Autokorrelation und Fehleranalyse von Monte Carlo Daten im Bereich eines Phasenübergangs erster Ordnung

Abschlussarbeit

zur Erlangung des akademischen Grades
Bachelor of Science

vorgelegt von

Alena Zmarsly

am 12.10.2017

Institut für theorethische Physik
J.W. Goethe Universität Frankfurt a.M.

Erstgutachter:	Prof. Dr. Owe Philipsen
Zweitgutachter:	Juniorprof. Dr. Marc Wagner
Betreuer:	Dr. Alessandro Sciarra

Inhaltsverzeichnis

Zusammenfassung

In dieser Arbeit wird die Autokorrelation von Monte Carlo Daten im Bereich eines Phasenübergangs erster Ordnung untersucht. Insbesondere soll die integrierte Autokorrelationszeit am Phasenübergang erster Ordnung untersucht werden. Hierzu werden zwei Methoden zur Bestimmung der integrierten Autokorrelationszeit dargestellt, die Jackknife-Methode und die Gamma-Methode. Die integrierte Autokorrelationszeit ist ein Maß für die Korrelation des Systems. Je höher die Autokorrelation zwischen den Daten ist, umso weniger effektiv unabhängige Zustände liegen vor. Das Verhalten der integrierten Autokorrelationszeit wird an Daten untersucht, die mit einem einfachen Metropolis-Algorithmus generiert werden. Dies hat den Vorteil, dass keine Äquilibrierung notwendig ist und die Daten schnell generiert werden können. Durch die Analyse lässt sich der benötigte Datenumfang abschätzen, der für eine ausreichend genaue Datenanalyse benötigt wird. Diese Abschätzung ist sinnvoll, da die Generierung von Daten in realen Simulationen zeitaufwändig ist.

Die Analyse wird auf Basis eines Modells für den Phasenübergang erster Ordnung durchgeführt. Dem Modell liegt zugrunde, dass ein System am Phasenübergang erster Ordnung durch die Summe zweier Normalverteilungen beschrieben werden kann, und weit entfernt vom Übergang durch eine Normalverteilung.

1 Statistische Analyse und numerische Physik

Führt man ein reproduzierbares physikalisches Experiment mehrmals durch, so sind aufeinanderfolgende und alle nachfolgenden Ereignisse voneinander unabhängig. Die Wiederholung und Datenerfassung eines solchen Experiments dient der Stichprobenentnahme. Jedes Ereignis liefert einen Datenpunkt der Stichprobe. Exakte Wahrscheinlichkeiten für die auftretenden Ereignisse können nicht angegeben werden, da die Anzahl der Durchführungen endlich ist. Daraus resultiert, dass auch die Wahrscheinlichkeitsverteilung nicht exakt ist und demnach auch der Erwartungswert nur mit einer gewissen Genauigkeit angegeben werden kann. Betrachtet man im Gegensatz zu einem realen Experiment eine Simulation am Computer, ergibt sich dabei häufig, dass die einzelnen Datenpunkte der Stichprobe korreliert sind. Die Generierung der Daten basiert auf der Verwendung des Markov Kette Monte Carlo Metropolis Algorithmus.

1.1 Fehleranalyse unabhängiger Zufallszahlen

Zur Generierung von Monte Carlo Daten werden (Pseudo-) Zufallszahlen verwendet[1]. Daher soll im Folgenden erörtert werden, wie eine Fehleranalyse von unabhängigen Zufallszahlen durchgeführt werden kann[2].

Man betrachtet eine Funktion $f^r = f(x^r)$, die von einer Zufallsvariablen x^r abhängt. Dann ist f^r selbst eine Zufallszahl. Der Erwartungswert von f^r kann bestimmt werden, indem man über das Produkt der Funktion und der Wahrscheinlichkeitsdichte $\rho(x)$ der Zufallsvariablen, integriert,

$$\langle f^r \rangle = \langle f(x^r) \rangle = \int_{-\infty}^{+\infty} f(x)\rho(x)dx \ . \tag{1.1}$$

Das n-te Moment ist definiert als

$$\lambda_n = \langle (x^r)^n \rangle = \int_{-\infty}^{+\infty} x^n \rho(x)dx, \quad n = 1, 2, 3, \dots \ . \tag{1.2}$$

Das erste Moment $\lambda_1 = \langle x^r \rangle$ ist der Erwartungswert einer Zufallsvariablen, der häufig mit μ bezeichnet wird.

Das zentrale Moment beschreibt die zu erwartende Abweichung einer Zufallsvariablen von ihrem Erwartungswert in n-ter Potenz. Es ist definiert als

$$\mu_n = \langle (x^r - \langle x \rangle)^n \rangle = \int_{-\infty}^{+\infty} (x^r - \langle x \rangle)^n \rho(x)dx, \quad n = 1, 2, 3, \dots \ . \tag{1.3}$$

[1] Pseudozufallszahlen werden häufig im Bereich $[0, 1)$ generiert. Dies ist hier der Fall.

[2] Die Theorie der Fehleranalyse unabhängiger Zufallszahlen basiert hauptsächlich auf [1], Kapitel 2.

Das zweite zentrale Moment ist die Varianz

$$\mu_2 = \langle (x^r - \langle x \rangle)^2 \rangle = \sigma^2(x^r) \ . \tag{1.4}$$

Die Varianz gibt die quadratische Streuung einer Größe um ihren Erwartungswert an. Die Quadratwurzel aus der Varianz ist die Standardabweichung

$$\sigma = \sqrt{\sigma^2(x^r)} \ , \tag{1.5}$$

die zur Bestimmung der Fehlerbalken herangezogen werden kann.

Die Schiefe einer Verteilung ist definiert als

$$\gamma = \frac{\mu_3}{\sigma^3} = \frac{\langle (x^r - \langle x \rangle)^3 \rangle}{\langle (x^r - \langle x \rangle)^2 \rangle^{3/2}} \ .$$

Sie ist ein Maß für die Asymmetrie der Verteilung. Für eine symmetrische Verteilung gilt $\gamma = 0$. Eine Verteilung wird als rechtsschief bezeichnet, wenn das Gewicht der Verteilung auf dem linken Peak liegt und auf der rechten Seite flacher abfällt als auf der linken Seite. Hier gilt $\gamma > 0$. Für eine linksschiefe Verteilung gilt umgekehrt $\gamma < 0$.

Die Fehleranaylse basiert darauf, dass die zugrunde liegenden Daten normal verteilt sind. Man betrachte eine beliebige Verteilungsfunktion einer zufälligen Größe x^r, deren Erwartungswert und Varianz existieren und endlich sind. Dann besagt der zentrale Grenzwertsatz, dass die Verteilungsfunktion für eine große Zahl $n \to \infty$ identisch verteilter unabhängiger Zufallsvariablen asymptotisch gegen die Normalverteilung konvergiert. Daher ist die Normalverteilung von hoher Relevanz in der Statistik.
Die **Wahrscheinlichkeitsdichte der Normalverteilung** ist definiert als

$$\rho(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\ \sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1.6}$$

mit dem Erwartungswert μ, der Standardabweichung σ und der Varianz σ^2.
Die Parallele zur y-Achse durch den Erwartungswert $x = \mu$ bildet die Symmetrieachse der Normalverteilung. Die Standardabweichung ist ein Maß für die Höhe und die Breite der Kurve.
Die Standardnormalverteilung ist ein Spezialfall der Normalverteilung, deren Werte mit einer quadratischen Abweichung $\sigma^2 = 1$ um den Erwartungswert $\mu = 0$ verteilt sind,

$$\rho(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \ . \tag{1.7}$$

Üblicherweise sind der exakte Erwartungswert und die exakte Varianz unbekannt. Mithilfe von Schätzern kann man Informationen über diese Unbekannten gewinnen[3] . Schätzer

[3] Der folgende Verlauf orientiert sich an [1].

werden im Folgenden mit einem Dach gekennzeichnet, z.B. " $\hat{x}$ ".
Der Schätzer für das n-te Moment ist

$$(\hat{x}^n)^r = \frac{1}{N}\sum_{i=1}^{N}(x_i^r)^n \ . \tag{1.8}$$

Als Schätzer für das erste Moment erhält man den arithmetischen Mittelwert $\bar{x}$,

$$\hat{x} = \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i^r \ . \tag{1.9}$$

Die Varianz des Probenmittels ist

$$\sigma_{\bar{x}}^2 = \left\langle \left(\frac{1}{N}\sum_{i=1}^{N}(x_i^r - \langle x \rangle) \right)^2 \right\rangle \ . \tag{1.10}$$

Sind die x_i^r unkorreliert und ist die Varianz für alle x_i^r gleich, $\sigma_{x_i^r}^2 = \sigma_x^2$, so ist die Varianz des Probenmittels um den Faktor N kleiner als die Varianz einer einzelnen Zufallsvariablen. Die Summe (1.10) vereinfacht sich zu dem Ausdruck

$$\sigma_{\bar{x}}^2 = \frac{\sigma_x^2}{N} \ . \tag{1.11}$$

Hieraus folgt, dass der statistische Fehler des Probenmittels unkorrelierter Daten mit $1/\sqrt{N}$ abnimmt,

$$\sigma_{\bar{x}} = \frac{\sigma_x}{\sqrt{N}} \ . \tag{1.12}$$

Man könnte einen Schätzer für die Varianz konstruieren, indem man in (1.10) statt des Erwartungswertes $\langle x \rangle$ dessen Schätzer $\hat{x}$ verwendet. Dieser Schätzer für die Varianz stimmt jedoch nicht mit dem exakten Wert der Varianz überein. Die Ungleichheit zwischen Schätzer und exaktem Wert wird als Bias bezeichnet. Im Fall der Varianz kann man den Bias korrigieren[4] und erhält den biasfreier Schätzer der Varianz:

$$(s_x^r)^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i^r - \hat{x})^2 \ . \tag{1.13}$$

Nun soll der Erwartungswert einer Funktion f abgeschätzt werden, die nicht-linear in $\langle x \rangle$ ist. Definiert man den Schätzer $\hat{f}$ analog zu (1.9), indem man x_i^r durch $f_i^r = f(x_i^r)$ ersetzt, so verschwindet der Bias nicht. Der Schätzer konvergiert nicht zum korrekten Erwartungswert. Ein geeigneter Schätzer für den Erwartungswert einer nicht-linearen Funktion ist $\hat{f} = f(\hat{x})$.

[4] vgl. [1] S.56ff

Wegen der Nicht-Linearität der Funktion f in $\langle x \rangle$ kann man die Fehlerfortpflanzungsformeln nicht anwenden[5]. Der Grund ist, dass eine nicht-lineare Funktion statistischen Fluktuationen unterliegt, die mit der linearen Näherung, auf der die Fehlerfortpflanzung basiert, kollidiert. Eine Möglichkeit zur Berechnung des Fehlerbalkens des Schätzers $\hat{f}$ liefert die Jackknife-Methode, mit der man den Bias und den Fehler des Bias korrigieren kann. Da die Anwendung der Jackknife-Methode verglichen mit herkömmlichen Berechnungen von Fehlerbalken ein gleiches oder besseres Ergebnis liefert, ist die Anwendung in jedem Fall sinnvoll. Es sei betont, dass die Unabhängigkeit der zugrundeliegenden Daten eine wichtige Voraussetzung für die Anwendung der Jackknife-Methode ist.

Die Jackknife-Methode ist ein Resampling Verfahren. Die Rohdaten werden verwendet, um neue Datensets zu bilden. Man erhält das i. Datenset, indem man den i. Wert des Datensets auslässt und den Funktionswert an diesem neuen Datenpunkt bestimmt,

$$f_i^J = f(x_i^J) \quad \text{und} \quad x_i^J = \frac{1}{N-1} \sum_{k \neq i} x_k \ . \tag{1.14}$$

Der Jackknife Schätzer wird durch Summation der neuen Datenpunkte ermittelt,

$$\hat{f}^J = \frac{1}{N} \sum_{i=1}^{N} f_i^J \ . \tag{1.15}$$

Die Varianz kann mithilfe der Jackknife Schätzer ermittelt werden,

$$s_J^2(\hat{f}^J) = \frac{N-1}{N} \sum_{i=1}^{N} (f_i^J - \hat{f}^J)^2 \ . \tag{1.16}$$

Häufig konvergiert der Bias schneller als der statistische Fehler und kann vernachlässigt werden. Eine Korrektur, die den Bias berücksichtigt, wird notwendig, wenn der Bias die Ordnung des statistischen Fehlers erreicht[6].

Nun ist bekannt, wie aus Zufallszahlen die Schätzer für den Erwartungswert und die Varianz bestimmt werden können. Man kann den Bereich, in dem sich der zu schätzende Parameter befindet, mithilfe des Konfidenzintervalls eingrenzen[7]. Hierzu betrachtet man zunächst die Verteilungsfunktion $F(x)$, die stetig differenzierbar ist mit $F'(x) = \rho(x)$,

$$F(x) = P(x^r \leq b) = \int_{-\infty}^{b} \rho(x) dx \ . \tag{1.17}$$

[5] Bei der Fehlerfortpflanzung verwendet man, dass die Auswirkung des Fehlers in x auf eine Funktion durch eine Taylorreihe genähert werden kann, wobei nach dem linearen Glied abgebrochen wird. Gilt die direkte Proportionalität, $f(x) = c \cdot x$, so ist auch der Fehler von f proportional zum Fehler von x: $\Delta f = c \cdot \Delta x$.

[6] Details zur Korrektur mittels Jackknife Schätzern zweiter Ordnung liefert [1], S. 106ff

[7] Struktur und Inhalt des Abschnitts über das Konfidenzintervall orientieren sich an [1] und [6].

$\rho(x)$ ist die Wahrscheinlichkeitsdichte. Die Verteilungsfunktion gibt die Wahrscheinlichkeit an, mit der man die Zufallsvariable x^r im Intervall $[-\infty, b]$ antrifft. In diesem Zusammenhang wird das Quantil x_q mit $0 \leq q \leq 1$ definiert als

$$F(x_q) = F(x^r \leq x_q) = q \ . \tag{1.18}$$

Der Wert einer zufällig gewählten Variablen x^r ist mit einer Wahrscheinlichkeit von q kleiner als das zugehörige Quantil x_q.
Die Wahrscheinlichkeit, eine zufällig gewählte Variable x^r im Bereich zwischen x_{q_1} und x_{q_2} zu finden, beträgt

$$P(x_{q_1} \leq x^r \leq x_{q_2}) = q_2 - q_1 \tag{1.19}$$

und heißt **Konfidenz**. Der zugehörige Bereich $[x_{q_1}, x_{q_2}]$ wird als **Konfidenzintervall** bezeichnet.
Zu einer Wahrscheinlichkeit p existiert ein korrespondierendes symmetrisches Konfidenzintervall

$$[x_q, x_{1-q}] \ \text{ mit } \ q = \frac{1}{2}(1-p) \ , \tag{1.20}$$

in dem ein zufällig gewähltes x^r mit einer Wahrscheinlichkeit p anzutreffen ist.
Für die Normalverteilung lässt sich dieses Intervall in Abhängigkeit der Standardabweichung σ angeben,

$$[-n\sigma, +n\sigma], \quad n = 1, 2, 3, ... \ . \tag{1.21}$$

Die **Tschebyschev'schen Ungleichung** sagt aus, dass die Wahrscheinlichkeit, eine Zufallsvariable in einem Bereich zu finden, der um mehr als $g\sigma$ vom Erwartungswert abweicht, kleiner ist als g^{-2},

$$P(|x^r - \langle x \rangle| > g\sigma) < g^{-2}, \quad g \in \mathbb{R} \ . \tag{1.22}$$

Daher ist es ausreichend, Werte bis $n = 3$ zu betrachten. In einem Bereich mit einer Abweichung von 3σ liegen $99,7\%$ der Werte. $95,45\%$ der Werte liegen im Intervall mit der Standardabweichung 2σ und im Intervall mit der Standardabweichung 1σ liegen $68,27\%$ der Werte.

Der exakte Erwartungswert $\langle x \rangle$ liegt nun im Konfidenzintervall mit einer Abweichung $n\sigma$ vom Schätzer des Erwartungswertes $\hat{x}$,

$$\langle x \rangle \ \in \ [\hat{x} - n\sigma, \hat{x} + n\sigma], \quad n = 1, 2, 3, ... \ . \tag{1.23}$$

Nach der Tschebyschev'schen Ungleichung ist die Wahrscheinlichkeit am größten, Werte in einem Abstand von $\pm\sigma$ um den Schätzer zu finden. Der exakte Erwartungswert befindet sich somit in diesem Bereich,

$$\langle x \rangle = \hat{x} \pm \Delta\hat{x} \ . \tag{1.24}$$

Der Fehlerbalken $\Delta\hat{x}$ ist entweder die exakte Standardabweichung σ oder ein Schätzer der Standardabweichung $\Delta\hat{x} = s^r_{\bar{x}}$.

Möchte man überprüfen, ob zwei empirisch gefundene Größen, zum Beispiel die Erwartungswerte, übereinstimmen, kann man den **Gauß'schen Differenztest**[8] anwenden. Seien $\bar{x}^r$ und $\bar{y}^r$ die normal verteilten Mittelwerte der Probe. Die Differenz der beiden Mittelwerte $D^r = \bar{x}^r - \bar{y}^r$ ist ebenfalls normal verteilt mit der Varianz $\sigma_D^2 = \sigma_{\bar{x}}^2 - \sigma_{\bar{y}}^2$. Zunächst bildet man den Quotienten aus der Differenz und der Standardabweichung als Wurzel aus der Varianz,

$$d^r = \frac{D^r}{\sigma_D} \ . \tag{1.25}$$

Die Wahrscheinlichkeit, dass dieser Quotient betragsmäßig kleiner ist als ein Schrankenwert d, kann durch die Fehlerfunktion beschrieben werden,

$$P = P(|d^r| \leq d) = \mathrm{erf}\left(\frac{d}{\sqrt{2}}\right) \ . \tag{1.26}$$

Da $d \in \mathbb{R}^+$ ist $P \in [0, 1)$. Hierbei hängt $d = 1, 2, 3$ direkt zusammen mit $b = 1\sigma, 2\sigma, 3\sigma$, wenn man das Argument der Fehlerfunktion als $b/\sigma\sqrt{2}$ definiert.
Die Wahrscheinlichkeit, dass die Differenz $|\bar{x} - \bar{y}|$ zufälliger Natur ist, beschreibt die Gegenwahrscheinlichkeit zu P,

$$Q = 1 - P = 1 - \mathrm{erf}\left(\frac{d}{\sqrt{2}}\right) \ . \tag{1.27}$$

Hat Q einen hohen Wert, so ist die Wahrscheinlichkeit hoch, dass die Differenz der beiden Mittelwerte zufällig ist. In diesem Fall stimmen die Erwartungswerte der beiden Observablen überein, $\langle x \rangle = \langle y \rangle$. Ist Q klein, so ist die Wahrscheinlichkeit klein, dass die Erwartungswerte zufällig voneinander abweichen. Daraus kann man schließen, dass $\bar{x}$ und $\bar{y}$ nicht übereinstimmen. Man erhält für $d = 1, 2, 3$ folgende Werte: $Q^{1\sigma} = 0,317$, $Q^{2\sigma} = 0,0455$ und $Q^{3\sigma} = 0,0027$. Die Wahl des Schrankenwerte Q_{cut} kann individuell gewählt werden. In der Auswertung werden Werte mit $Q < Q_{cut} = 0,0455$ noch zugelassen. Es sollen maximal Werte berücksichtigt werden, die im 3σ-Bereich liegen. Daher wird angenommen, dass die Werte signifikant variieren, sobald Q eine untere Schranke $Q_{cut,min} = 0,0027$ unterschreitet.

Häufig existieren mehrere unabhängige Messungen, Replika, die zusammengefasst werden sollen. Replika können erstellt werden, indem man die Simulation mehrmals durchführt, entweder nacheinander mit dem selben Computer oder (parallel) an verschiedenen Computern. Nun soll angenommen werden, die relativen Gewichte der verschiedenen Beobachtungen seien bekannt. Dies ist der Fall, wenn jede Beobachtung der Durchschnitt von N_i unabhängigen Messungen ist, die mit der gleichen Wachrscheinlichkeitsdichte generiert wurden. Der Mittelwert einer einzelnen Messung wird bestimmt durch

$$\bar{x}_i^r = \frac{1}{N_i} \sum_{j=1}^{N_i} x_j^r \ . \tag{1.28}$$

[8] vgl. [1], S. 60f

Den Mittelwert der verschiedenen Messungen berechnet man als gewichteten Durchschnitt,

$$\bar{x}^r = \sum_{i=1}^{N} w_i \bar{x}_i^r \text{ mit } w_i = \frac{N_i}{\sum_{i=1}^{N} N_i} \ . \tag{1.29}$$

w_i ist die Gewichtung und die Summe der w_i ist auf eins normiert. Der biasfreier Schätzer der Varianz wird analog wie zuvor berechnet, allerdings unter Berücksichtigung der jeweiligen Gewichtung,

$$(s_{\bar{x}}^r)^2 = \frac{1}{N-1} \sum_{i=1}^{N} w_i (x_i^r - \bar{x}^r)^2 \ . \tag{1.30}$$

In der Theorie wird die Fehleranalyse unabhängiger normal verteilter Zufallszahlen durchgeführt. In der Realität sind die Rohdaten, die in einer Monte Carlo Simulation generiert werden, weder unabhängig, noch normal verteilt. Das Binning schafft diese Voraussetzungen .
Die Rohdaten werden gebinnt, indem die N Rohdaten auf N_B Bins mit jeweils B Daten verteilt werden. Die Gesamtanzahl der Daten entspricht im Idealfall, aber nicht notwendigerweise, dem Produkt aus der Anzahl der Bins und der Anzahl der Daten pro Bin $N = N_B \cdot B$. Man berechnet die Mittelwerte der der Daten in den einzelnen Bins,

$$x_j^B = \frac{1}{B} \sum_{i=1+(j-1)B}^{jB} x_i \quad \text{für} \quad j = 1, ...N_B \ . \tag{1.31}$$

Jeder Mittelwert liefert einen neuen Datenpunkt. Häufig werden die Daten in der Reihenfolge zusammengefasst, in der die Daten generiert wurden $(x_1, ..., x_B;\ x_{B+1}, ..., x_{2B}; ...)$.

Bildet man das Probenmittel, so hat dieses den gleichen Erwartungswert wie eine einzelne Zufallszahl, aber einen um $\sqrt{N}$ kleineren Fehler, $\sigma_{\hat{x}} = \sigma/\sqrt{N}$. Dieser neue, durch Binning entstandene Datenpunkt hat einen geringeren Fehler als die Rohdaten. Je größer die Anzahl der Daten pro Bin ist, umso geringer ist die Korrelation zwischen den Bins. Bei ausreichend großer Anzahl B ist die Autokorrelation zu vernachlässigen. Die Anzahl der Bins darf nur so weit reduziert werden, dass eine ausreichende Zahl an Daten erhalten bleibt. Betrachtet man den Fall einer großen Anzahl an Daten pro Bin und einer ausreichend hohen Anzahl an Bins, so können die Daten als unabhängig angesehen werden. Die Korrelation, die bei Daten, die durch Markov Prozesse generiert wurden, auftritt, kann bei den gebinnten Daten vernachlässigt werden, sofern $B \gg \tau_{int}$. Unterliegen die Zufallszahlen der gleichen Wahrscheinlichkeitsverteilung und existieren der Erwartungswert und die Varianz und sind endlich, so konvergiert die Verteilungsfunktion der gebinnten Daten gegen die Normalverteilung.

1.2 Importance Sampling und Markov Ketten

Eine der zentralen Aufgaben in der numerischen Physik ist es, statistische Mittelwerte bzw. hochdimensionale Integrale zu bestimmen. Die beiden folgenden Abschnitte orientieren sich an [2], Kapitel 4.
Der Erwartungswert einer Funktion $f(x) : [a, b] \to \mathbb{R}$ bezüglich der Wahrscheinlichkeitsverteilung $\rho(x)$ ist definiert durch

$$\langle f \rangle = \frac{\int_a^b dx \rho(x) f(x)}{\int_a^b \rho(x) dx} \ . \tag{1.32}$$

Die *Importance Sampling* Monte Carlo Methode wird zur Approximation solcher Integrale herangezogen. Sie rechtfertigt es, große Summen durch eine vergleichsweise kleine Probe zu ersetzen, deren Beiträge gewichtet sind. Daher hat die *Importance Sampling* Monte Carlo Methode eine hohe Relevanz für die statistische Physik.
Man bestimmt den Probenmittelwert, indem man relevantere Konfigurationen höher gewichtet in die Summe einfließen lässt als weniger relevante Konfigurationen. In der statistischen Physik wird häufig der Boltzmannfaktor e^{-S} für die Gewichtung verwendet. Eine kleine Wirkung S führt zu einer hohen Gewichtung der Konfiguration. Unter Anwendung des *Importance Sampling* lässt sich der Erwartungswert der Funktion $f(x)$ durch die Mittelung der Funktionswerte $f(x_n)$ annähern,

$$\langle f \rangle \approx \hat{f} = \frac{1}{N} \sum_{n=1}^{N} f(x_n) \ . \tag{1.33}$$

$\hat{f}$ ist der Schätzer des Erwartungswertes $\langle f \rangle$ der Funktion f. Die x_n sollen nach einer normierten Wahrscheinlichkeitsdichte verteilt sein,

$$dP(x) = \frac{\rho(x) dx}{\int_a^b \rho(x) dx} \ . \tag{1.34}$$

Zufallszahlen, die einer normierten Verteilung folgen, kann man mit dem Markov Prozess [9] generieren. Im einfachsten Fall wird der Zustand U_{n+1} statistisch aus dem Zustand U_n gewonnen ohne Berücksichtigung des Zustandes U_{n-1} und aller vorhergehenden. Da der Übergang probabilistisch stattfindet, sind die Zustände U_n, U_{n+1} Zufallsvariablen, die nur vom vorhergehenden Zustand abhängen. Eine Markov Kette muss imstande sein, alle möglichen Zustände in einer endlichen Anzahl von Schritten zu erreichen. Die Wahrscheinlichkeit für den Übergang von U nach U' wird durch die Übergangsmatrix $T(U'|U)$ beschrieben. In der statistischen Physik betrachtet man häufig Übergangswahrscheinlichkeiten, die proportional zum Boltzmannfaktor $P(U'|U) \sim e^{-S}$ sind. Eine kleine Wirkung führt zu einer großen Wahrscheinlichkeit.
Die Wahrscheinlichkeit, von einem Zustand U in einen Zustand U' zu wechseln liegt zwischen Null und eins. Die Wahrscheinlichkeit, von einem Zustand U in einen beliebigen

[9] Details zum Markov Prozess können in [1], [2] und [4] gefunden werden.

Zustand U' zu wechseln ist 1, hierzu zählt auch $U' = U$. Betrachtet man einen Schritt $U_n \to U_{n+1}$, so muss die Wahrscheinlichkeit, in den Zustand U' überzugehen genauso groß sein, wie die Wahrscheinlichkeit, aus diesem Zustand heraus zu wechseln. Dies ist die Aussage der Gleichgewichtsgleichung,

$$\sum_U T(U'|U)P(U) = \sum_U T(U|U')P(U') \ . \tag{1.35}$$

Eine Lösung der Gleichgewichtsgleichung kann die termweise Gleichheit liefern,

$$T(U'|U)P(U) = T(U|U')P(U') \ . \tag{1.36}$$

Viele Algorithmen, auch der im Folgenden erläuterte Metropolis Algorithmus, verwenden diese Bedingung.

1.3 Metropolis Algorithmus

Der Metropolis Algorithmus ist ein Markov Kette - Monte Carlo - Verfahren, das dazu dient, eine Reihe aufeinanderfolgender Zustände zu generieren. Im Folgenden soll der Fall betrachtet werden, dass die Zustände gemäß der Boltzmann-Verteilung verteilt sind.

Der Metropolis Algorithmus besteht aus zwei Schritten:

1. Schritt: Sei $U_n = U$ die Konfiguration, die aktualisiert werden soll. Man wählt eine neue Konfiguration U' mit einer Übergangswahrscheinlichkeit $T_0(U'|U)$.

2. Schritt: U' wird mit einer Akzeptanzwahrscheinlichkeit

$$T_A(U'|U) = \min\left(1, \frac{T_0(U|U')e^{-S[U']}}{T_0(U'|U)e^{-S[U]}}\right) \tag{1.37}$$

als neue Konfiguration angenommen.

Geht man von einer symmetrischen Akzeptanzwahrscheinlichkeit aus, so kann man mit

$$T_0(U|U') = T_0(U'|U) \tag{1.38}$$

den Ausdruck für die Akzeptanzwahrscheinlichkeit vereinfachen zu

$$T_A(U'|U) = \min(1, e^{-\Delta S}) \quad \text{mit} \quad \Delta S = S[U'] - S[U] \ . \tag{1.39}$$

Die Akzeptanz des neuen Zustandes U' hängt somit von der Änderung der Wirkung ΔS ab. Es werden die folgenden zwei Fälle betrachtet:

$\Delta S < 0$: Dies ist äquivalent zu $e^{|\Delta S|} > 1$. In diesem Fall wird der neue Zustand U' akzeptiert. Das System befindet sich in einem energetisch günstigeren Zustand $U_{n+1} = U'$.

$\Delta S > 0$: In diesem Fall vergleicht man $e^{-|\Delta S|} < 1$ mit einer Zufallszahl $R \in [0, 1)$. Der neue Zustand wird akzeptiert, falls $e^{-|\Delta S|} \geq R$ und das System befindet sich im Zustand $U_{n+1} = U'$.
Gilt $e^{-|\Delta S|} < R$, so behält man den alten Zustand und das System befindet sich weiterhin im Zustand U, $U_{n+1} = U_n = U$.

1.4 Generierung korrelierter Daten

Nun soll die Anwendung des Metropolis Algorithmus zur Generierung korrelierter, normal verteilter Daten dargestellt werden. Meine Datenanalyse basiert auf dieser Anwendung. Sei x' der neue Zustand, x sei der alte Zustand. Zudem sei angenommen, dass die Standardabweichungen und der Erwartungswert beider Zustände gleich ist.

1. Schritt Der neue Zustand x' wird generiert, indem man mithilfe einer Zufallsvariablen den neuen Wert innerhalb eines Radius a um den alten Wert wählt:

$$x' = x + a(2x^r - 1) = x + aq, \tag{1.40}$$

mit $a \in \mathbb{R}^{+}$[10] und $x^r \in [0, 1] \rightarrow q \in [-1, 1] \rightarrow x' \in [x - a, x + a]$.

2. Schritt Die Wahrscheinlichkeitsverteilung sei $P(x)$. Daraus ergibt sich für die Akzeptanzwahrscheinlichkeit

$$P_{accept} = \frac{P(x')}{P(x)} \ . \tag{1.41}$$

Zur Generierung der Daten in der vorliegenden Arbeit wird die Gaußverteilung als Wahrscheinlichkeitsverteilung verwendet. Daraus erhält man die Akzeptanzwahrscheinlichkeit,

$$P_{accept} = \frac{P(x')}{P(x)} = \frac{e^{-\frac{1}{2}\left(\frac{x'-\mu}{\sigma}\right)^2}}{e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}} \ . \tag{1.42}$$

Für die Standardnormalverteilung mit $\sigma = 1$ und $\mu = 0$ ergibt sich eine vereinfachte Form für die Akzeptanzwahrscheinlichkeit,

$$P_{accept} = \exp\left(-\frac{x'^2 - x^2}{2}\right) \ . \tag{1.43}$$

[10] Wählt man $a = 0$, so ergibt sich aus obiger Gleichung für jeden Schritt $x' = x$. Der Zustand x ändert sich nicht. Die integrierte Autokorrelationszeit divergiert.

Für die Summe zweier gleichgewichteter Normalverteilungen mit Mittelwerten μ_1 und μ_2 und Varianzen σ_1^2 und σ_2^2 erhält man für die Akzeptanzwahrscheinlichkeit den folgenden Ausdruck,

$$P_{accept} = \frac{P(x')}{P(x)} = \frac{e^{-\frac{1}{2}\left(\frac{x'-\mu_1}{\sigma_1}\right)^2} + e^{-\frac{1}{2}\left(\frac{x'-\mu_2}{\sigma_2}\right)^2}}{e^{-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2} + e^{-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2}} \ . \tag{1.44}$$

Für die Akzeptanzwahrscheinlichkeit der Summe zweier Normalverteilungen hat es keinen Sinn, eine Differenz des Exponenten analog zur Wirkung zu definieren. Daher soll hier entsprechend das Verhältnis zwischen $P(x')$ und $P(x)$ betrachtet werden.
Falls $P(x') > P(x)$ ist, wird der neue Zustand x' akzeptiert, $x_{n+1} = x'$.
Ist $P(x') < P(x)$, so wird P_{accept} mit der Zufallsvariablen $R \in [0, 1)$ verglichen. Ist $P_{accept} > R$, so akzeptiert man den neuen Zustand x', $x_{n+1} = x'$. Andernfalls behält man den Zustand x, $x_{n+1} = x_n = x$.
Die Wahl des Parameters a, die der Generierung des neuen Zustandes dient, trägt zur Effizienz des hier verwendeten Algorithmus bei. Wählt man den Wert für a zu klein, ist die Akzeptanzrate für den neuen Zustand hoch, man erhält jedoch gleichzeitig einen hohen Wert für die integrierte Autokorrelationszeit, die im folgenden Kapitel näher untersucht werden soll. Wählt man den Wert für a zu hoch, werden neue Konfigurationen nur mit geringer Wahrscheinlichkeit akzeptiert. Ein geeigneter Wert ist $a = 3$, für den die Akzeptanzwahrscheinlichkeit bei $P_{accept} \approx 50\%$ liegt[11]. Dieser Wert wird durchgehend in dieser Arbeit verwendet.

[11] [1] S. 205

2 Integrierte Autokorrelationszeit

Im Folgenden soll genauer auf die Analyse autokorrelierter Monte Carlo simulierter Daten eingegangen werden. Im vorherigen Kapitel wurde die Fehleranalyse anhand unabhängiger Zufallsvariablen durchgeführt. Daten, die in einer Markov Kette generiert wurden, hängen vom vorhergehenden Zustand ab. Dies führt zu einer Autokorrelation der Daten. Im Folgenden wird die integrierte Autokorrelationszeit eingeführt. Sie ist ein Maß für die Stärke der Autokorrelation. Im Anschluss werden die Jackknife- und die Gamma-Methode zur Bestimmung der integrierten Autokorrelationszeit erörtert.

Zur Analyse der integrierten Autokorrelationszeit mit der Jackknife- und der Gamma-Methode habe ich ein Programm geschrieben, das zur Anwendung der Jackknife-Methode das `C++`-Programm von Dr. Alessandro Sciarra aus der Arbeitsgruppe von Prof. Dr. Owe Philipsen der J.W.Goethe-Universität Frankfurt am Main aufruft. Zur Anwendung der Gamma Methode wird die Python-Implementierung von Dirk Hesse[12] aufgerufen. Beide Programme müssen für das von mir geschriebene Programm verfügbar sein, damit eine Analyse durchgeführt werden kann. Der Code zur Bestimmung der integrierten Autokorrelationszeit befindet sich in Anhang A.2.

2.1 Theorie

Als Messobservablen werden im Folgenden die Zustände x_i betrachtet[13]. Die folgende Herleitung der integrierten Autokorrelationszeit orientiert sich an [1].
Der Abstand zwischen zwei Zuständen sei $t = |i-j|$, dabei ist der Abstand zwischen zwei aufeinanderfolgenden Messungen konstant. Der Zustand x_j ist derjenige Zustand, der den Abstand t zum Zustand x_i hat. Für unkorrelierte Daten gilt $\langle x_i x_j \rangle = \langle x_i \rangle \langle x_j \rangle$. Daten, die mit einem Markov Kette Monte Carlo Algorithmus generiert wurden, sind autokorreliert. Dies hat zur Folge, dass die Autokorrelationsfunktion $C(t)$ nicht verschwindet:

$$C(t) = C_{ij} = \langle (x_i - \langle x \rangle)(x_j - \langle x \rangle) \rangle = \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle = \langle x_0 x_t \rangle - \langle x \rangle^2 \ . \tag{2.1}$$

Die Autokorrelationsfunktion gibt an, wie stark der Zustand x_j nach t Updates vom Zustand x_i abweicht. Zum Zeitpunkt $t = 0$ liegt der Anfangszustand x_A vor. Die Autokorrelationsfunktion bei einem Abstand $t = 0$ ist die Varianz,

$$C(0) = \sigma^2(x) \ . \tag{2.2}$$

Abbildung 2.1 zeigt den Verlauf der Autokorrelationsfunktion. Zu Beginn hat sie den Wert $C(0) = 1$, unter Verwendung normal verteilter Zufallsvariablen mit Varianz $\sigma^2 = 1$. Mit wachsendem Abstand t zum Anfangszustand x_A sinkt die Übereinstimmung des

[12] Die Python-Implementierung der Gamma-Methode ist online über *GitHub* verfügbar: `https://github.com/dhesse/py-uwerr` und muss dem von mir geschriebenen Analyseprogramm zur Verfügung stehen.

[13] Für eine allgemeinere Behandlung unter Berücksichtigung von Funktionen $f_i = f(x_i)$ der Zustände x_i sei auf [1] verwiesen.

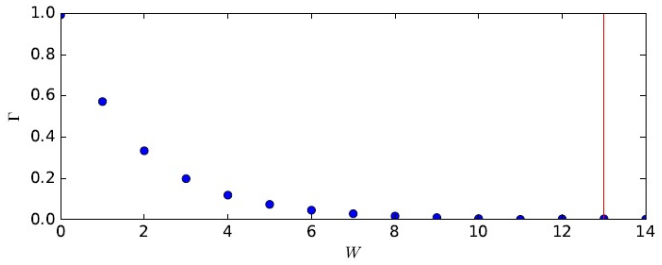


Abbildung 2.1: *Verlauf der Autokorrelationsfunktion in Abhängigkeit von t für* $200k$ *Daten.*

t	1	2	3	4	5	...	N
1	0	1	2	3	4		
2	1	0	1	2	3		
3	2	1	0	1	2		⋮
4	3	2	1	0	1		
5	4	3	2	1	0		
⋮						⋱	
N			...				0

Tabelle 2.1: *Darstellung der Häufigkeit der Abstände t in der Summe der Autokorrelationsfunktionen in* (2.3)*. In der ersten Zeile sind die Werte für i dargestellt, die erste Spalte zeigt die Werte für j. Die Zellen im Zwischenraum geben den Abstand t an. Auf der Diagonalen befindet sich der Abstand* $t = 0$*.*

Zustandes x_j vom Anfangszustand, die Autokorrelation nimmt ab und geht schließlich gegen Null. Der Zustand x_j kann dann als unabhängig vom Anfangszustand betrachtet werden.
Die Varianz des Schätzers ist proportional zur Summe über die Autokorrelationsfunktion,

$$\sigma^2(\hat{x}) = \langle(\hat{x} - \langle x\rangle)^2\rangle = \frac{1}{N^2}\sum_{i,j}\langle(x_i - \langle x\rangle)(x_j - \langle x\rangle)\rangle = \frac{1}{N^2}\sum_{i,j} C_{ij} \ . \tag{2.3}$$

Die Terme für $t = 0$, die die Varianz $C(0)$ beinhalten, treten N-mal in der Summe auf. Die Terme für $t \in [1, N-1]$ müssen in der Summe $2(N-t)$-mal berücksichtigt werden. Tabelle 2.1 veranschaulicht dies. Damit vereinfacht sich der Term zu

$$\sigma^2(\hat{x}) = \frac{1}{N^2}\left[NC(0) + \sum_{t=1}^{N-1} 2(N-t)C(t)\right] = \frac{\sigma^2(x)}{N}\left[1 + 2\sum_{t=1}^{N-1}\left(1 - \frac{t}{N}\right)c(t)\right] \ . \tag{2.4}$$

$c(t) = C(t)/C(0)$ ist die auf die Varianz normierte Autokorrelationsfunktion. Der Faktor vor der Klammer auf der rechten Seite in (2.4) ist die Varianz für unkorrelierte Daten,

der Term in der Klammer definiert die integrierte Autokorrelationszeit,

$$\tau_{int} = 1 + 2 \sum_{t=1}^{N-1} \left(1 - \frac{t}{N}\right) c(t) \ . \tag{2.5}$$

Die Varianz für korrelierte Daten ist somit um den Faktor τ_{int} größer als die Varianz unkorrelierter Daten,

$$\sigma^2(\hat{x}) = \frac{\sigma^2(x)}{N} \ \tau_{int} \ . \tag{2.6}$$

Für $N \to \infty$ strebt t/N gegen Null, man erhält für die integrierte Autokorrelationszeit

$$\lim_{N\to\infty} \tau_{int} = 1 + 2 \sum_{t=1}^{\infty} c(t) \ . \tag{2.7}$$

Die Varianz des nach (2.7) definierten Schätzers $\hat{\tau}_{int}$ divergiert für $N \to \infty$. Daher ist es sinnvoll, ein Fenster zu wählen, in dem τ_{int} nahezu unabhängig von t ist,

$$\hat{\tau}_{int}(t) = 1 + 2 \sum_{t'=1}^{t} \hat{c}(t') \ . \tag{2.8}$$

Man erhält den Schätzer für τ_{int}, indem man die Autokorrelationsfunktionen, angefangen bei $t' = 1$, bis t aufsummiert. Ist die Korrelation stärker, so bleibt die Autokorrelationsfunktion bis zu einem höheren Wert von t erhalten und ungleich Null. Viele Terme der Summe leisten einen Beitrag zum Wert für $\tau_{int}(t)$. $\tau_{int}(t)$ wird umso größer, je größer die Autokorrelation der Daten ist, da mehr Glieder der Summe berücksichtigt werden. Ab einem gewissen Wert für t_{cut} nähert sich die Autokorrelationsfunktion der Null. Autokorrelationsfunktionen für $t > t_{cut}$ leisten nur noch einen kleinen Beitrag zur Summe der integrierten Autokorrelationszeit. Der Verlauf der integrierten Autokorrelationszeit saturiert. Bis zu diesem t_{cut} wird die Summe ausgeführt. Der Wert aus diesem Fenster ergibt den Endwert für den Schätzer $\hat{\tau}_{int}$. Man muss jedoch berücksichtigen, dass $t \gg \tau_{int}$, um den statistischen Fehler so gering wie möglich zu halten. Zudem darf t nicht zu groß gewählt werden, da ansonsten Terme mit starkem Rauschen berücksichtigt werden.

Die integrierte Autokorrelationszeit legt die Anzahl der Daten fest, die effektiv als unkorreliert betrachtet werden können. Die Anzahl der ursprünglichen Daten reduziert sich auf

$$N_{unkorr} = \frac{N}{\tau_{int}} \ . \tag{2.9}$$

Demnach ist τ_{int} ein Maß für die Effizienz des zugrundeliegenden Algorithmus. Je kleiner τ_{int} ist, umso mehr effektiv unkorrelierte Daten liegen vor.

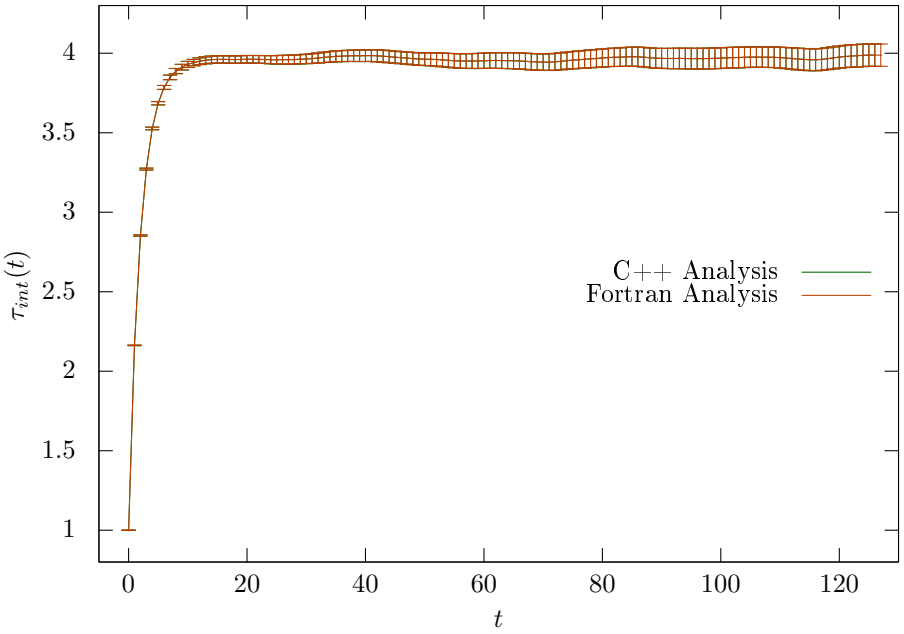


Abbildung 2.2: *Vergleich des Verlaufs der integrierten Autokorrelationszeit für $N = 2^{21}$. Analysiert mit den Analyse-Programmen von Berg und Dr. Sciarra.*

2.2 Abschätzung von τ_{int} mit der Jackknife-Methode

Die Darstellung der Jackknife-Methode basiert auf [1]. Berg liefert eine detaillierte Beschreibung seines Codes zur Bestimmung der integrierten Autokorrelationszeit unter Verwendung von Jackknife Schätzern der Autokorrelationsfunktion. Hier soll das Vorgehen kurz skizziert werden und anschließend auf die praktische Auswertung eingegangen werden. Zunächst wird der Schätzer der Autokorrelationsfunktion $\hat{C}(t)$ berechnet. Daraus wird die integrierte Autokorrelationsfunktion ermittelt. Die Jackknife Schätzer der Autokorrelationsfunktion werden ermittelt und verwendet, um die Jackknife Schätzer der integrierten Autokorrelationszeit zu gewinnen. Man erhält die Jackknife Schätzer für τ_{int}, definiert nach (2.8).

Die Berechnung der integrierten Autokorrelationszeit wird mit dem Analyse-Programm in `C++` durchgeführt.
Zunächst wurde überprüft, ob das Analyseprogramm von Dr. Sciarra das gleiche Resultat liefert wie das Programm von Berg. Dazu wurden die von Berg generierten Daten sowohl mit dem in `FORTRAN` geschriebenen Programm, das Berg in seinem Buch verwendet, als auch mit dem in `C++` geschriebenen Programm von Dr. Sciarra analysiert. Die Ergebnisse sind in Abbildung 2.2 zu sehen. Da beide Kurven übereinander liegen, führen beide Analysen zum gleichen Ergebnis.

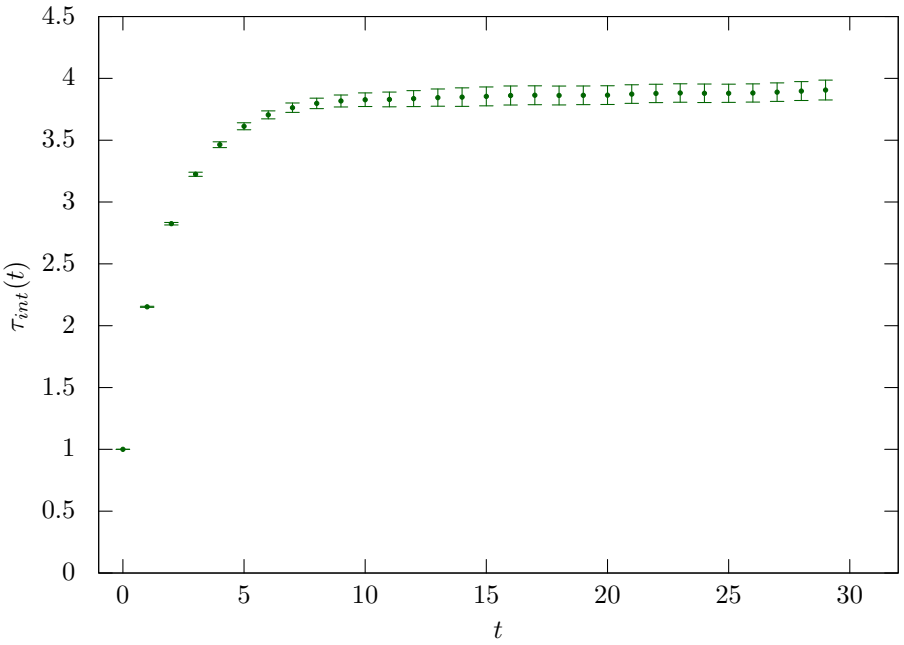


Abbildung 2.3: *Verlauf der integrierten Autokorrelationszeit in Abhängigkeit von t für* $200k$ *normal verteilte Daten.*

Nun soll exemplarisch die Analyse anhand von $N = 200k$ normal verteilter Daten durchgeführt werden.
Die Analyse der Daten mit Dr. Sciarras Programm liefert Werte für $\tau_{int}(t)$ und dessen Fehler bis zu einem gewählten Endwert t_{max}. Diese Werte werden geplottet. Abbildung 2.3 zeigt den Plot. Der Verlauf von $\tau_{int}(t)$ erreicht nach einiger Zeit ein Plateau. Die Kurve saturiert ab etwa $t = 10$. Man wählt nach Augenmaß einen einzelnen Wert und dessen Fehlerbalken innerhalb dieses Plateaus als Schätzer. Für $t = 15$ erhält man $\tau^J_{int}(15) = 3,85(8)$.

2.3 Abschätzung von τ_{int} mit der Gamma-Methode

Die Gamma-Methode ist eine von Wolff in [7] eingeführte Methode zur Abschätzung der Mittelwerte und deren Fehler für Funktionen von Elementarobservablen in Monte Carlo Simulationen. Sie beruht auf der expliziten Abschätzung und Summierung der relevanten Autokorrelationsfunktionen und -zeiten. Die Fehlerschätzer sind zuverlässiger als beispielsweise die der Binning Methode[14].

[14] Die integrierte Autokorrelationszeit kann mit der Binning Methode bestimmt werden. Dazu betrachtet man $\tau^{N_b}_{int}$ als Verhältnis der Varianz autokorrelierter Daten und der naiven Varianz unkorrelierter Daten $\tau_{int} = \sigma^2_{korr}/\sigma^2_{naiv}$. Nähere Informationen liefert [1], S.202ff

Die Notation in [7] weicht von der bisher verwendeten Notation nach [1] ab. Die Autokorrelationsfunktion, die zuvor mit $C(t)$ bezeichnet wurde, entspricht $\Gamma_{\alpha\beta}$ bei Wolff. Die Autokorrelationsfunktion spielt eine zentrale Rolle in der Fehleranalyse und liefert den Namen für die Gamma-Methode von Wolff. Die Autokorrelationsfunktion $\Gamma_{\alpha\beta}$ unter Berücksichtigung mehrerer Replika r ist definiert als

$$\langle (x_\alpha^{i,r} - X_\alpha)(x_\beta^{j,s} - X_\beta) \rangle = \delta_{rs} \Gamma_{\alpha\beta}(j-i) \ , \tag{2.10}$$

mit den Schätzern $x_\alpha^{i,r}$ der Primärobservablen X_α, $i = 1, ...N$, $r = 1, ...R$. Die Autokorrelationsfunktion verknüpft die Abweichung des i. Schätzers für den i. Zustand X_α mit der Abweichung des j. Schätzers vom j. Zustand X_β nach $t = j - i$ Schritten. Betrachtet man Daten innerhalb eines Replikums, so gilt $r = s$ und man erhält die Autokorrelationsfunktion analog zu (2.1). Für unterschiedliche Replika $r \neq s$ ergibt das Kroneckerdelta 0, da Daten verschiedener Replika nicht korreliert sind. Ich beschränke mich hier auf den Fall eines Replikums.
$C_{\alpha\beta}$ ist die Summe über alle Autokorrelationsfunktionen,

$$C_{\alpha\beta} = \sum_{t=-\infty}^{\infty} \Gamma_{\alpha\beta}(t) \ . \tag{2.11}$$

Wollf gibt diese Definition in [7] an. Eine Summe über unendlich große Abstände t kann in Simulationen nicht realisiert werden. Es würde daher mehr Sinn ergeben, von $-(N-1)$ bis $N-1$ zu summieren.

Die Varianz der Primärobservablen[15] X ist

$$\sigma_X^2 \simeq \frac{C_{\alpha\beta}}{N} = \frac{v_X}{N} 2\tau_{int,X} \ . \tag{2.12}$$

Man erhält auch hier wie in der Diskussion zuvor die integrierte Autokorrelationszeit als Verhältnis der naiven Varianz unkorrelierter Daten v_X/N und der Varianz korrelierter Daten σ_X^2. Dies kann unter Berücksichtigung Autokorrelationsfunktion umgeschrieben werden,

$$\tau_{int,X} = \frac{\sigma_X^2}{2\frac{v_X}{N}} = \frac{C_{\alpha\beta}}{2v_X} \ . \tag{2.13}$$

Nun soll darauf eingegangen werden, wie der Wert für die integrierte Autokorrelationszeit abgeschätzt werden kann. Zunächst wird die Autokorrelationsfunktion ermittelt,

$$\hat{\Gamma}_{\alpha\beta}(t) = \frac{1}{N-t} \sum_{i=1}^{N-t} (x_\alpha^i - \hat{x}_\alpha)(x_\beta^{i+t} - \hat{x}_\beta), \quad 0 \leq t \ll N \ . \tag{2.14}$$

[15] Wolff liefert in [7] eine Beschreibung zur Bestimmung der Autokorrelationsfunktion und -zeit für abgeleitete Größen. Hier wird, wie oben beschrieben, nur das Verhalten von Primärobservablen untersucht.

Als Schätzer für die Varianz und $C_{\alpha\beta}(t)$ wählt man

$$\hat{v}_X = \hat{\Gamma}_{\alpha\beta}(0) \quad \text{und} \quad \hat{C}_{\alpha\beta}(W) = \hat{\Gamma}_{\alpha\beta}(0) + 2\sum_{t=1}^{W} \hat{\Gamma}_{\alpha\beta}(t) \ . \tag{2.15}$$

Daraus ergibt sich der Schätzer für τ_{int},

$$\hat{\tau}_{int,X}(W) = \frac{\hat{C}_{\alpha\beta}}{2\hat{v}_X}(W) = \frac{1}{2} + \sum_{t=1}^{W} \frac{\hat{\Gamma}_{\alpha\beta}(t)}{\hat{\Gamma}_{\alpha\beta}(0)} \ . \tag{2.16}$$

Für die Abschätzung von $\hat{\tau}_{int}$ werden nur Autokorrelationsfunktionen innerhalb des Summationsfensters W betrachtet. Um den Wert für die integrierte Autokorrelationszeit zu erhalten, wird der Wert für das Summationsfenster W automatisch gewählt. Dabei wird derjenige Wert für W als optimal definiert, der die "Summe der absoluten Fehler minimiert"[16]. Die integrierte Autokorrelationszeit wird ausschließlich für den optimalen Wert von W bestimmt.

Der Vergleich von (2.16) mit (2.8) ergibt, dass nach Definition der Wert für $\hat{\tau}^{\Gamma}_{int}$ in [7] halb so groß ist wie $\hat{\tau}^{J}_{int}$ in [1]. Um die Ergebnisse beider Analysen vergleichen zu können, werden die Werte aus der Analyse von $\hat{\tau}^{\Gamma}_{int}$ im Folgenden bereits *verdoppelt* angegeben. Nach der Fehlerfortpflanzung verdoppelt man die zugehörigen Werte der Fehler ebenfalls[17].

Wolff stellt eine Matlab-Implementierung der Gamma-Methode zur Verfügung. Ich verwende die bereits erwähnte Python-Implementierung von Dirk Hesse. Der Code beinhaltet einen Algorithmus zur automatischen Wahl des Summationsfensters. Die zugrunde liegende These ist, dass $\tau \sim S\ \tau_{int}$, wobei τ und τ_{int} von der gleichen Größenordnung seien. Der Default liegt bei[18] $S = 1,5$ und wurde in der gesamten vorliegenden Analyse verwendet. Die Ausgabe des Analyseprogramms umfasst den Mittelwert der Probe und einen Wert für die integrierte Autokorrelationszeit mit den jeweiligen Fehlern. Zudem liefert das Programm einen Plot für die integrierte Autokorrelationsfunktion und -zeit. Eine vertikale rote Linie kennzeichnet den Wert W, für den $\hat{\tau}^{\Gamma}_{int}$ ermittelt wurde. Durch den Vergleich der berechneten Werte und die graphische Analyse des Plots kann man überprüfen, ob die Werte ungefähr übereinstimmen. Für $200k$ Daten erhält man Abbildung 2.4. Der Wert für τ^{Γ}_{int} wurde für $t = 12$ ermittelt. Hierfür erhält man den Wert $\tau^{\Gamma}_{int}(12) = 3,85(6)$.

[16] [7], S. 10 (Mitte)
[17] $\hat{\tau}^{\Gamma}_{int} = 2 \cdot \hat{\tau}^{\Gamma,*}_{int} \Longrightarrow \Delta\hat{\tau}^{\Gamma}_{int} = 2 \cdot \Delta\hat{\tau}^{\Gamma,*}_{int}$
[18] Details siehe [7] S. 12f

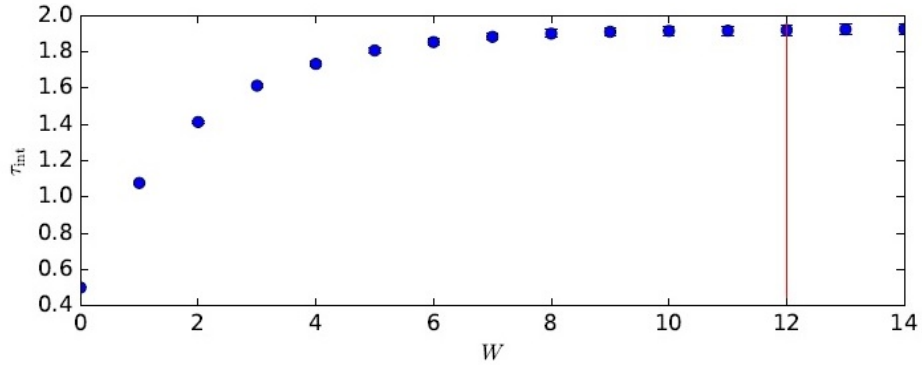


Abbildung 2.4: *Verlauf der integrierten Autokorrelationszeit in Abhängigkeit von t für* $200k$ *Daten.*

2.4 Vergleich der Jackknife- und der Gamma-Methode

Nun sollen die Jackknife-Methode und die Gamma-Methode verglichen werden.
Bei der Jackknife-Methode wurde die integrierte Autokorrelationszeit graphisch ermittelt. Bei der Gamma-Methode wurde eine automatische Wahl des Fensters verwendet. Der optimale Wert für das Fenster soll "die Summe der Absolutwerte der Fehler minimieren"[19]. Nach Augenmaß kann dies nicht abgeschätzt werden. Eine Automatisierung des Summationsfensters hat den Vorteil, dass beispielsweise die Fehlerminimierung berücksichtigt werden kann. Zudem spart man Zeit bei der Auswertung, da der Entscheidungsprozess per Auge entfällt und ein direkter Wert für $\tau_{int}(W)$ geliefert wird.

Der Unteschied in der Effizienz der Methoden liegt in der Bestimmung der integrierten Autokorrelationszeit. Bei der Jackknife Methode werden die Werte der integrierten Autokorrelationszeit für jeden Wert von t bis zu einem Schrankenwert t_{max} berechnet. Der Rechenaufwand ist vor Allem für große Werte für t_{max} sehr intensiv. Bei der Gamma-Methode wählt man zunächst das Summationsfenster W und berechnet den Wert für die integrierte Autokorrelationszeit lediglich für W, $\hat{\tau}_{int}(W)$.

[19] [7], S. 10 (Mitte). Gleichung (42) stellt den Fehler dar, der minimiert werden soll.

3 Verhalten von τ_{int} am Phasenübergang erster Ordnung

Nun werden die vorangegangene Theorie und die Methoden dazu verwendet, um einen Phasenübergang erster Ordnung zu untersuchen. Es wird ein vereinfachtes Modell verwendet, um Systeme in der Umgebung eines Phasenübergangs zu beschreiben.
Die Analyse basiert auf Daten, die durch einen einfachen Metropolis-Algorithmus generiert wurden. Dies ist eine schnelle und effiziente Weise, um viele Daten zu erzeugen. Der Algorithmus eignet sich daher, um beispielsweise zu überprüfen, wie viele Daten in einer realen Simulation notwendig sind, um aussagekräftige Ergebnisse zu liefern. Ferner soll in diesem Abschnitt das Verhalten der integrierten Autokorrelationszeit im Bereich des Phasenübergangs untersucht werden.

3.1 Modell zur Beschreibung eines Phasenübergangs erster Ordnung

Als Phasenübergang wird der Wechsel zwischen zwei Phasen bezeichnet. Der Phasenübergang findet bei einer kritischen Temperatur T_C statt, die von der Größe des Systems abhängt.
Ein Phasenübergang erster Ordnung findet statt, wenn die Zustandssumme nicht analytisch ist. Nach den Sätzen von Yang und Lee existiert kein Phasenübergang für ein System endlicher Größe, da die Zustandssumme analytisch ist. Dennoch kann man den Phasenübergang anhand endlicher Volumina untersuchen. Beispielsweise ist der Siedepunkt von Wasser mit $T_S = 100°C$ für ein unendlich großes System definiert. Für ein endliches System weicht die Siedetemperatur leicht ab. Durch Extrapolation der Messwerte von Experimenten mit Volumina endlicher Größe kann man auf das Verhalten eines unendlichen Volumens schließen. Dies rechtfertigt es, den Phasenübergang an Systemen endlicher Größe zu untersuchen, auch wenn dieser theoretisch nicht existiert. Im Folgenden meint *Phasenübergang* immer einen *Phasenübergang erster Ordnung.*

Ordnungsparameter sind abstrakte Größen, die der Beschreibung von Phasenübergängen dienen, wenn Zustandsgrößen zur Beschreibung des Systems nicht mehr ausreichen. Zur Lokalisierung eines Phasenübergangs muss die kritische Temperatur T_C ermittelt werden. Dazu kann man den Erwartungswert des Ordnungsparameters, die Varianz und die Schiefe betrachten.
Jede Phase weist einen anderen Wert für den Erwartungswert des Ordnungsparameters auf. In einem unendlich großen System tritt am Phasenübergang ein Sprung in diesem Parameter auf. Für ein endliches Volumen ändert sich der Erwartungswert beim Übergang von einer Phase in eine andere kontinuierlich. Daher ist diese Größe in einem endlichen System allein ist nicht ausreichend, um einen Phasenübergang zu lokalisieren. An einem Phasenübergang erster Ordnung hat die Varianz ein Maximum. Dies könnte man ebenfalls verwenden, um einen Phasenübergang zu lokalisieren. Es ist jedoch einfacher, die

Schiefe des Systems zu betrachten. Die Schiefe beschreibt die Asymmetrie der Verteilung und ist temperaturabhängig. Es soll nun kurz erläutert werden wie sich die Verteilung bezüglich der Schiefe ändert, wenn man die Temperatur erhöht. Abbildung 3.1 zeigt, wie sich die Verteilung dabei ändert. Hierbei wird ein System betrachtet, das sich im Bereich eines Phasenübergangs erster Ordnung befindet. Die Mittelwerte der Phasen befinden sich um $\mu_1 = 0$ in der ersten Phase und um $\mu_2 = 10$ in der zweiten Phase.
Es wird zuerst der Fall kleiner Temperaturen betrachtet, $T \ll T_C$. Die Schiefe hat den Wert $\gamma_{\ll} = 0$, die Zustände sind nach der Normalverteilung symmetrisch um den Mittelwert μ_1 verteilt. Erhöht man die Temperatur auf $T \lesssim T_C$, so erhält man eine rechtsschiefe Verteilung um μ_1. Die Schiefe hat einen positiven Wert, $\gamma_{\lesssim} > 0$. Bei der kritischen Temperatur $T = T_C$ nimmt die Schiefe den Wert $\gamma_{=} = 0$ an. Man erhält zwei gleich hohe Peaks in der Verteilung, einen um μ_1 und einen um μ_2. Für $T \gtrsim T_C$ erhält man eine linksschiefe Verteilung um μ_2 mit einer negativen Schiefe, $\gamma_{\gtrsim} < 0$. Befindet man sich weit entfernt von einem Phasenübergang für $T \gg T_C$, so erhält man eine symmetrische Normalverteilung um den Mittelwert μ_2, die Schiefe verschwindet, $\gamma_{\gg} = 0$.
Um die kritische Temperatur zu finden, bestimmt man die Schiefe bei verschiedenen Temperaturen. Die Temperatur, bei der die Schiefe die Nullstelle hat, ist die kritische Temperatur.
Die Art des Phasenübergangs kann durch den Wert der Kurtosis bei $T = T_C$ ermittelt werden. Die Kurtosis hat für jeden Phasenübergang einen anderen Wert. Da hier der Phasenübergang erster Ordnung untersucht wird, soll die Kurtosis an dieser Stelle nur erwähnt werden[20].

Im Modell wird ein System, das sich weit entfernt von einem Phasenübergang erster Ordnung befindet, durch eine Normalverteilung beschrieben. Das System befindet sich dabei weit entfernt von der kritischen Temperatur des Systems, $T \ll T_C$ oder $T \gg T_C$.
Ein System, das sich bei einer kritischen Temperatur $T \approx T_c$ an einem Phasenübergang erster Ordnung befindet, kann durch die Summe zweier Normalverteilungen beschrieben werden, deren Peaks gleich hoch sind. Der Erwartungswert der ersten Verteilung ist μ_1, der der zweiten Verteilung ist μ_2, die Standardabweichung σ beider Verteilungen ist identisch. Das Verhalten bei einer Erhöhung des Abstandes der Peaks $d = |\mu_2 - \mu_1|$ ist übertragbar auf das Verhalten bei einer Erhöhung des Systemvolumens[21] .

Zur Untersuchung des Phasenübergangs nach dem beschriebenen Modell habe ich ein Programm geschrieben, das korrelierte Daten nach dem in Unterabschnitt 1.4 beschriebenen einfachen Metropolis-Algorithmus generiert. Der Code befindet sich in Anhang A.1. Für den Fall $T \ll T_C$ habe ich als Wahrscheinlichkeitsverteilung die Standardnormalvertei-

[20] Für Details siehe [5], S.101ff

[21] Die Variation des Abstandes d ist ein vereinfachtes Vorgehen. Für eine realistischere Beschreibung des Systems erhöht man das Volumen bei fixierten μ_1 und μ_2. Dabei verringern sich die Standardabweichungen der Verteilungen. Man betrachtet nun nicht mehr den Abstand zwischen den Peaks d, sondern den Abstand $d^* = |(\mu_1 + \sigma_1) - (\mu_2 - \sigma_2)|$. Dieser nimmt dadurch zu, dass die Standardabweichung mit wachsendem Volumen abnimmt. Für ein unendliches Volumen entspricht die Verteilung einer Dirac-Distribution.

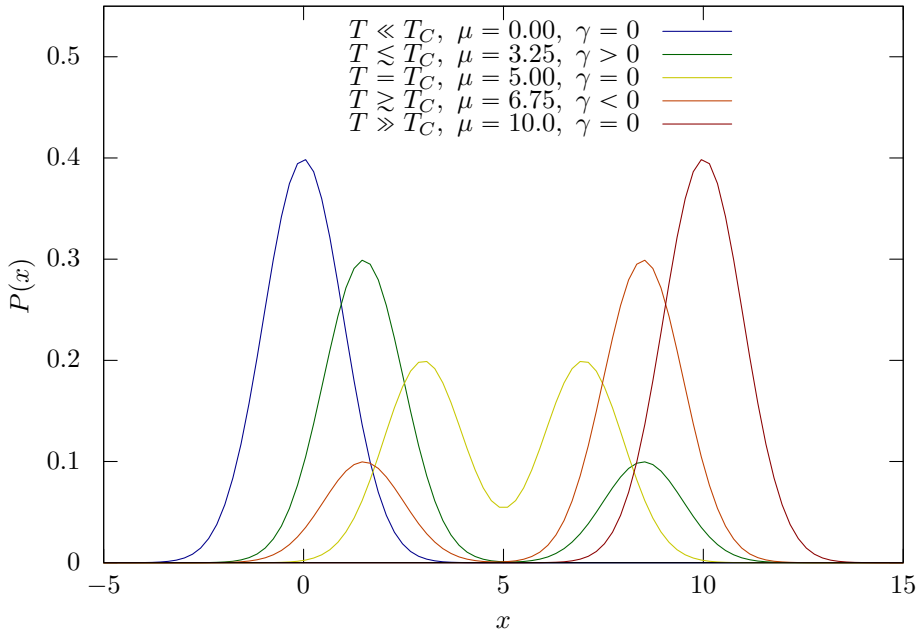


Abbildung 3.1: *Modell zur Beschreibung eines Phasenübergangs erster Ordnung.*

lung verwendet. Wie zuvor beschrieben lässt sich ein System an einem Phasenübergang erster Ordnung durch die Summe zweier Normalverteilungen beschreiben, deren Peaks die gleiche Höhe haben,

$$P(x) = \frac{1}{\sqrt{8\pi}} \left(e^{-\frac{1}{2}(x-\mu_1)^2} + e^{-\frac{1}{2}(x-\mu_2)^2} \right) \ . \tag{3.1}$$

Der Erwartungswert der ersten Verteilung ist $\mu_1 = 0$ und der Erwartungswert der zweiten Verteilung variiert, $\mu_2 = 3, 5, 10$. Der Wert der Varianz ist für beide Verteilungen identisch und fixiert, $\sigma^2 = 1$. Der Code, den ich zur Generierung der Daten geschrieben habe, befindet sich im Anhang. Hierbei kann man entweder normal verteilte Daten oder Daten die nach Summe zweier gleich hoher Normalverteilungen generieren. Die Erwartungswerte und Varianzen sind frei wählbar. Die Ausgabe liefert eine Textdatei mit den generierten Daten. Die Verwendung dieses Algorithmus ermöglicht eine schnelle Generierung von Daten, um ein reales System nachzubilden.

3.2 τ_{int} weit entfernt vom Phasenübergang erster Ordnung

Nun wird ein System betrachtet, das sich weit entfernt von einem Phasenübergang erster Ordnung befindet, bei einer Temperatur klein gegenüber der kritischen Temperatur, $T \ll T_C$. Zur Untersuchung dieses Systems werden nach dem Modell normal verteilte Daten verwendet mit $\mu = 0$.

Zunächst soll diskutiert werden, wie viele Daten für eine aussagekräftige Analyse notwendig sind. Hierfür wird das Histogramm der Daten an eine Gaußkurve gefittet. Die Fitparameter für μ und σ werden mit den theoretischen Werten verglichen, die zur Generierung der Daten verwendet wurden. Dies geschieht mithilfe des Gauß'schen Differenztests. Die Werte für $Q^{2\sigma} \leq Q \leq 1 - Q^{2\sigma}$ sind in der Tabelle grün markiert. Die grün markierten Werte weisen darauf hin, dass die Fit-Werte der Parameter mit hoher Wahrscheinlichkeit mit den theoretischen Werten übereinstimmen. Die Werte für $Q^{3\sigma} \leq Q < Q^{2\sigma}$ und $Q > 1 - Q^{2\sigma}$ sind gelb markiert. Sie liegen noch innerhalb des 3σ- Bereichs um den theoretischen Wert. Die zugehörigen Fitparameter werden auch als übereinstimmend mit den theoretischen Werten betrachtet. Werte für $Q < Q^{3\sigma}$ sind rot markiert. Die zugehörigen Fitparameter stimmen mit sehr hoher Wahrscheinlichkeit nicht mit den theoretischen Werten überein.
Eine weitere Überprüfung der Daten basiert auf der Methode des reduzierten χ^2. In der Auswertung werden Werte für $0,1 \leq \chi^2 < 5$ als ausreichende Übereinstimmung akzeptiert.
Die Anzahl der verwendeten Bins hat Relevanz für die Qualität des Fits. In Zusammenarbeit mit Dr. Sciarra ist ein Programm entstanden, das für eine gewählte Verteilung die Fitparameter Q und χ^2 ermittelt. Der Code befindet sich in Anhang A.3. Ein weiteres Programm wurde von Dr. Sciarra bereitgestellt, bei dem verschieden große Datensets und unterschiedliche Bingrößen eingelesen werden, für die die Analyse der Fitparameter durchgeführt werden soll. Dies dient dem systematischen Vergleich der Fitparameter bei unterschiedlicher Bingröße und Datengröße. Die Analyse einer geeigneten Bin- und Datengröße wurde nicht so systematisch wie möglich durchgeführt, sie liefert dennoch eine ausreichende Abschätzung. Die Fitparameter werden für verschiedene Datengrößen N und Bingrößen Bins_{Hist} verglichen. Tabelle 3.1 zeigt das Ergebnis der Analyse. Man erkennt, dass eine Anzahl von $50k$ Daten nicht ausreichend ist, um eine gute Übereinstimmung zwischen theoretischen Werten und Fitparametern zu erzielen. Die Werte für Q liegen im roten Bereich. Die Fluktuationen in der Bingröße bei Erhöhung der Datengröße N resultieren daraus, dass leere Bins im Histogramm nicht berücksichtigt werden. Bei $50k$ sind beispielsweise 2 Bins leer. Die Q-Werte für das Datenset mit $200k$ Daten liegen mindestens im gelben Bereich und auch das reduzierte χ^2 liefert einen guten Wert. Die Analyse mit diesem Datenset wird ein ausreichende genaues Ergebnis liefern. Mit steigendem Umfang der Daten auf $500k$ erhöht sich die Übereinstimmung von theoretischen Werten und Fitparametern. Bei realen Simulationen ist der Aufwand zur Produktion umfangreicher Datensets jedoch aufwendig und langwierig. Daher sollte ermittelt werden, wie groß ein Datenset mindestens sein muss, um aussagekräftige Ergebnisse zu liefern. Im Folgenden, und auch im nächsten Abschnitt, soll daher die Analyse auf der Verwendung des kleinstmöglichen Datensets basieren. Die Bewertung der geeigneten Datengröße beruft sich auch im Folgenden auf die Markierungen rot, gelb und grün. Es sei angemerkt, dass die eben genannten Werte bezüglich der benötigten Datengröße keine Absolutwerte sind, sondern ungefähre Größenordnungen darstellen.

Für die hier vorliegende Normalverteilung wurde die Auswertung anhand von $200k$ Daten

N	Bins_{Hist}	$Q(\mu)$	$Q(\sigma)$	χ^2
$50k$	48	$0,003$	3e-5	$3,8$
$200k$	49	$0,040$	$0,046$	$2,8$
$500k$	50	$0,23$	$0,81$	$2,7$

Tabelle 3.1: *Auswertung der benötigten Statistik für die Normalverteilung unter Verwendung des Gauß'schen Differenztest und des reduzierten χ^2-Tests.*

bereits im 2. Kapitel durchgeführt. Für die Jackknife Methode wurde der Wert

$$\tau_{int}^{J}(15) = 3,85(8)$$

ermittelt, die Gamma-Methode lieferte den Wert

$$\tau_{int}^{\Gamma}(12) = 3,85(6)\ .$$

3.3 τ_{int} am Phasenübergang erster Ordnung

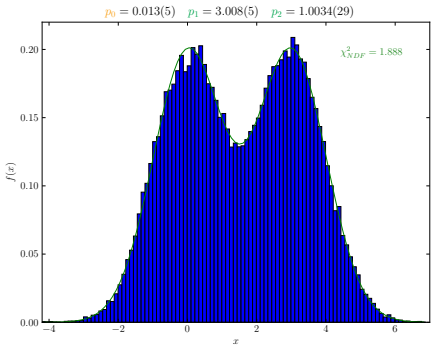
Nun soll ein System untersucht werden, das sich an einem Phasenübergang erster Ordnung befindet. Nach dem Modell entspricht dies der Beobachtung von Daten, deren Verteilung der Summe zweier Gaußverteilungen folgt. Es soll wie im vorherigen Abschnitt untersucht werden, welcher Datenumfang benötigt wird und welche Ergebnisse die Jackknife- und die Gamma-Methode für die integrierte Autokorrelationszeit liefern. Dies soll vor Allem in Hinblick auf die Variation des Abstandes d geschehen.

Zunächst soll auch hier diskutiert werden, wie viele Daten notwendig sind, um eine aussagekräftige Analyse zu erhalten. Wie bei der Normalverteilung im vorhergehenden Abschnitt werden die Fitparameter nun für μ_1, μ_2 und σ mit den theoretischen Werten verglichen.

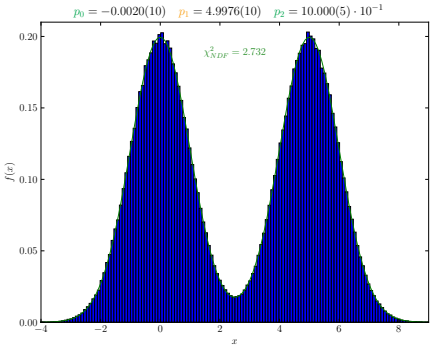
Tabelle 3.2 kann man entnehmen, dass für $d = 3$ ungefähr $100k$ Daten notwendig sind, um eine ausreichend genaue Analyse durchzuführen. Für $d = 5$ ist eine Datengröße von $\approx 2M$ ausreichend. Für $d = 10$ liefern $5M$ ausreichend gute Werte für die Fitparameter Q zur Abschätzung der Lage der Maxima der Verteilung. Hier ist der Wert für χ^2 sehr hoch. Für $10M$ Daten liegt dieser Wert sogar im dreistelligen Bereich. Für $20M$ Daten ist der Wert für χ^2 gut, jedoch weicht $Q(\mu_2)$ ab.
Nun soll geklärt werden, warum man für die korrekte Simulation eines Systems mit $d = 10$ im Vergleich zu $d = 5$ und $d = 3$ verhältnismäßig viele Daten benötigt.
Abbildung 3.2 zeigt die Histogramme für $d = 3$, $d = 5$ und $d = 10$. Betrachtet man das Histogramm für $d = 3$, so fällt auf, dass die Wahrscheinlichkeit, einen Zustand in der Mitte zwischen den Maxima zu finden, mit mehr als 10% relativ wahrscheinlich ist. Für $d = 5$ liegt die Wahrscheinlichkeit nur noch bei etwa 2%. Die Wahrscheinlichkeit, bei einem Abstand $d = 10$ einen Zustand in der Mitte anzutreffen ist sehr gering. Verwendet

d	N	Bins_{Hist}	$Q(\mu_1)$	$Q(\mu_2)$	$Q(\sigma)$	χ^2
3	$10k$	91	$0,08$	$0,026$	9e-5	$3,3$
	$100k$	99	$0,017$	$0,16$	$0,25$	$1,9$
	$700k$	95	$0,19$	$0,77$	$0,47$	$2,3$
5	$500k$	148	$0,80$	4e-5	$0,007$	$3,4$
	$2M$	150	$0,06$	$0,03$	$0,98$	$2,8$
	$10M$	149	$0,10$	$0,65$	$0,20$	$3,8$
10	$500k$	233	$0,60$	$0,59$	9e-5	66
	$5M$	239	$0,997$	$0,20$	$0,50$	47
	$10M$	244	$0,90$	$0,013$	$0,64$	550
	$20M$	247	$0,47$	$0,005$	$0,76$	$3,0$

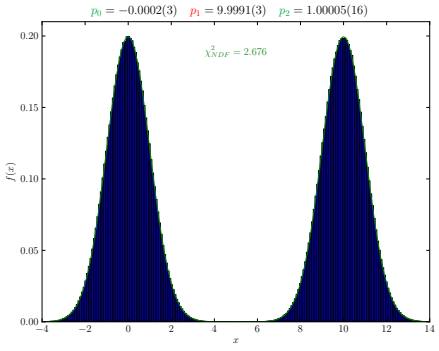
Tabelle 3.2: *Auswertung der benötigten Statistik für $d = 3$, $d = 5$ und $d = 10$ unter Verwendung des Gauß'schen Differenztest und des reduzierten χ^2.*



(a) Histogramm für $d = 3$ mit $100k$ Daten



(b) Histogramm für $d = 5$ mit $2M$ Daten



(c) Histogramm für $d = 5$ mit $20M$ Daten

Abbildung 3.2: *Histogramme für $d = 3$, $d = 5$ und $d = 10$.*

man ausreichend viele Daten, so wird man dennoch Zustände in diesem Bereich antreffen.

Zum Verständnis dieses Phänomens betrachtet man die Funktionsweise des verwendeten Metropolis-Algorithmus am Beispiel $d = 10$. Angenommen, man startet bei $x = 0 = \mu_1$. Man wechselt in einen Zustand in der Umgebung von x innerhalb eines Radius a. Die Wahrscheinlichkeit ist am größten, Zustände um μ_1 zu wählen. Wir betrachten den Fall, dass man sich in positiver x-Richtung von μ_1 entfernt. Die Wahrscheinlichkeit, einen Zustand anzutreffen, sinkt, je mehr man sich vom Erwartungswert μ_1 entfernt. Zwischen $x \approx 4$ und $x \approx 6$ befindet sich eine Lücke, in der Zustände nur mit geringer Wahrscheinlichkeit auftreten. Da diese Wahrscheinlichkeit nicht Null ist, ist es, ähnlich wie beim Tunneln, möglich, die Lücke zu überwinden. Die Wahrscheinlichkeit, Zustände für $x > 6$ anzutreffen nimmt wieder zu und hat ein Maximum bei $x = 10 = \mu_2$. Das System ist von einer Phase im Bereich um μ_1 in die andere Phase um μ_2 übergegangen.

Betrachtet man den zeitlichen Verlauf einer großen Anzahl an Daten, so kann man beobachten, dass das System häufig zwischen den Phasen wechselt. In Abbildung 3.3 ist der zeitliche Verlauf der Zustände für $50k$, $100k$ und $500k$ Daten dargestellt. Man könnte die y-Achse in drei Teile teilen. Der unterste Teil beinhaltet den grünen Balken. Auf der Höhe der Mitte des grünen Balkens könnte man auf der y-Achse die Null als Mittelwert der Verteilung eintragen. Die Zustände innerhalb des grünen Balkens fluktuieren um den Mittelwert $\mu_1 = 0$. Analog Könnte man auf Höhe der Mitte des unteren orangefarbenen Balkens den Wert 0 und auf Höhe der Mitte des oberen orangefarbenen Balkens eine 10 eintragen, genauso bei den roten Balken. Die Zustände sind in einem Bereich um die Erwartungswerte verteilt.

Betrachtet man den zeitlichen Verlauf von $50k$ Daten, so fällt auf, dass alle Zustände im selben Bereich fluktuieren. Ein gleiches Ergebnis erhielte man, wenn man den zeitlichen Verlauf einer einzelnen Normalverteilung darstellen würde. Das System befindet sich in einer Phase und kann die Lücke nicht überwinden. Es exisitert keine zweite Phase. Der Datenumfang ist zu gering, um das System korrekt zu beschreiben. Für $100k$ Daten befinden sich die ersten knapp $100k$ Zustände im Bereich um $x = 0$. Die Lücke wird überwunden und die Zustände sind im Bereich um $x = 10$ anzutreffen. Dies entspricht dem Wechsel zwischen zwei Phasen. Bei $500k$ Daten findet der Phasenübergang häufiger statt, im Durchschnitt etwa alle $50k$ Schritte.

Nun soll anhand des Abstandes $d = 10$ zwischen den Maxima der doppelten Gaußverteilung untersucht werden, wie sich τ_{int} verhält, wenn man die Datenmenge erhöht. Abbildung 3.4 zeigt das Verhalten. Die Ergebniss der Analyse sind in Tabelle 3.3 dargestellt. Man sieht, dass die integrierte Autokorrelationszeit mit zunehmender Datengröße stark ansteigt. Dies resultiert aus den eben betrachteten Phasenübergängen. Für $10k$ und $50k$ befinden sich alle Zustände im Bereich um den Mittelwert $\mu_1 = 0$. Das System bleibt in einer Phase. Betrachtet man einen kleinen Abstand t zwischen zwei Zuständen, so können die Werte zweier Zustände stark voneinander abweichen. Dies erklärt den steilen

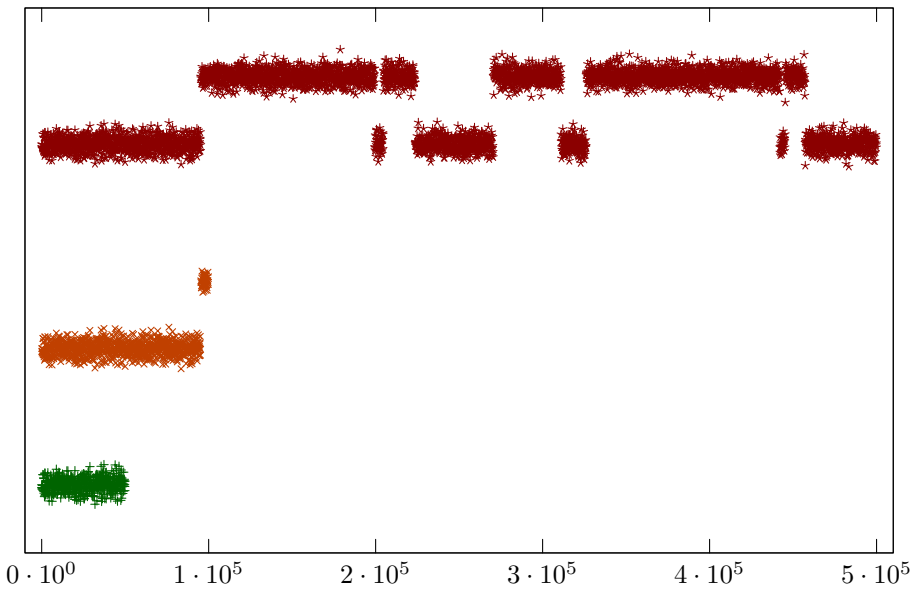


Abbildung 3.3: *Zeitlicher Verlauf der Konfigurationen für* $50k$ *(grün),* $100k$ *(orange) und* $500k$ *(rot) Daten.*

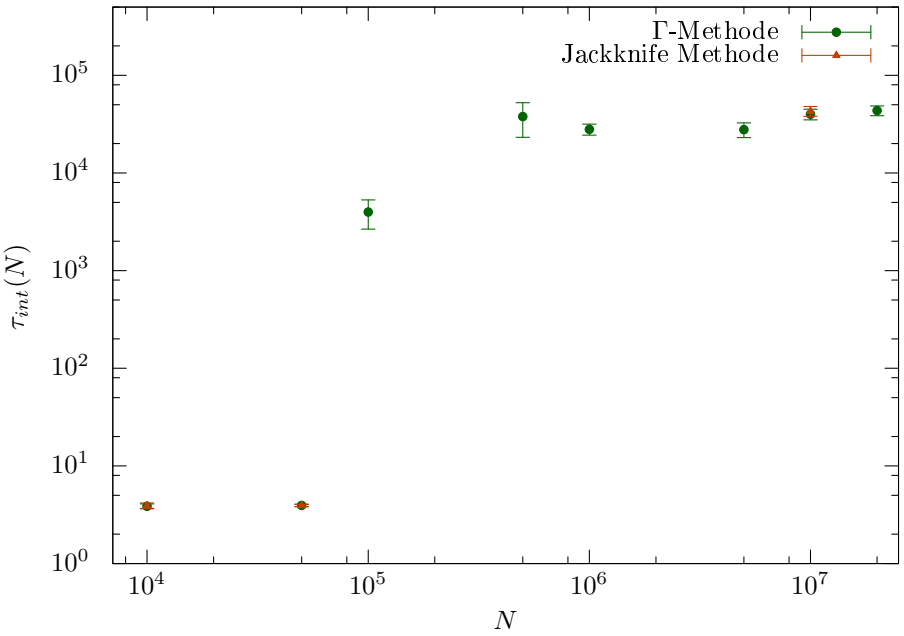


Abbildung 3.4: *Integrierte Autokorrelationszeit in Abhängigkeit der Anzahl der Daten pro Datenset für* $d = 10$.

d	N	t^J	τ_{int}^J	τ_{int}^Γ
3	$100k$	30	$6,84(25)$	$6,67(18)$
5	$2M$	100	$3,23(4) \cdot 10^1$	$3,24(5) \cdot 10^1$
10	$10k$	15	$3,93(25)$	$3,86(24)$
	$50k$	12	$3,93(10)$	$3,94(12)$
	$100k$	–	–	$4,0(13) \cdot 10^3$
	$500k$	–	–	$3,8(15) \cdot 10^4$
	$1M$	–	–	$2,9(9) \cdot 10^4$
	$5M$	–	–	$2,8(5) \cdot 10^4$
	$10M$	$100k$	$4,3(5) \cdot 10^4$	$4,1(6) \cdot 10^4$
	$20M$	–	–	$4,4(6) \cdot 10^4$

Tabelle 3.3: *Auswertung der integrierten Autokorrelationszeit nach der Jackknife Methode und der* Γ*-Methode für* $d = 3$*,* $d = 5$ *und* $d = 10$*.*

Anstieg im Verlauf der integrierten Autokorrelationszeit mit t. Betrachtet man Zustände in einem größeren Abstand t, so weichen diese nicht mehr so stark voneinander ab. Die starken Fluktuationen, die für kleine t auftreten, werden minimiert. Zustände, die sich weit entfernt voneinander befinden, sind nur schwach korreliert. τ_{int} erreicht relativ schnell ein Plateau bei einem relativ kleinen Wert, hier bei $\tau_{int} \approx 4$. Die integrierte Autokorrelationszeit ist in der gleichen Größenordnung wie in Unterabschnitt 3.2.
Für $100k$ Daten findet ein Phasenwechsel statt. Nun kann es bei der Bestimmung der Autokorrelationsfunktion vorkommen, dass zwei Zustände betrachtet werden, von denen sich einer in der ersten und der andere in der zweiten Phase befindet. Die Autokorrelation zwischen zwei Zuständen verschiedener Phase ist hoch. Diese Beiträge erhöhen den Wert der Summe der Autokorrelationszeit. Dies erklärt den rapiden Anstieg der Autokorrelationszeit im Vergleich zu einem System, das sich in nur einer Phase befindet.
Für $500k$ Daten finden mehrere Phasenwechsel statt. In diesem Fall werden in der Summe für τ_{int} mehr Summanden mit einem hohen Autokorrelation berücksichtigt als bei einem Phasenwechsel für $100k$ Daten. Daher findet ein weiterer Sprung in der Größenordnung der integrierten Autokorrelationszeit von 10^3 auf 10^4 statt. τ_{int} ist ein Maß für Zeitskalierung des Systems. Im Durchschnitt findet alle $\tau_{int} \approx 50k$ Zeitschritte ein Phasenwechsel statt.

Bei der Auswertung der integrierten Autokorrelationszeit für $d = 10$ stößt die Jackknife-Methode an ihre Grenze. Die Analyse der Daten benötigt sehr viel Zeit. Für $10M$ Daten benötigte das Programm 20 Stunden für die Analyse. Eine grobe Abschätzung der integrierten Autokorrelationszeit kann getroffen werden. Der Wert stimmt gut mit dem Wert von τ_{int}^Γ überein. Daher wird auf weitere zeitintensive Ausführungen des Programms verzichtet. Die Größenordnung der integrierten Autokorrelationszeit wird mit der Gamma-Methode adäquat beschrieben.

Fazit

Abschließend soll eine kurze Zusammenfassung und eine Bewertung der Ergebnisse gegeben werden.
Die Autokorrelationszeit wurde zunächst theoretisch hergeleitet. Die Jackknife- und die Gamma-Methode wurden zur Bestimmung der integrierten Autokorrelatinoszeit verwendet. Als Modell eines Phasenübergangs erster Ordnung wurden Daten mit einem Metropolis-Algorithmus generiert, die entweder normal verteilt sind oder nach der Summe zweier Normalverteilungen mit gleichem Gewicht. Die Vorteile des Metropolis-Algorithmus zur Untersuchung des Phasenübergangs sind die schnelle Generierung von Daten und dass keine Äquilibrierung des Systems notwendig ist. Unter Zuhilfenahme des Gauß'schen Differenztests und des reduzierten χ^2 wurde der benötigte Datenumfang abgeschätzt. Diese Notwendigkeit zeigt sich in realen Simulationen. Es ist aufwändig, große Datenmengen zu generieren. Daher hat es Sinn, zunächst mit einfacheren Mitteln zu prüfen, wie viele Daten notwendig sind, um eine korrekte Analyse zu ermöglichen.

Der klare Vorteil der Gamma-Methode gegenüber der Jackknife-Methode ist die Effizienz. Bei der Gamma-Methode wird die integrierte Autokorrelationszeit nur für ein gewähltes W bestimmt wird. Bei der Jackknife-Methode wird die integrierte Autokorrelationszeit für alle t innerhalb des Summationsfensters ermittelt. Dieser hohe Rechenaufwand führt zu einer langen Laufzeit des Programms. Für den Fall $d = 10$, bei dem das Plateau bei einer ausreichenden Datengöße erst nach einigen Tausend bis Zehntausend Zeitschritten erreicht wird, ist der Zeitaufwand bei der Berechnung mit der Gamma-Methode deutlich geringer.

Nun sollen die Ergebnisse zusammengefasst und bewertet werden. Die integrierte Autokorrelationszeit ist klein, wenn man ein System weit entfernt von einem Phasenübergang erster Ordnung untersucht. Das System verharrt in einer Phase. Vergrößert man den Abstand d, so steigt der Wert für die integrierte Autokorrelationszeit, die generierten Daten weisen eine höhere Korrelation auf. Je größer der Abstand d ist, umso mehr Daten werden benötigt, um das System korrekt zu beschreiben. Dabei entspricht das Verhalten bei Vergrößerung des Abstandes d dem Verhalten bei einer Vergrößerung des Systemvolumens. Verwendet man zur Analyse ein Datenset von zu geringer Größe, wird der Wert für die integrierte Autokorrelationszeit unterschätzt. Das System befindet sich in nur einer Phase, es liegt keine Kenntnis über die zweite Phase vor. Die Anzahl der Daten ist nicht ausreichend, um das System korrekt zu beschreiben. Erhöht man die Anzahl der Daten, finden Phasenwechsel statt. Der zeitliche Verlauf der generierten Daten sollte einige Phasenwechsel aufweisen, damit das System adäquat beschrieben wird. Häufige Phasenwechsel führen in diesem Fall zu einer höheren Autokorrelationszeit. Am Phasenübergang erster Ordnung steigt die integrierte Autokorrelationszeit rasant an. Dies wird in der Realität nicht nur bei der kritischen Temperatur stattfinden, sondern auch in einem Bereich oberhalb und unterhalb der kritischen Temperatur.

A Anhang

A.1 Python-Implementierung des Metropolis-Algorithmus

```
import random
import argparse
from math import exp, expm1, sqrt, pi
import math
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import numpy as np
from scipy.stats import norm
import datetime
import time
import os
import errno


def CreateCommandLineOptionParser():
    parser = argparse.ArgumentParser(prog='tool', formatter_class=lambda prog: argparse.
        HelpFormatter(prog, max_help_position=40, width=100, indent_increment=5))
    parser.add_argument("--filename", help="Name of the file that will contain the data. For
        example 'data' ", type=str, required=True)
    parser.add_argument("--var_a", help="Value for a in x' = x + 2ax^r - a. If you choose a
        small value for a, the acceptance rate will be high but also will the autocorrelation
        be high. If the value for a is too high, the acceptance rate will be small. For a=3.0
        the acceptance probability is approxamately p=0.5. (default: a=%(default)s)", type=
        float, default=3.0)
    parser.add_argument("--ndat", help="Number of the produced data.", type=int)
    parser.add_argument("--typeOfProbDistribution", help="Normal Gaussian distribution: '1',
        distribution of the sum of two Gaussian distributions: '2'. (default: %(default)s)",
        type=int, choices=[1,2], default="1")
    parser.add_argument("--mu", help="Expectation value of the gaussian distribution. Insert a
        value for mu, if you chose the Normal Gaussian distribution. (default: mu=%(default)s)
        ", type=float, default=0.0)
    parser.add_argument("--sigma", help="Standard deviation of the gaussian distribution.
        Insert a value for sigma, if you chose the Normal Gaussian distribution. (default:
        sigma=%(default)s)", type=float, default=1.0)
    parser.add_argument("--mu1", help="Expectation value of the first gaussian. Insert a value
        for mu1, if you chose the Sum of two Gaussian distribution. (default: mu1=%(default)s)
        ", type=float, default=0.0)
    parser.add_argument("--sigma1", help="The standard deviation of the first gauss.Insert a
        value for sigma1, if you chose the Sum of two Gaussian distribution. (default: sigma1
        =%(default)s)", type=float, default=1.0)
    parser.add_argument("--mu2", help="Expectation value of the second gaussian. Insert a value
         for mu2, if you chose the Sum of two Gaussian distribution. (default: mu2=%(default)s
        )", type=float, default=0.0)
    parser.add_argument("--sigma2", help="The standard deviation of the second gauss. Insert a
        value for sigma2, if you chose the Sum of two Gaussian distribution. (default: sigma2
        =%(default)s)", type=float, default=1.0)
    parser.add_argument("--activateHistogram", help="Add this command wihout any option to the
        command line to activate the histogram of the data. (default: False)", action="
        store_true", default=False)
    return parser.parse_args()


def GetCommandLineOptionValues(args):
    return args.filename, args.typeOfProbDistribution, args.var_a, args.ndat, args.mu, args.
        sigma, args.mu1, args.sigma1, args.mu2, args.sigma2, args.activateHistogram


def gaussian(xVariable, mu, sigma):
    return ( exp( -( (xVariable-mu)**2) / (2*sigma**2) ))


def sumOfTwoGaussians(xVariable, mu1, sigma1, mu2, sigma2):
    return ( gaussian(xVariable, mu1, sigma1) + gaussian(xVariable, mu2, sigma2) )


def probabilityDistribution(xVariable):
    if(typeOfProbDistribution=="1"):
        return ( gaussian( xVariable, mu, sigma)  )
    else:
        return ( sumOfTwoGaussians(xVariable,mu1,sigma1,mu2,sigma2))


def CreateDirectoryAndGetOutputFilenameAsGlobalPath(typeOfProbDistribution, filename, mu, sigma
    , mu1, sigma1, mu2, sigma2):
    now = datetime.datetime.now()
    nowDateHourMinute = now.strftime("%y-%m-%d-%H-%M")
```

```
    if(typeOfProbDistribution==1):
        dirname = "1Gauss_"+str(ndat)+"data"+"_mu-"+str(mu)+"_sigma-"+str(sigma)+"_"+str(
            nowDateHourMinute)
        filename = filename+"_"+str(ndat)+"_"+"1Gauss"+"_mu-"+str(mu)+"_sigma-"+str(sigma)+".
            txt"
    else:
        dirname = "2Gauss_"+str(ndat)+"data"+"_mu1-"+str(mu1)+"_s1-"+str(sigma1)+"_mu2-"+str(
            mu2)+"_s2-"+str(sigma2)+"_"+str(nowDateHourMinute)
        filename = filename+"_"+str(ndat)+"_"+"2Gauss"+"_mu1-"+str(mu1)+"_s1-"+str(sigma1)+"
            _mu2-"+str(mu2)+"_s2-"+str(sigma2)+".txt"
    try:
        os.mkdir(dirname)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise
    dirPath = os.getcwd()
    completeName = os.path.join(dirPath, dirname, filename)
    return completeName


def PrintHeaderToFile(filename):
    form1="{: <24}"
    text1 = "#x"
    text2="x**2"
    text3="x**3"
    saveFile = open(filename,"w")
    saveFile.write(form1.format(text1) + form1.format(text2) + form1.format(text3))
    saveFile.close()


def ProduceDataUsingMetropolisAndSaveThemToFile(filename):
    form1="{: <24}"
    appendFile = open(filename,"a")
    random.seed(1234)
    xOld = 0.0

    for i in range(0,ndat):                 # range(x,y) --> [x,y-1]
        randomNumber=random.random()        # generates random number in [0,1)
        xNew=xOld+2.0*a*(randomNumber-0.5)
        acceptanceProbability = probabilityDistribution(xNew)/probabilityDistribution(xOld)
        if( acceptanceProbability > 1 ):
            xOld=xNew
            accepted=1
        else:
            randomNumber=random.random()
            if ( acceptanceProbability >= randomNumber ):
                xOld=xNew
                accepted=1
            else:
                accepted=0

        # xAsString more general so one can choose what powers of x shall be produced. Input
            similar to first and last column in the calculate
        # script.
        xAsString = str(xOld)
        xSquaredAsString = str(xOld**2)
        xCubedAsString = str(xOld**3)
        appendFile.write("\n")
        appendFile.write(form1.format(xAsString))
        appendFile.write(form1.format(xSquaredAsString))
        appendFile.write(form1.format(xCubedAsString))

    appendFile.close()


def ProduceHistogramPlot(filename):

    def fctPlotHistogram(data, title, titleXAxis, titleYAxis, powerOfX):
        plt.hist(data, bins='auto', normed=True)
        plt.title(title)
        plt.xlabel(titleXAxis)
        plt.ylabel(titleYAxis)
        plt.savefig(filename+"_histogram_"+powerOfX+".pdf")

    def oneNormGauss(xRange, mu, sigma):
        gauss = (1.0/(np.sqrt(2.0*math.pi) * sigma)) * np.exp(-((xRange-mu)/sigma)**2/2.0)
        return gauss

    def plotOneNormGauss(mu, sigma):
        x1 = np.linspace(mu-3*sigma**2, mu+3*sigma**2, 100)
        Gauss = oneNormGauss(x1, mu, sigma)
        plt.plot(x1, Gauss)
```

```
131     def twoNormGauss(xRange, mu1, sigma1, mu2, sigma2):
            normalisationFactor = 1.0/2
            gauss1 = oneNormGauss(xRange, mu1, sigma1)
            gauss2 = oneNormGauss(xRange, mu2, sigma2)
            gaussSum = gauss1 + gauss2
136         normedGauss = normalisationFactor * gaussSum
            return normedGauss

        def plotTwoNormGauss(mu1, sigma1, mu2, sigma2):
            x2 = np.linspace(mu1-3*sigma**2, mu2+3*sigma**2, 100)
141         twoGauss = twoNormGauss(x2, mu1, sigma1, mu2, sigma2)
            return plt.plot(x2, twoGauss)

        for column in range(0, 3):
            plt.clf()
146         correlatedData = np.loadtxt(filename, usecols=[column])
            if(column==0):
                if(typeOfProbDistribution==1):
                    envelopeGauss = plotOneNormGauss(mu, sigma)
                else:
151                 envelopeGauss = plotTwoNormGauss(mu1, sigma1, mu2, sigma2)
            fctPlotHistogram(correlatedData, "$Histogram$", "$x^"+str(column+1)+"$", "$Q(x^"+str(
                column+1)+")$", "xTo" + str(column+1))


#=============================================================================#
156
    if __name__ == "__main__":
        arguments = CreateCommandLineOptionParser()
        filename, typeOfProbDistribution, a, ndat, mu, sigma, mu1, sigma1, mu2, sigma2,
            activateHistogram = GetCommandLineOptionValues(arguments)
        completeName = CreateDirectoryAndGetOutputFilenameAsGlobalPath(typeOfProbDistribution,
            filename, mu, sigma, mu1, sigma1, mu2, sigma2)
161     PrintHeaderToFile(completeName)
        ProduceDataUsingMetropolisAndSaveThemToFile(completeName)
        if( bool(activateHistogram) == True ):
            ProduceHistogramPlot(completeName)
        else:
166         exit()
```

A.2 Python-Implementierung zur Berechnung von τ_{int}

```
    # This program will contain subprocesses to calculate the autocorrelation time as in Berg's
        book (mentioned up from p. 199)
    # and like Ulli Wolff in his paper.
3

    import argparse
    import os.path
    import os
8   import numpy as np
    import matplotlib.pyplot as plt
    from matplotlib import rc

    def CreateCommandLineOptionParser():
13      parser = argparse.ArgumentParser(prog='tool', formatter_class=lambda prog: argparse.
            HelpFormatter(prog, max_help_position=40, width=100, indent_increment=5))
        parser.add_argument("--systempath", help="Path where your file is located.", type=str,
            required=True)
        parser.add_argument("--filename", help="Please insert the name of the file you want to
            analyze.", type=str, required=True)
        parser.add_argument("--method", help="For the method of Berg choose '1', for the Gamma-
            method of Wolff '2' and to apply both methods choose '3'. The file of tau_int
            calculated from Jackknife binning will have the name of the old file with the suffix '
            _auto'; the file with tau_int and its error calculated with the Gamma method will have
             the suffix '_tauintWolff'. (default:1/ Berg Method)", type=int, choices=[1,2,3],
            default=1)
        parser.add_argument("--firstColumn", help="All columns of your file from firstColumn to
            lastColumn will be used for Bergs method. Value shuold be >=0. (if data created with
            generalMetropoisAlgorithm.py: 0: data, 1: squared data, 2: cubed data.) (default:0)",
            type=int, default=0)
18      parser.add_argument("--lastColumn", help="All columns of your file from firstColumn to
            lastColumn will be used for Bergs method. Value shuold be >=0. 0: data, 1: squared
            data, 2: cubed data. (default:0)", type=int, default=0)
```

```
    parser.add_argument("--binsJackknife", help="Number of bins, which will be used for the
        Jackknife method. It should be much smaller than ndat. (default:10)", type=int,
        default=10)
    parser.add_argument("--tMax", help="Maximum time of the Autocorrelation function.(default
        :10)", type=int, default=10)
    parser.add_argument("--activateBergPlot", help="Adding this command without a option
        activates the plot.", action='store_true', default=False)
    parser.add_argument("--factorS", help="For the Gamma-method: To choose the summation window
         W Wolff constructs the hypothesis: tau~S*tau_int with some factor S. Since a too
        large S leads to large statistical error of the error please choose S in [1,2]. (
        default:1.5)", default=1.5)
    parser.add_argument("--activateWolffPlot", help="Adding this command without a option
        activates the plot.", action='store_true', default=False)
  # parser.set_defaults(activateWolffPlot=False)
    return parser.parse_args()


def GetCommandLineOptionValues(args):

    return args.systempath, args.filename, args.method, args.firstColumn, args.lastColumn, args
        .binsJackknife, args.tMax, args.activateBergPlot, args.factorS, args.activateWolffPlot


def CheckIfFileExists(systempath, filename):
    if(os.path.exists(systempath+filename)):
        pass
    else:
        print("The file '" + systempath + filename + "' doesn't exist. Aborting.")
        exit()

def analysisBerg(externalExecutableName, optionsToBePassed, systempath, filename, powerOfX,
    activateBergPlot):
    try:
        import subprocess
        print("#Calling external subprocess with options: \"{0}\"".format( optionsToBePassed ))
        output = subprocess.check_output("{0}{1}".format(externalExecutableName,
            optionsToBePassed), shell=True)
    except Exception, e:
        print("Error:  " + e.message)
        print("Got exception with external program! Aborting...")
    os.rename(systempath+filename+"_auto", systempath+filename+"_auto_"+powerOfX)
    if activateBergPlot == True:
        rc('text', usetex=False)
        dataTau = np.loadtxt(systempath+filename+"_auto_"+powerOfX, usecols=[0])
        dataDeltaTau = np.loadtxt(systempath+filename+"_auto_"+powerOfX, usecols=[1])
        y = dataTau[0:tMax]
        x = np.arange(0, tMax, 1)
        yerr = dataDeltaTau
        plt.figure()
        plt.errorbar(x, y, yerr)
        plt.title('TauInt')
        plt.xlabel('t')
        plt.ylabel(r'$\tau_{int}(t)$')
        plt.savefig(systempath+filename+"_auto_"+powerOfX+".pdf")


def analyssisWolff(systempath, filename, correlatedData, powerOfX, factorS, activateWolffPlot):
    import sys
    sys.path.append("/home/phil-shared/sciarra_zmarsly/py-uwerr")
    from puwr import tauint, correlated_data
    #plt.title("Gamma method of Wolff")
    #plt.savefig(systempath+filename+"_"+powerOfX+"_WolffWindow_S"+str(factorS)+".pdf")
    mean, delta, tint, d_tint = tauint([[correlatedData]], 0, False, plots=activateWolffPlot)
            # instead of 0 one could add a function to apply
    #plt.close()
    saveFile = open(systempath+filename+"_"+powerOfX+"_tauintWolff_S"+str(factorS),"w")
    saveFile.write("#This is the mean and it's error and the integrated autocorrelation time
        tau_int of the data from the file '" + systempath + filename + "' and the error of
        tau_int.")
    saveFile.write("\n")
    saveFile.write("mean = {0}  +/- {1}".format(mean, delta))
    saveFile.write("\n")
    saveFile.write("tau_int = {0}  +/- {1}".format(tint, d_tint))
    saveFile.close()


def AnalyseTheData(firstColumn, lastColumn, systempath, filename, method, binsJackknife, tMax,
    factorS, activateBergPlot, activateWolffPlot):
    for column in range(firstColumn,lastColumn+1):
        correlatedData = np.loadtxt(systempath+filename, usecols=[column])
        powerOfX = "xTo" + str(column+1)
        columnAuto = column+1
```

```
        optionsToBePassed = "-f"+" "+systempath+filename+" "+"-c"+" "+ str(columnAuto)+" "+"-a
            1"+" "+ "--numberOfBinsForAutocorrelation="+str(binsJackknife)+" "+"--
            timeMaxAutocorrelationFunction="+str(tMax)
        externalExecutableName = "/home/phil-shared/sciarra_zmarsly/dataAnalysis "
88
        if (method==1):
            analysisBerg(externalExecutableName, optionsToBePassed, systempath, filename,
                powerOfX, activateBergPlot)
        elif (method==2):
            analyssisWolff(systempath, filename, correlatedData, powerOfX, factorS,
                activateWolffPlot)
93      elif (method == 3):
            analysisBerg(externalExecutableName, optionsToBePassed, systempath, filename,
                powerOfX, activateBergPlot)
            analyssisWolff(systempath, filename, correlatedData, powerOfX, factorS,
                activateWolffPlot)
        else:
            exit()
98

#=============================================================================#


103 if __name__ == "__main__":
    arguments = CreateCommandLineOptionParser()
    systempath, filename, method, firstColumn, lastColumn, binsJackknife, tMax,
        activateBergPlot, factorS, activateWolffPlot = GetCommandLineOptionValues(arguments)
    CheckIfFileExists(systempath, filename)
    AnalyseTheData(firstColumn, lastColumn, systempath, filename, method, binsJackknife, tMax,
        factorS, activateBergPlot, activateWolffPlot)
```

A.3 Python-Implementierung zur Anwendung des Gauß'schen Differenztest und des reduzierten χ^2

```
import argparse
import numpy as np
3 from matplotlib import rc, rcParams
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import math
import printDataWithError as prErr
8

def CreateCommandLineOptionParser():
    parser = argparse.ArgumentParser(prog='tool', formatter_class=lambda prog: argparse.
        HelpFormatter(prog, max_help_position=40, width=100, indent_increment=5))
    parser.add_argument("--systempath", help="Insert the systempath where the file is located:
        /home/.../filename.", type=str, required=True)
    parser.add_argument("--typeOfProbDistribution", help="For the gaussian distribution choose
        '1', for the sum of two gaussians choose '2'.", type=int, choices=[1,2], required=True
        )
13  parser.add_argument("--muTheoretical", help="muTheoretical is the same value as you used to
        generate your data. Insert a value for muTheoretical if you want to apply the
        Gaussian Difference Test. This is a method to check whether the difference between the
        value for muTheoretical and the value for mu you get from fitting the histogrammed
        data is due to chance or significant. If you chose one Gaussian as distribution,
        muTheoretical will be compared with the value for mu from the fit. If you chose the
        sum of two Gaussians as distribution, muTheoretical will be compared with mu1 from the
        fit. (default: muTheoretical=%(default)s", type=float, default=0.0)
    parser.add_argument("--mu2Theoretical", help="mu2Theoretical is used for the Gaussian
        Difference Test in the same way than muTheoretical, but it will be compared with mu2
        from the fit if you chose the sum of two gaussians as distribution. (default:
        mu2Theoretical=%(default)s", type=float, default=0.0)
    parser.add_argument("--sigmaTheoretical", help="sigmaTheoretical is used for the Gaussian
        Difference Test in the same way than muTheoretical, but it will be compared with sigma
        from the fit. It will be used for both distributions. (default: mu2Theoretical=%(
        default)s", type=float, default=1.0)
    parser.add_argument("--nbins", help="The number of bins used to histogram the data. If 0 is
        given, an ideal number of bins is automatically found. (default: %(default)s)", type=
        int, default=0)
    parser.add_argument("--activatePlot", help="Insert the command without a option to activate
        the plot of the histogram and the fit curve.", action="store_true", default=False)
```

```
    parser.add_argument("--xyrange", help="Insert the following information for the plot: xMin, xMax, yMin, yMax. If not given, they are automatically determined.", nargs=4, type=float)
    return parser.parse_args()

def GetCommandLineOptionValues(args):
    if args.nbins == 0: args.nbins = 'auto'
    if args.xyrange == None:
        args.xyrange = [0,0,0,0]
    return  args.systempath, args.typeOfProbDistribution, args.muTheoretical, args.mu2Theoretical, args.sigmaTheoretical, args.nbins, args.activatePlot, args.xyrange

def ReadDataFromFile(filename):
    return np.loadtxt(filename, usecols=[0])

def GetValuesForXAndYFromTheHistogram(correlatedData, nbins, quiet=False):
    # get the height for every x value as a tuple of a data point. bins (actually binCenters): x data, n: y data
    n, bins, patches = plt.hist(correlatedData, bins=nbins, normed=False)
    binCenters = bins[:-1] + 0.5 * (bins[1:] - bins[:-1])
    binWidth = bins[1] - bins[0] #They are all equals
    areaOfHistogram = sum(n)*binWidth
    xValue = binCenters
    yValue = n
    eyValue = np.sqrt(yValue)
    #Normalize data to be fitted
    yValue  /= areaOfHistogram
    eyValue /= areaOfHistogram
    #Remove points referring to empty bins which break fit
    zeroY = []
    for _ in range(len(yValue)):
        if yValue[_] == 0.0:
            zeroY = np.append(zeroY, _)
    if not quiet:
        print("\n  Found {0} bin(s) with no entries => excluding them from fit!".format(len(zeroY)))
    xValue  = np.delete(xValue,zeroY)
    yValue  = np.delete(yValue,zeroY)
    eyValue = np.delete(eyValue,zeroY)
    return xValue, yValue, eyValue

# define a model function (gauss) which is the basis for the fit
def gaussFct(x, mu, sigma):
    gauss = (1.0/(np.sqrt(2.0 * math.pi)*sigma)) * np.exp( -((x - mu) / sigma)**2/2.0 )
    return gauss

def gaussSumFct(x, mu1, sigma1, mu2, sigma2):
    gaussPrefactor = 1.0/2
    gauss1 = gaussFct(x, mu1, sigma1)
    gauss2 = gaussFct(x, mu2, sigma2)
    gauss = gaussPrefactor * (gauss1 + gauss2)
    return gauss

def gaussSumFctSameSigma(x, mu1, mu2, sigma):
    gaussPrefactor = 1./2
    gauss1 = gaussFct(x, mu1, sigma)
    gauss2 = gaussFct(x, mu2, sigma)
    gauss = gaussPrefactor * (gauss1 + gauss2)
    return gauss

def FunctionForFit(typeOfProbDistribution):
    if(typeOfProbDistribution == 1):
        return gaussFct
    else:
        return gaussSumFctSameSigma

def MakeFitAndGetFitResult(typeOfProbDistribution, xValue, yValue, eyValue, theoreticalValues):
    function = FunctionForFit(typeOfProbDistribution)
    popt, pcov = curve_fit(function, xValue, yValue, p0=theoreticalValues, sigma=eyValue, absolute_sigma=True)
    perr = np.sqrt(np.diag(pcov))
    chi2 = sum( ( (function(xValue, *popt) - yValue) / eyValue )**2 )
    NDF = len(xValue) - len(popt)
    reducedChi2 = chi2/NDF
    return popt, perr, chi2, NDF, reducedChi2

def PrintResultOfFitToScreen(params, errors, reducedChi2, NDF):
    print("\n =====================================")
    for _ in range(len(params)):
        print("      p[{0}] = {1:6.4f} +- {2:5.4f}".format(_, params[_], errors[_]))
    print("  ---------------------------------- ")
    print("      The reduced chi2 is {0:.3f} for NDF={1}".format(reducedChi2, NDF))
    print(" =====================================")
```

```
def PlotNormalizedHistogramAndFitFunction(typeOfProbDistribution, data, xValue, params, errors,
    reducedChi2, nbins, Q, xyrange):
    # See https://stackoverflow.com/a/42768093 for the following bunch of lines
    from matplotlib import backend_bases
    from matplotlib.backends.backend_pgf import FigureCanvasPgf
    backend_bases.register_backend('pdf', FigureCanvasPgf)
    pgf_with_latex = {
        "text.usetex": True,            # use LaTeX to write all text
        "pgf.rcfonts": False,           # Ignore Matplotlibrc
        "pgf.preamble": [ r'\usepackage[dvipsnames, usenames]{color}' ]    # xcolor for colours
    }
    rcParams.update(pgf_with_latex)

    myTitle = ''
    for _ in range(len(params)):
        if Q[_] < 0.01:
            color = 'red'
        elif Q[_] < 0.05 or Q[_] > 0.99:
            color = 'YellowOrange'
        else:
            color = 'Green'
        myTitle += r'$\textcolor{%s}{p_{%d}} = %s\quad$'%(color, _, prErr.
            GetValueAndErrorInHumanLikeWay(params[_], errors[_], True))
    plt.clf() #Because we had done a not normalized histogram
    fig = plt.figure()
    ax = fig.add_subplot(111)
    if reducedChi2 < 0.1 or reducedChi2 > 5:
        color = 'red'
    elif reducedChi2 >= 4:
        color = 'darkorange'
    else:
        color = 'forestgreen'
    plt.text(0.8, 0.9, "$\chi^2_{{NDF}} = {0:.3f}$".format(reducedChi2), horizontalalignment='
        center', verticalalignment='center', transform=ax.transAxes, color=color)
    #Set axes limits
    if not xyrange == [0,0,0,0]:
        axes = plt.gca()
        axes.set_xlim([xyrange[0],xyrange[1]])
        axes.set_ylim([xyrange[2],xyrange[3]])
    plt.hist(correlatedData, bins=nbins, normed=True)
    function = FunctionForFit(typeOfProbDistribution)
    plt.plot(xValue, function(xValue, *params))
    plt.title(r'%s' %myTitle, y=1.01)
    plt.xlabel("$x$")
    plt.ylabel("$f(x)$")
    from os import path
    plt.savefig(path.basename(systempath)+"_fitHistogram.pdf")

def GaussianDifferenceTest(valueA, errorA, valueB, errorB):
    sigma = np.sqrt(errorA**2 + errorB**2)
    differenceMean = abs(valueA - valueB)/(sigma * np.sqrt(2.0))
    QValue = 1.0-math.erf(differenceMean)
    #print("Gaussian Difference Test of ", valueA, " and ", valueB, "provides the Value for Q:
        ", QValue)
    return QValue

def GetTheoreticalValuesAsArray(typeOfProbDistribution, mu, mu2, sigma):
    if(typeOfProbDistribution == 1):
        return [mu, sigma]
    else:
        return [mu, mu2, sigma]

def AdjustOrderOfTheoreticalValuesForCompatibilityTest(typeOfProbDistribution, params,
    theoreticalValues):
    if(typeOfProbDistribution == 2) and (params[0]-theoreticalValues[0]) > 0.5:
        theoreticalValues[0], theoreticalValues[1] = theoreticalValues[1], theoreticalValues[0]
    return theoreticalValues

def MakeCompatibilityTest(parameters, errors, theoreticalValues, quiet=False):
    import sys, os
    def blockPrint():
        sys.stdout = open(os.devnull, 'w')
    def enablePrint():
        sys.stdout = sys.__stdout__
    if(quiet):
        blockPrint()
    print("\n ================================================")
    print(" Gaussian Difference Test:")
    print(" -------------------------")
    Qvalues=[]
    for _ in range(len(parameters)):
        Qvalue = GaussianDifferenceTest(parameters[_], errors[_], theoreticalValues[_], 0.0)
```

```
173             Qvalues.append(Qvalue)
                if Qvalue < 0.01:
                    colorCode = '\033[91m\033[1m'
                elif Qvalue < 0.05 or Qvalue > 0.99:
                    colorCode = '\033[93m\033[1m'
178             else:
                    colorCode = '\033[92m\033[1m'
                Qvalue = "{0}{1:.5g}\033[0m".format(colorCode, Qvalue)
                print("      p_theo[{0}] = {1:6.4f}   =>    Q = {2}".format(_, theoreticalValues[_],
                    Qvalue))
        print(" ================================================\n")
183     enablePrint()
        return Qvalues


#=============================================================================#
188
if __name__ == "__main__":
    arguments = CreateCommandLineOptionParser()
    systempath, typeOfProbDistribution, muTheoretical, mu2Theoretical, sigmaTheoretical, nbins,
        activatePlot, xyrange = GetCommandLineOptionValues(arguments)
    correlatedData = ReadDataFromFile(systempath)
193 xValue, yValue, eyValue = GetValuesForXAndYFromTheHistogram(correlatedData, nbins)
    theoreticalValues = GetTheoreticalValuesAsArray(typeOfProbDistribution, muTheoretical,
        mu2Theoretical, sigmaTheoretical)
    parameters, errors, chi2, NDF, reducedChi2 = MakeFitAndGetFitResult(typeOfProbDistribution,
        xValue, yValue, eyValue, theoreticalValues)
    PrintResultOfFitToScreen(parameters, errors, reducedChi2, NDF)
    AdjustOrderOfTheoreticalValuesForCompatibilityTest(typeOfProbDistribution, parameters,
        theoreticalValues)
198 Qvalues = MakeCompatibilityTest(parameters, errors, theoreticalValues)
    if(activatePlot):
        PlotNormalizedHistogramAndFitFunction(typeOfProbDistribution, correlatedData, xValue,
            parameters, errors, reducedChi2, nbins, Qvalues, xyrange)
    else:
        exit()
```

Literatur

[1] Berg, Bernd A.: *Markov Chain. Monte Carlo Simulations and their statistical analysis.* World Scientific, 2004.

[2] Gattringer, C.; Lang, C.B.: *Quantum Chromodynamics on the Lattice. An Introductory Presentation.* Springer Berlin Heidelberg, 2010.

[3] Nolting, W.: *Grundkurs Theoretische Physik 6. Statistische Physik.* Springer-Verlag Berlin Heidelberg, 7. Auflage, 2014.

[4] Rothe, Heinz J.: *Lattice Gauge Theory. An Introduction.* World Scientific Lecture Notes in Physics, 2005.

[5] Sciarra, Alessandro: *The QCD phase diagram at purely imaginary chemical potential from the lattice.* Dissertation. Frankfurt: J.W.Goethe Universität, 2017.

[6] Waerden, B. L. van der: *Mathematische Statistik.* Springer-Verlag Berlin Heidelberg, 3. Auflage, 1971.

[7] Wolff, Ulli: *Monte Carlo errors with less errors.* Comput. Phys. Commun. 156, 2004. Verfügbar via [arXiv:hep-lat/0306017].

Selbständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit gemäß §30(12) der Ordnung für den Bachelor- und den Masterstudiengang Physik der Johann Wolfgang Goethe-Universität Frankfurt am Main vom 24.04.2013 selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst zu haben. Alle Stellen, Bilder und Zeichnungen, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderen fremden Texten entnommen wurden, wurden als solche kenntlich gemacht. Diese Abschlussarbeit ist nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet worden.

Datum	Unterschrift