The fractal dimensionalities of the figures in the lattices, are the slopes of these fits, giving us the resulting dimensionalities pictured in Table 1.

6.2 Critical Temperature

We determined $T_c$ for each of the simulations $90 \times 90$, $180 \times 180$ and $512 \times 512$, by looking at the graphs for both spin difference and max cluster size, to find the largest increase in these parameters.
Figure 12: Linear fit on double logarithmic plot of the box count, which are the results from Figure 8, the slopes of these fits are the desired fractal dimensionalities. For the values of these fractal dimensionalities see Table 1.

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Dimensionality</th>
<th>R-Square</th>
</tr>
</thead>
<tbody>
<tr>
<td>5400*5400</td>
<td>1.87 ± 0.02</td>
<td>0.99</td>
</tr>
<tr>
<td>Third party</td>
<td>1.66 ± 0.03</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 1: Results from fits pictured in Figure 8, where we have calculated the slope, which is written as the dimension.

The results of this are a temperature range rather than a specific number, as shown in Table 2.

<table>
<thead>
<tr>
<th>L</th>
<th>Determination Parameter</th>
<th>$T_c$ Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>Spin Difference</td>
<td>2.2 to 2</td>
</tr>
<tr>
<td>512</td>
<td>Largest Cluster</td>
<td>2.8 to 2.7</td>
</tr>
<tr>
<td>180</td>
<td>Spin Difference</td>
<td>2.2 to 2.1</td>
</tr>
<tr>
<td>180</td>
<td>Largest Cluster</td>
<td>2.9 to 2.7</td>
</tr>
<tr>
<td>90</td>
<td>Spin Difference</td>
<td>2.5 to 2.2</td>
</tr>
<tr>
<td>90</td>
<td>Largest Cluster</td>
<td>2.5 to 2.2</td>
</tr>
</tbody>
</table>

Table 2: The critical temperature $T_c$, as determined from figures 9, 10 and 11.

### 6.3 Self Similarity and Scale Invariance

Figure 13 ought to show scale invariance in our fractal on the 5400 * 5400 lattice. As a result of the chosen number of timesteps, the system never reached equilibrium and is not scale invariance. We chose the size of the lattice by deciding that the smallest size of zoom on the lattice should be 200 * 200, and that each part should have a size three times bigger than the one before. Figure 14 shows self similarity on the zooms 180 * 180, 90 * 90, 60 * 60 and 40 * 40. The number of timesteps was sufficiently high for this lattice to be in equilibrium at $T_c$. 

13
Figure 13: Images of a 5400*5400 lattice at four different zoom levels. This is to show scale invariance, but the lattice did not reach equilbrium because of the small number of time steps (935), which means this does not show scale invariance.

7 Discussion

7.1 Fractal Dimensionality

7.1.1 5400*5400

Unfortunately our biggest lattice 5400*5400 did not run for enough timesteps to be in equilibrium, which have affected our final results. The fact that the lattice did not reach equilibrium means, the spins have not had enough time to interact with each other. It takes at least 5400 timesteps for spin information to traverse the lattice, which means we as a minimum should have used 5400 timesteps.

7.1.2 Earlier Simulated Results

We found an article, where N. Ito and M. Suzuki write that they earlier have estimated the fractal dimension numerically at the critical temperature $T_c$ as $D = 1.86 \pm 0.01$ for the two-dimensional square lattice[6].
Figure 14: Image of a 180x180 lattice at four different zoom levels. This lattice was simulated for 36000 time steps, with a gradually falling temperature. This amount of time steps is enough for the lattice to reach equilibrium, and is therefore a better example for scale invariance than Figure 13.

Our results shows the fractal dimension for the 5400x5400 lattice at the critical temperature $T_c$ as $D = 1.87 \pm 0.02$. This is comparable as both are for fractal dimensionality at the critical temperature for the Ising model. When compared, we find, that these two results are very alike, with both having relatively small uncertainties as well.

### 7.1.3 Third Party

The use of the third party fractal makes it easy to show a great difference in fractal dimensionality. Fractal dimensions are as expected not always alike for fractals resulting from experiments. It is seen easily that it matters how you measure on the fractal dimension and how exact the measurement is. The group measured the dimension of the experimental fractal as $D = 1.72 \pm 0.02$, but when we measured it, we found the fractal dimension as $D = 1.65 \pm 0.03$. Though the difference is relatively small, especially including the uncertainties, this difference makes it clear how big a difference it makes who and how results are measured. The difference luckily is of a size that makes the measurements comparable, and almost within each others uncertainties.
The reason for this difference in results, could be the parameters used for the box count algorithm, and of course differences in the box count algorithm.

7.2 Critical Temperature

From the results and figures 9, 10 and 11 it is clear that it makes a difference in our simulation what size the lattice is and how many timesteps are used. Simulation of lattice $512 \times 512$ would have had a better result if it had run with more timesteps, and the results from lattice $180 \times 180$ are problematic for determining given how it ended with a solid horizontal stripe of spin downs. This means that the results from the $90 \times 90$ lattice provides the best measurement of the critical temperature, since it reached equilibrium, and did not form any anomalies. 

This is supported by comparing our results from Table 2 with the table value $T_c = 2.269$. The temperature ranges for the $90 \times 90$ lattice, are the only ones which encompass this value, with the spin difference ranges for the $512 \times 512$ and $180 \times 180$.

7.3 Self Similarity and Scale Invariance

As the $5400 \times 5400$ lattice did not reach equilibrium it did not become self similar as seen in Figure 13. However our simulation of a $180^*180$ lattice reached equilibrium and can be used to show self similarity for fractals made in the Ising model at the critical temperature. The problem with the $180 \times 180$ lattice is the size of it as the zooms, the scale between the $180 \times 180$, $90 \times 90$, $60 \times 60$ and the $40 \times 40$ lattice, is easily seen in figure 14. To show both scale invariance and self similarity, it would be necessary to run a lattice of a larger size, for a sufficient number of time steps. Experimental and simulated fractals will however never be completely scale invariant, since they always have a level at which their structure disappear. What we see, when we zoom on the fractals created by our simulation, we see that the structure at some point disappear, and we can distinguish the individual pixels.

As mentioned in the definition of fractals, scale invariance only is a characteristic of mathematical fractals.

8 Conclusion

We conclude that fractals may appear in the Ising model at the critical temperature, which is seen by scale invariance, self-similarity and non integer dimensionality. Scale invariance can unfortunately not be determined from our results.

By the use of box count on our fractal made in a $5400 \times 5400$ lattice in the Ising model, we found the fractal dimension as $D = 1.87 \pm 0.02$, which is close to an earlier numerically estimated fractal dimension, where the fractal dimension was found as $D = 1.86 \pm 0.01$.

Box count has further more been used to determine the dimensionality of a third party fractal, created in an experiment, with the dimensionality $D = 1.66 \pm 0.03$. This, however, is not completely consistent with the third party result of $D = 1.72 \pm 0.02$. This difference could however be ascribed to variation of boxcount method.

From our results, we found the range of the critical temperature to be between $T_c = 2.2$ and $T_c = 2.5$. This corresponds well to the table value of $T_c = 2.269$.

9 Appendix

9.1 MATLAB Functions
function [ lattice ] = GenLattice(n)
%Generates a lattice of n*n sites, of either 1 or -1
lattice = randi(2,n);
    lattice = lattice*2-3; %Change values from 1 and 2 to 1 and -1
end

Error using GenLattice (line 3)
Not enough input arguments.

Published with MATLAB® R2013a
function [lattices] = IsingFunc(numberOfTimeSteps, Temp, lattices)
% Simulate the Ising model for a numberOfTimeSteps, at the temperature
% Temp on a given lattice
n = length(lattices);
m = n;
outerField = 0; % The model is set up to use an outerfield, but it is not used for
% of this report
neighs = CreateNeighs(n, m); % Create the Neighbour matrix
% simulation
for s = 1:numberOfTimeSteps*m*n; % Flip m*n sites for each time step
    lin = randi(m*n); % Generate index of site to be flipped
    spin = lattices(lin); % Spin of the site
    deltaE = -2*spin*(outerField + sum(lattices(neighs(lin,:)))); % Calculate change
    if deltaE < 0 || rand() < exp(-deltaE/Temp) % Flip if energy change is negative
        lattices(lin) = -spin;
    end
end
end

Error using IsingFunc (line 4)
Not enough input arguments.

Published with MATLAB® R2013a
function [ neighs ] = CreateNeighs(m,n)
%Generates a matrice containing the linear index for the four neighbours,
%of each individual site on the m*n lattice including periodic boundary s.
neighs = zeros(n*m,4); %Create matrice to store neighbours
for  i =1:n;
    for  j = 1:m;
        linindex = i+(j-1)*m; %linear index of the site
        if (i < n) %Lower neighbour
            neighs(linindex,1) = i+1+(j-1)*m;
        else
            neighs(linindex,1) = 1+(j-1)*m; %Lower periodic boundary
        end
        if (i == 1) %Upper neighbour
            neighs(linindex,2) = n+(j-1)*m; %Upper periodic boundary
        else
            neighs(linindex,2) = i-1+(j-1)*m;
        end
        if (j < m) %Right neighbour
            neighs(linindex,3) = i+(j*m);
        else
            neighs(linindex,3) = i; %Right periodic boundary
        end
        if (j == 1) %Left neighbour
            neighs(linindex,4) = i+(m-1)*m; %Left periodic boundary
        else
            neighs(linindex,4) = i+(j-2)*m;
        end
    end
end
end

Error using CreateNeighs (line 4)
Not enough input arguments.

Published with MATLAB® R2013a
function [clusterLat, clusters] = ClusterFunc(lattices)
% Generate the spin up and spin down clusters on the lattice, and return a
% copy of the lattice with cluster numbers, and the size of each cluster.
clusterLat = lattices;
n = length(lattices);
size = n*n;
clusters = zeros(size,1);
neighs = CreateNeighs(n,n);
clusterNumber = 2; % Start with cluster number two
clusterIds = randperm(size)+1; % Create vector of randomly generated cluster numbers
for lin = 1:size
    if clusterLat(lin)< 2 % If number in site is not a cluster number
        spin = clusterLat(lin); % Spin of the site
        clusterLat(lin) = clusterIds(clusterNumber); % Change the number in the site
        clustSize = 1;
        queue = zeros(size+1,1);
        queue(1) = lin;
        read = 1; % Read pointer for the queue
        write = 2; % Write pointer for queue
        while queue(read) > 0
            indexNeighbours = neighs(queue(read),:); % Find neighbours to site at read pointer
            for i = 1:4
                if clusterLat(indexNeighbours(i)) == spin
                    queue(write) = indexNeighbours(i); % Write neighbour is spin corresponds to original site
                    clusterLat(indexNeighbours(i)) = clusterIds(clusterNumber); % Change number
                    clustSize = clustSize+1;
                    write = write+1;
                end
            end
            read = read+1;
        end
        clusters(clusterNumber) = clustSize;
        clusterNumber = clusterNumber+1;
    end
end
[~,index] = max(clusters); % Find index of largest cluster
clusterLat(clusterLat == clusterIds(index)) = 1; % Rename largest cluster to 1 for easier visualization
clusters = clusters(2:clusterNumber-1);
end

Error using ClusterFunc (line 4)
Not enough input arguments.

Published with MATLAB® R2013a
function [ count ] = BoxCount(clusters, side)
% Count the number boxes with side length that are needed to cover the % largest cluster in clusters
[n,m] = size(clusters);
remain = mod(n,side);
if  remain ~= 0
    n = floor(n/side)*side;
    clusters = clusters(1:n,1:n); %Cut down cluster matrix to fit a multiplier of
end
boxes = GenerateBoxes(n, side) ;
[k,l] = size(boxes);
count = 0;
for  i = 1:k
    for  j = 1:l %loop through each cell in a box
        if  clusters(boxes(i,j)) == 1;
            count = count+1;
            break; %exit loop through cells if a hit is found
        end
    end
end
end

Error using BoxCount (line 4)
Not enough input arguments.

Published with MATLAB® R2013a
function [areas] = CalcFracDim(clusters, sideRange)
%Calculate the number of boxes with a side length in sideRange needed for covering
nBoxCounts = length(sideRange);
areas = zeros(nBoxCounts,1);
for i = 1:nBoxCounts
    side = sideRange(i);
    count = BoxCount(clusters,side);
    areas(i) = count;
end
end

Error using CalcFracDim (line 3)
Not enough input arguments.

Published with MATLAB® R2013a
function [ boxes ] = GenerateBoxes(n,r)
%GenerateBoxes Generate the coordinates for the mesh of r*r boxes,covering the n*n lattice.
nBoxes = floor(n^2/(r^2)); %Compute the number of boxes needed
boxes = zeros(nBoxes,r^2); %Create the matrix to store the boxes
box = 0;
for i = 1:r^2*n/r:n*n %Loop through the rows of boxes
    if box >= nBoxes %Terminate early if there would be generated to many boxes
        break
    end
    for j = 1:n/r %Loop through the start of each box in a row
        boxstart = i+(j-1)*r; %Calculate the start coordinate for the box
        for k = 0:r-1 %Loop through the rows in the box
            for l = 0:r-1 %Loop through the elements in each row
                cell = boxstart+k*r*n/r+l; %Calculate the coordinate of the individual cell
                boxes(box,k*r+l+1) = cell;
            end
        end
    end
end
end

Error using GenerateBoxes (line 3)
Not enough input arguments.

Published with MATLAB® R2013a
References


