Karl-Franzens Universität Graz
Institut für Physik

Projekt für Computational Physics I

Implementierung und Auswertung des Potts-Modells

Andreas Windisch

Graz, am 22. September 2009

Erklärung

Diese Ausarbeitung entstand im Rahmen der Lehrveranstaltung Computational Physics. Dabei sind die ersten Kapiteln aus der Vorlesungsmitschrift aufgebaut, wie sie Prof. Gattringer präsentiert hat. Im zweiten Abschnitt befindet sich das ausgearbeitete Projekt, welches ebenfalls unter der Anleitung von Prof. Gattringer entwickelt wurde.

Inhaltsverzeichnis

1 Monte-Carlo-Simulation einfacher Verteilungen

1.1 Problemstellung/Beispiele

1.1.1 Spinsysteme (Ising-Modell)

Spinvariable

$$S_{\vec{n}} \in \{-1, +1\}; \quad \vec{n} \in \Lambda. \tag{1}$$

$$\vec{n} = \begin{pmatrix} n_1 \\ n_2 \\ \vdots \\ n_d \end{pmatrix} \tag{2}$$

Dabei ist d die Dimension des Gitters, Λ das Gitter selbst. Ferner ist:

$$H[S] = \frac{\alpha}{2} \sum_{\vec{n}} \sum_{\nu=1}^{d} [1 - S_{\vec{n}} \cdot S_{\vec{n}+\hat{\nu}}] \tag{3}$$

$$\hat{\nu} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow \nu\text{-te Stelle} \tag{4}$$

$\hat{\nu}$ ist dabei der Einheitsvektor in die ν-Richtung.

Grundzustand: Alle Spins sind parallel $\Rightarrow H[S] = 0$.

Ein Spin umgeklappt: $H[S] = 2 \cdot d \cdot \alpha$. Dabei steht S für die Konfiguration.

Die Wahrscheinlichkeit für das Auftreten der Konfiguration S ist dann:

$$P[S] = \frac{1}{Z} \cdot e^{-\frac{H[S]}{k_B \cdot T}} \tag{5}$$

mit $k_B = 1.3806505(24) \cdot 10^{-23} J/K$ Boltzmannkonstante und T der Temperatur in Kelvin.

1.

$$T \to 0 \Rightarrow \frac{1}{T} \to \infty \Rightarrow e^{-\frac{H[S]}{k_B \cdot T}} \to 0 \text{ außer} \quad H = 0. \tag{6}$$

Im Limes $T \to 0$ hat nur der Grundzustand eine Wahrscheinlichkeit die ungleich 0 ist.

2.

$$\to \infty \Rightarrow \frac{1}{T} \to 0 \Rightarrow e^{-\frac{H[S]}{k_B \cdot T}} \to e^0 = 1 \quad \forall H[S] \tag{7}$$

Der Boltzmannfaktor liefert keine Einschränkung mehr und alle Zustände können angeregt werden.
Boltzmannfaktor:

$$\text{Wahrscheinlichkeit } P[S] = \frac{1}{Z} \cdot e^{-\frac{H[S]}{k_B \cdot T}} \tag{8}$$

$$\text{mit der Zustandssumme } Z = \sum_{S'} e^{-\frac{H[S']}{[k_B \cdot T}} \tag{9}$$

(Summe über alle Konfigurationen.)

Nun muss Überprüft werden ob $P[S]$ eine Wahrscheinlichkeit ist:

1.

$$0 \leq P[S] \leq 1 \quad \checkmark \tag{10}$$

2.

$$\sum_S P[S] = 1 \tag{11}$$

$$\sum_S P[S] = \frac{1}{Z} \cdot \sum_S e^{-\frac{H[S]}{k_B \cdot T}} = \frac{1}{Z} \cdot Z = 1 \quad \checkmark \tag{12}$$

1.1.2 Erwartungswerte und Observablen

Die Magnetisierung $M[S]$ ist:

$$M[S] = \sum_{\vec{n} \in \Lambda} S_{\vec{n}} = \#\uparrow \text{ Spins} - \#\downarrow \text{ Spins} \tag{13}$$

Der thermische Erwartungswert ist dann:

$$\langle M \rangle = \sum_S M[S] \cdot P[S] = \frac{1}{Z} \cdot \sum_S M[S] \cdot e^{-\frac{H[S]}{k_B \cdot T}} \tag{14}$$

Im Folgenden soll eine kleine Abschätzung zeigen warum eine Monte-Carlo-Simulation eines solchen Problems sinnvoll ist.
Sei ein winziger Magnet als Kubus ausgeführt und weise Kantenlänge $L = 100$ auf.
Wir wollen nun die Anzahl der möglichen Konfigurationen abschätzen:

$$L = 100 \Rightarrow L^3 = 10^6 \tag{15}$$

$$2^{10^6} = 2^{10 \cdot 10^5} = (1024)^{100000} \approx 10^{3 \cdot 100000} = 10^{300000} \tag{16}$$

Wir fassen dies nun in Worte:

- Typische Summen in der statistischen Physik sind viel zu groß um komplett summiert zu werden. Man kann unter Verwendung von Zufallsprozessen (Monte Carlo) die wichtigsten Konfigurationen berücksichtigen. ⇒ sehr genaue Approximation $\langle M \rangle$

- systematisch verbesserbar

- Aufgabe: Erzeuge Konfigurationen mit Wahrscheinlichkeit $P[S]$

1.1.3 Beispiel: QCD (Quanten Chromodynamik) auf dem Raum-Zeit-Gitter

Gitterkonstante liegt im Bereich $a = 0.1fm = 0.1 \cdot 10^{-15}m$. Wir betrachten das Feynmansche Pfadintegral (Integral über 10^6 bis 10^9 Variablen):

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \prod_{\vec{n} \in \Lambda} dQ_{\vec{n}} \cdot dG_{\vec{n}} \cdot \mathcal{O}[Q, G] \cdot e^{-S[Q,G]} \tag{17}$$

Dabei sind $Q_{\vec{n}}$ Quarkfelder, $G_{\vec{n}}$ Gluonfelder und $S[Q, G]$ ist die Wirkung. Auch in diesem Fall wird Approximiert. Das Integral wird durch zufällig gewählte Konfigurationen mit Verteilung $\propto e^{-S[Q,G]}$ genähert.

1.1.4 Beispiel: Optimierungsproblem (Traveling Salesman Problem)

Auch dieses bekannte Problem lässt sich in guter Näherung mit einem Monte-Carlo-Prozess lösen. Die Problemstellung ist schnell skizziert:
Man denke sich n Städte und einen reisehungrigen Staubsaugervertreter. Dieser will so schnell wie möglich alle n Städte auf einer Rundreise besuchen und ist aus Gründen der Effizienz darauf bedacht die schnellste Route zu nehmen. Dies ist jedoch mit großem n ein Problem, welches sich nicht leicht lösen läßt. Denn:
Die Anzahl der Möglichkeiten explodiert förmlich mit $\propto n!$.
Bei einer möglichen Formulierung dieses Problems sieht die Wahrscheinlichkeitsverteilung etwa so aus:

$$P[S] = \frac{1}{Z} \cdot e^{-\frac{K(S)}{T}} \tag{18}$$

Dabei ist $K(S)$ die Kostenfunktion (etwa die Länge des Weges) und S eine Konfiguration (etwa Anordnung der Städte). Senkt man nun bei diesem stochastischen Prozess die Temperatur T langsam ab ($T \to 0$), so bleibt der Prozess bei einer guten Lösung hängen.

1.2 Simple Sampling, Reweighting

1.2.1 Simple Sampling

Sei S Konfiguration, $P[S]$ die Wahrscheinlichkeit mit der die Konfiguration S auftritt.

$$\langle \mathcal{O} \rangle = \sum_S \mathcal{O}[S] \cdot P[S] \tag{19}$$

Monte-Carlo-Strategie:

Ersetze die Summe über alle Konfigurationen durch die Summe über einige wenige repräsentative Konfigurationen $S^{(i)}, i = 1, \ldots, N$.

$S^{(i)}$ gleichverteilt:

$$\langle \mathcal{O} \rangle_N = \frac{\sum_{i=1}^N \mathcal{O}[S^{(i)}] \cdot P[S^{(i)}]}{\sum_{i=1}^N P[S^{(i)}]} \quad \text{Simple Sampling} \tag{20}$$

Problem: Zufällig gewählte (gleichverteilte) Konfigurationen haben meist sehr kleines $P[S^{(i)}] \Rightarrow$ keine repräsentative Auswahl an physikalisch wichtigen Konfigurationen.

$S^{(i)}$ verteilt mit Wahrscheinlichkeit $P[S^{(i)}]$:

$$\langle \mathcal{O} \rangle_N = \frac{1}{N} \sum_{i=1}^N \mathcal{O}[S^{(i)}] \quad \text{Importance Sampling} \tag{21}$$

Wahrscheinlichkeit für Konfiguration: $S^{(i)} = P[S^{(i)}]$
Für den Limes gilt dann:

$$\lim_{N \to \infty} \langle \mathcal{O} \rangle_N = \langle \mathcal{O} \rangle \tag{22}$$

Besseres Simple Sampling:

Bei der Erzeugung der Konfigurationen S wird der Gewichtsfaktor $P[S]$ nicht verwendet.

Mögliche Implementation:

- Erzeuge eine Konfiguration S zufällig
- Berechne für diese S die $P[S]$
- Akzeptiere die Konfiguration mit Wahrscheinlichkeit $P[S]$, dh. wenn $r \leq P[S]$, wobei $r \in [0, 1]$ gleichverteilte Zufallszahl
- $\langle \mathcal{O} \rangle_N = \frac{1}{N} \sum_{i=1}^N \mathcal{O}[S^{(i)}]$, mit $S^{(i)}$ akzeptierte Konfigurationen

1.2.2 Übung: Ein Experiment zu Simple Sampling

Gegeben sei folgende Problemstellung:

Sei $(x, y) \in \mathbb{R}^2$ und $P(x, y) = \frac{1}{2\pi} \cdot e^{-\frac{x^2+y^2}{2}}$

Der mittlere quadratische Abstand ist dann gegeben durch:

$$\langle r^2 \rangle = \langle x^2 + y^2 \rangle = \int d^2x(x^2 + y^2) \cdot P(x, y) = \frac{1}{2\pi} \int d^2x(x^2 + y^2) \cdot e^{-\frac{x^2+y^2}{2}}$$

Es soll nun versucht werden, das Ergebnis $\langle r^2 \rangle = 2$ mit der Methode des Simple Sampling zu reproduzieren.

Dabei soll folgendermaßen vorgegangen werden:

- Zahlenpaar (x, y) zufällig bestimmen
- Zahlenpaar (x, y) akzeptieren, falls $\rho \leq P(x, y)$, ρ Zufallszahl zwischen 0 und 1
- $r^2 = x^2 + y^2$ über die akzeptierten Paare (x, y) mitteln $\Rightarrow \langle r^2 \rangle$

Der C++-Code dieses Problems sieht so aus:

```
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <time.h>

int main (void)
{
ofstream fout("gauss.dat");
srand( time (NULL));
double x;//Zufallsvariablen
double y;
double rho; //Akzeptanzkriterium
double p; //Wahrscheinlichkeit fuer ein Paar (x,y)
const float L = 4.; //Intervallgrenze fuer Zufallszahlen
const int loop = 1000000;//Anzahl der Durchlaeufe
float n = 0; //Zaehlvariable
double rsquare = 0; //Observable

cout << "Dieses Programm berechnet den mittleren quadratischen\n";
cout << "Abstand fuer ein zweidimensionales Gausspaket.\n";
cout << "Zur Berechnung kommt ein Monte Carlo Algorithmus mit \n";
cout << "Simple Sampling zum Einsatz.\n";

for ( int i=0; i<loop; i++)
{
x = -L + (2*L)*rand() / (RAND_MAX + 1. );
y = -L + (2*L)*rand() / (RAND_MAX + 1. );
rho = rand()/ ((float)RAND_MAX + 1.);
p = 1/(2*M_PI)*exp(-(x*x + y*y)/2);

if( rho <= p)
{
```

```
        rsquare = rsquare + x*x + y*y;
        n++;
        fout << x << " " << y << "\n";
}
}

rsquare = rsquare / n;
cout << "rsquare: " << rsquare << "\t" << n << " akzeptierte Werte von "
     << loop << " Durchlaeufen\n" ;
return 0;
}
```

Wertet man dies nun graphisch aus (hier für L=4), so findet man:

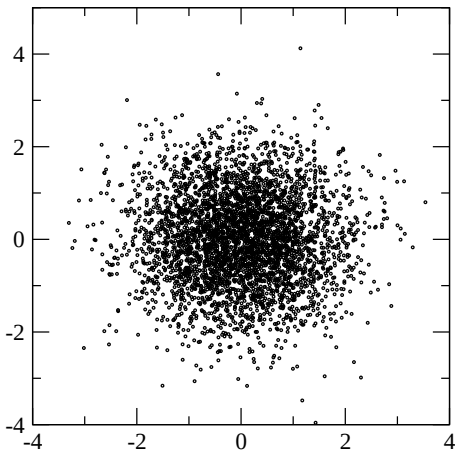


Abbildung 1: Gausspaket in zwei Dimensionen, genähert durch Simple Sampling

1.3 Importance Sampling

Die Verteilungsfunktion $P[S]$ wird direkt bei der Erzeugung der Konfiguration $S^{(i)}$ verwendet.

Stochastischer Prozess: (Markov-Kette)

Man geht schrittweise auf einem Pfad von Konfigurationen durch den Raum aller Konfigurationen. $S^{(t)}$, mit t der sogenannten Monte-Carlo-Zeit.
Die Konfiguration $S^{(t+1)}$ wird dabei durch einen Zufallsprozess aus der Konfiguration $S^{(t)}$ erzeugt. Der Zufallsprozess ist durch Übergangswahrscheinlichkeiten definiert.

$$W(S^{(t+1)} = S' | S^{(t)} = S) = W(S \to S') \tag{23}$$

Auf der linken Seite finden wir die bedingte Wahrscheinlichkeit, auf der rechten formuliert als Übergangswahrscheinlichkeit.
Alle Punkte im Konfigurationsraum sind durch Übergangswahrscheinlichkeiten verknüpft.
$W(S \to S')$ ist Wahrscheinlichkeit:

$$0 \leq W(S \to S') \leq 1 \tag{24}$$

$$\sum_{S'} W(S \to S') = 1 \tag{25}$$

Wir wollen nun die Übergangswahrscheinlichkeit $W(S \to S')$ so konstruieren, dass die Konfigurationen $S^{(t)}$ gemäß $P[S^{(t)}]$ verteilt sind.

1.3.1 Bilanzgleichung

Die summierte Wahrscheinlichkeit über Prozesse die im Schritt $t \to t+1$ das System in den Punkt S' hineinführen muss gleich der summierten Wahrscheinlichkeit über Prozesse die im Schritt $t \to t+1$ aus S' herausführen sein:

$$\sum_S P[S] \cdot W(S \to S') = \sum_S P[S'] \cdot W(S' \to S) \tag{26}$$

(Vgl.: Vektor auf Matrixmultiplikation)
$W(S \to S')$ bildet Wahrscheinlichkeiten aufeinander ab:

$$\tilde{P}[S] = \sum_S P[S] \cdot W(S \to S') \tag{27}$$

z.z.: $\tilde{P}[S']$ ist Wahrscheinlichkeit:

$$0 \leq \tilde{P}[S'] \leq 1 \quad \checkmark \tag{28}$$

$$1 = \sum_{S'} \tilde{P}[S'] = \sum_S \sum_{S'} P[S] \cdot W(S \to S') \tag{29}$$
$$= \sum_S P[S] \sum_{S'} W(S \to S') = \sum_S P[S] = 1 \quad \checkmark$$

Interpretation der Bilanzgleichung:

$$\sum_S P[S] \cdot W(S \to S') = P[S'] \sum_S W(S' \to S) = P[S'] \tag{30}$$

$$\sum_S P[S] \cdot W(S \to S') = P[S'] \quad \text{ist Fixpunktgleichung} \tag{31}$$

Bilanzgleichung: $P[S]$ ist ein Fixpunkt unserer Abbildung, dh. ein Fixpunkt der Markov-Kette. Man kann zeigen, dass der Fixpunkt attraktiv ist, der Markov-Prozess läuft also in die gewünschte Verteilung $P[S]$ hinein wenn die Bilanzgleichung erfüllt ist.

Lösung der Bilanzgleichung:

$$\sum_S [P[S] \cdot W(S \to S') - P[S'] \cdot W(S' \to S)] = 0 \tag{32}$$

Eine hinreichende Lösung ist gegeben wenn man fordert dass der Klammerausdruck gleich 0 ist, dh.:

$$P[S] \cdot W(S \to S') = P[S'] \cdot W(S' \to S) \tag{33}$$

Dies nennt man auch 'detailed balance equation'.

1.3.2 Metropolisalgorithmus (1953)

1. Die Markovkette steht bei einer Konfiguration $S^{(t)} = S$
2. Erzeuge eine Angebotskonfiguration $\tilde{S}$ 'in der Nähe' von S
3. Berechne eine Größe $\rho = \frac{P[\tilde{S}]}{P[S]} \in [0, \infty)$
4. Fall 1: $\rho \geq 1$: akzeptiere $\tilde{S}$ als neue Konfiguration $S' = S^{(t+1)}$
   Fall 2: $\rho < 1$: akzeptiere $\tilde{S}$ als neue Konfiguration $S' = S^{(t+q)}$ nur mit Wahrscheinlichkeit ρ. Sonst: $S' = S = S^{(t+1)}$. Bemerkung: In der Praxis ist r Zufallszahl, $r \in [0, 1)$ gleichverteilt, akzeptiere falls $r \leq \rho$.
5. $t \to t + 1$, goto 1

1.3.3 Beweis von 'detailed balance' für Metropolis

z.z.:$P[S] \cdot W(S \to S') = P[S'] \cdot W(S' \to S)$

Fall A: $P[S'] < P[S]$

$$\rho = \frac{P[S']}{P[S]} < 1 \tag{34}$$

$$W(S \to S') = \rho \tag{35}$$

$$W(S' \to S) = 1 \tag{36}$$

$$\underline{P[S] \cdot W(S \to S')} = P[S] \cdot \rho = P[S] \cdot \frac{P[S']}{P[S]} = P[S'] \cdot 1 = \underline{P[S'] \cdot W(S' \to S)} \quad \checkmark \tag{37}$$

Fall B: $P[S'] > P[S]$

Wie Fall A, mit $S \to S', \quad S' \to S$.

□

Bemerkungen:

- $\tilde{S}$ in der Nähe von S, sonst ist zumeist $\rho << 1$ ⇒Akzeptanz zu gering
- Oft verwendet man Symmetrien des Systems um $\tilde{S}$ zu erzeugen
- Ergodizität: Die Angebotskonfigurationen müssen so konstruiert werden, dass jede Konfiguration in endlich vielen Schritten erreicht werden kann
- Am Anfang benötigt das System eine Anzahl von Equilibrierungsschritten um in die Gleichgewichtsverteilung zu laufen

1.4 Monte-Carlo-Simulation des Potts-Modells

Λ sei Gitter (regulär) in d Dimensionen.

$$\vec{n} = \begin{pmatrix} n_1 \\ \vdots \\ n_d \end{pmatrix} \tag{38}$$

$$n_\nu = 1, \ldots, L \quad \text{und} \quad \nu = 1, \ldots, d;$$

Spineinstellungen $S_{\vec{n}} \in \{1, 2, 3, \ldots, q\}$ liefert ein q-state-Potts-Modell.

1.4.1 Hamiltonfunktion

$$H[S] = \alpha \sum_{n \in \Lambda} \sum_{\nu=1}^{d} [1 - \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}})] + \mu \sum_{\vec{n} \in \Lambda} [1 - \delta(S_{\vec{n}}, 1)] \tag{39}$$

Dabei ist

$$\delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}}) = \begin{cases} 1 & S_{\vec{n}} = S_{\vec{n}+\hat{\nu}} \\ 0 & \text{sonst} \end{cases} \tag{40}$$

das gute alte Kronecker-Delta, α ist die Kopplungsstärke der Nachbarn, μ repräsentiert das externe Magnetfeld. Ferner sind $\alpha, \mu \in \mathbb{R}$.

Grundzustand:

$S_{\vec{n}} = 1 \quad \forall \vec{n} \in \Lambda, \quad \mathcal{E} = 0$ ist eindeutig

Erster angeregter Zustand:

Ein Spin ist ungleich 1.
$\mathcal{E} = 2d \cdot \alpha + \mu, \quad V = L^d, \quad L^d \cdot (q-1)$ entartet.

Zweiter angeregter Zustand

$\vdots$

führt zur sogenannten Tieftemperaturentwicklung:
Diese ist eine Liste der niedrigsten Anregungen und Entartungen.
Zustandssumme:

$$Z = \sum_S e^{-\frac{H[S]}{k_B \cdot T}} = 1 + L^d \cdot (q-1) \cdot e^{-\frac{2d \cdot \alpha + \mu}{k_B \cdot T}} + \ldots \tag{41}$$

Anzahl der Konfigurationen: q^V

2 Berechnung thermischer Erwartungswerte, Implementierung, Beispiele

2.1 Observable

$$P[S] = \frac{1}{Z} \cdot e^{-\beta \cdot H[S]} \tag{42}$$

mit $\beta = \frac{1}{k_B \cdot T}$, $T \ldots$ Temperatur in Kelvin und $k_B \ldots$ Boltzmannkonstante.
Wieder ist

$$Z = \sum_S e^{-\beta \cdot H[S]} \tag{43}$$

Die Observable ist dann:

$$\langle \mathcal{O} \rangle_T = \sum_S \mathcal{O}[S] \cdot P[S] = \frac{1}{Z} \sum_S e^{-\beta \cdot H[S]} \cdot \mathcal{O}[S] \tag{44}$$

2.1.1 Mögliche Observable

$$M_{q_0[S]} = \sum_{\vec{n} \in \Lambda} \delta(S_{\vec{n}}, q_0) \quad \text{mit} \quad 1 \leq q_0 \leq q \tag{45}$$

Diese Observable entspäche (bei vorgegebenem q_0) der Anzahl jener Spins, die die Einstellung q_0 besitzen.

$$C_{q_0(\vec{r})} = \frac{1}{V} \sum_{\vec{n} \in \Lambda} \delta(S_{\vec{n}}, q_0) \cdot \delta(S_{\vec{n}+\vec{r}}, q_0) \tag{46}$$

ist Korrelationsfunktion: diese liefert nur einen Beitrag wenn zwei um den Vektor $\vec{r}$ entfernte Spins die gleiche Einstellung haben.
Notation:

$$\beta \cdot H[S] = \beta \cdot \alpha \sum_{\vec{n}} \sum_{\nu=1}^{d} [1 - \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}}] + \beta \cdot \mu \sum_{\vec{n}} [1 - \delta(S_{\vec{n}}, 1)$$

$$= \beta \cdot \alpha \cdot V \cdot d - \beta \cdot \alpha \sum_{\vec{n}} \sum_{\nu} \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}}) + \beta \cdot \mu \cdot V - \beta \cdot \mu \sum_{\vec{n}} \delta(S_{\vec{n}}, 1) \tag{47}$$

$$= C - \underbrace{J \sum_{\vec{n}} \sum_{\nu} \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}}) - M \sum_{\vec{n}} \delta(S_{\vec{n}}, 1)}_{\tilde{H}[S]}$$

mit $J = \alpha \cdot \beta$ und $M = \mu \cdot \beta$.
Weiters:

$$Z = \sum_S e^{-\beta \cdot H[S]} = \sum_S e^{-C - \tilde{H}[S]} = e^{-C} \sum_S e^{-\tilde{H}[S]} = \underline{e^{-C} \cdot \tilde{Z}} \tag{48}$$

So folgt:

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \sum_S e^{-C-\tilde{H}[S]} \cdot \mathcal{O}[S] = \frac{1}{e^{-C} \cdot \tilde{Z}} \cdot e^{-C} \sum_S e^{-\tilde{H}[S]} \cdot \mathcal{O}[S] \tag{49}$$

und damit schließlich:

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \sum_S e^{-\tilde{H}[S]} \cdot \mathcal{O}[S] \tag{50}$$

C wird weggelassen: $\tilde{H}$ liefert dasselbe Ergebnis wie H (Vgl.: freie Wahl des Nullpunkts)

2.2 Monte-Carlo-Simulation

Wie bereits im vorigen Kapitel besprochen gilt es, Konfigurationen $S^{(t)}$ mit $t = 1, 2, \ldots, N$ zu erzeugen, mit einer Verteilung

$$P[S^{(t)}] = \frac{1}{Z} e^{-\beta \cdot H[S^{(t)}]} \tag{51}$$

Für die Observablen gilt dann:

$$\langle \mathcal{O} \rangle_N = \frac{1}{N} \sum_{t=1}^{N} \mathcal{O}[S^{(t)}] \tag{52}$$

Im Limes gilt:

$$\lim_{N \to \infty} \langle \mathcal{O} \rangle_N = \langle \mathcal{O} \rangle \tag{53}$$

2.2.1 Metropolisalgorithmus für das Potts-Modell

Wir haben:

$$P[S] = e^{-\beta \cdot H} = e^{J \sum_{\vec{n}} \sum_{\nu=1}^{d} \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}}) + M \sum_{\vec{n}} \delta(S_{\vec{n}}, 1)} \tag{54}$$

$\frac{1}{Z}$ kann weggelassen werden.

1. Wir wählen einen Gitterpunkt $\vec{n}_0$

2. Angebotskonfiguration: $\tilde{S}_{\vec{n}} = S_{\vec{n}} \quad \forall \vec{n} \neq \vec{n}_0, \quad \tilde{S}_{\vec{n}_0} \in \{1, 2, \ldots, q\}$ zufällig

3. Berechne $\rho = \frac{P[\tilde{S}]}{P[S]} = \frac{e^{-\beta \cdot H[\tilde{S}]}}{e^{-\beta \cdot H[S]}} = e^{-\beta [H[\tilde{S}] - H[S]]}$, ist nur lokal, dh. nur Nachbarn tragen bei, Rest fällt weg.

$$\rho = e^{J \sum_{\nu=1}^{d} [\delta(\tilde{S}_{\vec{n}_0}, \tilde{S}_{\vec{n}_0+\hat{\nu}}) + \delta(\tilde{S}_{\vec{n}_0}, \tilde{S}_{\vec{n}_0-\hat{\nu}}) - \delta(S_{\vec{n}_0}, S_{\vec{n}_0+\hat{\nu}}) - \delta(S_{\vec{n}_0}, S_{\vec{n}_0-\hat{\nu}})] + M[\delta(\tilde{S}_{\vec{n}_0}, 1) - \delta(S_{\vec{n}_0}, 1)]] \tag{55}$$

4. Akzeptiere, wenn $r \leq \rho, \quad r \in [0, 1]$ gleichverteilte Zufallszahl

5. goto 1

Bemerkungen:

- Periodische Randbedingungen:

  → reduzieren finite-size-Effekte

  → lassen wichtige Symmetrien intakt

- Für $\vec{n}_0$ geht man oft systematisch durch das Gitter. Ein Rundgang durch das Gitter bezeichnet man als einen 'sweep'.

- even/odd-Aufteilung bei Parallelcomputern

- Bevor wir beginnen Observable auszuwerten müssen sweeps zur Equilibrierung des Systems durchgeführt werden. Die Equilibrierungsdauer hängt von der Observable ab.

- Zwischen zwei Messungen, dh. Berechnungen von $\mathcal{O}[S^{(t)}]$, werden viele Zwischenupdates eingeschoben (viele sweeps).

2.3 Aufgerollte Indizes

Wir gehen aus von einem d-Dimensionalen Gitter.

$$\vec{n} = \begin{pmatrix} n_1 \\ \vdots \\ n_d \end{pmatrix}, \quad n_i = 0, 1, \ldots, L-1 \tag{56}$$

Der aufgerollte Index ist nun folgender Gestalt:

$$i = n_1 + n_2 \cdot l + n_3 \cdot l^2 + \ldots + n_d \cdot l^{d-1}, \quad i = 0, 1, 2, \ldots, l^d - 1 \tag{57}$$

Die Stärke dieses Konzeptes liegt vor allem darin, Objekte die viele Indizes besitzen leichter handlebar zu machen. In unserem Falle dominiert also der Lern- und Kennenlerneffekt. Halten wir nun für unseren konkreten Fall des Pottsmodells die Anwendung dieses Konzeptes auf die Nachbarn eines jeden Gitterpunktes in weniger konkretem Pseudo-Code fest: (z.B. $d = 2$)

$for(n_1 = 0 \to L-1)$
$for(n_2 = 0 \to L-1)$
$n_{1P} = n_1 + 1$
$n_{1M} = n_1 - 1$
$n_{2P} = n_2 + 1$
$n_{2M} = n_2 11$
$if(n_{1P} == L) \quad n_{1P} = 0$
$if(n_{1M} == -1) \quad n_{1M} = L - 1$
$if(n_{2P} == L) \quad n_{2P} = 0$
$if(n_{2M} == -1) \quad n_{2M} = L - 1$

$i = n_1 + l \cdot n_2 \quad //zentralerPunkt$
$neib[i][0] = n_{1P} + l \cdot n_2 \quad //oestl.Nachbar$
$neib[i][1] = n_1 + l \cdot n_{2P} \quad //noerd.Nachbar$
$neib[i][2] = n_{1M} + l \cdot n_2 \quad //westl.Nachbar$
$neib[i][3] = n_1 + l \cdot n_{2M} \quad //suedl.Nachbar$

$endfor$
$endfor$

Damit finden wir:

$$\sum_{\nu=1}^{d}[\delta(S_{\vec{n}_0}, S_{\vec{n}_0+\hat{\nu}}) + \delta(S_{\vec{n}_0}, S_{\vec{n}_0-\hat{\nu}})] \Rightarrow \sum_{k=0}^{3}\delta(S[i_0], S[neib[i_0][k]]), \quad \text{mit} \quad i_0 \ldots \text{aufgerollter Index} \tag{58}$$

2.4 Die Struktur des Programms

2.4.1 Unterprogramme

1. $fillspins(int \quad S[V])$

   hotstart: Zufallsverteilte Spineinstellungen

   coldstart: alle Spins auf 1

2. $sweeps(int \quad S[V], \quad double \quad J, M, neib, \quad int \quad nsweeps)$
   $for(n = 0 \rightarrow nsweeps - 1)$
   $for(i = 0 \rightarrow V - 1)$
   $\tilde{S} = rand(1, \ldots, q)$
   $Metropolis$
   $endfor$
   $endfor$

   macht einen $nsweeps$ sweep

3. $fillneib(int \quad neib[V][2d])$

   Lernt Nachbarn kennen, Vgl. Kapitel Aufgerollte Indizes

2.4.2 Aufbau des Hauptprogramms

- Einlesen der Parameter $\leftarrow L, J, M, nequi, nmeas, nskip, \ldots$

  $\rightarrow$ eventuell alte Konfiguration
  $\rightarrow$ eventuell Status des Random Generators

- Initialisieren

  Nachbarfelder$\leftarrow fillneib$

  Spinkonfiguration (eventuell alte Konfiguration$\leftarrow fillspins$)

eventuell Zufallszahlengenerator initialisieren

- Equilibrieren

  $sweeps(nequi)$ →nicht notwendig wenn alte Konfiguration vorliegt

- Messen und Zwischenupdates

  $for(n = 0 \rightarrow nmeas - 1)$

  $sweeps(nskip)$ $//Schrittweite$ im $Konfigurationsraum$

  $Observable$ $berechnen$ und $schreiben$

  $endfor$

- Beenden des Programmes

  Konfiguration ausschreiben

  Random Generator Status ausschreiben

  Files schließen, etc...

2.5 Weitere Anwendungen der Monte-Carlo Methoden

2.5.1 Kontinuierliche Spinsysteme

Sei $\vec{n} \in \Lambda$ d-dimensionales Gitter.
$\vec{S}_n$ seien $\vec{N}$ komponentige Vektoren mit $|\vec{S}_n| = 1$

$$\vec{n} = \begin{pmatrix} x \\ y \end{pmatrix}, x^2 + y^2 = 1 \Rightarrow S^1 \tag{59}$$

$$H = \alpha \sum_{\vec{n} \in \Lambda} \sum_{\nu=1}^{d} [1 - \vec{S}_{\vec{n}} \cdot \vec{S}_{\vec{n}+\hat{\nu}}] + \mu \sum_{\vec{n}} [1 - \vec{S}_{\vec{n}} \cdot \vec{l}_d] \tag{60}$$

Auch hier ist der Grundzustand 0. Die erste Anregung ist jedoch nicht wie beim diskreten Modell. Sie ist beliebig gering, da aufgrund der Kontinuität ein Spin beliebig wenig ausgelenkt werden kann.
Die Monte-Carlo-Simulation sähe dann etwa so aus:

- Gehe durch das Gitter

- Angebotskonfiguration: einen Spin ändern:

  $\vec{S} \rightarrow \tilde{\vec{S}}_{\vec{n}}$, $\tilde{\vec{S}}$ soll nur gering abweichen.

- Variante 1:

$$\tilde{\vec{S}} = M \cdot \vec{S}, \quad M = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \quad \text{Drehmatrix} \tag{61}$$

dh. $M^T = M^{-1}$, und $\phi \in [-\epsilon, \epsilon]$ zufällig wählen, $r \in [0,1]$ zufällig
z.B. $\phi = (r - \frac{1}{2}) \cdot 2\epsilon$

- Variante 2:

Addition eines Zufallsvektors:

$$\tilde{\vec{S}} = \frac{\vec{S} + 2\epsilon(r - \frac{1}{2})\vec{e}_k}{|\vec{S} + 2\epsilon(r - \frac{1}{2})\vec{e}_k|}, \quad k = 1, \ldots, d \quad \text{zufällig} \tag{62}$$

In beiden Varianten ist ϵ ein freier Parameter der die Abweichung vom alten Spin $\vec{S}$ steuert. Dies bedeutet, dass wenn ϵ klein ist, ist die Akzeptanz hoch und die Markov-Schritte sind klein, andererseits, wenn ϵ groß ist, ist die Akzeptanz gering, die Markov-Schritte jedoch sind groß.

2.5.2 Gitter-Feldtheorien

z.B. skalares Feld $\Phi(\vec{x}, t) = \Phi(x)$, mit x 4-Vektor
$\vec{B}, \vec{E}(\vec{x}, t), A_\mu(\vec{x}, t)$
Betrachten die Wirkung der Φ^4-Feldtheorie:

$$S = \int d^4x \underbrace{[-\frac{1}{2}\Phi(x)\Delta\Phi(x) + \frac{m^2}{2}\Phi^2(x) + \frac{g}{4}\Phi^4(x)]}_{\mathcal{L}} \tag{63}$$

Dabei ist der erste Teil der kinetische, der mittlere Masse- und der letzte Selbstwechselwirkungsterm.
Euler-Lagrange-Gleichungen:

$$\frac{\partial}{\partial t}\frac{\delta\mathcal{L}}{\delta\dot{\Phi}(x)} - \frac{\delta\mathcal{L}}{\delta\Phi(x)} = 0 \quad \text{mit} \quad \delta \ldots \text{Variationsableitung} \tag{64}$$

$$\Rightarrow -\Delta\Phi(x) + m^2\Phi(x) + g\Phi^3(x) = 0 \quad \text{Klein-Gordon-Gleichung} \tag{65}$$

Die Klein-Gordon-Gleichung ist invariant unter Lorentz-Transformationen.

Pfadintegralquantisierung:

$$\langle\mathcal{O}\rangle = \frac{1}{Z}\int D[\Phi] \cdot e^{-S[\Phi]}\mathcal{O}[\Phi] \quad \text{mit} \quad D[\Phi] \ldots \text{Integration über Feldkonfigurationen} \tag{66}$$

$$\sim \prod_{\vec{x}\in\mathbb{R}^3}\prod_{t\in\mathbb{R}} d\Phi(\vec{x}, t) \tag{67}$$

Die zentrale Idee das Pfadintegral korrekt zu definieren:
4-dimensionales Gitter:

$$x = a \cdot n, \quad n = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{pmatrix}, \quad n_i = 1, 2, \ldots, L \quad \text{und} \quad a \ldots \text{Gitterkonstante} \tag{68}$$

$$D[\Phi] = \prod_{n \in \Lambda} d\Phi(an) \quad \text{wohldefiniert} \quad \checkmark \tag{69}$$

Die Wirkung S wird diskretisiert:

$$S = a^4 \sum_{n \in \Lambda} [-\Phi(an) \underbrace{\sum_{\nu=1}^{d} \frac{\Phi(a(n+\hat{\nu})) - 2\Phi(an) + \Phi(a(n-\hat{\nu}))}{a^2}}_{diskretisierter \Delta} + \frac{1}{2} m^2 \Phi^2(an) + \frac{g}{4} \Phi^4(an)] \tag{70}$$

Wir haben nun ein System mit derselben Struktur wie ein Spinsystem ⇒Monte-Carlo-Methoden

Metropolis Update:

- Gehe systematisch durch das Gitter
- erzeuge Angebotskonfiguration $\tilde{\Phi}(an) = \Phi(an) + \epsilon(2r - 1)$, ϵ random
- $\rho = \frac{e^{-\tilde{S}}}{e^{-S}}$, akzeptiere wenn random $< \rho$

2.5.3 Simulated Annealing bei Optimierungsproblemen

z.B. Traveling Salesman Problem
Dieses Beispiel wurde schon im vorigen Kapitel aufgegriffen und soll nun ein wenig präzisiert werden. Relevant ist eine solche Fragestellung u.a. auch wenn es darum geht, Komponenten auf einem Chip optimal anzuordnen. (Anmerkung: Annealing ist ein Begriff aus der Stahlindustrie und bedeutet 'anlassen', bzw. 'tempern')
Wir erinnern uns:
Wir haben $(N-1)!$ Möglichkeiten (weil der Startwert fix vorgegeben ist).

Simulated Annealing:

- kanonisches Ensemble:
$$P[S] = e^{\frac{1}{k_B \cdot T} \cdot L[S]} \tag{71}$$
- $S \ldots$ Liste der Städte in der Reihenfolge in der sie besucht werden. z.B. 500 Städte, numeriert von 1 bis 500: Eine mögliche Liste wäre dann etwa: $S = 1, 347, 17, 13, 212, 416, 91, 42, 15, \ldots, 7, (1)$.

499! Möglichkeiten (Listen). Ordne nun der Liste die Länge $L(S)$ (Kostenfunktion) der entsprechenden Rundreise zu:

$$P[S] = e^{-\frac{1}{k_B \cdot T} \cdot L[S]} \tag{72}$$

Kurze Rundwege haben dann eine hohe Wahrscheinlichkeit.

Monte-Carol-Simulation:

Die Temperatur wird langsam abgesenkt. → System soll dem Grundzustand nahe kommen, dh. nahe an die Liste mit der kürzesten Rundreise. Dazu muß T kontinuierlich abgesenkt werden. (Mögliche Modifikation: 'Pendeln' der Temperatur).
Wir betrachten nun erlaubte Angebotskonfigurationen:

1. Tausche benachbarte Einträge der Liste

2. Schneide eine Subliste heraus und drehe diese um:

$$1, 347, 17, |\underbrace{13, 212, 416, 91,}_{91,416,212,13}|42, 15, \ldots \tag{73}$$

3. Schneide eine Subliste aus und füge diese an einer anderen Stelle wieder ein

4. Wende 1 bis 3 immer wieder an, senke dabei die Temperatur langsam ab

3 Thermodynamik und Phasenübergänge in Spinsystemen

3.1 Grundlegende Formeln

3.1.1 Symmetrien

Wir wenden uns wieder unserem Hauptbeispiel, dem q-state Potts-Modell zu.

$$S_{\vec{n}} \in \{1, \dots, q\}, \quad \vec{n} \in \Lambda \tag{74}$$

$$H[S] = \alpha \sum_{\vec{n}} \sum_{\nu=1}^{d} [1 - \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}})] + \mu \sum_{\vec{n}} [1 - \delta(S_{\vec{n}}, 1)], \quad \alpha, \mu > 0 \tag{75}$$

Ein zentraler Punkt bei der Diskussion von Phasenübergängen sind die Symmetrien des Systems.
Hier: Das äußere Magnetfeld stört die Symmetrie, weil es eine Raumrichtung auszeichnet.
$\Rightarrow \mu = 0$
Dann hat das Potts-Modell eine Permutationssymmetrie:
Symmetrie im Potts-Modell: $\mu = 0$

$$\begin{aligned} 1 &\leftrightarrow P_{(1)} \\ 2 &\leftrightarrow P_{(2)} \\ &\vdots \\ q &\leftrightarrow P_{(q)} \end{aligned} \tag{76}$$

dh. wir vertauschen die Zuordnung der Spins.
$s - \vec{n} \to S_{\vec{n}'} = P(S_{\vec{n}}), \quad P \dots$ Permutationsoperator
$\Rightarrow$ Energie bleibt invariant, weil:

$$\delta(S_{\vec{n}'}, S_{\vec{n}'+\hat{\nu}}) = \delta P(S_{\vec{n}}), P(S_{\vec{n}+\hat{\nu}}) = \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}}) \tag{77}$$

Symmetrie im Ising-Modell: $\mu = 0$

$$H = \frac{\alpha}{2} \sum_{\vec{n} \in \Lambda} \sum_{\nu=1}^{d} [1 - S_{\vec{n}} \cdot S_{\vec{n}+\hat{\nu}}] \tag{78}$$

Dann ergeben sich zwei Möglichkeiten:

- $$S_{\vec{n}} \to S'_{\vec{n}} = -S_{\vec{n}} \tag{79}$$
- $$S'_{\vec{n}} \to S_{\vec{n}} = id \tag{80}$$

Die Multiplikation mit -1 bzw. $\underbrace{+1}_{id} \Rightarrow \{1, -1\} = \mathbb{Z}_2$.

Symmetrie im kontinuierlichen Spinsystem: $\mu = 0$

$$H = \frac{\alpha}{2} \sum_{\vec{n} \in \Lambda} \sum_{\nu=1}^{d} [1 - \vec{S}_{\vec{n}} \cdot \vec{S}_{\vec{n}+\hat{\nu}}], \quad S_{\vec{n}} \in \mathbb{R}^k : |\vec{S}_{\vec{n}}| = 1 \tag{81}$$

Kontinuierliche Symmetrie:

$$\vec{S}_{\vec{n}} \to \vec{S}'_{\vec{n}} = M \cdot \vec{S}_{\vec{n}}, \quad M^T = M^{-1} \quad \text{orthogonale Matrix} \tag{82}$$

Globale Transformation: gleiches $M \quad \forall \vec{n}$
Wichtig: $|\vec{S}'_{\vec{n}}| = 1$

$$1 = |\vec{S}'_{\vec{n}}|^2 = \vec{S}'^T_{\vec{n}} \vec{S}'_{\vec{n}} = (M\vec{S}_{\vec{n}})^T M \vec{S}_{\vec{n}} = \vec{S}^T_{\vec{n}} M^T M \vec{S}_{\vec{n}} = \vec{S}^T_{\vec{n}} \vec{S}_{\vec{n}} = |\vec{S}_{\vec{n}}|^2 \tag{83}$$

Magnetfelder brechen diese Symmetrien immer:

- Potts: $\mu \sum_{\vec{n}} \delta(S_{\vec{n}}, 1)$
- Ising: $\mu \sum_{\vec{n}} S_{\vec{n}}$
- Kontinuierlich: $\mu \sum_{\vec{n}} \vec{S}_{\vec{n}} \cdot \hat{e}_z$

Diese Terme sind nicht invariant unter den jeweiligen Symmetrien.

3.2 Observablen

Wir betrachten das kanonische Ensemble: (System im Wärmebad, T ist konstant)

$$P[S] = \frac{1}{Z} \cdot e^{-\beta H[S]}, \quad \beta = \frac{1}{k_B \cdot T}, \quad Z = \sum_S e^{-\beta \cdot H[S]} \tag{84}$$

3.2.1 Freie Energie

$\sim$ *thermodynamisches Potential*

$$\begin{aligned} \langle \mathcal{O} \rangle &= \sum_S \mathcal{O}[S] \cdot P[S] = \frac{1}{Z} \sum_S \mathcal{O}[S] \cdot e^{-\beta \cdot H[S]}, \quad z = e^{-\beta \cdot F} \\ \Rightarrow F &= -\frac{1}{\beta} \cdot \ln z \quad \text{Definitionsgleichung der freien Energie F} \end{aligned} \tag{85}$$

Wir bilden nun Observablen und identifizieren diese als Ableitung(en) der freien Energie nach den Parametern.

3.2.2 Innere Energie

Mittelwert der Energie

$$\begin{aligned} \mathcal{U} = \langle H \rangle &= \frac{1}{Z} \sum_S H[S] \cdot e^{-\beta \cdot H[S]} \\ &= \frac{1}{Z} (-\frac{\partial}{\partial \beta}) \underbrace{\sum_S e^{-\beta \cdot H[S]}}_{Z} \\ &= -\frac{1}{Z} \frac{\partial}{\partial \beta} Z = -\frac{\partial}{\partial \beta} \ln Z \\ &= \frac{\partial}{\partial \beta} (\beta \cdot F) = \langle H \rangle = \mathcal{U} \end{aligned} \tag{86}$$

dh. freie Energie abgeleitet nach β liefert die innere Energie.

3.2.3 Magnetisierung in die $\vec{r}$-Richtung

$$M_r[S] = \sum_{\vec{n}\in\Lambda} \delta(S_{\vec{n}}, r) \tag{87}$$

Magnetfeld in Richtung r:

$$\Rightarrow H[S] = -\alpha \sum_{\vec{n}} \sum_{\nu=1}^{d} \delta(S_{\vec{n}}, S_{\vec{n}+\hat{\nu}}) - \mu \sum_{\vec{n}} \delta(S_{\vec{n}}, r) \tag{88}$$

$$\langle M_r \rangle = \frac{1}{Z} \sum_S M_r[S] \cdot e^{-\beta \cdot H[S]} = \frac{1}{\beta} \frac{\partial}{\partial \mu_r} \underbrace{\sum_s e^{-\beta \cdot H[S]}}_{Z}$$

$$= \frac{1}{\beta} \frac{1}{Z} \frac{\partial}{\partial \mu_r} Z = \frac{1}{\beta} \frac{\partial}{\partial \mu_r} \ln Z = -\frac{1}{\beta} \frac{\partial}{\partial \mu_r} (\beta \cdot F) \tag{89}$$

$$\langle M_r \rangle = -\frac{1}{\beta} \frac{\partial}{\partial \mu_r} (\beta \cdot F) = -\frac{\partial}{\partial \mu_r} F$$

3.2.4 Fluktuationen

C_V . . .Wärmekapazität

$$\frac{1}{\beta^2} \cdot C_V = \langle (H - \langle H \rangle)^2 \rangle = \langle H^2 - 2H\langle H \rangle + \langle H \rangle^2 \rangle$$

$$= \langle H^2 \rangle - 2 \underbrace{\underbrace{\langle H \langle H \rangle \rangle}_{\langle H \rangle \langle H \rangle}}_{\langle H \rangle^2} + \underbrace{\langle \langle H \rangle^2 \rangle}_{\langle H \rangle^2} \tag{90}$$

$$= \langle H^2 \rangle - \langle H \rangle^2$$

$$\langle H \rangle = -\frac{\partial}{\partial \beta} \ln Z = \frac{1}{Z} \sum_S H[S] \cdot e^{-\beta \cdot H[S]} \tag{91}$$

$$\frac{\partial}{\partial \beta} \langle H \rangle = -\frac{\partial^2}{\partial \beta^2} \ln Z = \frac{\partial}{\partial \beta} \frac{1}{Z} \sum_S H[S] \cdot e^{-\beta \cdot H[S]}$$

$$= -\frac{1}{Z^2} (\frac{\partial}{\partial \beta} Z) \sum_S H[S] \cdot e^{-\beta \cdot H[S]} = \frac{1}{Z} \sum_S H[S] \cdot e^{-\beta H[S]} \cdot H[S]$$

$$= -\frac{1}{Z} \sum_S H[S]^2 \cdot e^{-\beta \cdot H[S]} - \frac{1}{Z} \sum_S H[S] \cdot e^{-\beta \cdot H[S]} \cdot \frac{1}{Z} \frac{\partial}{\partial \beta} \sum_S e^{-\beta \cdot H[S]} \tag{92}$$

$$= -\underbrace{\frac{1}{Z}\sum_S H[S]^2 \cdot e^{-\beta \cdot H[S]}}_{\langle H^2 \rangle} + \underbrace{\frac{1}{Z}\sum_S H[S] \cdot e^{-\beta \cdot H[S]}}_{\langle H \rangle} \cdot \underbrace{\frac{1}{Z}\sum_S H[S] \cdot e^{-\beta \cdot H[S]}}_{\langle H \rangle}$$

$$\Rightarrow -\frac{\partial}{\partial \beta} H = \frac{\partial^2}{\partial \beta^2} \ln Z = \langle H^2 \rangle - \langle H \rangle^2 = \frac{1}{\beta^2} \cdot C_V \tag{93}$$

$$\Rightarrow C_V = \beta^2 \frac{\partial^2}{\partial \beta^2} \ln Z = -\beta^2 \frac{\partial^2}{\partial \beta^2}(\beta \cdot F) = \beta^2[\langle H^2 \rangle - \langle H \rangle^2] \tag{94}$$

3.2.5 Magnetische Suszeptibilität

$$\frac{1}{\beta}\chi_r = \langle (M_r - \langle M_r \rangle)^2 \rangle = \langle M_r^2 \rangle - \langle M_r \rangle^2 \tag{95}$$

$$\frac{\partial}{\partial \mu_r}\langle M_r \rangle = \frac{\partial}{\partial \mu_r}\frac{1}{Z}\sum_S M_r[S] \cdot e^{-\beta \cdot H[S]}$$

$$= -\frac{1}{Z^2}\frac{\partial Z}{\partial \mu_r}\sum_S M_r[S] \cdot e^{-\beta \cdot H[S]} + \frac{1}{Z}\sum_S M_r[S] \cdot e^{-\beta \cdot H[S]} \cdot H[S]$$

$$= -\underbrace{\frac{1}{Z}\sum_S M_r[S] \cdot e^{-\beta \cdot H[S]}}_{\langle M_r \rangle} \cdot \underbrace{\frac{1}{Z}\sum_S M_r[S] \cdot e^{-\beta \cdot H[S]}}_{\langle M_r \rangle} \cdot \beta + \beta \langle M_r^2 \rangle \tag{96}$$

$$= \beta[\langle M_r^2 \rangle - \langle M_r \rangle^2] = \chi_r$$

$$\chi_r = \frac{\partial}{\partial \mu_r}\langle M_r \rangle = \frac{\partial}{\partial \mu_r}(-\frac{\partial}{\partial \mu_r}\frac{1}{\beta}(\beta \cdot F)) = -\frac{1}{\beta}\frac{\partial^2}{\partial \mu_r^2}(\beta \cdot F) \tag{97}$$

3.3 Phasenübergänge

In vielen Phasenübergängen beobachtet man einen Übergang zwischen einer geordneten und einer ungeordneten Phase. Diese Übergänge können als Funktion der Temperatur, aber auch als Funktion anderer Parameter (Kopplung, Magnetfeld,...) auftreten.
Die Phasen werden oft durch unterschiedliche Symmetrien gekennzeichnet.
Magnetisierung als Funktion von J:

$$M_r = \sum_{\vec{n}} \delta(S_{\vec{n}}, r) \tag{98}$$

3.3.1 Phasenübergänge zweiter Ordnung

Man nennt diese auch kontinuierliche Phasenübergänge.
Eine zweite Ableitung der freien Energie F ist unstetig. In diesem Fall: M ist erste Ableitung, nochmaliges Ableiten führt zu einer Unstetigkeit.
Weitere Beispiele sind etwa C_V und χ_r.

3.3.2 Phasenübergänge erster Ordnung

Bereits eine erste Ableitung der freien Energie ist unstetig.
Beispiele: $\langle M_r \rangle$, $\mathcal{U} = \langle H \rangle$.

3.3.3 Kritische Kopplungen im Potts-Modell

q	J_0	Ordnung des Phasenüberganges
2	0.881	2
3	1.005	2
4	1.099	2
5	1.174	1
6	1.238	1
7	1.298	1
8	1.342	1
9	1.386	1
10	1.426	1

Wichtig: Phasenübergänge, dh. Unstetigkeiten von Ableitungen von F, treten nur bei Systemen mit ∞ vielen Freiheitsgraden auf. (Vgl.: Thermodynamischer Limes)
Denn:
q-state Potts-Modell auf einem Gitter mit V Punkten ($\Rightarrow q^V$ Zustände):

$$e^{-\beta \cdot F} = Z = \sum_S e^{-\beta \cdot H[S]} = \sum_S e^{-\beta\alpha \cdot H_1[S]} \cdot e^{-\beta\mu \cdot H_2[S]} \tag{99}$$

ist ein endliches Polynom und kann als solches keine unstetigen Ableitungen besitzen.
Die Ableitungen konvergieren jedoch gegen die Sprungstelle.

3.3.4 Kritische Exponenten

Bei einem Phasenübergang zeigen viele Observablen kritisches Verhalten. Dieses Verhalten ist universell und wird durch 'kritische Exponenten' beschrieben.
Wir führen die reduzierte Temperatur t ein:

$$t = \frac{T - T_C}{T_C} \tag{100}$$

Es ist also $t > 0$ wenn $T > T_C$, und $t < 0$ wenn $T < T_C$.
Für t sehr nahe an T_C:

$$M = M_0(-t)^3 \tag{101}$$
$$\chi = \chi_0|t|^{-\gamma} \tag{102}$$
$$C = C_0|t|^{-\alpha} \tag{103}$$
$$\xi = \xi_0|t|^{-\nu} \tag{104}$$

Wir haben die kritischen Exponenten $\beta, \gamma, \alpha, \nu$ und η. (Letzterer wird hier nicht näher spezifiziert).
Es gilt: $\beta, \gamma, \alpha, \nu, \eta \geq 0$

Beispiel: 2-d Ising-Modell:
Die kritischen Exponenten für dieses System sind exakt bekannt:
$\alpha = 0, \beta = \frac{1}{8}, \gamma = \frac{7}{8}, \nu = 1, \eta = \frac{1}{4}$
Kritische Exponenten sind nicht unabhängig voneinander: Es gibt 'Scaling Laws'.

Scaling Laws:

$$\alpha + 2\beta + \gamma \geq 2 \tag{105}$$
$$d \cdot \nu \geq 2 - \alpha \quad d \ldots \text{Dimension} \tag{106}$$
$$\gamma \leq \nu(2 - \eta) \tag{107}$$

Systeme mit gleichen kritischen Exponenten gehören zur gleichen Universalitätsklasse. Diese hängt ab von:

- Raumdimension
- Gitterstruktur (Dreieck, Viereck, Hexagon)
- Dimensionalität der Spins
- Symmetrie der geordneten Phase

Berechnung der kritischen Exponeneten:

Endliche Gitter produzieren keine Phasenübergänge. In der Wärmekapazität und der Suszeptibilität gibt es aber ausgeprägte Maxima die mit der Systemgröße divergieren.
Für kontinuierliche Übergänge:

$$\underbrace{\frac{1}{L^d}}_{Vol.} \cdot \chi_{max}(T_C) \propto L^{\gamma_C}, \quad \frac{1}{L^d} \cdot C_V(T_C) \propto L^{\alpha_C} \tag{108}$$

Phasenübergänge erster Ordnung können als Gemisch von zwei Phasen verstanden werden. Am Phasenübergang kippt das System von einer Phase in die andere Phase.

Zustandssumme: $Z = Z_+ + Z_-$
Freie Energie: $F_i(\beta) = -\frac{1}{\beta} ln Z_i$
Energiedichte: $f_i(\beta) = F_i(\beta)/V$, intensiv

$\Rightarrow Z_i = e^{-\beta f_+(\beta) \cdot V} + e^{-\beta f_-(\beta) \cdot V}$

$\beta > \beta_c : f_+(\beta) < f_-(\beta)$
$\beta < \beta_c : f_-(\beta) < f_+(\beta)$
$\beta = \beta_c : f_+(\beta) = f_-(\beta)$

Mit $U = \langle H \rangle = -\frac{\partial}{\partial \beta} ln Z$ folgt:

$$\begin{aligned} \frac{U}{V} &= -\frac{1}{V} \frac{\partial}{\partial \beta} ln Z \\ &= -\frac{1}{V} \frac{1}{Z} \frac{\partial}{\partial \beta} Z \\ &= -\frac{1}{V} \frac{1}{Z} \frac{\partial}{\partial \beta} [e^{-\beta f_+(\beta) \cdot V} + e^{-\beta f_-(\beta) \cdot V}] \end{aligned}$$

$$= -\frac{1}{V}\cdot\frac{1}{Z}[e^{-\beta f_+(\beta)\cdot V}(-1)\underbrace{[f_+(\beta)+\beta f'_+(\beta)]}_{h_+}V + e^{-\beta f_-(\beta)\cdot V}(-1)\underbrace{[f_-(\beta)+\beta f_-(\beta)]}_{h_-}V] \tag{109}$$

$$= \frac{h_+ e^{-\beta f_+(\beta)\cdot V} + h_- e^{-\beta f_-(\beta)\cdot V}}{e^{-\beta f_+(\beta)\cdot V} + e^{-\beta f_-(\beta)\cdot V}} = \frac{U}{V}$$

Nun können wir betrachten was bei $V \to \infty$ passiert:

$$\frac{U}{V} = \begin{cases} \frac{e^{-\beta f_+ V}}{e^{-\beta f_+ V}}\frac{h_+ + h_- e^{-\beta(f_- - f_+)V}}{1+e^{-\beta(f_- - f_+)V}}, & \beta > \beta_c \Rightarrow f_- - f_+ > 0 \\ \frac{e^{-\beta f_- V}}{e^{-\beta f_- V}}\frac{h_+ e^{-\beta(f_+ - f_-)V} + h_-}{e^{-\beta(f_+ - f_-)V}+1}, & \beta < \beta_c \Rightarrow f_+ - f_- > 0 \end{cases} \tag{110}$$

$$\lim_{V\to\infty}\frac{U}{V} = \begin{cases} h_+ = f_+ + \beta f'_+, & \beta > \beta_c \\ h_- = f_- + \beta f'_-, & \beta < \beta_c \end{cases} \tag{111}$$

Nun ist $\frac{\Delta U}{V}$ die latente Wärme (Sprunghöhe):

$$\frac{\Delta U}{V} = |h_+ - h_-|\Big|_{\beta_c} = |f_+(\beta_c) + \beta_c f'_+(\beta_c) - (f_-(\beta_c) + \beta_c f'_-(\beta_c))| = \underline{\underline{\beta_c |f'_+(\beta_c) - f'_-(\beta_c)|}} \tag{112}$$

3.3.5 Numerische Methode zur Unterscheidung von Phasenübergängen

Zur Unterscheidung von Phasenübergängen erster Ordnung und kontinuierlichen Übergängen geht man wie folgt vor.

Histogramme von Observablen

Bei einem Phasenübergang pendelt das System zwischen zwei Phasen. Diese sind durch unterschiedliche Werte von Observablen, also etwa der Energie, gekennzeichnet.

Histogramme:

- Erzeuge N Monte-Carlo Konfigurationen
- Berechne eine Observable $\mathcal{O}_i, i = 1, \ldots, N$
- $\frac{\mathcal{O}_{max} - \mathcal{O}_{min}}{n} = \Delta, \quad n \ldots$ Anzahl der Bins (freier Parameter)
- $ibin = nint[\frac{\mathcal{O}_i - \mathcal{O}_{min}}{\Delta}], \quad mit \quad nint \ldots$ nearest int
- $H[ibin] \leftarrow H[ibin] + 1$
- $H \to H/(N \cdot \Delta) \rightsquigarrow \sum_{i=0}^{n} H[i] \cdot \Delta = 1$
- Plot $H[i]$ versus $\mathcal{O}_i = \mathcal{O}_{min} + \Delta i$

Auf diese Weise findet man für Phasenübergänge erster Ordnung eine Doppelpeakstruktur, für kontinuierliche Übergänge ist dies nicht vorzufinden.

4 Statistische Auswertung der Monte-Carlo Daten

4.1 Grundlegende Formeln

4.1.1 Mittelwert, Varianz, Standardabweichung

$x_i, \quad i = 1, 2, \ldots, N$ Rohdaten.

Mittelwert: $\bar{X_N} = \frac{1}{N} \sum_{i=1}^{N} x_i$

Varianz: $S_N^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x_N})^2$

Standardabweichung: S_N

Weiters sei eine wichtige Relation festgehalten:

$$\begin{aligned} S_N^2 &= \frac{1}{N-1} \sum_{i=1}^{N} (x_i^2 - 2x_i\bar{x_N} + \bar{x_N}^2) \\ &= \frac{1}{N-1} \underbrace{\sum_{i=1}^{N} (x_i^2)}_{N \cdot \bar{x_N^2}} - \frac{2}{N-1} \bar{x_N} \cdot \underbrace{\sum_{i=1}^{N} x_i}_{N \cdot \bar{x_N}} + \frac{1}{N-1} \bar{x_N}^2 \cdot N \\ &= \frac{N}{N-1} [\bar{x_N^2} - (\bar{x_N})^2] \sim \bar{x_N^2} - (\bar{x_N})^2 \end{aligned} \tag{113}$$

Dennoch ist eine Implementierung auf diese Weise ungünstig, da Rundungsfehler auftreten.

Gaussverteilung

$x_i \to y \in \mathbb{R}$ kontinuierliche Verteilung

$$P(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(y-\mu)^2}{2\sigma}} \tag{114}$$

Wahrscheinlichkeitsverteilung:

$$P(y)\Big|_{y=\mu\pm\delta} = \frac{1}{\sqrt{2\pi}\delta} \cdot e^{-\frac{1}{2}} = \frac{1}{\sqrt{2\pi}\delta} \cdot 0.606 \tag{115}$$

Weiters ist

$$\int_{-\infty}^{\infty} P(y)dy = 1, \tag{116}$$

und

$$\bar{y} = \int_{-\infty}^{\infty} P(y) \cdot y dy = \mu. \tag{117}$$

$$\bar{y^2} = \int_{-\infty}^{\infty} P(y) \cdot y^2 = \mu^2 + \delta^2. \tag{118}$$

Schließlich gilt

$$S^2 = \bar{y^2} - (\bar{y})^2 = \mu^2 + \delta^2 - \mu^2 = \delta^2 \tag{119}$$

$\Rightarrow$ Varianz δ^2, Standardabweichung δ. Dann ist die Wahrscheinlichkeit y im Intervall $[\mu - n\delta, \mu + n\delta]$ zu finden, mit $n = 1, 2, 3, \ldots$:

$$\int_{\mu-n\delta}^{\mu+n\delta} P(y)dy = \begin{cases} 0.68 & n = 1 \\ 0.954 & n = 2 \\ 0.997 & n = 3, \end{cases} \tag{120}$$

und die Wahrscheinlichkeit für Events $\notin [\mu - n\delta, \mu + n\delta]$ ist
$\sim 30\%$ $n = 1$,
$\sim 5\%$ $n = 2$,
$\sim 0.3\%$ $n = 3$.

4.2 Fehler bei einer endlichen Menge von Messungen

Aus einer Monte-Carlo Simulation (od. Labor, etc.): $\Rightarrow x_1, x_2, \ldots, x_N$ Werte einer Observablen. Wir haben dann

$$\bar{x_N} = \frac{1}{N}\sum_{i=1}^{N} x_i; \quad S_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2. \tag{121}$$

Endresultat: $\bar{x_N} \pm \epsilon$, wobei ϵ der Fehler ist.

Für ϵ nicht die Standardabweichung angeben!

Der Grund hierfür liegt darin, dass das Grundprinzip 'Anzahl der Messungen hoch $\Rightarrow$ Fehler klein' von der Standardabweichung nicht erfüllt wird. Wir werden zeigen: $\epsilon = \frac{S_N}{\sqrt{N}}$.
Wir halten N fest und führen weitere Messreihen mit N Messungen durch (im Prinzip unendlich viele).
$\Rightarrow$ unendlich viele $\bar{x_N}$. Wie sind nun die Mittelwerte $\bar{x_N}$ um den korrekten Mittelwert $\langle x \rangle$ verteilt?

Der Zentrale Grenzwertsatz zeigt uns, dass die $\bar{x_N}$ Gaussverteilt sind, mit Mittelwert $\langle x \rangle$ und Breite $\frac{\delta}{\sqrt{N}}$. In führender Näherung gilt: $\delta \sim S_N$.
Wenn wir unsere Messreihe unendlich oft wiederholen, so können wir aus jedem Wert $x_j, \quad j = 1, \ldots, N$ den exakten Wert $\langle x \rangle$ berechnen:

$$\langle x_i \rangle = \langle x \rangle. \tag{122}$$

Annahme: Unsere Messwerte seien perfekt dekorreliert:

$$\forall i \neq j : \langle x_i x_j \rangle = \langle x_i \rangle \langle x_j \rangle = \langle x \rangle^2. \tag{123}$$

(Dies gilt nicht für den Metropolis-Algorithmus!)

$$\begin{aligned}
\delta^2_{\bar{x_N}} &= \langle (\bar{x_N} - \langle x \rangle)^2 \rangle = \langle (\frac{1}{N}\sum_{i=1} N x_i - \langle x \rangle)^2 \rangle = \\
&= \langle \frac{1}{N^2}\sum_{i,j} x_i x_j - \frac{2}{N}\sum_{i=1}^{N} x_i \langle x \rangle + \langle x \rangle^2 \rangle = \\
&= \frac{1}{N^2}\sum_{i,j}\langle x_i x_j \rangle - 2\langle x \rangle \underbrace{\frac{1}{N}\sum_{i=1}^{N}\underbrace{\langle x_i \rangle}_{=\langle x \rangle}}_{\langle x \rangle} + \langle x \rangle^2 = \\
&= \frac{1}{N}\sum_{i=1}^{N}\underbrace{\langle x_i^2 \rangle}_{\langle x^2 \rangle} + \frac{1}{N^2}\sum_{i \neq j}\underbrace{\langle x_i x_j \rangle}_{\langle x_i \rangle \langle x_j \rangle = \langle x \rangle^2} - \langle x \rangle^2 = \\
&= \frac{N}{N^2}\langle x^2 \rangle + (\underbrace{\frac{1}{N^2}N(N-1)}_{\frac{N^2_N}{N^2} = 1 - \frac{1}{N}} - 1)\langle x \rangle^2 = \\
&\quad \frac{1}{N}(\langle x^2 \rangle - \langle x \rangle^2) = \frac{1}{N}\delta^2.
\end{aligned} \tag{124}$$

Also ist

$$\delta^2_{\bar{x_N}} = \langle(\bar{x_N} - \langle x\rangle)^2\rangle = \frac{1}{N}\delta^2, \tag{125}$$

und somit folgt

$$\Rightarrow \epsilon = \delta_{\bar{x_N}} = \frac{\delta}{\sqrt{N}} \sim \frac{S_N}{\sqrt{N}}(1 + \frac{1}{\sqrt{N}}) \sim \frac{S_N}{\sqrt{N}}. \tag{126}$$

Autokorrelation

Bei einer Monte-Carlo Simulation sind die Messwerte x_i und $x_j = x_{i+t}$ leider meist nicht dekorreliert, dh. es gilt nicht $\langle x_i x_{i+t}\rangle = \langle x_i\rangle\langle x_{i+t}\rangle$. Man definiert die Autokorrelationsfunktion

$$C_x(t) = \langle x_i x_{i+t}\rangle - \underbrace{\langle x_i\rangle\langle x_{i+t}\rangle}_{\langle x\rangle^2}, \tag{127}$$

wobei x Observable ist.

$$C_x(0) = \langle x^2\rangle - \langle x\rangle^2 = \delta^2 \tag{128}$$

Ein typisches Verhalten sieht etwa so aus:

$$C_x(t) = c_x(0) \cdot e^{-\frac{t}{\tau_{x,exp}}}, \quad C_x(t) = C_x(-t) = C_x(|t|) \tag{129}$$

und $\tau_{x,exp} \ldots$ exponentielle Autokorrelationszeit der Observable x.

$$\begin{aligned}
\delta^2_{\bar{x_N}} &= \langle(\bar{x_N} - \langle x\rangle)^2\rangle = \\
&= \langle\frac{1}{N^2}\sum_{i,j} x_i x_j - \langle x\rangle^2\rangle = \frac{1}{N^2}\sum_{i,j}[\langle x_i x_j\rangle - \langle x\rangle^2] \\
&= \frac{1}{N^2}\sum_{i,j=1}^{N} C_x(|i-j|) = \Big|t = |i-j|\Big| = \frac{1}{N^2}\sum_{t=-(N-1)}^{N-1}\sum_{k=1}^{N-|t|} C_x(|t|) = \\
&= \frac{1}{N^2}\sum_{t=-(N-1)}^{(N-1)}(N-|t|)C_x(|t|) = \frac{1}{N}\sum_{t=-N}^{N}\frac{N-|t|}{N}C_x(|t|) \\
&= \frac{C_x(0)}{N}\sum_{t=-N}^{N}\frac{C_x(|t|)}{C_x 0}(1-\frac{|t|}{N}) \approx \frac{\overbrace{C_x(0)}^{\delta^2}}{N}\sum_{t=-N}^{N}\frac{C_x(|t|)}{C_x(0)} \\
&= 2\frac{\delta^2}{N}[\overbrace{\frac{1}{2} + \sum_{t=1}^{N}\underbrace{\frac{C_x(|t|)}{C_x(0)}}_{e^{-\frac{t}{\tau_{x,exp}}}}}^{\tau_{x,int}}] = 2\frac{\delta^2}{N}[\int_0^N e^{-\frac{t}{\tau_{x,exp}}}dt] = \\
&= 2\frac{\delta^2}{N}(-\tau_{x,exp})\cdot e^{-\frac{t}{\tau_{x,exp}}}\Big|_0^N = \frac{\delta^2}{N}\cdot 2\tau_{x,exp}[1 - e^{-\frac{N}{\tau_{x,exp}}}] = \\
&= \frac{\delta^2}{N}2\tau_{x,exp} = \delta^2_{\bar{x_N}} \Rightarrow \delta_{\bar{x_N}} = \frac{\delta}{\sqrt{N}}\sqrt{2\tau_{x,exp}} \\
&\Rightarrow \langle x\rangle = \bar{x_N} \pm \frac{S_N}{\sqrt{N}}\sqrt{2\tau_{x,exp}}.
\end{aligned} \tag{130}$$

Dabei ist zu beachten, die Summation, welche vorher über i und j ging durch eine äquivalente Summation, nun über den Parameter t, ersetzt wurde. Graphisch anschaulich bedeutet dies einen

Wechsel von Summation von Punkten auf einem $i-j$ Gitter zur Summation über die Gitterdiagonalen.
Ferner ist $\tau_{x,int}\ldots$ integrierte Autokorrelationszeit.

4.3 Fehler für sekundäre Observablen

Allgemeine Formeln zur Kombination von Fehlern: Observable $\bar{z}$: $\bar{z} = f(\bar{x}^{(1)}, \bar{x}^{(2)}, \ldots, \bar{x}^{(r)}$.
Dann gilt:

$$\delta_z^2 = \sum_{i=1}^{r} (f_i - \underbrace{f}_{\bar{z}})^2 \tag{131}$$

und

$$f_i = (\bar{x}^{(1)}, \bar{x}^{(2)}, \ldots, \underbrace{\bar{x}^{(i)}\delta^{(i)}}_{Fehler \quad nur \quad i-tes \quad Element}, \bar{x}^{(i+1)}, \ldots, \bar{x}^{(r)}). \tag{132}$$

$$\Rightarrow \langle z \rangle = \bar{z} \pm \delta z. \tag{133}$$

Vgl.: Fehlerfortpflanzung:

$$\begin{aligned} f_i &= f(\bar{x}^{(1)}, \ldots, \bar{x}^{(i-1)}, \bar{x}^{(i)} + \delta^{(i)}, \bar{x}^{(i+1)}, \ldots, \bar{x}^{(r)}) \\ &= \underbrace{f(\bar{x}^{(1)}, \ldots, \bar{x}^{(r)})}_{f} + \frac{\partial f}{\partial \bar{x}^{(i)}}\delta^{(i)} + \mathcal{O}(\delta^2) \end{aligned} \tag{134}$$

$$\Rightarrow \delta_z^2 = \sum_{i=1}^{r}(f_i - f)^2 = \sum_{i=1}^{r}(f + \frac{\partial f}{\partial \bar{x}^{(i)}}\delta^{(i)} - f) = \sum_{i=1}^{r}(\frac{\partial f}{\partial \bar{x}^{(i)}})^2(\delta^{(i)})^2.$$

Sekundäre Observablen: Vgl.: Isingmodell:

$$\langle S_{\vec{n}} \quad S_{\vec{n}+\vec{r}} \rangle \sim C \cdot e^{-\frac{|\vec{r}|}{\xi}}, \tag{135}$$

dh., je weiter Orte voneinander entfernt sind, desto weniger wissen sie voneinander.

Beispiel:

Angenommen man wolle ein radioaktives Element vermessen und misst einen Tag lang die Anzahl der Clicks in Abhängigkeit der Zeit. Das Histogramm, welches man dann erhält, ist aber nicht fitbar, da kein exponentielles Gesetz erkennbar ist (hohe Halbwertszeit). Man kann nun aber immer wieder messen, etwa z.B. 160 Tage lang, und die erhaltenen Werte mitteln. Dann kann gefittet werden, so folgt τ:

$$\bar{H}(t) = \frac{1}{N}\sum_{i=1}^{N} H^{(i)}(t) \Rightarrow \quad Fit \quad \Rightarrow \tau. \tag{136}$$

Jackknife-Methode

Wir haben Datensätze $S^{(1)}, S^{(2)}, \ldots, S^{(N)}$. Ferner seien:
$\bar{\mathcal{O}}$ ... Observable berechnet aus allen N Datensätzen und
$\mathcal{O}^{(i)}$ ... Observable berechnet ohne den Datensatz Nummer (i), $\quad i = 1, \ldots, N$.

$$S^2 = (N-1)\sum_{i=1}^{N}(\bar{\mathcal{O}} - \mathcal{O}^{(i)})^2. \tag{137}$$

ACHTUNG: $(N-1)$ tritt nun im Zähler auf!

$$\langle \mathcal{O} \rangle = \bar{\mathcal{O}} \pm \frac{S}{\sqrt{N}} \tag{138}$$

Nachrechnen der Jackknife Methode für einfache Mittelwerte

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i; \quad x^{(j)} = \frac{1}{N-1}\sum_{i=1, i\neq j}^{N} x_i = \frac{1}{N-1}\sum_{i=1}^{N} x_i - \frac{1}{N-1}x_j.$$

$$\begin{aligned} S^2 &= (N-1)\cdot\sum_{j=1}^{N}(\bar{x}-x^{(j)})^2 = (N-1)\cdot\sum_{j=1}^{N}(\frac{1}{N}\sum_{i=1}^{N}x_i - \frac{1}{N-1}\sum_{i=1}^{N}x_i + \frac{1}{N-1}x_j)^2 \\ &= (N-1)\sum_{j=1}^{N}(\frac{N-1-N}{N(N-1)}\cdot\sum_{i=1}^{N}x_i + \frac{1}{N-1}x_j)^2 = \frac{(N-1)}{(N-1)^2}\sum_{j=1}^{N}(-\frac{1}{N}\sum_{i=1}^{N}x_i + x_j)^2 \\ &= \frac{1}{N-1}\sum_{j=1}^{N}(-\bar{x}+x_j)^2 = \frac{1}{N-1}\sum_{j=1}^{N}(\bar{x}-x_j)^2 = S^2, \end{aligned} \tag{139}$$

wie bereits vorher definiert.

4.4 Modellierung von Daten

Datensätze (x_i, y_i, δ_i), $\quad i = 1, 2, \ldots, N$, Fit ist gerade Linie $y = a + bx$.
χ^2-Funktional:

$$\chi^2(a,b) = \sum_{i=1}^{N}(\frac{y_i - a - bx_i}{\delta_i})^2, \tag{140}$$

wir berücksichtigen den Fehler als Gewicht. Wir minimieren nun $\chi^2(a,b)$ durch Variation der Fit-Parameter a und b. Dann ist das Minimum der beste Fit, dh. wir haben dann die optimalen Werte für a und b gefunden.

$$\begin{aligned} \frac{\partial\chi^2}{\partial a} = 0, \quad \frac{\partial\chi^2}{\partial b} = 0, \quad a = \frac{S_{xx}S_y - S_xS_xy}{\Delta}, \quad b = \frac{SS_{xy} - S_xS_y}{\Delta}, \\ S = \sum_{i=1}^{N}\frac{1}{\delta_i^2}, \quad S_x = \sum_{i=1}^{N}\frac{x_i}{\delta_i^2}, \quad S_y = \sum_{i=1}^{Ny_i}\delta_i^2, \\ S_{xx} = \sum_{i=1}^{N}\frac{x_i^2}{\delta_i^2}, \quad S_{xy} = \sum_{i=1}^{N}\frac{x_iy_i}{\delta_i^2}, \quad \Delta = S\cdot S_{xx} - S_x^2. \end{aligned} \tag{141}$$

χ^2-Test:

Seien x_i normalverteilte Zufallszahlen mit Varianz 1 und Mittelwert 0.

$$P(x_i) = \frac{1}{2\pi}e^{-\frac{x_i^2}{2}}, \quad x^2 = \sum_{i=1}^{\nu}x_i^2. \tag{142}$$

Sei ferner χ^2 vorgegebene Zahl >0. Dann ist die Wahrscheinlichkeit dass $x^2 \geq \chi^2$ gegeben durch

$$Q(\chi^2|\nu) = \frac{2}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})}\int_{\chi^2}^{\infty} t^{\frac{\nu}{2}-1}e^{-\frac{t}{2}}dt. \tag{143}$$

Die Werte können aus Tabellen ersehen werden. Vernünftige Werte liegen bei $\frac{\chi^2}{\nu} \sim 1-2$.

Wir nehmen an, dass die Fehler Gaussverteilt sind, dh. die Fehlerbalken können als Standardabweichung interpretiert werden.

$$P(y_i) = \frac{1}{\sqrt{2\pi}\delta_i} \cdot e^{-\frac{(y_i - f_i)^2}{2\delta_i^2}} \tag{144}$$

$\underline{\chi^2\text{-Test:}}$ $x_i = \frac{y_i - f_i}{\delta_i} \Rightarrow P(x_i) \ldots$ Normalverteilt.

$\underline{\underline{\text{Allgemein:}}}$

$\underline{\text{Daten:}}$ (x_i, y_i, δ_i), $i = 1, \ldots, N$
$\underline{\text{Fit-Funktion:}}$ $y = f(x)_{a_1, a_2, \ldots, a_r}$ (Parameter des Fits)

$$\chi^2(a_1, a_2, \ldots, a_r) = \sum_{i=1}^{N} \left(\frac{f(x_i)_{a_1, a_2, \ldots, a_r}}{\delta_i}\right)^2 \tag{145}$$

$\underline{\text{Minimierung von } \chi^2\text{:}}$ (i.A. numerisch) $\Rightarrow a_1, a_2, \ldots, a_r$.
$\underline{\chi^2\text{-Test:}}$ $\frac{\chi^2_{min}}{\nu} \sim 1-2$ ok✓
$\nu = N - r$
Fehler an Fitparameter, z.B. mit Jackknife.

5 Aufgabenstellung und Umsetzung

5.1 Aufgabenstellung

Die Aufgabenstellung für dieses Projekt sieht wie folgt aus:

Simulation des Potts Modelles

1. Implementation des Metropolis Algorithmus für das 2-d Potts Modell mit $q = 2, 3, \ldots, 8$.
2. Plots von typischen Konfigurationen für $J > J_c$ und $J < J_c$.
3. Plots von:
   - $\langle M_r \rangle$
   - $\langle H_r \rangle$
   - $\frac{\chi_r}{\beta \cdot V} = \langle (M_r - \langle M_r \rangle)^2 \rangle \cdot \frac{1}{V}$
   - $\frac{1}{V} C = \langle (H - \langle H \rangle)^2 \rangle \cdot \frac{1}{V}$,

   jeweils als Funktion von J für $q = 2$ und $q = 8$.
4. Berechnung der statistischen Fehler für $\langle M_r \rangle$, $\langle H \rangle$.
5. Analyse des Phasenüberganges bei $q = 2$ und $q = 8$ mit der Histogrammtechnik.
6. Dokumentation.

Im Folgenden sollen nun ein Punkt nach dem anderen behandelt werden, bzw. die Ergebnisse dargestellt werden. Die Betrachtung endet auf natürliche Weise mit Abschluss von Punkt 5, da damit Punkt 6 automatisch erfüllt ist. Ferner ist der Source-Code in seiner Gesamtheit im Anhang zu ersehen.

5.2 Punkt 1: Metropolis-Algorithmus

Hier nun ein Ausschnitt aus dem Code, welcher den Metropolis-Algortihmus repräsentiert.

```
for(int n=0; n<nsweeps; n++)
{
 for(int i=0; i<L*L; i++)
 {
  offer = rand()%q + 1;                    //waehle einen der q Spins
  rho = exp(J*( kronecker(offer, S[neib[i][0]])
           + kronecker(offer, S[neib[i][1]])
           + kronecker(offer, S[neib[i][2]])
           + kronecker(offer, S[neib[i][3]])
           - kronecker(S[i], S[neib[i][0]])
           - kronecker(S[i], S[neib[i][1]])
           - kronecker(S[i], S[neib[i][2]])
           - kronecker(S[i], S[neib[i][3]]))
           + M*( kronecker(offer, 1)
           - kronecker(S[i], 1)));         //Bilde Verhaeltnis der WS-keiten

   r = rand()/(RAND_MAX + 1.);             //Ziehe Zufallszahl zw. 0,1 glvtlt.
   if ( r <= rho)
   {
    S[i] = offer;                          //Akzeptiere
   }

 }
}
```

Es handelt sich hierbei um das Unterprogramm `update`, welches ein Metropolis-Update durchführt. Dies läuft wie folgt ab: Die äußerste Schleife ist die Anzahl der `sweeps`, also der Durchläufe durch das Gitter. Die zweite Schleife von außen läuft genau einmal durch das Gitter, wobei wir im Spinarray `S[i]` mit einem aufgerollten Index arbeiten. Als Angebot (`offer`) wird nun ein möglicher Spin mit dem Zufallsgenerator gewählt. ρ, das Verhältnis der Wahrscheinlichkeiten aus Angebots- und aktueller Konfiguration, wird im Folgenden gebildet. Nach dem Ziehen einer gleichverteilten Zufallszahl r zwischen 0 und 1 wird schließlich entschieden, ob die Angebotskonfiguration akzeptiert wird oder nicht.

5.3 Punkt 2: Typische Plots

Es folgen einige Plots für Konfigurationen. Zuerst ein 'Initial Plot' mit 7 sates, danach Konfigurationen eines 2-state Potts-Modelles, da sich dieses auf anschauliche Weise auf ein Ising Modell reduziert.

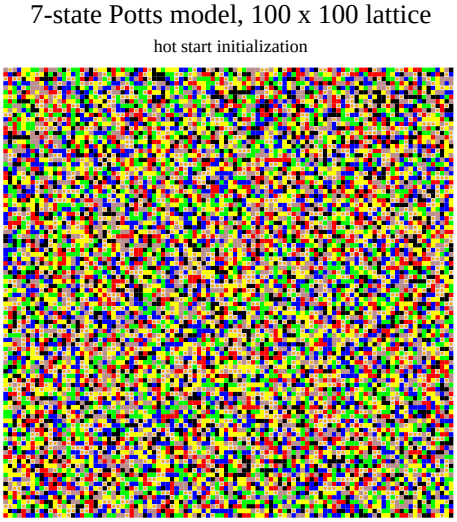


Abbildung 2: Konfiguration über der kritischen Temperatur

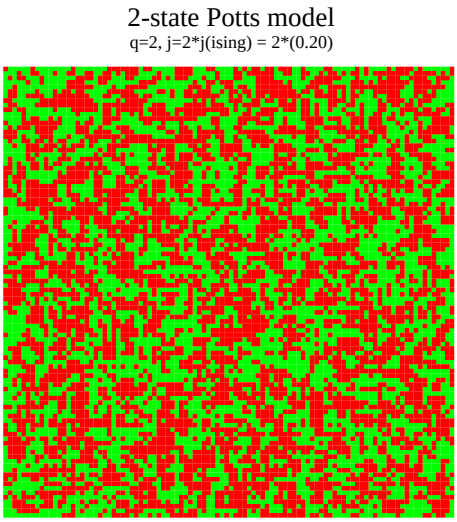


Abbildung 3: Konfiguration über der kritischen Temperatur

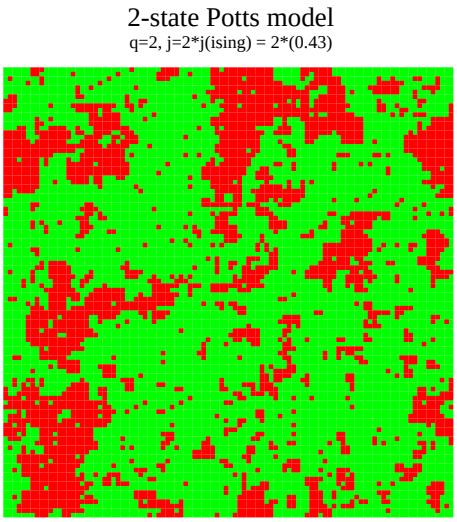


Abbildung 4: In der Nähe des Phasenüberganges beginnen plötzlich alle Skalen mitzuspielen

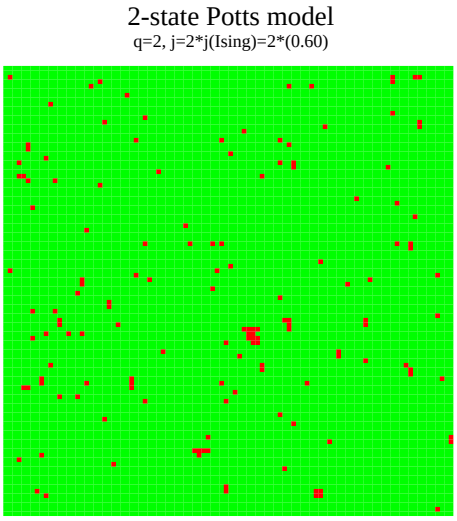
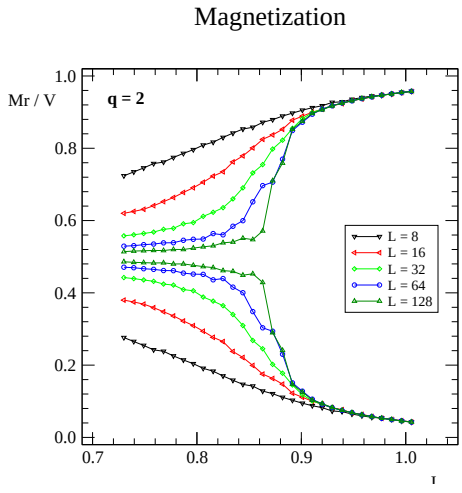


Abbildung 5: Unter der kritischen Temperatur dominiert eine Spinrichtung

5.4 Punkt 3: Observablen

Wir betrachten zunächst das Verhalten der Magnetisierung für $q = 2$ und $q = 8$. Ausserdem wurden verschiedene Gittergrößen verwendet.



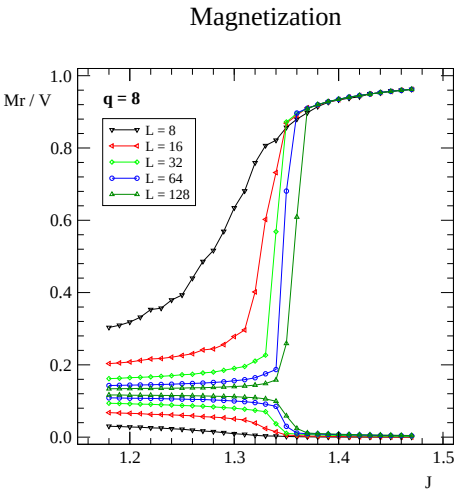
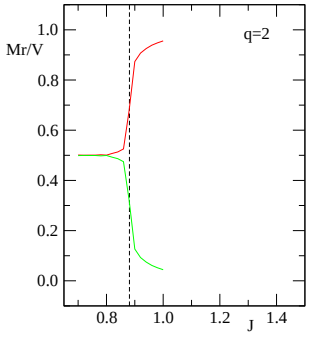
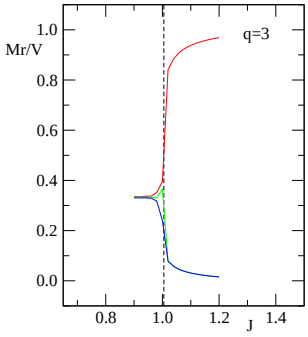
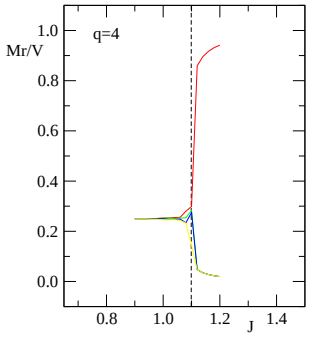
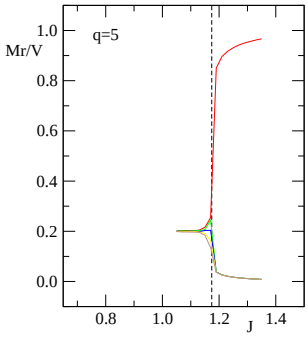


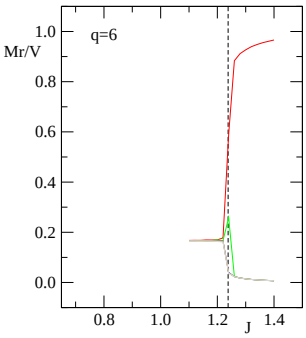
Abbildung 6: Magnetisierung als Funktion von j für unterschiedliche Gittergrößen bei $q = 2$ und $q = 8$











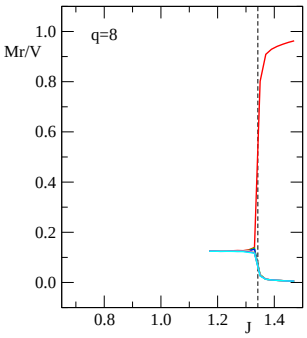


Abbildung 7: Magnetisierung als Funktion von j für verschiedene q

Der Erwartungswert für H bei unterschiedlichen Gittergrößen ist aus diesen Plots zu ersehen.

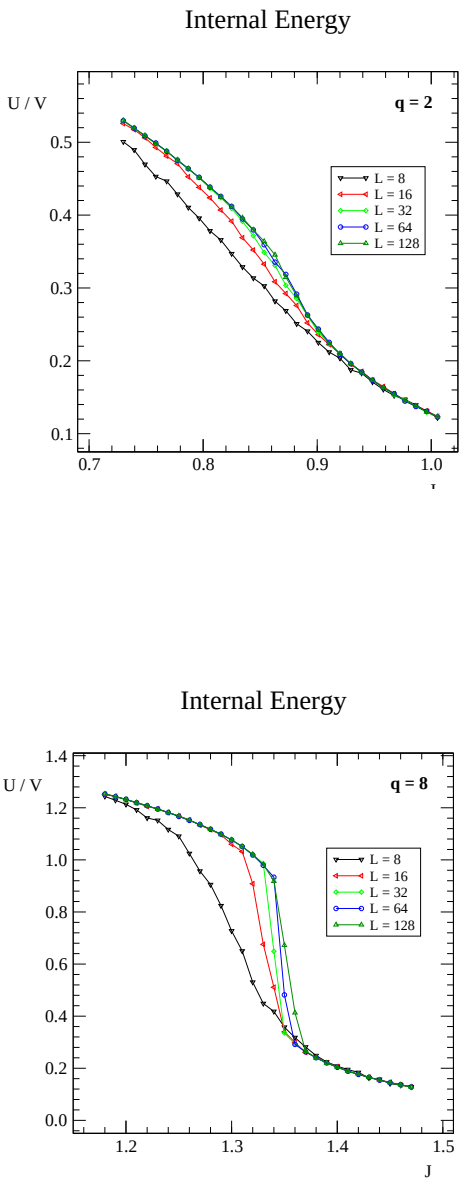


Abbildung 8: Innere Energie als Funktion von j für unterschiedliche Gittergrößen bei $q = 2$ und $q = 8$

Es folgt nun die Observablen χ, also die Suszeptibilität:

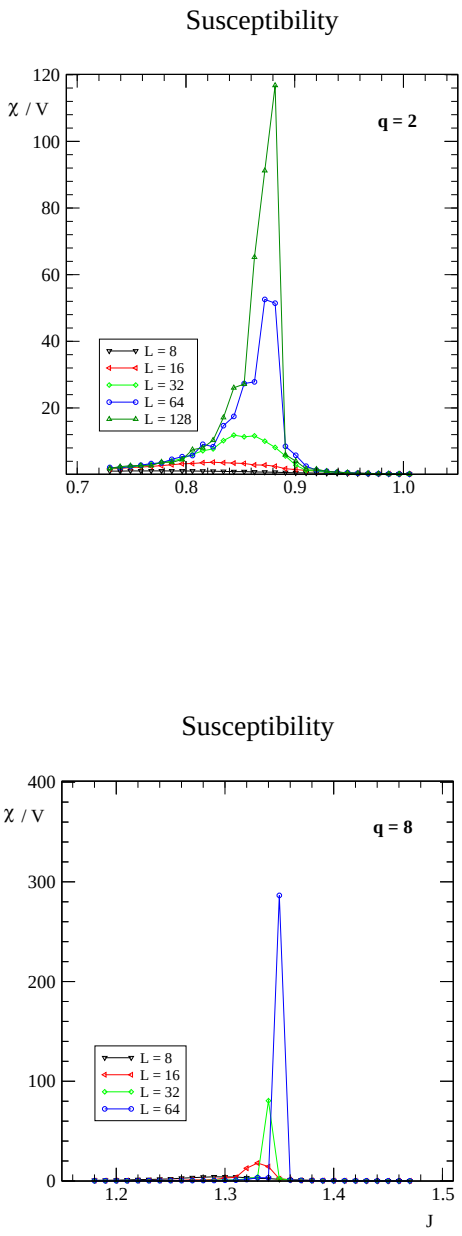


Abbildung 9: Suszeptibilität als Funktion von j für unterschiedliche Gittergrößen bei $q = 2$ und $q = 8$

Nun fehlt lediglich noch die Wärmekapazität:

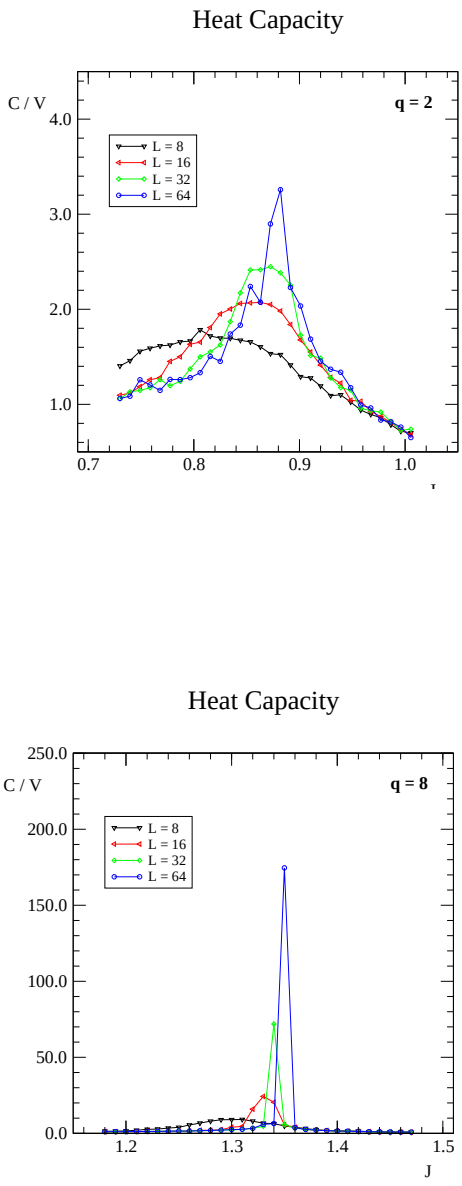


Abbildung 10: Wärmekapazität als Funktion von j für unterschiedliche Gittergrößen bei $q = 2$ und $q = 8$

5.5 Punkt 4: Berechnung der statistischen Fehler für die Magnetisierung und die innere Energie

Bei der Berechnung der statistischen Fehler wurde selbiger mit $\epsilon = \pm\frac{S_N}{\sqrt{N}}$ angenommen. Dabei erweist es sich als Vorteil, dass ja bereits die Observablen χ und C berechnet wurden, die wiederum zur Berechnung der jeweiligen Varianz herangezogen werden können. Es waren Fehler für die Observablen von $\langle H\rangle$, sowie $\langle M_r\rangle$ zu bestimmen. Dabei wurde wie folt vorgegangen:

$$S_N^2 = \frac{N}{N-1}[\bar{x_N^2} - (\bar{x_N})^2]$$

$$\Rightarrow \tag{146}$$

$$S_N = \frac{\sqrt{N}}{\sqrt{N-1}}[\bar{x_N^2} - (\bar{x_N})^2]^{\frac{1}{2}}.$$

(147)

Der Fehler ist aber gegeben durch:

$$\epsilon = \pm\frac{S_N}{\sqrt{N}}. \tag{148}$$

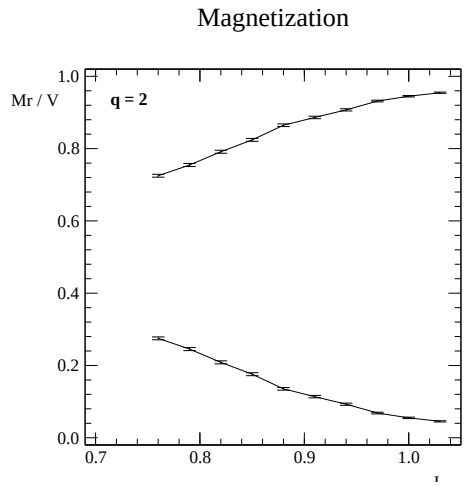
So folgt:

$$\epsilon = \frac{S_N}{\sqrt{N}} =$$

$$= \frac{1}{\sqrt{N-1}}[\bar{x_N^2} - (\bar{x_N})^2]^{\frac{1}{2}}\cdot \tag{149}$$

(150)

Konkret muss nun lediglich noch die jeweilige Observable herangezogen werden, die ja schon vorher berechnet wurde. Weiters wurde noch durch V dividiert, da ja die Observablen immer auf das Volumen bezogen werden. Die Fehlerbalken fallen sehr klein aus, aus diesem Grunde wurden nur zwei Plots, jeweils auf einem 8×8 Gitter, erzeugt.



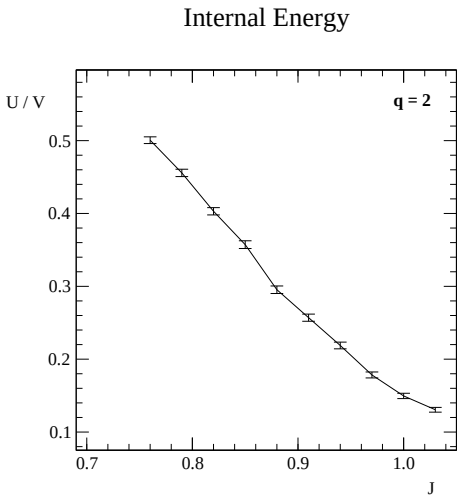
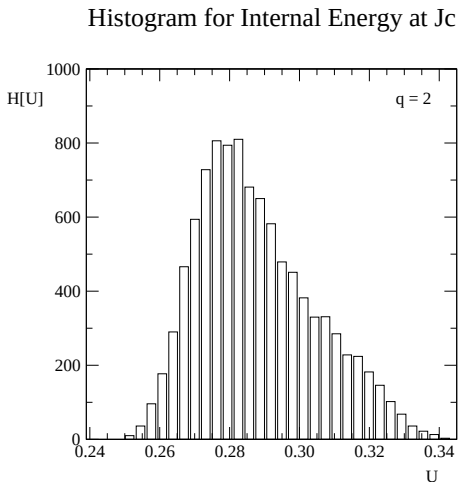


Abbildung 11: Fehlerbalken für Magnetisierung und innere Energie, jeweils auf einem 8×8 Gitter

5.6 Punkt 5: Analyse des Phasenüberganges mit der Histogrammtechnik

Auch hier bekommen wir die erwarteten Ergebnisse. Bei $q = 2$ liegt die kritische Kopplung bei $j_C = 0.881$, der Plot für $q = 8$ entstand auf einem 64×64 Gitter bei $j_C = 1.341$.



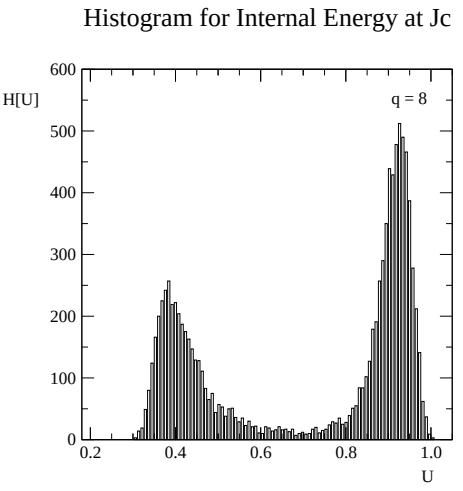


Abbildung 12: Histogramme für j_C bei $q = 2$ und $q = 8$

6 Appendix A: Source Codes

6.1 potts.cpp

```
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

const int L = 8;                              //Seitenlaenge des 2-dim Gitters
const int q = 2;                              //q-state potts model

//Funktionsprototypen

void    fillspins(int S[L*L], int start); //initialisiert die spins
void    update(int S[L*L], double J, double M, int neib[L*L][4], int nsweeps);
//macht einen nsweep sweep
void    neibinit(int neib[L*L][4]);           //initialisiert die Nachbarn
int     kronecker(int, int);                  //Ye ole' Delta
void    magnetization(int S[L*L], double mag[q]); //Misst Magnetisierung

int main(void)
{

//Variablen

const int start = 0;                          //1...hot start, 0...cold start
int S[L*L];                                   //Spinarray
int neib[L*L][4];                             //Nachbarn
double J0 = 0.73;                             //J0...Startwert
double JINC = 0.03;                           //Increment
double JSTEPS = 10;                           //Anzahl der Schritte
double J = J0;                                //J Wert
double M = 0;                                 //ext. Feld
double H = 0;                                 //Energie
int nequi = 500;                              //Equilibrierungsschritte
int nmeas = 1000;                             //Anzahl Messungen
int nskip = 30;                               //Anzahl Zwischenupdates
double aux = 0;                               //Hilfsvariable
double mag[q] = {0};                          //Magnetisierungsarray

ofstream outfile ("potts.dat");               //Ausgabestream

cout << "Simmulating q = " << q << " states Potts Model.\n";
cout << "Parameters: \n";
cout << "length L  ................. " << L << "\n";
cout << "states q  ................. " << q << "\n";
cout << "Jstart    ................. " << J0 << "\n";
cout << "Jsteps    ................. " << JSTEPS << "\n";
cout << "J-incr    ................. " << JINC << "\n";
cout << "M (ext.)  ................. " << M << "\n";
cout << "nmeas     ................. " << nmeas << "\n";
cout << "nskip     ................. " << nskip << "\n";
cout << "nequi     ................. " << nequi << "\n";

outfile << "Simmulating q = " << q << " states Potts Model.\n";
outfile << "Parameters: \n";
outfile << "length L  ................. " << L << "\n";
outfile << "states q  ................. " << q << "\n";
outfile << "Jstart    ................. " << J0 << "\n";
outfile << "Jsteps      ................. " << JSTEPS << "\n";
outfile << "J-incr    ................. " << JINC << "\n";
outfile << "M (ext.)  ................. " << M << "\n";
outfile << "nmeas     ................. " << nmeas << "\n";
outfile << "nskip     ................. " << nskip << "\n";
outfile << "nequi     ................. " << nequi << "\n";
```

```
outfile.width(18);                                  //Parameter fuer Ausgabe
outfile.precision(10);
outfile.setf(ios_base::fixed);
outfile << "\n";

neibinit(neib);                                     //Nachbarn kennenlernen
cout<< "Neighbours initialzed.\n";

fillspins(S, start);                                //Spins initialisieren
cout<< "Spins initialized.\n";


for (int i=0; i<JSTEPS; i++)                        //Beginn Messungsabschnitt
{
 J = J0 + i*JINC;

 update(S, J, M, neib, nequi);                      //Equilibrierung

 cout << "Equilibrated for j = " << J << ".\n";

 for( int imeas=0; imeas < nmeas; imeas++)          //Messungen
 {
  update(S, J, M, neib, nskip);                     //leere Updates
  magnetization(S, mag);                            //Observable Magnetisierung



  for( int i=0; i<q-1; i++)                         //sortieren der Ausgabe
  {
   for( int ii=i+1; ii<q; ii++)
   {
    if( mag[i]<mag[ii])
    {
     aux = mag[ii];
     mag[ii] = mag[i];
     mag[i] = aux;
    }
   }
  }                                                 //Ende der Sortierung

  for( int ii=0; ii<q; ii++)
  {
   outfile << mag[ii] << " ";                       //schreiben auf File
   mag[ii] = 0;
  }

  for( int iii=0; iii<L*L; iii++)                   //Energiemessung
  {
   H += kronecker(S[iii],S[neib[iii][0]])
      + kronecker(S[iii],S[neib[iii][1]]) ;
  }

  H = 2*L*L - H;                                    //Observable Energie
  outfile << H << "\n";                             //schreiben auf File
  H = 0;
 }
 cout << "j = " << J << " done.\n";

}

 cout << "Fertig!\n";
 return 0;
}


void fillspins(int S[L*L], int start)   //Spininitialisierung
```

```
{
if(start == 1)
{
 for(int i=0; i<L*L; i++)
 {
  S[i] = rand()%q + 1;                    //Zufaellige Spinausrichtung
 }
}
else
{
 for(int ii=0; ii<L*L; ii++)
 {
  S[ii] = 1;                              //Alles Spins Richtung 1
 }
}
}

void neibinit(int neib[L*L][4])           //Nachbarinitialisierung
{
int i, n1p, n1m, n2p, n2m;
for( int n1=0; n1<L; n1++)
{
 for( int n2=0; n2<L; n2++)
 {
  n1p = n1 + 1;
  n1m = n1 - 1;
  n2p = n2 + 1;
  n2m = n2 - 1;

  if(n1p == L)
   {n1p = 0;}                             //periodische RB
  if(n1m == -1)
   {n1m = L-1;}
  if(n2p == L)
   {n2p = 0;}
  if(n2m == -1)
   {n2m = L-1;}

  i = n1 + L*n2;                          //zentraler punkt (aufger. Index)
  neib[i][0] = n1p + L*n2;                //oestl. Nachbar
  neib[i][1] = n1 + L*n2p;                //noerdl. Nachbar
  neib[i][2] = n1m + L*n2;                //westl. Nachbar
  neib[i][3] = n1 + L*n2m;                //suedl. Nachbar
 }
}
}

void update(int S[L*L], double J, double M, int neib[L*L][4], int nsweeps)
{
int offer;                                //Angebotskonfiguration
double rho, r;                            //Wahrscheinlichkeiten

for(int n=0; n<nsweeps; n++)
{
 for(int i=0; i<L*L; i++)
 {
  offer = rand()%q + 1;                   //waehle einen der q Spins
  rho = exp(J*( kronecker(offer, S[neib[i][0]])
           + kronecker(offer, S[neib[i][1]])
           + kronecker(offer, S[neib[i][2]])
           + kronecker(offer, S[neib[i][3]])
           - kronecker(S[i], S[neib[i][0]])
           - kronecker(S[i], S[neib[i][1]])
           - kronecker(S[i], S[neib[i][2]])
           - kronecker(S[i], S[neib[i][3]]))
           + M*( kronecker(offer, 1)
           - kronecker(S[i], 1)));        //Bilde Verhaeltnis der WS-keiten
```

```
   r = rand()/(RAND_MAX + 1.);          //Ziehe Zufallszahl zw. 0,1 glvtlt.
   if ( r <= rho)
   {
    S[i] = offer;                       //Akzeptiere
   }

 }
}

}


int kronecker(int a, int b)             //Ye ole' Delta
{
 if(a == b)
 { return 1;}
 else
 {return 0;}
}

void magnetization(int S[L*L], double mag[q])
{
 for( int i=0; i<L*L; i++)              //Messe Magnetisierung einer Konf.
 {
  mag[S[i]-1]++;
 }
}
```

6.2 analyzer.cpp

```
#include<iostream>
#include<math.h>
#include<fstream>

using namespace std;

istream &nxl(istream& is)       //Routine schreitet eine Zeile weiter
{
        char xcr[257];
        is.getline(xcr,256);
        return is;
}


int main(void)
{
//Variablen

int ibin = 50;                  //Anzahl der Saeulen
string dummy;                   //Leere Variable zum Einlesem
int L=0;                        //Kantenlaenge
int q=0;                        //Anzahl states
double J0=0;                    //Startwert J
double JSTEPS=0;                //Anzahl Schritte
double JINC=0;                  //Inkrement
double M=0;                     //ext. Feld
double H=0;                     //Energie
double epsH=0;                  //Fehler fuer H
double C=0;                     //Waermekapazitaet
double aux=0;                   //Hilfsvariable
double j=0;                     //j-Wert
double mag[8]={0};              //Magnetisierungsarray
double epsm[8]={0};             //Fehler fuer Mr
double chi[8]={0};              //Suszeptibilitaet
double delta = 0;               //Delta fuer Histogramm
double hmin = 0;                //Minimalwert d. Energie
double hmax = 0;                //Maximalwert d. Energie
double hist[300]={0};           //Histogramm
double h[2000]={0};             //H-Mittelwerte
int nmeas = 0;                  //Anzahl Messungen
int nskip = 0;                  //Anzahl Zwischenupdates
int nequi = 0;                  //Equilibrierungsschritte

cout << "Reading parameters...\n";   //Einlesen der Rohdaten
ifstream infile ("potts.dat");
ifstream infile2 ("potts.dat");
infile >> nxl;                      //nextline
infile2 >> nxl;                     //nextline fuer zweiten Stream
infile >> nxl;
infile2 >> nxl;
infile >> dummy >> dummy >> dummy >> L;
infile2 >> nxl;
infile >> dummy >> dummy >> dummy >> q;
infile2 >> nxl;
infile >> dummy >> dummy >> J0;
infile2 >> nxl;
infile >> dummy >> dummy >> JSTEPS;
infile2 >> nxl;
infile >> dummy >> dummy >> JINC;
infile2 >> nxl;
infile >> dummy >> dummy >> dummy >> M;
infile2 >> nxl;
infile >> dummy >> dummy >> nmeas;
infile2 >> nxl;
infile >> dummy >> dummy >> nskip;
```

```
infile2 >> nxl;
infile >> dummy >> dummy >> nequi;
infile2 >> nxl;
infile >> nxl;
infile2 >> nxl;

cout << "length L  ................. " << L << "\n";
cout << "states q  ................. " << q << "\n";
cout << "Jstart    ................. " << J0 << "\n";
cout << "Jsteps    ................. " << JSTEPS << "\n";
cout << "J-incr    ................. " << JINC << "\n";
cout << "M (ext.)  ................. " << M << "\n";
cout << "nmeas     ................. " << nmeas << "\n";
cout << "nskip     ................. " << nskip << "\n";
cout << "nequi     ................. " << nequi << "\n";
cout << "\n\n\n";
cout << "Ausgabemuster:\n";
cout << "j\tMr/V\tEpsilon Mr\tChir/(beta*V)\tC/V\tH/V\tEpsilon H\t\n\n";

ofstream outfile ("finaldata.dat");      //Vorbereitung der Ausgabe
outfile.width(18);                       //fuer Magnetisierung
outfile.precision(10);
outfile.setf(ios_base::fixed);

ofstream outfile2 ("histogram.dat");     //Vorbereitung der Ausgabe
outfile2.width(18);                      //fuer Histogramm
outfile2.precision(10);
outfile2.setf(ios_base::fixed);


j = J0;                                  //Startwert setzen

for (int i=0; i<JSTEPS; i++)
{

 j += JINC;
 H = 0;                                  //H loeschen

 for (int k=0; k<8; k++)
 {
  mag[k]=0;                              //Mag-Array loeschen
  chi[k]=0;                              //Chi loeschen
 }

 for (int ii=0; ii<nmeas; ii++)          //Observable < M >
 {
  for (int iii=0; iii<q; iii++)
  {
   infile >> aux;
   mag[iii] += aux;
   //cout << aux << " ";
   aux = 0;
  }

  infile >> aux;                         //Observable < H >
  H += aux;
  aux = 0;
 }                                       //endfor nmeas-Schleife


 H = H/nmeas;                            //Mittelung Energie
 h[i] = H/(L*L);                         //Mittelwert des Datensatzes speichern

 for (int iv=0; iv<q; iv++)              //gemittelte Magnetisierung
 {
  mag[iv] = mag[iv]/nmeas;
 }
```

```
 for (int ii=0; ii<nmeas; ii++)          //Observable Chi
 {
  for (int iii=0; iii<q; iii++)
  {
   infile2 >> aux;
   chi[iii] += (aux - mag[iii])*(aux - mag[iii]);
   aux = 0;
  }

  infile2 >> aux;                        //Observable C
  C += (aux - H)*(aux - H);
  aux = 0;
 }//endfor nmeas Schleife chi

 for(int i=0; i<q; i++)
 {
  mag[i]  = mag[i]/(L*L);
  chi[i]  = chi[i]/nmeas;
  epsm[i] = sqrt(chi[i]/(nmeas-1));      //Fehler fuer Mr
  epsm[i] /= (L*L);
  chi[i]  = chi[i]/(L*L);
 }

 C = C/nmeas;
 epsH = sqrt(C/(nmeas-1));               //Fehler fuer H berechnen
 epsH /= (L*L);
 C = C/(L*L);

 cout    << j << " ";                   //Ausgabeblock fuer Screen und File
 outfile << j << " ";

 H = H/(L*L);

 for (int i=0; i<q; i++)                 //Ausgabe Magnetisierung und Fehler
 {
  cout << mag[i] << " " << epsm[i] << " ";
  outfile << mag[i] << " " << epsm[i] << " ";
 }

 for (int i=0; i<q; i++)                 //Ausgabe Chi
 {
  cout << chi[i] << " ";
  outfile << chi[i] << " ";
 }

 cout << C << " ";                       //Ausgabe C
 outfile << C << " ";
 cout << H << " " << epsH << "\n";       //Ausgabe H und Fehler
 outfile << H << " " << epsH << "\n";
}                                        //Ende Observablen


return 0;
}
```

6.3 pottshistogramm.cpp

```
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

const int L = 64;                          //Seitenlaenge des 2-dim Gitters
const int q = 2;                           //q-state potts model

//Funktionsprototypen

void    fillspins(int S[L*L], int start); //initialisiert die spins
void    update(int S[L*L], double J, double M, int neib[L*L][4], int nsweeps); //macht einen nsweep
void    neibinit(int neib[L*L][4]);        //initialisiert die Nachbarn
int     kronecker(int, int);               //Ye ole' Delta

int main(void)
{

//Variablen

const int start = 0;                       //1...hot start, 0...cold start
int S[L*L];                                //Spinarray
int neib[L*L][4];                          //Nachbarn
double J0 = 0.881;                         //J0...Startwert
int RUNS = 10000;                          //Anzahl der Durlaeufe
double M = 0;                              //ext. Feld
double H = 0;                              //Energie
int nequi = 500;                           //Equilibrierungsschritte
int nmeas = 20;                            //Anzahl Messungen
int nskip = 20;                            //Anzahl Zwischenupdates
double aux = 0;                            //Hilfsvariable

ofstream outfile ("potts_hist.dat");              //Ausgabestream

cout << "Simmulating q = " << q << " states Potts Model. Ermittle Histogrammwerte.\n";
cout << "Parameters: \n";
cout << "length L  ................. " << L << "\n";
cout << "states q  ................. " << q << "\n";
cout << "J          ................. " << J0 << "\n";
cout << "RUNS       ................. " << RUNS << "\n";
cout << "M (ext.)   ................. " << M << "\n";
cout << "nmeas      ................. " << nmeas << "\n";
cout << "nskip      ................. " << nskip << "\n";
cout << "nequi      ................. " << nequi << "\n";

outfile << "Simmulating q = " << q << " states Potts Model. Ermittle Histogrammwerte.\n";
outfile << "Parameters: \n";
outfile << "length L  ................. " << L << "\n";
outfile << "states q  ................. " << q << "\n";
outfile << "J          ................. " << J0 << "\n";
outfile << "RUNS       ................. " << RUNS << "\n";
outfile << "M (ext.)   ................. " << M << "\n";
outfile << "nmeas      ................. " << nmeas << "\n";
outfile << "nskip      ................. " << nskip << "\n";
outfile << "nequi      ................. " << nequi << "\n";

outfile.width(18);                                 //Parameter fuer Ausgabe
outfile.precision(10);
outfile.setf(ios_base::fixed);
outfile << "\n";

neibinit(neib);                                    //Nachbarn kennenlernen
cout<< "Neighbours initialzed.\n";
```

```
fillspins(S, start);                                //Spins initialisieren
cout<< "Spins initialized.\n";


for (int i=0; i<RUNS; i++)                          //Beginn Messungsabschnitt
{

 update(S, J0, M, neib, nequi);                     //Equilibrierung

 cout << "Equilibrated. Noch " << RUNS - i << " Messdurchlaeufe.\n";

 for( int imeas=0; imeas < nmeas; imeas++)          //Messungen
 {
  update(S, J0, M, neib, nskip);                    //leere Updates


  for( int iii=0; iii<L*L; iii++)                   //Energiemessung
  {
   H += kronecker(S[iii],S[neib[iii][0]])
      + kronecker(S[iii],S[neib[iii][1]]) ;
  }

  H = 2*L*L - H;                                    //Observable Energie
  outfile << H << "\n";                             //schreiben auf File
  H = 0;
 }
}



 cout << "Fertig!\n";
 return 0;
}


void fillspins(int S[L*L], int start)   //Spininitialisierung
{
if(start == 1)
{
 for(int i=0; i<L*L; i++)
 {
  S[i] = rand()%q + 1;                  //Zufaellige Spinausrichtung
 }
}
else
{
 for(int ii=0; ii<L*L; ii++)
 {
  S[ii] = 1;                            //Alles Spins Richtung 1
 }
}
}

void neibinit(int neib[L*L][4])         //Nachbarinitialisierung
{
int i, n1p, n1m, n2p, n2m;
for( int n1=0; n1<L; n1++)
{
 for( int n2=0; n2<L; n2++)
 {
  n1p = n1 + 1;
  n1m = n1 - 1;
  n2p = n2 + 1;
  n2m = n2 - 1;

  if(n1p == L)
   {n1p = 0;}                           //periodische RB
```

```
  if(n1m == -1)
   {n1m = L-1;}
  if(n2p == L)
   {n2p = 0;}
  if(n2m == -1)
   {n2m = L-1;}

  i = n1 + L*n2;                          //zentraler punkt (aufger. Index)
  neib[i][0] = n1p + L*n2;                //oestl. Nachbar
  neib[i][1] = n1 + L*n2p;                //noerdl. Nachbar
  neib[i][2] = n1m + L*n2;                //westl. Nachbar
  neib[i][3] = n1 + L*n2m;                //suedl. Nachbar
 }
}
}

void update(int S[L*L], double J, double M, int neib[L*L][4], int nsweeps)
{
int offer;                                //Angebotskonfiguration
double rho, r;                            //Wahrscheinlichkeiten

for(int n=0; n<nsweeps; n++)
{
 for(int i=0; i<L*L; i++)
 {
  offer = rand()%q + 1;                   //waehle einen der q Spins
  rho = exp(J*( kronecker(offer, S[neib[i][0]])
           + kronecker(offer, S[neib[i][1]])
           + kronecker(offer, S[neib[i][2]])
           + kronecker(offer, S[neib[i][3]])
           - kronecker(S[i], S[neib[i][0]])
           - kronecker(S[i], S[neib[i][1]])
           - kronecker(S[i], S[neib[i][2]])
           - kronecker(S[i], S[neib[i][3]]))
           + M*( kronecker(offer, 1)
           - kronecker(S[i], 1)));        //Bilde Verhaeltnis der WS-keiten

   r = rand()/(RAND_MAX + 1.);            //Ziehe Zufallszahl zw. 0,1 glvtlt.
   if ( r <= rho)
   {
    S[i] = offer;                         //Akzeptiere
   }

 }
}

}


int kronecker(int a, int b)               //Ye ole' Delta
{
 if(a == b)
 { return 1;}
 else
 {return 0;}
}
```

6.4 analyzerhistogramm.cpp

```
#include<iostream>
#include<math.h>
#include<fstream>

using namespace std;

istream &nxl(istream& is)       //Routine schreitet eine Zeile weiter
{
        char xcr[257];
        is.getline(xcr,256);
        return is;
}


int main(void)
{
//Variablen

int ibin = 30;                  //Anzahl der Saeulen
string dummy;                   //Leere Variable zum Einlesem
int L=0;                        //Kantenlaenge
int q=0;                        //Anzahl states
double J0=0;                    //Startwert J
double RUNS=0;                  //Anzahl Durchlaeufe
double M=0;                     //ext. Feld
double H=0;                     //Energie
double aux=0;                   //Hilfsvariable
double delta = 0;               //Delta fuer Histogramm
double hmin = 0;                //Minimalwert d. Energie
double hmax = 0;                //Maximalwert d. Energie
double hist[300]={0};           //Histogramm
double h[20000]={0};            //H-Mittelwerte
int nmeas = 0;                  //Anzahl Messungen
int nskip = 0;                  //Anzahl Zwischenupdates
int nequi = 0;                  //Equilibrierungsschritte

cout << "Reading parameters...\n";   //Einlesen der Rohdaten
ifstream infile ("potts_hist.dat");
infile >> nxl;                       //nextline
infile >> nxl;
infile >> dummy >> dummy >> dummy >> L;
infile >> dummy >> dummy >> dummy >> q;
infile >> dummy >> dummy >> J0;
infile >> dummy >> dummy >> RUNS;
infile >> dummy >> dummy >> dummy >> M;
infile >> dummy >> dummy >> nmeas;
infile >> dummy >> dummy >> nskip;
infile >> dummy >> dummy >> nequi;
infile >> nxl;

cout << "length L  ................. " << L << "\n";
cout << "states q  ................. " << q << "\n";
cout << "J         ................. " << J0 << "\n";
cout << "RUNS      ................. " << RUNS << "\n";
cout << "M (ext.)  ................. " << M << "\n";
cout << "nmeas     ................. " << nmeas << "\n";
cout << "nskip     ................. " << nskip << "\n";
cout << "nequi     ................. " << nequi << "\n";
cout << "\n\n\n";
cout << "Ausgabemuster:\n";
cout << "H\tAnzahl\n";

ofstream outfile ("histogram.dat");      //Vorbereitung der Ausgabe
outfile.width(18);                       //fuer Histogramm
outfile.precision(10);
```

```
outfile.setf(ios_base::fixed);


for (int i=0; i<RUNS; i++)
{
 H = 0;                                  //H loeschen
 for (int ii=0; ii<nmeas; ii++)
 {
  infile >> aux;                         //H einlesen
  H += aux;                              //Mittelung Energie
 // cout << aux << " " << i << " " << ii << "\n";
 }
 H = H/nmeas;
 h[i] = H/(L*L);
}

hmin = h[0];                             //Setzen von hmin und hmax
hmax = h[0];

for (int p=0; p<RUNS; p++)               //Minimum und Maximum suchen
{                                        //und speichern
 if( h[p] < hmin) hmin = h[p];
 if( h[p] > hmax) hmax = h[p];
}

cout << "\n\n";
cout << "Histogrammteil:\n";
cout << "\n\n hmin = " << hmin << " und hmax = " << hmax << ".\n\n";

delta = (hmax - hmin)/ibin;              //Energieintervall bestimmen
for (int r=0; r<RUNS; r++)
{
 for( int q=0; q<ibin; q++)              //Energiewert einordnen
 {
  if((h[r]>=(hmin + q*delta)) && (h[r]<=(hmin +(q+1)*delta))) hist[q]++;
 }
}

cout << "\n";

for (int s=0; s<ibin; s++)               //Ausgabe Histogramm
{
 outfile << hmin + delta*s << " " << hist[s] << "\n";
 cout << hmin + delta*s << " " << hist[s] << "\n";
}

return 0;
}
```