BERGISCHE UNIVERSITÄT WUPPERTAL

PROJEKTPRAKTIKUM

PROJEKT IN DER THEORETISCHEN PHYSIK IM RAHMEN VON 6 LP

Untersuchung des zweidimensionalen Ising-Modells mit Monte-Carlo Methoden

Autor
Kai BENNING

Betreuer
PD Dr. Christian HÖLBLING

24. Januar 2018

Inhaltsverzeichnis

1 Einführung

Das Ising-Modell ist ein Modell der theoretischen Physik zur Beschreibung des Ferromagnetismus. Es wurde von Ernst Ising im Jahre 1924 auf Anregung seines Doktorvaters genauer untersucht [1]. Das vom ihm untersuchte eindimensionale Modell zeigte keinen Phasenübergang. Dieses Verhalten zeigt sich erst in höheren Dimensionen und wurde später entdeckt.

2 Das Modell

Es werden magnetische Spins $s = \pm 1$ auf einem NxN großen Gitter behandelt, welche untereinander nur mit dem nächsten Nachbarn wechselwirken. Jeder Spin hat in zwei Dimensionen vier nächste Nachbarn (oben, unten, links, rechts), die Spins am Rand beziehungsweise die Spins in den Ecken des Modells wechselwirken für die fehlenden Nachbarn mit den jeweils gegenüberliegenden Spins (periodische Randbedingungen). Der Hamiltonian für das System lautet:

$$\mathcal{H} = -\frac{1}{2} \cdot J \sum_{\langle ij \rangle} s_i s_j - H \sum_i s_i \tag{1}$$

[2]. Der Ausdruck $\langle ij \rangle$ steht dafür, dass nur direkt benachbarte Spins in der Summe gezählt werden. Der zweite Term berücksichtigt die Wechselwirkung der magnetischen Spins mit einem äußeren Feld. In dieser Arbeit ist $H = 0$. Der Hamiltonian vereinfacht sich damit zu:

$$\mathcal{H} = -\frac{1}{2} \cdot J \sum_{\langle ij \rangle} s_i s_j \tag{2}$$

Das negative Vorzeichen ist Konvention, der Faktor $\frac{1}{2}$ ist wichtig, damit die Spins nicht doppelt gezählt werden, und J ist die Kopplungskonstante, diese ist für den ferromagnetischen Fall $J = 1$. Das zweidimensionale Ising-Modell (mit dem verschwindenen Magnetfeld) wurde 1944 von Lars Onsager gelöst.

3 Simulation

Die numerische Berechnung des Ising-Modells erlaubt einerseits Lösungen für das zweidimensionale Gitter mit Feld zu finden und andererseits auch höherdimensionale Gitter zu untersuchen, für die noch keine analytsiche Lösung bekannt ist. Auch gelangt man aufgrund der „Einfachheit"des Ising-Modells sehr schnell zu Ergebnissen. Die Zeit des Modells wird hierbei diskretisiert. Das Modell startet bei $t_0 = 0$ und endet bei einer beliebigen Zeit $t_N = N\,;\, N \in \mathbb{N}^+$. Die Wahrscheinlichkeit, ob sich die Konfiguration zwischen zwei „sweeps"ändert, hängt von dem Energieunterschied ab. Die „Vorgeschichte"einer Konfiguration beschränkt sich hierbei nur auf den direkten Vorgänger t_{n-1}. Dieser stochastische Prozess lässt sich durch eine Markow-Kette beschreiben. In dem Projekt soll diese Kette mit Hilfe von zwei bekannten Algorithmen modelliert werden. Für die Programmierung wurde die Julia-Language verwendet.

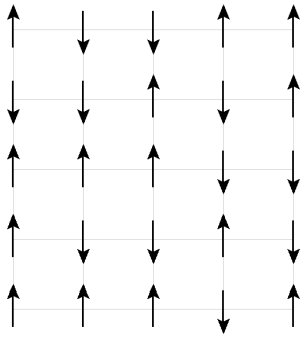

Abbildung 1: Zur Veranschaulichung ein 5 x 5 Spin-Gitter, auf welchem eine solche Simulation für verschiedene Temperaturen ausgeführt werden könnte. Auf den insgesamt 25 Gitterplätzen ist ein magnetisches Moment platziert, welches entweder nach oben oder nach unten zeigt. Ein einzelner Spin sieht hierbei nur die nächsten Nachbarn (oben, unten, rechts, links) beziehungsweise am Rand die gegenüberliegenden Spins.

Der erste Algorithmus bezieht für die Wahrscheinlichkeit, dass ein Spin flipt, nur die nächsten Nachbarn hinzu. Der zweite Algorithmus erzeugt zusammenhängende Gebiete aus Spins gleicher Orientierung, die anschließend gemeinsam geflipt werden.

3.1 Metropolis-Algorithmus

Der Metropolis-Algorithmus ist ein lokaler Update Algorithmus. In jedem „sweep“wird ein zufälliger Spin aus dem Gitter ausgewählt. Wenn die Energie der Spinkonfiguration bei einem Flip niedriger ist als vorher, wird dieser sofort geflippt. Ist sie höher wird der Flip nur mit einer Wahrscheinlichkeit P durchgeführt.

$$P = \exp\{-\beta \Delta E\} \tag{3}$$

[3] Der Faktor $\beta = (k_B T)^{-1}$ (in diesem Projekt ist $k_B = 1$) verhält sich reziprok zur Temperatur. Für große Temperaturen ist also ein spontaner Spin-Flip wahrscheinlicher. Wenn ΔE maximal ist, also alle vier benachbarten Spins parallel zum zentralen Spin ausgerichtet sind, ist die Wahrscheinlichkeit für festes β minimal.

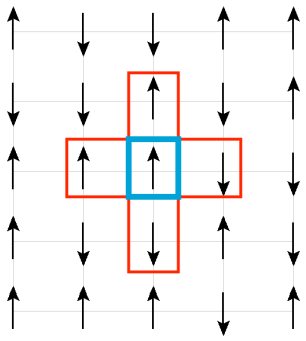

Abbildung 2: Hier wurde der blau eingerahmte Spin zufällig ausgewählt. Seine vier Nachbarn sind zweimal nach unten, und zweimal nach oben gerichtet, damit ist der Energieunterschied ΔE bei einem Flip Null und demnach die Wahrscheinlichkeit $P = 1$. Der Spin wird also gedreht.

Dies wird solange gemacht, bis sich die zu messenden Obversablen sich nicht mehr ändern, also bis sich das System im Gleichgwicht befindet.

3.2 Wolff-Algorithmus

Es stellt sich heraus, dass der Ansatz, nur einen Spin zur selben Zeit zu flippen, insbesondere im Bereich der kritischen Temperatur T_c zu einer langsamen Entwicklung „critical-slowing-down“. Auch hier wird wieder ein zufälliger Spin aus dem Gitter ausgewählt.

Aber nun werden die nächsten Nachbarn daraufhin überpfüft, ob diese dieselbe Orientierung aufweisen. Ist das der Fall wird der Spin mit der Wahrscheinlichkeit $P = 1 - \exp(-2/T)$ der Auswahl hinzugefügt [4].

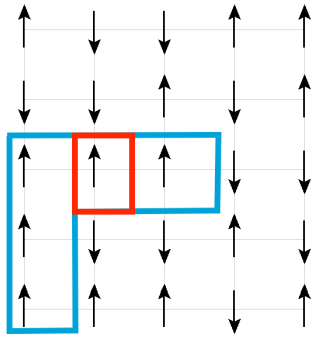

Abbildung 3: Bei einem Step des Wolff-Algorithmus wurde der zufällige Spin (rot eingerahmt) ausgewählt. Die nächsten Nachbarn, die auch nach oben zeigen, werden mit der Wahrscheinlichkeit P dem Gebiet hinzugefügt. Gelingt dies, so werden die nächsten Nachbarn der neu hinzugefügten Spins ebenfalls überprüft.

Dies wird solange ausgeführt bis sich entweder keine neuen Nachbarn mit gleicher Orientierung finden oder die verbleibenden Erweiterungsversuche fehlschlagen.

3.3 Autokorrelation

Die Autokorrelation beschreibt die Korrelation einer zum Beispiel zeitabhängigen Funktion mit sich selbst zu einem früheren Zeitpunkt. Im Rahmen der Modellierung des Isings Systems mit Monte Carlo Methoden, bei welchem nach Erreichen des Gleichgewichtszustands Messungen der Obversablen vorgenommen werden, ist es wichtig die Korrelation der Messwerte zwischen den diskreten Zeitpunkten zu kennen. Die Formel der Autokorrelation lautet:

$$Corr(t_1, t_2) = \frac{\langle S_i(t) * S_i(t') \rangle - \langle S_i(t) \rangle^2}{\sigma^2} \tag{4}$$

[5]. Dies wird für verschiedene Temperaturen sowohl beim Metropolis- als auch beim Wolff-Algorithmus berechnet. Es wird erwartet, dass die Autokorrelation bei dem Wolff-Algorithmus viel schneller abfällt als beim Metropolis-Algorithmus.

3.4 Mittlere Energie

Die mittlere Energie berechnet sich durch die Wirkung des Hamilton-Operators auf das Spingitter. Wegen der periodischen Randbedingung reicht es aus zu dem Gitter selbiges einmal in x- und einmal in y-Richtung verschoben hinzu zu addieren. Die mittlere Energie pro Spin ist maximal, wenn alle Spins in dieselbe Richtung zeigen und minimal, wenn alle Spins zueinander antiparallel liegen. Im ferromagnetischen Fall erwartet man für niedrige Temperaturen eine „rasche“Ausrichtung aller Spins in eine Orientierung, da die thermische Anregung und ein Spin-Flip aufgrund der Boltzmann-Statistik unwahrscheinlicher ist.

3.5 Wärmekapazität

Die Wärmekapazität lässt sich als Varianz der Energie herleiten:

$$C = \beta^2 \frac{\partial^2 \ln(Z)}{\partial \beta^2} = \beta^2 \frac{\partial}{\partial \beta} \left(\frac{1}{Z} \frac{\partial Z}{\partial \beta} \right) = \beta^2 \left[\frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2} - \frac{1}{Z^2} \left(\frac{\partial Z}{\partial \beta} \right)^2 \right] \tag{5}$$

wir identifizieren den Subtrahenten in der Klammer mit:

$$\frac{1}{Z^2} \left(\frac{\partial Z}{\partial \beta} \right)^2 = \langle E \rangle^2 \tag{6}$$

und den Minuenden mit:

$$\frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2} = \langle E^2 \rangle \tag{7}$$

somit erhält man:

$$C = \beta^2 \left(\langle E^2 \rangle - \langle E \rangle^2 \right) = \beta^2 \operatorname{Var}[E] \tag{8}$$

3.6 Finite Size Scaling und Reweighting Technique

Im oberen Abschnitt sieht man, dass für größere Gittergrößen sich der Peak der Wärmekapazität zu kleineren Temperaturen hin verschiebt. Allerdings kann man die genaue Position kaum ablesen. Hier kann mit der „Reweighting Technique“angesetzt werden. Da wir den ungefähren Punkt der kritischen Temperatur für die jeweilige Gittergröße kennen sowie die mittlere Energie der Konfigurationen $\Phi_i, i \in \mathbb{N}$, ist es ausreichend die Obversable O_i und die Energie E_i für eine Temperatur in der Nähe zu berechnen. Die Wahrscheinlichkeit $\mathcal{P}(\beta, E)$, dass ein System mit der Temperatur β im Zustand mit einer Energie E ist, lautet:

$$\mathcal{P}(\beta, E) = \frac{1}{Z} \cdot N(E) \cdot \exp(-\beta E) \tag{9}$$

Die Wahrscheinlichkeit einen Zustand mit derselben Energie, bei einem anderen β' zu finden, ist statistisch um die Temperatur β verteilt. Man kommt anschließend auf die Formel:

$$\langle O \rangle_\beta = \frac{\langle O \exp(-(\beta^{'} - \beta) \cdot E) \rangle_\beta}{\langle \exp(-(\beta^{'} - \beta) \cdot E) \rangle_\beta} \tag{10}$$

In einer Monte-Carlo-Simulation kann der Erwartungswert einer Obserable $\langle O_i \rangle$ durch den Mittelwert über die zeitdiskreten Zustände $(\Phi_1, \Phi_2, ..., \Phi_N)$ ermittelt werden.

$$\frac{1}{N} \sum_{i=1}^{N} O_i \tag{11}$$

Damit vereinfacht sich die Formel zu;

$$\langle O \rangle_\beta = \frac{\sum_i O_i \exp(-(\beta^{'} - \beta) \cdot E_i)}{\sum_i \exp(-(\beta^{'} - \beta) \cdot E_i)} \tag{12}$$

Über diese Formeln werden nun als Obversablen O_i die einfachen- und die quadratische Energien der Konfigurationen genommen. Anschließend wird die Summe über alle Konfigurationen ausgeführt. Dies wird für ein Intervall um die geschätzte kritische Temperatur herum ermittelt. Daraufhin wird die Position des Maximums bestimmt und gegen die reziproke Gittergröße aufgetragen. Anschließend kann die kritische Temperatur für ein unendlich ausgedehntes Ising-Gitter extrapoliert werden [6] [7].

4 Ergebnisse

Es sollen die Ergebnisse der Berechnungen des zweidimensionalen Isingsmodells mit dem Wolff-Algorithmus vorgestellt werden. Um erst einmal ein Gefühl für die unterschiedliche Wirkungsweise der Algorithmen zu bekommen, soll die Evolution des Ising Gitters bei der Temperatur $T = 1$ mit dem Wolff-Algorithmus und dem Metropolis-Algorithmus betrachtet werden. Bei beiden wird aus einer zufälligen Konfiguration aus Spin-Up und Spin-Down Spins gestartet. Das Ergebnis für den Metropolis Algorithmus ist in Abbildung 4 zu sehen. Nach $t = 12.500$ Schritten erkennt man wie sich die Spins auf dem $50x50$ Gitter Gruppen aus Spin-Up und Spin-Down Spins bilden. Diese Muster bleiben in den weiteren Konfigurationen stabil und werden tendenziell kleiner. Nach $t = 125.000$ Schritten ist die „weiße Insel“in der unteren linken Ecke verschwunden. Dieses Verhalten setzt sich bis $t = 500.000$ fort. In der letzten Konfiguration ($t = 1.000.000$) sind alle Spins bis auf einem parallel zueinander ausgerichtet.

Das Ergebnis für den Wolff-Algorithmus ist in Abbildung 5 zu sehen. Hier wird direkt klar, dass die Simulationszeit bei derselben Gittergröße $50x50$ und Temperatur $T = 1$ erheblich kürzer ist. Während beim Metropolis-Algorithmus das Gleichgewicht erst nach $t = 1.000.000$ erreicht ist, ist dies beim Wolff-Algorithmus bereits nach $t = 24$ erreicht. Die Inselbildung ist bereits schon bei $t = 4$ in der oberen rechten Ecke zu beobachten. In $t = 10$ wurde erneut ein Spin aus dieser Gruppierung ausgewählt und das Gebiet erweitert und geflipt. Es entstehen mit der „Zeit“immer mehr kleine Grüppchen, bis in $t = 24$ diese Gebiete zusammenwachsen. Im letzten Schritt $t = 24$ sind fast alle Spins zueinander parallel ausgerichtet.

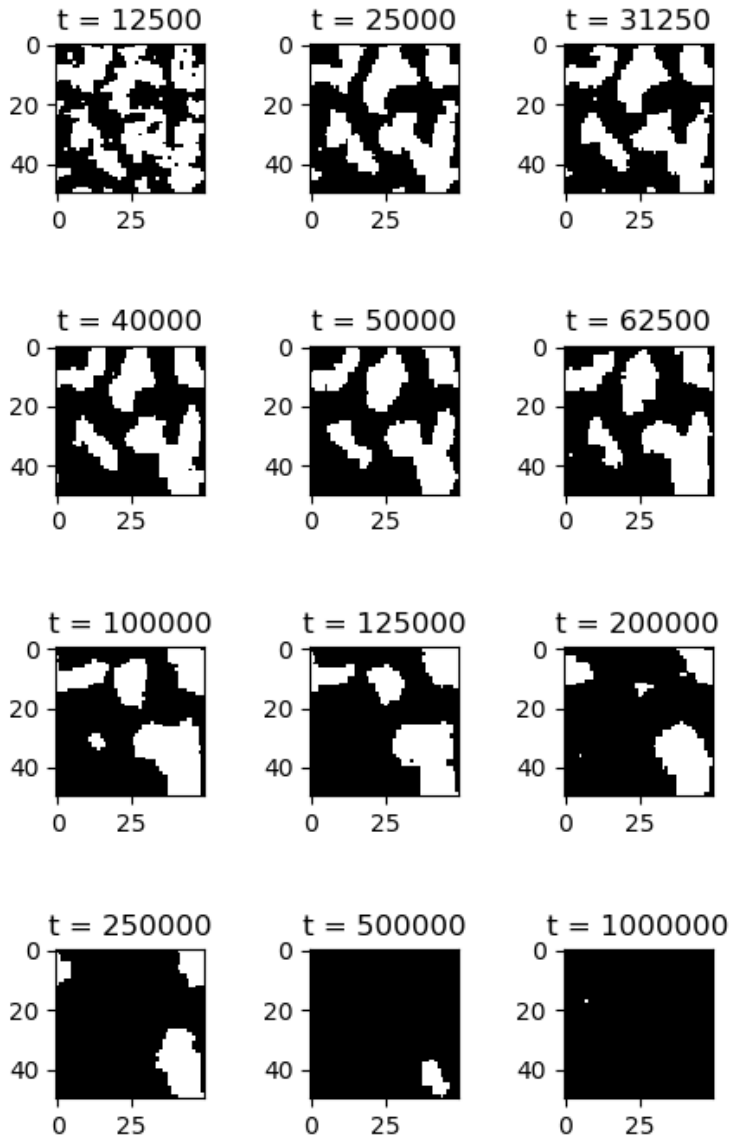

Abbildung 4: Evolution des Spingitters bei der Temperatur $T = 1$ bei Verwendung des Metropolis-Algorithmus. Es ist eine deutliche Strukturbildung zu erkennen. Bei langen Laufzeiten richten sich alle Spins parallel zueinander aus.

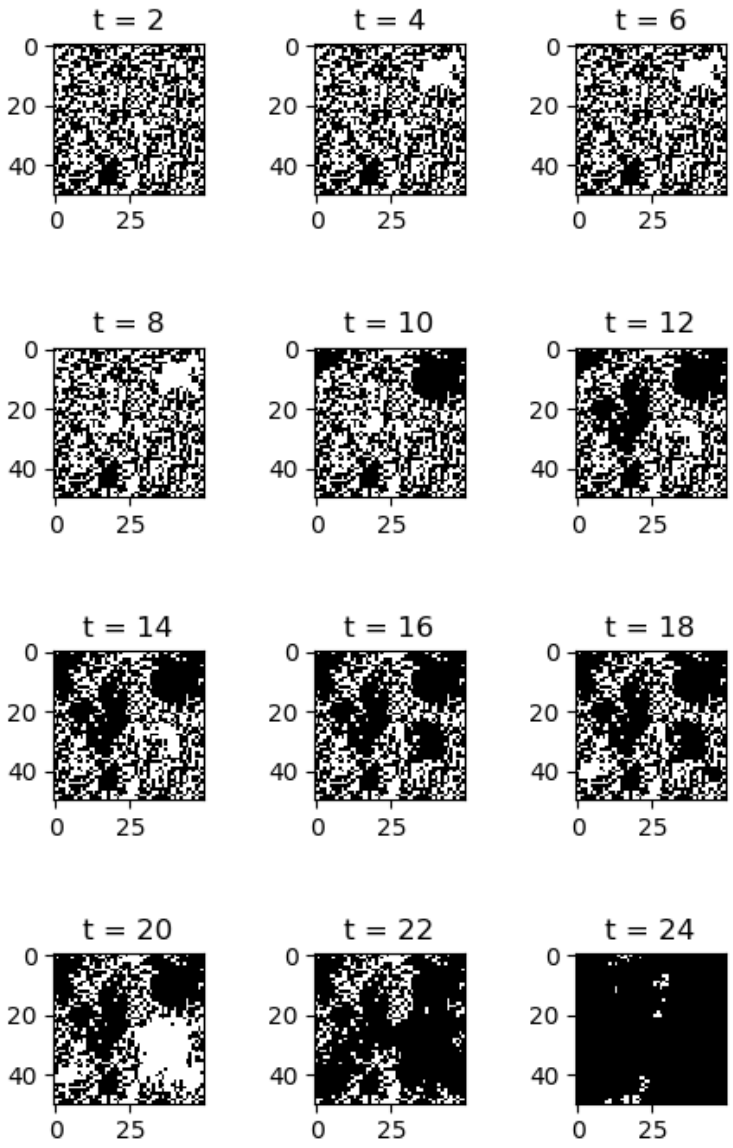

Abbildung 5: Evolution des Spingitters bei der Temperatur $T = 1$ bei Verwendung des Wolff-Algorithmus. Es ist eine deutliche Strukturbildung zu erkennen. Bei langen Laufzeiten richten sich alle Spins parallel zueinander aus.

Nun soll die Autokorrelation der beiden Algorithmen miteinander verglichen werden. Als Gittergröße wird $N = 8x8$ gewählt. Das Ergebnis für den Metropolis-Algorithmus ist in Abbildung 6 zu sehen.

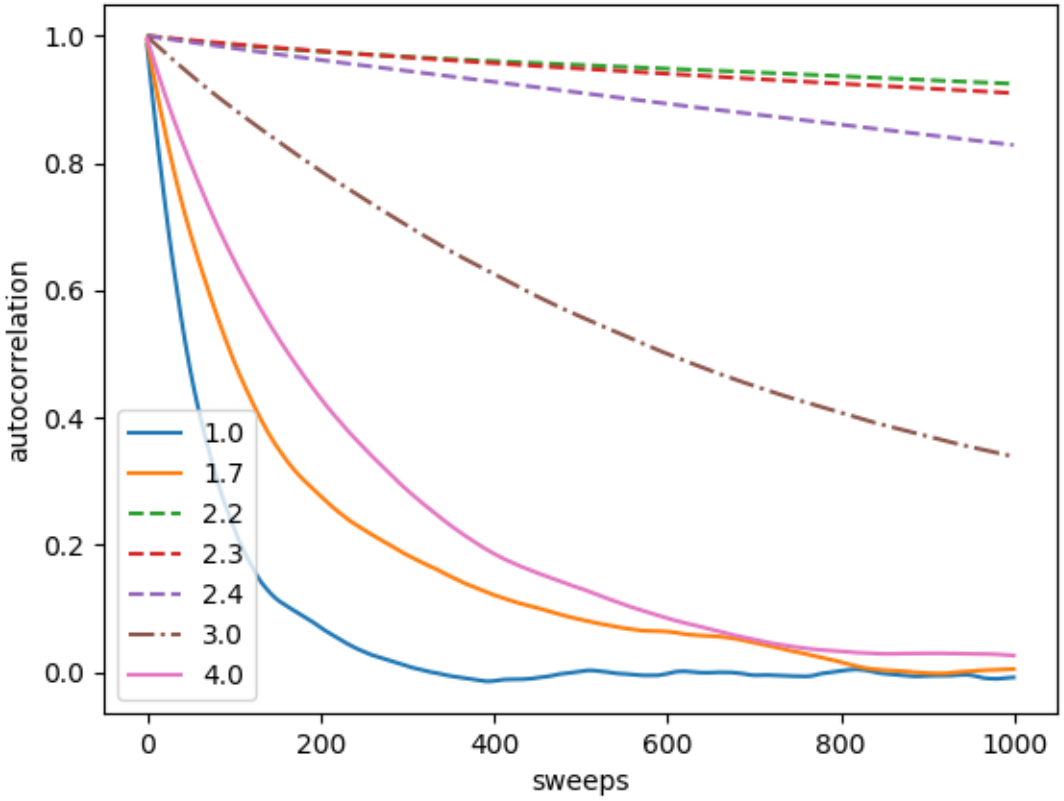

Abbildung 6: Autokorrelation des Metropolis-Algorithmus bei verschiedenen Temperaturen

Für die Temperaturen T = 1.0, T = 1.7, und T = 4.0 fällt die Autokorrelation exponentiell ab. Bei der Temperatur T = 3.0 ist das Abfallen wesentlich langsamer, und bei den Temperaturen T = 2.2, T = 2.3, und T = 2.4 divergiert die Autokorrelation und bleibt auf 1. Dass der Wert leicht absinkt, hängt an der begrenzten Größe des Systems.

Beim Wolff-Algorithmus sieht die Korrelation komplett anders aus. Wie man in Abbildung 7 sieht, alterniert die Autokorrelation zwischen 1 und -1, was daran liegt, dass bei einem sweep ein ganzes Cluster geflipt wird.

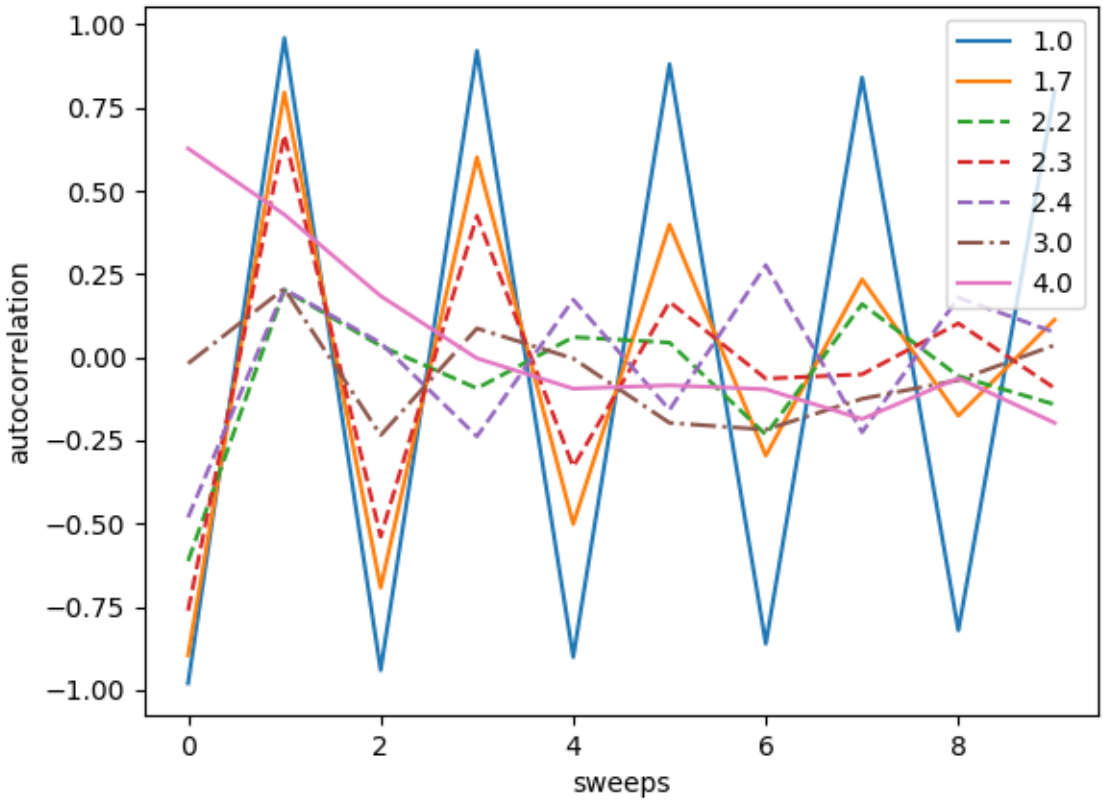

Abbildung 7: Autokorrelation des Wolff-Algorithmus bei verschiedenen Temperaturen

Ein Plot für die integrierte Autokorrelation über den Temperaturbereich $T = 1.0$ bis $T = 4.0$ beider Algorithmen befindet sich im Anhang.

Die Mittlere Energie pro Spin wird wie beschrieben für die Gittergrößen $4x4$, $8x8$,$16x16$, $32x32$, und $64x64$ berechnet. Das Ergebnis ist in Abbildung 8 zu sehen. Bei niedrigen Temperaturen ist der Betrag der Energie maximal, da die Spins alle parallel zueinander ausgerichtet sind. Erhöht man die Temperatur, ist das System im Gleichgewicht ungeordnet, und die absolute Energie pro Spin nimmt ab.

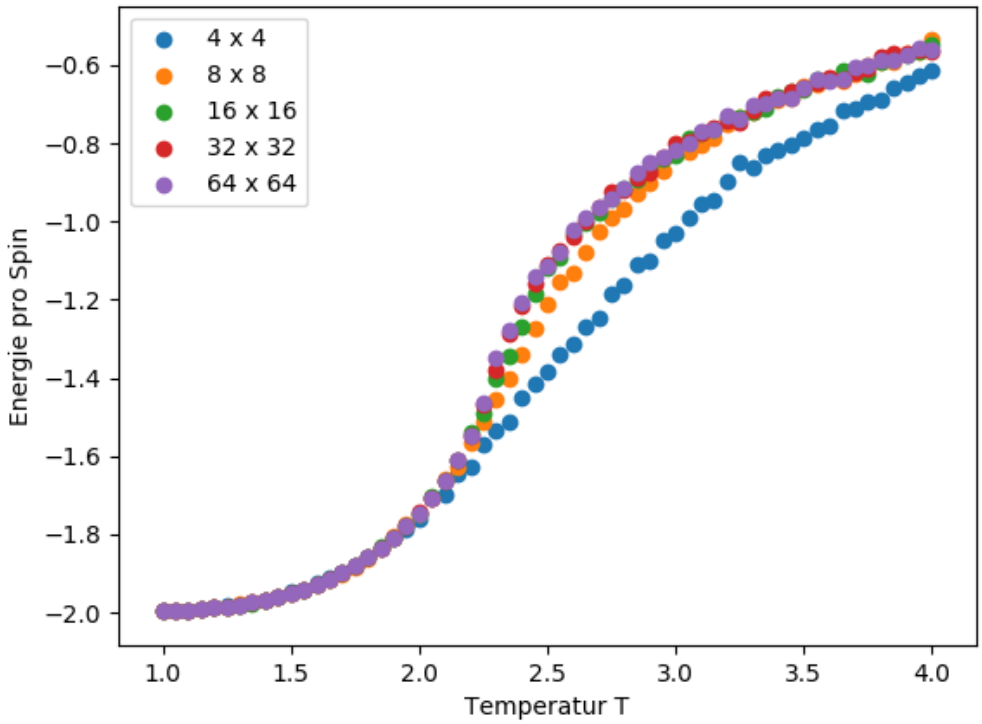

Abbildung 8: Mittlere Energie pro Spin für verschiedene Gittergrößen

Zwischen der Temperatur $T = 2.0$ und $T = 3.0$ nimmt die absolute Energie pro Spin rasch ab. Der Verlauf ist umso steiler desto größer das Gitter ist. Hier ist ein Phasenübergang zwischen den beiden Temperaturen auszumachen.

Der Phasenübergang lässt sich auch bei der Wärmekapazität beobachten. Das Ergebnis ist in Abbildung 9 zu sehen. Für die Gittergröße $4x4$ hat die Wärmekapazität nur ein schwaches Maximum um $T = 2.4$. Wenn die Gittergröße vergrößert wird, verschiebt sich das Maximum zu niedrigeren Temperaturen und wird spitzer.

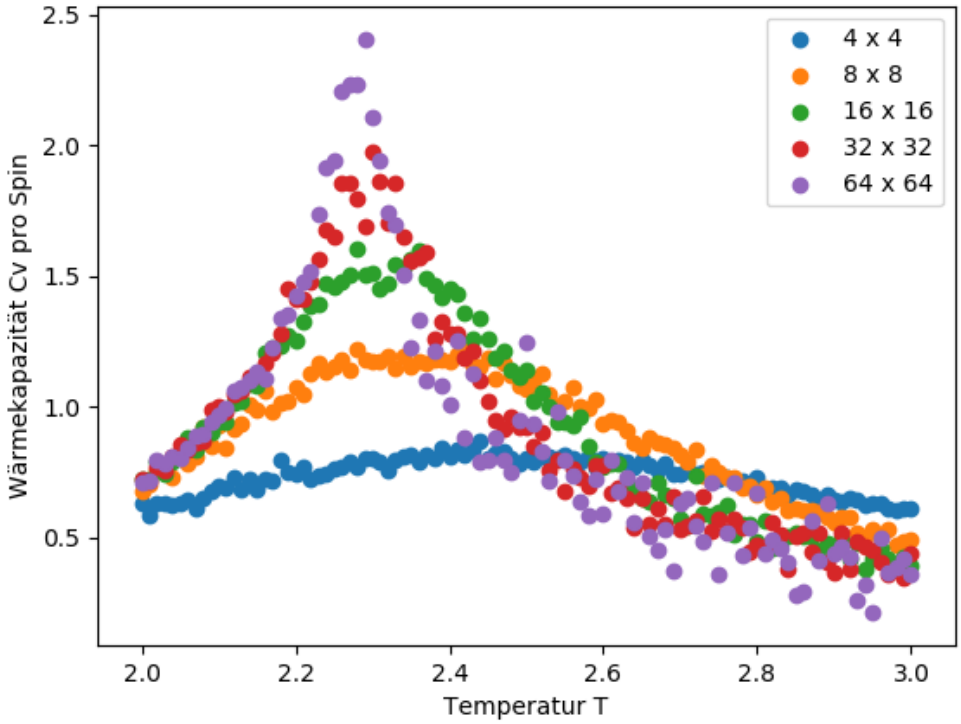

Abbildung 9: Wärmekapazität pro Spin für verschiedene Gittergrößen

Das liegt daran, dass die Wärmekapazität bei $T_{Literatur} = 2.269$ divergiert [1]. Die Position des Maximums verändert sich dabei immer weniger. Mit der Reweighting-Technik sollen nun die genauen Maxima bestimmt werden. Die Maxima werden gegen die reziproke Gittergröße aufgetragen und mit einem Power-Law gefittet.

Um ein Gefühl für das Reweighting zu bekommen wird die Wärmekapazität C_v für ein $N = 16x16$ Gitter bei zwei Temperaturen $T = 2.2$ und

[1] Onsagers Lösung für die kritische Temperatur beim zweidimensionalen Ising-Modell

$T = 2.3$ bestimmt, und mit der Reweighting-Technik die Wärmekapazität des Systems für Temperaturen darüber beziehungsweise darunter rekonstruiert. Das Ergebnis ist in Abbildung 10 zu sehen.

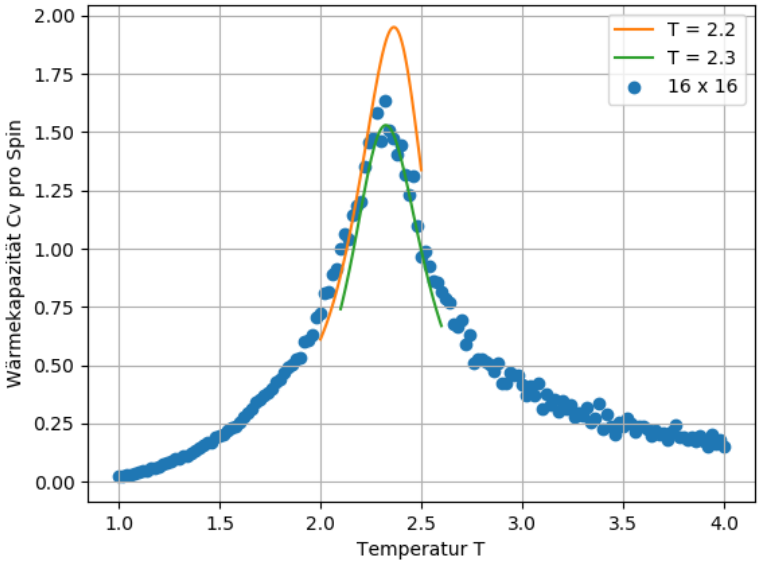

Abbildung 10: Ergebnisse der Reweighting-Technik für T = 2.2 und T = 2.3

Es ist ein deutlicher Unterschied in der Höhe der orangen und grünen Kurve zu sehen, und auch ein kleiner Shift zwischen den Maxima zu erkennen. Unser Interesse ist dabei die Position des Maximums der Wärmekapazität. Um die Verschiebung des Maximums grob abzuschätzen, wird die Simulation fünfmal ausgeführt. Die Verschiebung betrug 0.03 ± 0.01. Für das Finite-Size-Scaling der Wärmekapazität Cv wird keine Fehlerabschätzung in dieser Form vorgenommen, aber es wird versucht die Ausgangstemperatur T für die verschiedenen Gittergrößen (Abbildung 9) möglichst genau abzulesen.

Das Reweighting in der Nähe der kritschen Temperatur für die Gittergrößen $8x8$,$16x16$, $32x32$, und $64x64$ ist in Abbildung 11 zu sehen. Ebenso der Plot der inversen kritischen Temperatur β_{max} gegen die inverse Gittergröße $1/L$ mit der angesprochenen Fitfunktion.

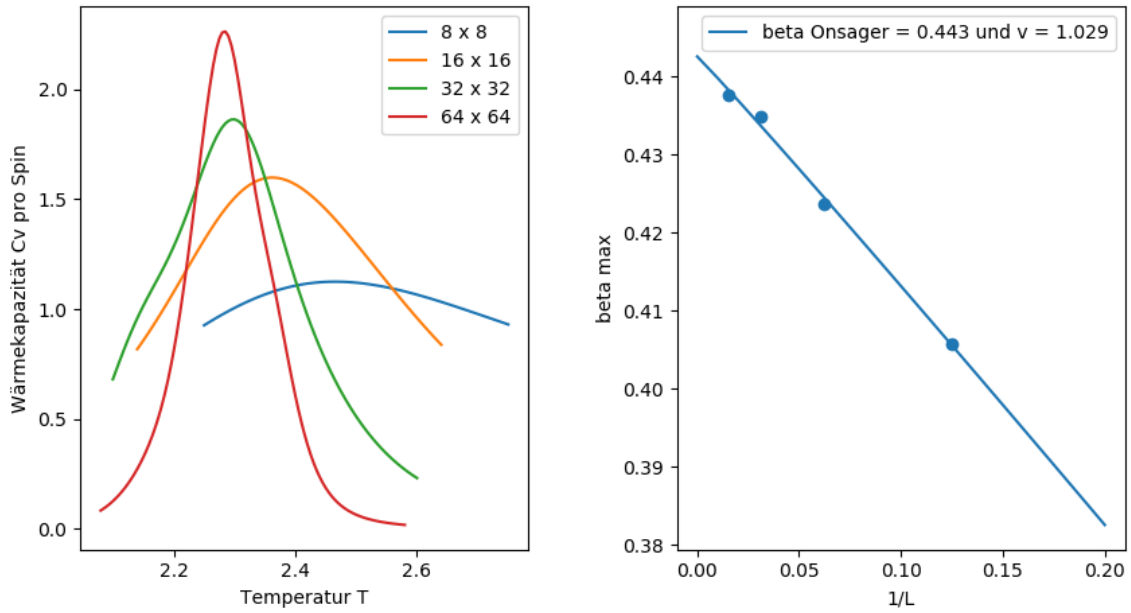

Abbildung 11: Reweighting der Wärmekapazität für verschiedene Gittergrößen in der Nähe der kritischen Temperatur und Fitfunktion.

Im linken Plot ist zu erkennen wie die Maxima für größere Gitter immer weiter nach links (zu kleineren Temperaturen hin) wandern. Es ist auch zu erkennen wie die Peakhöhe größer wird. Beides scheint für $N \to \infty$ gegen einen Grenzwert zu streben. Im rechten Plot sieht der Zusammenhang zwischen β_{krit} und $1/L$ linear aus. Der y-Achsenabschnitt sollte hierbei Onsagers Lösung $\beta_{Onsager}$ für die kritische Temperatur entsprechen. Für die Fitparameter ergibt sich für den gezeigten Durchlauf (ohne Fehlerbetrachtung) als Exponent $v = 1.029$ und als y-Achsenabschnitt $\beta_{Onsager} = 0.443$. Damit ergibt sich eine kritische Temperatur von

$$T_{Onsager} = \left(\beta_{Onsager}\right)^{-1} = 2.257 \tag{13}$$

.

5 Zusammenfassung

In diesem Projektpraktikum wurde das Ising-Modell mit zwei verschiedenen Algorithmen realisiert. Es wurde ein lokaler- (Metropolis-Algorithmus) und ein Cluster-Algorithmus (Wolff-Algorithmus) verwendet. Dabei konnte festgestellt werden, dass der Wolff-Algorithmus wesentlich effizienter ist (ca. Faktor 10^5) und die Obversablen weniger miteinander korreliert sind. Mit letzterem wurde dann die „mittlere Energie pro Spin“und die „Wärmekapazität pro Spin“für verschiedene Gittergrößen ermittelt. Dabei konnte ein Phasenübergang beobachtet werden.

Mittels Reweighting wurde die genaue Position der Maxima der Wärmekapazität für die verschiedenen Gittergrößen berechnet. Die inversen Maxima ($1/\mathrm{T} = \beta$) wurden gegen die inversen Gittergrößen ($1/\mathrm{L}$) geplottet und mit einem Power-Law gefittet.

Dieses Finite Size Scaling ergab einen Exponenten von $\nu = 1.029$ (linearer Zusammenhang) und einen y-Achsenabschnitt von $\beta_{max} = 0.443$. Damit ergibt sich eine kritische Temperatur $T_{Onsager} = \left(\beta_{Onsager}\right)^{-1} = 2.257$.

Literatur

[1] E. Ising, "Beitrag zur theorie des ferromagnetismus," *Zeitschrift für Physik, Band 31*, 1925.

[2] W. Nolting, *Grundkurs Theoretische Physik 6 Elektronische Ressource: Statistische Physik*. Springer-Lehrbuch, Berlin, Heidelberg: Springer Berlin Heidelberg, 7. aufl. 2014 ed., 2014. Verfasserangabe: von Wolfgang Nolting ; Online-Ressource Kann nicht per Fernleihe bestellt werden! ; Quelldatenbank: UBWU-x ; Format:marcform: print ; Umfang: XVI, 642 S. 114 Abb.

[3] N. Metropolis, A. Rosenbluth, M. Rosenbluth, and A. T. und E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics. Band 21*, 1953.

[4] U. Wolff, "Collective monte carlo updating for spin systems," 1989.

[5] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. Englewood Cliffs, NJ: Prentice Hall, 3rd ed ed., 1994.

[6] K. Rummukainen, "Reweighting," 2003.

[7] K. Rummukainen, "Phase transitions and finite size scaling," 2003.

6 Anhang

Ein weiteres Messergebnis zur Autokorrelation und der Julia-Code, welcher in diesem Projekt erstellt wurde.

6.1 Integrierte Autokorrelation

Die Autokorrelation kann auch über die gemessenen Konfigurationen integriert werden. Damit ergibt sich eine integrierte Autokorrelation. Diese kann dann anschaulich in einem Temperaturbereich dargestellt werden. Für den Metropolis-Algorithmus ist ein Peak um die kritische Temperatur zu erwarten, da dort die Autokorrelationszeit divergiert. Für den Wolff-Algorithmus sollte die integrierte Autokorrelationszeit nahe Null sein. Das Ergebnis ist in Abbildung 12 zu sehen.

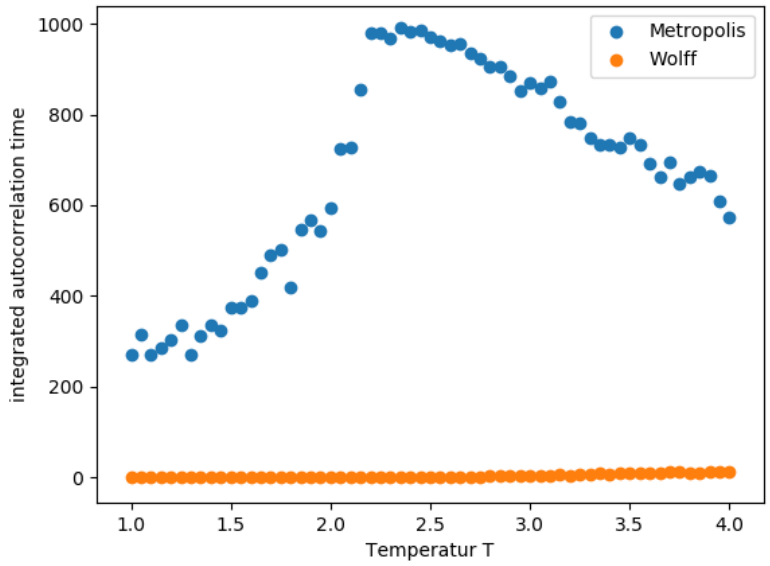

Abbildung 12: Integrierte Autokorrelation für Metropolis- und Wolff-Algorithmus für ein $N = 16x16$ Gitter.

Für den Metropolis-Algorithmus ist ein breiter Peak ab $T = 2$ zu sehen. Dieser ist asymmetrisch, beginnt steil, und fällt ab $T = 2.3$ langsam (linear) fällt.

6.2 Julia-Code

```
using PyPlot, StatsBase, LsqFit

#periodic boundary conditions
function Periodic(shift,size)
    if shift == 0
        shift = size
    elseif shift == size+1
        shift = 1
    end
    return shift
end

#calculate the energy
function energy(S)
    sum(- S .* (circshift(S, (1, 0)) .+ circshift(S, (0, 1))))
end

#metropolis step
function metropolis_step!(S,T)
    N = size(S,1)
    r_x = rand(1:N)
    r_y = rand(1:N)
    delta_E = 2 * S[r_x,r_y] .* (circshift(S, (0, 1)) .+
                  circshift(S, (0, -1)) .+
                  circshift(S, (1, 0)) .+
                  circshift(S, (-1, 0)))
    if delta_E[r_x,r_y] <= 0 || rand() < exp(- (1/(T)) * delta_E[r_x,r_y])
        S[r_x,r_y] = - S[r_x,r_y]
    end
end

#wolf step
function wolff_step!(S,T)
    grow_probability = 1 - exp(-2/T)
    rand_x,rand_y = rand(1:size(S,1)),rand(1:size(S,1))
    stack = [[rand_x,rand_y]]
    value_spin = copy(S[rand_x,rand_y])
    S[rand_x,rand_y] *= -1
```

```
    while !isempty(stack)
        x,y = pop!(stack)
        above = [x,Periodic(y+1,size(S,1))]
        below = [x,Periodic(y-1,size(S,1))]
        left = [Periodic(x-1,size(S,1)),y]
        right = [Periodic(x+1,size(S,1)),y]
        neighbors = [above, below, left, right]
        for i in neighbors
            if S[i[1],i[2]] == value_spin
                if rand() < grow_probability
                    S[i[1],i[2]] *= -1
                    push!(stack,i)
                else
                end
            end
        end
    end
end

#get statistics
function energy_stats(S, T, samples, modus)
    Original = copy(S)
    energy_config = 0
    magnetization_config = 0
    energies_therma = Float64[]
    magnies_therma = Float64[]
    energies = Float64[]
    magnies = Float64[]
    autocorr = Float64[]
    for sample in 1:samples
        if modus == 1
            metropolis_step!(S,T)
        elseif modus == 2
            wolff_step!(S,T)
        end
        energy_config = energy(S)
        magnetization_config = sum(S)
        push!(energies_therma, energy_config)
        push!(magnies_therma, magnetization_config)
        if sample > samples/2
```

```
            push!(energies, energy_config)
            push!(magnies, magnetization_config)
        end
    end
    meanE = mean(energies)
    varEvec = (energies - meanE).^2
    varE = mean(varEvec)
    meanM = mean(magnies)
    varM = mean((i - meanM)^2 for i in magnies)
    autocorr = (autocor(magnies, range(1,1000); demean=true))
    erg = Dict("meanE"=>meanE / length(S),
               "varE"=>varE / length(S),
               "energies"=>energies_therma,
               "magnies"=>magnies_therma,
               "meanM"=>meanM / length(S),
               "varM"=>varM / length(S),
               "autocorr"=>autocorr,
               "varEvec"=> varEvec,
               "Evec" => energies )
    return erg
end

#sweep trough and collect statistics
function sweep(N, Trange, samples, modus)
    S = rand([1.0, -1.0], (N, N))
    meanEhist = Float64[]
    varEhist = Float64[]
    meanMhist = Float64[]
    varMhist = Float64[]
    Cvhist = Float64[]
    Sushist = Float64[]
    autocorr_hist = Float64[]

    for (i, T) in enumerate(Trange)
        erg = energy_stats(S, T, samples, modus)
        push!(meanEhist, erg["meanE"])
        push!(varEhist, erg["varE"])
        push!(meanMhist, erg["meanM"])
        push!(varMhist, erg["varM"])
        push!(autocorr_hist, sum(erg["autocorr"]))
```

```
    end

    Cvhist = varEhist ./ Trange.^2
    Sushist = varMhist ./ Trange

    erg = Dict("meanEhist"=>meanEhist,
               "varEhist"=>varEhist,
               "Cvhist"=>Cvhist,
               "Sushist"=>Sushist,
               "autohist"=>autocorr_hist)
    return erg
end

function plot_scale_e(T_array,sizes,sweeps,modus)
    index = 1
    for size in sizes
        stats_hist = sweep(size, T_array, sweeps[index], modus)
        scatter(T_array,stats_hist["meanEhist"],
                label = "$size x $size")
        index +=1
    end

    legend()
    xlabel("Temperatur T")
    ylabel("Energie pro Spin")
    show()
end

function plot_scale_cv(T_array,sizes,sweeps,modus)
    index = 1
    for size in sizes
        stats_hist = sweep(size, T_array, sweeps[index],modus)
        scatter(T_array,stats_hist["Cvhist"],
                label = "$size x $size")
        index +=1
    end
    legend()
    xlabel("Temperatur T")
    ylabel("Waermekapazitaet Cv pro Spin")
end
```

```
function plot_integrated_auto(T_array,samples,modus)
    size_1 = 16
    stats_hist_1 = sweep(size_1, T_array, samples, modus)
    if modus == 1
        scatter(T_array,stats_hist_1["autohist"],label = "Metropolis")
    elseif modus == 2
        scatter(T_array,stats_hist_1["autohist"],label = "Wolff")
    end
    legend()
    xlabel("Temperatur T")
    ylabel("integrated autocorrelation time")
end

function plot_time_series_images_metro(T)
    N = 50
    samples = 1e6
    S = rand([1.0, -1.0], (N, N))
    for i in 1:samples
        metropolis_step!(S, T)
        if i == samples/80
            subplot(4,3,1)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == samples/40
            subplot(4,3,2)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == samples/32
            subplot(4,3,3)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == samples/25
            subplot(4,3,4)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
```

```
if i == samples/20
    subplot(4,3,5)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
if i == samples/16
    subplot(4,3,6)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
if i == samples/10
    subplot(4,3,7)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
if i == samples/8
    subplot(4,3,8)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
if i == samples/5
    subplot(4,3,9)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
if i == samples/4
    subplot(4,3,10)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
if i == samples/2
    subplot(4,3,11)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
if i == samples
    subplot(4,3,12)
    title("t = $(@sprintf("%d", i))")
    imshow(S, vmin=-1, vmax=1, cmap="Greys")
end
```

```
    end
    tight_layout()
    show()
end

function plot_time_series_images_wolff(T)
    N = 50
    samples = 24
    S = rand([1.0, -1.0], (N, N))
    for i in 1:samples
        wolff_step!(S,T)
        if i == 2
            subplot(4,3,1)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 4
            subplot(4,3,2)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 6
            subplot(4,3,3)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 8
            subplot(4,3,4)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 10
            subplot(4,3,5)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 12
            subplot(4,3,6)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
```

```
        end
        if i == 14
            subplot(4,3,7)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 16
            subplot(4,3,8)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 18
            subplot(4,3,9)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 20
            subplot(4,3,10)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 22
            subplot(4,3,11)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
        if i == 24
            subplot(4,3,12)
            title("t = $(@sprintf("%d", i))")
            imshow(S, vmin=-1, vmax=1, cmap="Greys")
        end
    end
    tight_layout()
    show()
end

function thermalization(T,Ns,samples,modus)
    for N in Ns
        S = rand([1.0, -1.0], (N, N))
        thermo = energy_stats(S, T, samples, modus)
```

```
        plot(abs.(thermo["energies"]/(N*N)),label = "$N x $N")
    end
        legend()
        xlabel("sweeps")
        ylabel("energy per spin")
end

function plot_auto(samples, modus)
    N = 8
    T = 1.0
    S = rand([1.0, -1.0], (N, N))
    thermo = energy_stats(S, T, samples, modus)
    plot(thermo["autocorr"],label = "$T")
    T = 1.7
    S = rand([1.0, -1.0], (N, N))
    thermo = energy_stats(S, T, samples, modus)
    plot(thermo["autocorr"],label = "$T")
    T = 2.2
    S = rand([1.0, -1.0], (N, N))
    thermo = energy_stats(S, T, samples, modus)
    plot(thermo["autocorr"],linestyle = "--",label = "$T")
    T = 2.3
    S = rand([1.0, -1.0], (N, N))
    thermo = energy_stats(S, T, samples, modus)
    plot(thermo["autocorr"],linestyle = "--",label = "$T")
    T = 2.4
    S = rand([1.0, -1.0], (N, N))
    thermo = energy_stats(S, T, samples, modus)
    plot(thermo["autocorr"],linestyle = "--",label = "$T")
    T = 3.0
    S = rand([1.0, -1.0], (N, N))
    thermo = energy_stats(S, T, samples, modus)
    plot(thermo["autocorr"],linestyle = "-.",label = "$T")
    T = 4.0
    S = rand([1.0, -1.0], (N, N))
    thermo = energy_stats(S, T, samples, modus)
    plot(thermo["autocorr"],label = "$T")
    legend()
    xlabel("sweeps");ylabel("autocorrelation")
end
```

```
function plot_rew_hat(N,T,samples,modus,lab)
    S = rand([1.0, -1.0], (N, N))
    stats = energy_stats(S, T, samples, modus)
    varEvec = stats["varEvec"]
    Evec = stats["Evec"]

    T_finer = (T-0.2):0.005:(T+0.3)
    e_finer = zeros(length(T_finer))
    e2_finer = zeros(length(T_finer))

    for i in eachindex(T_finer)
        oben_e = sum(exp.(-(1/T_finer[i]-1/T) .* Evec) .* Evec)
        oben_e2 = sum(exp.(-(1/T_finer[i]-1/T) .* Evec) .* (Evec .^2))
        unten = sum(exp.(-(1/T_finer[i]-1/T) .* Evec))
        e_finer[i] = oben_e / unten
        e2_finer[i] = oben_e2 / unten
    end

    cv_finer = (e2_finer - e_finer.^2) ./ (T^2) ./ N^2
    plot(T_finer,cv_finer,label = lab)
    xlabel("Temperatur T")
    ylabel("Waermekapazitaet Cv pro Spin")
    return T_finer[indmax(cv_finer)]
end

function plot_rew_demo()
    modus = 2
    samples = 10000
    N = 16
    T = 2.2
    lab = "T = $T"
    plot_rew_hat(N,T,samples,modus,lab)

    modus = 2
    samples = 10000
    N = 16
    T = 2.3
    lab = "T = $T"
    plot_rew_hat(N,T,samples,modus,lab)
```

```
    T_array = 1.0:0.02:4.0
    sweeps = [samples]
    sizes = [N]
    modus = 2
    plot_scale_cv(T_array,sizes,sweeps,modus)
    xlabel("Temperatur T")
    ylabel("Waermekapazitaet Cv pro Spin")
    grid()
end

function finite_size_scaling(modus,samples,sizes,T_array)
    subplot(1,2,1)
    max_values = zeros(length(sizes))
    for i in eachindex(sizes)
        size = sizes[i]
        lab = "$size x $size"
        max_values[i] = plot_rew_hat(sizes[i],
                        T_array[i],samples,modus,lab)
    end
    legend()
    subplot(1,2,2)
    rec_lattice = 1 ./ sizes
    rec_temperature = 1 ./ max_values
    scatter(rec_lattice,rec_temperature)

    model(x, p) = p[1] - (p[3]) .* x .^ (p[2])
    p0 = [0.44,0.5,1]
    fit = curve_fit(model, rec_lattice, rec_temperature, p0)
    print(fit.param)
    print(fit)
    beta = round(fit.param[1],3)
    v = round(fit.param[2],3)
    rec_lattice_fit = 0:0.01:0.2
    rec_temperature_fit = model(rec_lattice_fit,fit.param)
    plot(rec_lattice_fit,rec_temperature_fit,
        label = "beta Onsager = $beta und v = $v")
    xlabel("1/L")
    ylabel("beta max")
    legend()
```

```
    tight_layout()
end

#seed the random number generator
srand()

###plot integrated autocorrelation for metro and wolff
#T_array = 1.0:0.05:4.0;samples = 1e6;
#modus = 1
#plot_integrated_auto(T_array,samples,modus)
#modus = 2
#plot_integrated_auto(T_array,samples,modus)
#show()

###plot autocorrelation of wolff-algorithm
#samples = 1e6;
#modus = 2;
#plot_auto(samples,modus)
#xlim([0,10])
#show()

###plot autocorrelation of metropolis-algorithm
#samples = 1e6;
#modus = 1;
#plot_auto(samples,modus)
#show()

###heat_capacity with wolff algorithm
#T_array = 2.0:0.01:3.0
#modus = 2
#sweeps = [10000,10000,10000,10000,10000]
#sizes = [4,8,16,32,64]
#plot_scale_cv(T_array,sizes,sweeps,modus)
#show()

###mean energy with wolff algorithm
#T_array = 1.0:0.05:4.0
#modus = 2
#sweeps = [10000,10000,10000,10000,10000]
#sizes = [4,8,16,32,64]
```

```
#plot_scale_e(T_array,sizes,sweeps,modus)
#show()

###thermalization with wolff algorithm
#T = 1.8
#samples = 1000
#modus = 2
#Ns = [4,8,16,32,64,128,256]
#thermalization(T,Ns,samples,modus)
#xlim([0,1000])
#show()

###thermalization with metropolis algorithm
#T = 1
#samples = 50000
#modus = 1
#Ns = [4,8,16,32]
#thermalization(T,Ns,samples,modus)
#show()

###time series
#T = 1
#plot_time_series_images_metropolis(T)
#T = 1
#plot_time_series_images_wolff(T)

###heat_capacity with wolff algorithm_reweighting
#modus = 2
#samples = 20000
#sizes = [8,16,32,64]
#T_array = [2.45,2.34,2.30,2.28]
#finite_size_scaling(modus,samples,sizes,T_array)
#show()

#plot_rew_demo()
#show()
```