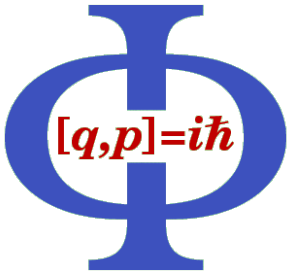
GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN



Institut für Theoretische Physik

Fakultät für Physik
Friedrich-Hund-Platz 1
37077 Göttingen

Skriptum zur Vorlesung

Computergestütztes wissenschaftliches Rechnen Teil 2

von Andreas Honecker

Sommersemester 2011

basierend auf der Vorlage
„Einführung in die Rechnerbedienung und Programmierung in den Naturwissenschaften“
von Thomas Pruschke aus dem Sommersemester 2008

(Stand: 15. Juli 2011)

PDF-Fassung dieses Skripts (mit Hyperlinks etc.) unter

http://www.theorie.physik.uni-goettingen.de/~honecker/CWR2skript.pdf

Inhaltsverzeichnis

Kapitel 1

Erste Schritte

1.1 Einführung

Die zunehmende Leistungsfähigkeit der Computer, speziell die Entwicklung von „Supercomputern“, bestehend aus Netzwerken von oftmals mehreren 10000 einzelner Prozessoren, haben in den letzten 20 Jahren zu einem extremen Anwachsen der Bedeutung dieser Maschinen für die Wissenschaft geführt. Speziell in den Naturwissenschaften erlauben Computer die numerische Behandlung von sehr komplexen Problemen auf den verschiedensten Längen- und Zeitskalen. Eine kleine Auswahl:

- Die Berechnung von Wechselwirkungswahrscheinlichkeiten von Elementarteilchen („Monte-Carlo für Quantenchromodynamik“).
- Das Verhalten schwerer, instabiler Kerne unter extremen Bedingungen (spart Tests z.B. in Nevada) („Quanten Monte-Carlo“).
- Die Berechnung elektrischer, mechanischer und magnetischer Eigenschaften fester Körper („Klassische und Quanten Monte-Carlo“, „Dichtefunktionaltheorie“, „Molekulardynamik“).
- Die Untersuchung von Strömungen (Flüssigkeiten oder Gase) um unregelmäßige Körper („Lösung partieller Differentialgleichungen“, „Finite Elemente“).
- Bewegung von Körpern im Gravitationsfeld, elektrischen und magnetischen Feldern („Systeme (partieller) Differentialgleichungen“).

Außerdem kann man hier noch etliche ähnliche Problemkreise aus den Bereichen der Chemie, Biologie, Wirtschaftswissenschaften, Mathematik, etc. aufzählen.
Interessanterweise gibt es trotz der Vielfalt in Themen nur eine relativ kleine Anzahl von Verfahren, die in verschiedenen Verkleidungen immer wieder auftreten. Ziel dieser Vorlesung ist es, Ihnen einen Überblick über diese grundlegenden Verfahren zu geben und Sie mit den Eigenheiten und Fallstricken vertraut zu machen. Am Ende der Vorlesung werden Sie sicher kein ausgewiesener Experte in auch nur einem dieser sog. Algorithmen sein, haben aber eigenständig Programme für jeden Einzelnen entwickelt und sollten somit in der Lage sein, sich schnell in komplexere numerische Probleme einarbeiten und die zugehörigen Algorithmen einigermaßen effizient in lauffähige Programme umsetzen.

1.1.1 Werkzeuge

Obwohl es immer wieder sehr befriedigend ist, schwierige mathematische Probleme mit Papier und Bleistift analytisch zu lösen, werden Sie im Studium sehr schnell feststellen, daß die interessanten und für die Anwendung relevanten Fragen ohne die Hilfe eines Computers oft nicht mehr zu lösen sind.

Um Computer zu sinnvollen Werkzeugen der Wissenschaft zu machen, braucht man Kenntnisse in Programmierung. Die Bedienung von existierenden Programmen kann für wissenschaftliche Fragestellungen nicht genügen. Andererseits kann man nicht alles selber programmieren und immer wieder das berühmte, steinzeitliche Rad neu erfinden. Daher muss man auch mit existierenden Softwarewerkzeugen umgehen können. Dies trifft insbesondere auf Softwarebibliotheken und Visualisierungssoftware zu (wer will schon die Computergrafik neu erfinden?). Anfänger sollten sich allerdings von der überwältigenden Vielzahl existierenden Werkzeuge nicht blenden lassen, sondern lieber diszipliniert eigene Programmiererfahrung aufbauen.
In dieser Vorlesung und den zugehörigen Übungen setze ich Erfahrungen mit Programmiersprache voraus. Sinnvoll sind einigermaßen solide Kenntnisse in C oder C++, aber im Prinzip tut's auch jede andere Sprache (PASCAL, ADA, FORTRAN, BASIC, ...) oder Umgebungen wie Matlab.
Die Programmiersprache sollte eigentlich nicht die Hauptrolle bei der Lösung wissenschaftlicher Probleme spielen, sie bestimmt aber auf subtile Weise das weitere Denken. Heutzutage dominieren sogenannte *prozedurale* und *objekt-orientierte* Sprachen im numerischen Teil des wissenschaftlichen Rechnens, während in der Computeralgebra *funktionale* Sprachen gefragt sind. Die Elemente prozeduraler Sprachen erlauben es, die Speicherinhalte in Computern auf genau definierte Art zu manipulieren. Wichtige Vertreter im wissenschaftlichen Rechnen sind FORTRAN und C. Objekt-orientierte Sprachen (C++, JAVA, ...) arbeiten mit Sprachkonstrukten, die Datenstrukturen und die Prozeduren, mit denen sie manipuliert (angelegt, gelesen, geändert, gelöscht,...) werden, zu sinnvollen Einheiten verbinden. Funktionale Sprachen (LISP, ...) arbeiten mit Input-Output Beziehungen statt mit Prozeduren zur Veränderung von Speicherinhalten.
Wir werden in dieser Vorlesung C++ verwenden. Um es nochmals deutlich zu machen: Die Vorlesung bietet keine Einführung in eine Programmiersprache oder in Grundbegriffe der Programmierung. Dazu fehlt hier die Zeit. Sie sollten also Kenntnisse in einer Sprache mitbringen und schon einmal Programme geschrieben und zum Laufen gebracht haben.
Neben einer Programmiersprache (natürlich einschließlich des zugehörigen Compilers) sind Werkzeuge der Visualisierung besonders wichtig. Lernen Sie mit einigen, wenigen dieser Werkzeuge richtig umzugehen. In der Vorlesung verwenden wir *xmgr*, gelegentlich auch *gnuplot*, und für manche Zwecke auch mal ein Computergafikprogramm namens *povray*.
Übrigens: Zum Entwurf und zur Programmierung von graphischen Benutzeroberflächen finden Sie hier nichts.
Andere Werkzeuge des wissenschaftlichen Rechners sind:

- ein *Editor*. Benutzen Sie, was Sie wollen und vor allem, was Sie bedienen

können.

- ein sog. *build*-Werkzeug, mit dem Sie immer wieder anfallende Arbeiten im codieren-kompilieren-laufenlassen-abstürzen-Fehlersuche-codieren Zyklus automatisieren können. Ich benutzen *make*. Eine Mikroeinführung ergibt sich aus den Beispielen.

- Einen *Debugger* benutzen wir in dieser Vorlesung nicht. Entweder Sie können mit so einem Werkzeug umgehen, dann benutzen Sie es oder nicht, dann lassen Sie es bleiben. Seien Sie aber gewarnt: ohne Erfahrung und Praxis hilft Ihnen ein Debugger bei der Fehlersuche auch nicht weiter.

- *Integrierte Entwicklungsumgebungen* sind erst dann anzuraten, wenn man genügend viel Erfahrung beim Programmieren hat um abschätzen zu können, ob die einem wirklich weiterhelfen. Riesige Plattformen gibt es von kommerziellen Anbieter, aber auch als freie Software. Wer sich traut, kann *kdevelop*, *anjuta* oder *eclipse* mal versuchen. Wirklich benötigen werden wir so etwas für die Vorlesung aber nicht.

- Umgebungen für wissenschaftliches Arbeiten, die Computeralgebra, Numerik, Grafik, wissenschaftliche Textverarbeitung und mehr integrieren können nützliche Werkzeuge sein. Wir können *maple* und *matlab* empfehlen, allerdings erfordern diese „mächtigen“ Werkzeuge auch eigene Vorlesungen und Übungen, um ihren Gebrauch zu erlernen. Hier kommt nichts vor, was Sie in die Lage versetzen kann, diese Software sinnvoll einzusetzen. Andererseits können Sie, wenn Sie mit diesen Umgebungen bereits umgehen können, die anfallenden Übungen auch damit erledigen.

- *Programmbibliotheken*. Selbstverständlich muss man nicht das sprichwörtliche Rad immer wieder neu erfinden und kann von guten Programmbibliotheken profitieren. Allerdings sollte man Bibliotheksroutinen als Wissenschaftler (!) erst verwenden, wenn man vollständig versteht, was sie tun. Das kann man aber nur lernen, indem man selbst einige Räder erfindet. Daher ist der Einsatz von Bibliotheksfunktionen in den Übungen oder der Modulprüfung nicht gestattet.

1.1.2 Literatur

[1] T. Pruschke, „*Einführung in die Rechnerbedienung und Programmierung in den Naturwissenschaften*“, Skript, Göttingen, Sommersemester 2008.

[2] F.J. Vesely, „*Computational Physics: An Introduction*“, Kluwer Academics.

[3] W. Kinzel und G. Reents, „*Physics by Computer*“, Springer Verlag; „*Physik per Computer*“, Spektum Akademischer Verlag (1996).

[4] H. Gould, J. Tobochnik, W. Christian, „*An Introduction to Computer Simulation Methods*“, 3rd edition, Addison-Wesley (2007).

[5] W.H. Press, S.A. Teukolsky, V.T. Vetterling, B.P. Flannering, „*Numerical Recipes*“, Third Edition, Cambridge University Press (2007), www.nr.com

[6] A. Honecker, „*Computergestütztes wissenschaftliches Rechnen*“, Skript, Göttingen, Sommersemester 2007.

[7] J.J. Barton und L.R. Nackman, „*Scientific and Engineering C++*“, Addison-Wesley.

[8] Daoqui Yang, „*C++ and Object-Oriented Numeric Computing for Scientists and Engineers*“, Springer Verlag, New York.

[9] S. Meyers, „*Effective C++. 50 Ways to Improve Your Programs and Design*“, Addison-Wesley.

1.2 Ein erstes Beispiel

Viele Fragestellungen in den Naturwissenschaften lassen sich auf das Lösen eines Anfangswertproblems (deutsch: AWP, englisch: IVP) gewöhnlicher Differentialgleichungen (ODE: Ordinary Differential Equation) zurückführen.
Auf die Struktur und Lösung solcher ODEs gehen wir später ein. Hier soll anhand eines konkreten Beispiels gleich ein generelles Problem aller numerischer Verfahren illustriert werden.

1.2.1 Das Modell

Wir betrachten im Folgenden ein Modell für das Wachstum einer biologischen Population, das von *Pearl, Verhulst* und *Lotka* eingeführt und untersucht wurde. Mit $n(t)$ bezeichnen wir die Anzahldichte der Individuen einer Population, die wir als kontinuierliche Variable betrachten (gut für viele Individuen pro Fläche oder Volumen, schlecht für wenige). Pro Zeiteinheit nimmt die Dichte aufgrund von „Geburten“ zu und aufgrund von „Todesfällen“ ab. Die *Geburtenrate=Anzahl der Geburten pro Individuum pro Zeit* sei r_+ , die *Todesrate* r_-. Dann erfüllt $n(t)$ die sogenannte *Ratengleichung*:

$$\frac{dn}{dt} = (r_+ - r_-)n \tag{1.1}$$

$r = r_+ - r_-$ bezeichnet man auch als *Netto-Reproduktionsrate*. Um von der Ratengleichung zu einer ODE für unser System zu kommen, muss man die Raten $r_\pm$ noch durch die Dichte zur Zeit t ausdrücken.
Das einfachste Modell nimmt an, dass die Raten unabhängig von den Dichten sind (*Malthus'sches Wachstum*). Das Pearl-Verhulst Modell nimmt an, dass für große Dichten *limitierende Faktoren* auftreten und die Netto-Reproduktionsrate abnimmt. Dabei gelte der einfache Ansatz

$$r_+(n) - r_-(n) = r_0(1 - Kn)$$

Der Parameter r_0 kontrolliert das (exponentielle) Wachstum der Population: Für $K = 0$ hat man die einfache DGL

$$\frac{dn}{dt} = r_0 \cdot n \ ,$$

die ein unbeschränktes exponentielles Anwachsen der Individuen liefert. Der Parameter K heißt in der Ökologie-Literatur auch *Kapazität* des Lebensraums. Er sorgt im Term $K \cdot n$ dafür, dass bei zu hoher Population die Netto-Reproduktionsrate effektiv abnimmt (entweder weil weniger geboren werden oder – z.B. Nahrungsmittelmangel – mehr sterben).

Unsere ODE für das Populationsmodell hat also die Form

$$\frac{dn}{dt} = r_0(1 - Kn(t))n(t) \tag{1.2}$$

Diese ODE ist einfach genug, um die Lösung des AWP $n(t = 0) = n_0 > 0$ geschlossen in analytischer Form angeben zu können. Sie lautet (rechnen Sie es mal nach)

$$n(t) = \frac{n_0 \exp(r_0 t)}{1 + Kn_0(\exp(r_0 t) - 1)} \tag{1.3}$$

Solche exakten Lösungen einfacher Probleme sind immer sehr geeignet zum Testen numerischer Verfahren, und genau das wollen wir jetzt ausnutzen.
Beachten Sie, dass weitere Modifikationen, wie z.B. eine etwas kompliziertere Abhängigkeit der Reproduktionsrate von der Dichte, wie z.B.

$$r(n) = r_0(1 - Kn + bn^2) \ ,$$

schon keine Lösung in geschlossener Form durch elementare Funktionen mehr zulassen.

> Je realistischer ein Modell, desto weniger kann man auf analytische Lösungen hoffen.

1.2.2 Der Algorithmus

Wie Sie bereits wissen sollten, kann kein Computer mit kontinuierlichen Zahlenkörpern umgehen. Intern werden alle reellen Zahlen in Form so genannter *Gleitkommazahlen* abgelegt, die einen endlichen Bereich mit einer endlichen Auflösung überstreichen (z.B. doppelt genaue Zahlen von $10^{-308} \cdots 10^{308}$, kleinste auflösbare Differenz 10^{-16}). Zudem will man in endlicher Zeit mit seinen Rechnungen fertig sein. Die Grundstrategie, um ODEs numerisch zu lösen besteht in der *Diskretisierung*. Dazu denke man sich das Zeitintervall $[0, T]$ von der Anfangszeit $t = 0$ bis zur gewünschten Zeit T in „hinreichend kleine“ Teilintervalle zerlegt, so dass

$$t_i \leq t < t_{i+1}, \quad i = 0, \ldots, N - 1$$

wobei $t_N = T$. In jedem kleinen Teilintervall löst man nun die ODE *approximativ* unter Benutzung von Taylorentwicklungen, deren Fehler kontrollierbar mit kleiner werdender *Schrittweite* $t_{i+1} - t_i$ verschwinden. Dann geht man zum nächsten Schritt weiter. Wichtig ist dabei, dass die Fehler innerhalb der kleinen Schritte sich nicht zu einem riesigen Gesamtfehler akkumulieren.
Der einfachste Algorithmus dieser Art für eine ODE

$$\frac{dn}{dt} = f(n(t), t) \tag{1.4}$$

ist der *Euler-Cauchy'sche*. Zur Vereinfachung der Notation schreiben wir

$$n(t_i) = n_i \quad .$$

Nun betrachten wir einen Zeitschritt $t_i \to t_{i+1}$. Bei bekanntem n_i erhalten wir n_{i+1} formal durch Integration als

$$n_{i+1} = n_i + \int_{t_i}^{t_i+1} f(n(s), s)ds \quad .$$

Natürlich kennen wir die Funktion $n(t)$ im Intervall $[t_i, t_{i+1}]$ nicht explizit, d.h. wir können das Integral eigentlich nicht auswerten. Wir approximieren daher den Integranden durch eine Konstante, d.h. wir ersetzen die Funktion $f(n(s), s)$ durch ihren Wert am Anfang des Intervalls,

$$f(n(s), s) \approx f(n_i, t_i) \quad .$$

Somit ist

$$n_{i+1} \approx n_i + \Delta t_i f(n_i, t_i) + \mathcal{O}(h_i^2) \quad , \tag{1.5}$$

wobei $\Delta t_i = t_{i+1} - t_i$ die *Schrittweite* bezeichnet. Offensichtlich hängt die Güte der Näherung von Δt_i ab: Je kleiner Δt_i, desto genauer scheint der Algorithmus zu sein. Es ist aber auch klar, dass man Δt_i nicht beliebig klein machen kann, da man irgendwann an die Genauigkeit der Zahlendarstellung stößt.
Die häufigste Wahl sind *äquidistante* Schritte. Dazu zerlegt man das Intervall der Länge T in N Schritte der Länge $\Delta t = T/N$, d.h. $t_n = n\Delta t$. Man beachte, dass der *gesamte Fehler* nach N äquidistanten Euler-Schritten

$$\text{globalerror} = N\mathcal{O}(\Delta t^2) = N\mathcal{O}(T^2/N^2) = \mathcal{O}(1/N)$$

ist, d.h. er sollte bei hinreichend kleiner Schrittlänge Δt beliebig klein werden.

1.2.3 Ergebnisse

Die Implementierung dieses einfachen Algorithmus für die Populationsgleichung (1.2) überlassen wir Ihnen als erste Übung. Hier möchte ich nur das Verhalten des Algorithmus diskutieren. Dazu schreiben wir den Euler-Cauchy-Algorithmus etwas um als

$$n_{i+1} = n_i + \Delta t \cdot r_0 n_i(1 - Kn_i) = (1 + \Delta t\, r_0)n_i(1 - \frac{\Delta t\, r_0 K}{1 + \Delta t\, r_0} n_i) \quad ,$$

reskalieren gemäß

$$x_i = \frac{\Delta t\, r_0 K}{1 + \Delta t\, r_0} n_i$$

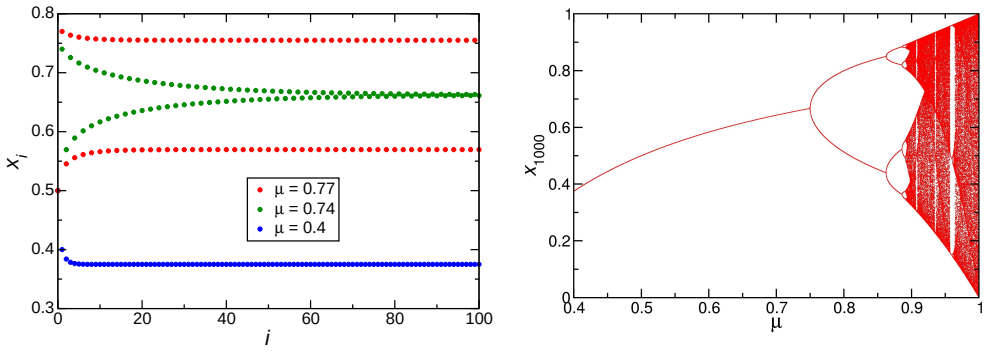


Abbildung 1.1: Zeitentwicklung der logistischen Abbildung (1.6) für verschiedene Werte von μ (links) bzw. als Folge von Punkten in der μ-x-Ebene („Poincaré-Schnitt“, rechts).

und führen die Bezeichnung

$$4\mu = 1 + \Delta t\, r_0$$

ein. So nimmt der Euler-Schritt die Form

$$x_{i+1} = 4\mu x_i(1 - x_i) \tag{1.6}$$

an, die man als *logistische Abbildung* bezeichnet. Bei festem Δt nimmt μ mit wachsendem r_0 zu. Sie können nun ein weiteres Programm schreiben, das den Euler-Algorithmus in der reskalierten Form als diskrete Abbildung auffasst und dann diese Abbildung untersuchen, wenn μ immer größer wird. Beschränken Sie sich auf $0 < \mu \leq 1$ und $0 < x \leq 1$. Dabei finden Sie ein erstaunlich komplexes, dynamisches Verhalten (Periodenverdopplung, Chaos, ...). Dieses Verhalten ist für das diskrete Modell (1.6) real, für die ODE (1.2) jedoch ein reines Artefakt der numerischen Behandlung!

Gerade bei „numerischen Entdeckungen“ ist es also äußerst wichtig (und nicht immer einfach) nachzuweisen, dass die Entdeckung real und kein *numerisches Artefakt* ist.

Bei nicht zu großen Raten r_0 erhalten wir eine monotone Annäherung an den Fixpunkt (so sollte es laut exaktem Resultat ja auch sein). Für etwas größere r_0 (s. Abb. 1.1 links, grüne Punkte) zeigt die Trajektorie gedämpfte Oszillationen, bei $\mu = 0.77$ entsteht eine *stabile Oszillation* (Abb. 1.1 links, rote Punkte). Für noch größere Werte von r_0 wird die Dynamik sehr komplex. Das rechte Bild in Abb. 1.1 zeigt die Dynamik des Systems nach 1000 Zeitschritten Voriteration. Man erkennt die Oszillation der Periode 2, die von einer Oszillation der Periode 4 abgelöst wird etc., bis sog. „chaotische“ Trajektorien auftreten, bei denen die Dynamik einen ganzen Bereich der x-μ-Ebene ausfüllt. Dieser chaotische Bereich ist dabei immer wieder

unterbrochen von Gebieten, in denen auch wieder periodische Lösungen auftreten. Obwohl diese komplexe Lösungsvielfalt für die eigentliche ODE nur numerisches Artefakt ist, ist die logistische Abbildung selbst ein berühmtes, ökologisches Modell. So könnte nämlich eine Populationsdynamik aussehen, bei denen die Generationen sich nicht überlappen (wie z.B. bei Maikäfern: Ist die neue Generation von Käfern da, ist die alte weg) und bei der der Generationswechsel sich für alle Individuen synchron vollzieht (eben Maikäfer!).

1.3 Effizienz von Algorithmen

Neben der Stabilität von Algorithmen ist ein nicht zu unterschätzendes Merkmal auch die Effizienz. Obwohl heutzutage z.B. Speicher kein eigentliches Problem mehr ist (Ihre Armbanduhr hat wahrscheinlich mehr Speicher als ein Supercomputer vor 20 Jahren), ist ein überlegter Umgang mit Ressourcen – wie auch im täglichen Leben – immer sinnvoll. Speziell der Rechenzeitbedarf als Funktion der Komplexität einer Aufgabe kann auch auf modernen Systemen zu einem entscheidenden Faktor werden. Im Folgenden wollen wir uns daher an einem konkreten Beispiel überlegen, wie man den Zeitbedarf eines Algorithmus abschätzen kann und welche Maßnahmen eventuell helfen können, ein Programm effizienter zu machen.
Ein häufiges Problem ist, einen Satz von Objekten nach einer vorgegebenen Ordnungsrelation zu sortieren. Im einfachsten Fall handelt es sich um Zahlen und die Ordnungsrelation ist die übliche „kleiner“ bzw. „gleich“; es kann sich aber z.B. auch um Orten von Atomen in einem Festkörper handeln, und die „Ordnungsrelation“ sagt etwas darüber aus, welche Atome mit einem gegebenen als nächstes Wechselwirken werden.

Gegeben ist eine Menge $\{7, 0, 5, 5, 1, 2, 6\}$ von Zahlen, die der Größe nach sortiert werden sollen. Eine naive Vorgehensweise ist die folgende: Man beginne mit dem 2. Element der Menge und sortiere es bezüglich des ersten Elements. Dann geht man Schritt für Schritt durch die Menge und sortiert jeweils das j-te Element relativ zu den $j-1$ vorherigen Elementen ein. Nehmen wir an, dass das Element $c(j) = a$ sei und ein $0 \leq i < j$ existiert, so dass $c(i) \leq a < c(i+1)$ gilt. Dann muss man folgende Ersetzung durchführen: $c(j-1) \to c(j)$, $c(j-2) \to c(j-1)$, …, $c(i+1) \to c(i+2)$, $c(i+1) = a$. Wie sieht es mit der Abhängigkeit der Rechenzeit von der Anzahl der Elemente N der Folge aus? Offensichtlich muss man N-mal die Operation „Verschieben und Einfügen“ ausführen. Dabei benötigt das Verschieben der Daten jeweils ebenfalls $\mathcal{O}(N)$ Operationen, d.h. insgesamt skaliert die Rechenzeit des Algorithmus $\propto N^2$.

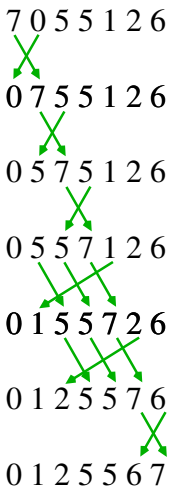


Abbildung 1.2: Naiver Sortieralgorithmus.

Nun ist diese Abhängigkeit von N für 5 Werte nicht weiter schlimm. Muss man aber riesige Mengen (typischerweise etliche 10^5 oder mehr) Daten sortieren, und muss man das etliche Male (typischerweise 10^5 oder mehr) tun, dann ist klar, dass eine Reduktion der Zeit für einen Sortiergang durchaus deutliche Einsparung an Rechenzeit bedeuten kann. Ein Trick, mit dem man dieses Ziel erreichen kann, be-

steht darin, die Folge zunächst in zwei gleichgroße[1] Teilfolgen aufzuspalten, jede dieser Teilfolgen dann auf dieselbe Art zu sortieren und zum Schluss beide Teilfolgen wieder zusammenzufügen. Diese Abfolge von „sort“ und „merge“ gibt dem Algorithmus seinen Namen: Sort-Merge-Algorithmus.
Warum sollte das nun einen Geschwindigkeitsvorteil bringen? Dazu sehen wir uns die Abfolge von Rechenzeiten in Abhängigkeit von der Länge N der Folge an:

$$\begin{aligned} T(N=1) &= C && \text{(nichts zu sortieren)} \\ T(N=2) &= 2C && \text{Teilfolgen zusammenfügen} \\ &\quad +2T(N/2) && \text{Sortieren der Teilfolgen} \\ &\vdots \\ T(N) &= N\cdot C + 2\cdot T(N/2) \end{aligned}$$

Die „Minimalzeit“ C für $N=1$ ergibt sich dabei z.B. aus dem Zugriff auf das Feld, und der Term $N\cdot C$ im letzten Schritt aus der Notwendigkeit, alle N Feldelemente in der richtigen Reihenfolge zu kopieren. Die Idee wird aus der nebenstehenden Skizze ersichtlich (zur Abwechslung einmal mit einer Folge von Buchstaben). Der zeitaufwendige Schritt ist hier offensichtlich das Mischen („merging“) der Zweige.

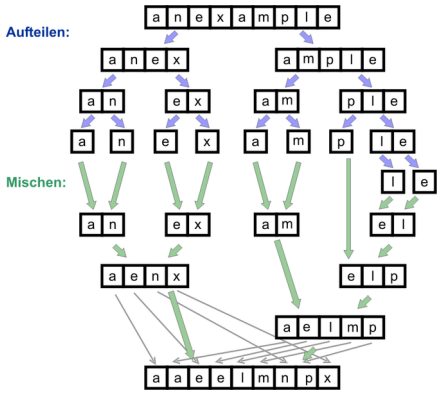


Abbildung 1.3: Beispiel für den Merge-Sort-Algorithmus.

Die Lösung der Rekursionsformel lautet

$$T(N) = \frac{C}{\ln 2} N\ln N \propto N\ln N \ ,$$

wie man durch Einsetzen leicht verifiziert. Das Ergebnis erscheint zunächst nicht sonderlich beeindruckend. Setzt man aber einmal Zahlen ein, z.B. $N=10^5$, dann ergibt sich das Verhältnis der Laufzeiten von Merge-Sort und naivem Sort zu $\mathcal{O}(10^{-6})$, d.h. der Merge-Sort-Algorithmus ist sehr viel schneller!
Leider gibt es wie üblich kein Patentrezept, wie man ein Programm für optimales Laufzeitverhalten strukturieren muss. Aber ein paar Grundregeln gibt es schon:

1. Konstrukte wie `2.0*a` mit Zahlenkonstanten vermeiden. Die Erzeugung von solchen Konstanten zur Laufzeit kann deutlich langsamer sein als der Zugriff auf einen Speicherbereich. Daher lieber eine Variable `two=2.0` definieren und `two*a` schreiben.

2. Die Zahl der Funktionsaufrufe so klein wie möglich halten, da jeder Aufruf einer Funktion neben dem eigentliche Sprung in die Funktion mit einem nicht

[1] Sollte N ungerade sein, dann schlägt man das überzählige Element oBdA der „linken“ Teilfolge zu.

unerheblichen administrativen Aufwand verbunden ist. Nicht zu umfangreichen Programmcode lieber an den entsprechenden Stellen einfügen. Wenn Funktionen sich nicht vermeiden lassen, diese wenn möglich auf Felder operieren lassen.

3. Symmetrien von Objekten ausnutzen. Z.B. kann man für eine symmetrische Matrix $A_{ij} = A_{ji}$ damit den Zugriffsaufwand[2] um einen Faktor 2 senken.

4. Darauf achten, dass Felder kontinuierlich im Speicher angelegt werden. Zugriff auf weit auseinanderliegende Speicherbereiche[3] ist normalerweise sehr ineffizient. Daher auch besonders bei mehrdimensionalen Feldern darauf achten, dass bei Schleifen über die Indizes so weit wie möglich aufeinander folgende Speicheradressen angesprochen werden.

5. Sich ständig wiederholende identische Rechnungen vermeiden. Lieber ein Feld definieren und die Werte darin einmal abspeichern. Speicherzugriffe sind zwar relativ langsam, aber immer noch sehr viel schneller als z.B. der Aufruf der Exponentialfunktion (oder jeder anderen mathematischen Funktion).

Wenn Sie diese einfachen Regeln beherzigen, dann können Sie bereits einigermaßen effiziente Programme schreiben. Mehr kann man dann noch durch Lesen der Literatur bzw. Zusammenarbeit mit Informatikern oder Mathematikern erreichen, bzw. durch die Nutzung von optimierten Bibliotheken (z.B. BLAS, LAPACK für Aufgaben der linearen Algebra).
Eine Warnung sei noch hinsichtlich der „Optimierungsflags“ bei Compilern angebracht. Hier gibt es typischerweise mehrere Stufen, beim Gnu-Compiler z.B. bis Stufe 6. Diese Optimierungen folgen Standardstrategien (Ersetzen von Funktionen durch deren Quellcode (Inlining), Umstellen von Schleifen und Variablen, u.v.m. abhängig vom Compiler. Dabei wird zunächst die Compilierzeit und der Speicherbedarf typischerweise riesig, und der Gewinn bleibt oftmals im Rahmen einiger Prozent[4]. Viel schlimmer, da der Compiler nichts über die Intention des Programmes weiß, kann er unter Umständen durch Umorganisation den Ablauf so sehr ändern,

[2] Bei modernen Computern mit CPU-Cache muss man allerdings mit der Reihenfolge der Zugriffe vorsichtig sein (vgl. den nächsten Punkt). So kostet z.B. eine derartige „Optimierung“ in der Routine `Symmeig:tred2()` aus Kapitel 11.3.2 von [5] grob eine Größenordnung bei der Programm-Laufzeit.

[3] Das gilt besonders dann, wenn Teile der Daten über den sog. Prozessorcache gespiegelt sind, der einen extrem schnellen Zugriff erlaubt. Jeder Zugriff auf Daten außerhalb des Caches führt zu einen herben Geschwindigkeitseinbruch.

[4] Andererseits würde ein Compiler Optimierungen wie in Punkt 1 aus obiger Liste wahrscheinlich automatisch durchführen. Es verbleibt also weiterhin zwischen Compiler- und manueller Optimierung abzuwägen – die beste Entscheidung setzt oftmals eine genaue Kenntnis sowohl des Systems wie auch des Compilers voraus.

dass die Ergebnisse falsch sind oder das Programm nicht mehr vernünftig läuft. Daher gilt die Daumenregel: Die Optimierung durch den Compiler so gering wie möglich halten. Die sicherste Variante ist meist `-O`. Wenn man mehr will, sollte man sich unbedingt die Dokumentation des Herstellers ansehen, und dessen Warnungen hinsichtlich der Aggressivität von Optimierungsoptionen sehr ernst nehmen.

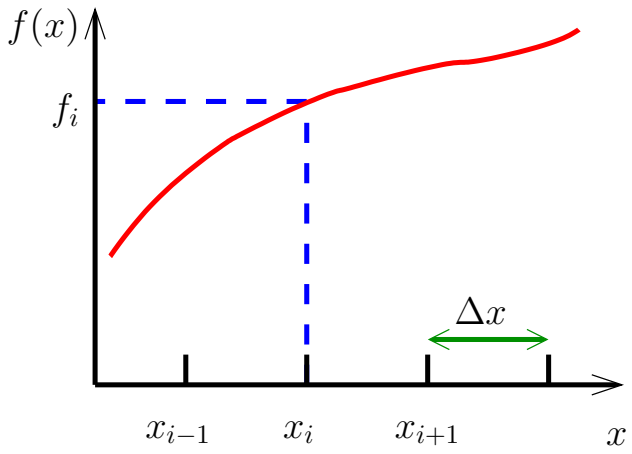


Abbildung 1.4: Zur Definition der Vektoren x und f.

1.4 Konzepte der numerischen Analysis

1.4.1 Numerische Differentiation

Die grundlegenden Fragestellungen der numerischen Analysis sind Differentiation, Integration, Nullstellensuche bei Funktionen und Auffinden von Extrema. Wir nehmen für das Folgende an, dass wir eine hinreichend gutmütige eindimensionale Funktion $f(x)$ gegeben haben sowie einen Vektor $\vec{x}$ mit Elementen $x_i = x_0 + i \cdot \Delta x$ für $i = 0, \ldots, N-1$. Dazu gehört dann ein Vektor $\vec{f}$ von Funktionswerten $f(x_i) \equiv f_i$ (siehe Abb. 1.4).

Numerische Ableitungen kann man nun auf zwei Wegen einführen. Einerseits kann man den Weg über Interpolationsformeln gehen (dieser Weg wird z.B. in Kapitel 1.4 von [1] beschritten). Hier wollen wir eine Abkürzung nehmen und betachten dazu die Taylor-Entwicklung der Funktion f um den Punkt x:

$$f(x+\delta) = f(x) + \delta\, f'(x) + \frac{\delta^2}{2} f''(x) + \frac{\delta^3}{6} f'''(x) + \ldots \tag{1.7}$$

$$f(x-\delta) = f(x) - \delta\, f'(x) + \frac{\delta^2}{2} f''(x) - \frac{\delta^3}{6} f'''(x) + \ldots . \tag{1.8}$$

Wir verwenden diese Formeln nun für $x = x_i$ und $\delta = \Delta x$, also $f(x) = f_i$ und $f(x \pm \delta) = f_{i\pm1}$.

In der praktischen Anwendung benötigen wir zunächst die ersten Ableitungen. Eine erste diskretisierte Fassung, die sogenannte *Newton-Gregory-Vorwärts-Ableitung*, erhält man direkt, indem man (1.7) nach $f'(x_i)$ auflöst:

$$f'(x_i) = \frac{f_{i+1} - f_i}{\Delta x} + \mathcal{O}\left[\Delta x\right] . \tag{1.9}$$

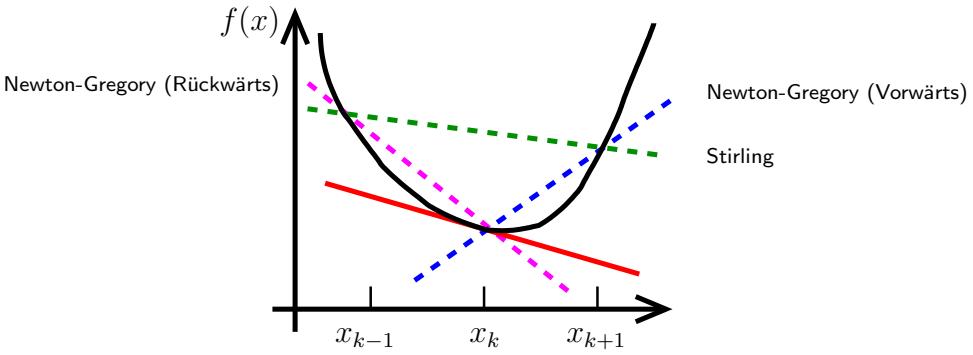


Abbildung 1.5: Vergleich der verschiedenen einfachen Näherungen für die erste Ableitung

Natürlich kann man genausogut (1.8) nach $f'(x_i)$ auflösen und findet die *Newton-Gregory-Rückwärts-Ableitung*

$$f'(x_i) = \frac{f_i - f_{i-1}}{\Delta x} + \mathcal{O}\left[\Delta x\right] . \qquad (1.10)$$

Am intelligentesten ist es, die Differenz von (1.7) und (1.8) zu bilden. Dies führt auf die symmetrische *Stirling-Ableitung*

$$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2\Delta x} + \mathcal{O}\left[(\Delta x)^2\right] . \qquad (1.11)$$

Die Tatsache, dass (1.11) eine Ordnung besser als (1.9) bzw. (1.10) ist, liest man direkt aus (1.7) und (1.8) ab. In der Abb. 1.5 sind die verschiedenen Näherungen für die erste Ableitung. schematisch dargestellt. Die bessere Güte der symmetrischen Formel wird dabei deutlich.
Eine Formel für die *zweite Ableitung* erhält man leicht, indem man die Summe von (1.7) und (1.8) bildet und nach $f''(x_i)$ auflöst:

$$f''(x_i) = \frac{f_{i+1} - 2\,f_i + f_{i-1}}{(\Delta x)^2} + \mathcal{O}\left[(\Delta x)^2\right] . \qquad (1.12)$$

Wir haben also aus f_i und $f_{i\pm1}$ Näherungsausdrücke für die drei Größen $f(x)$, $f'(x)$ und $f''(x)$ hergeleitet. Mehr kann man für drei Eingangsgrößen nicht erwarten. Die Vorgegensweise für den Fall, dass man höhere Ableitungen oder Formeln höherer Ordnung benötigt ist die folgende: man verwendet neben f_i und $f_{i\pm1}$ auch $f_{i\pm2}$ etc., d.h. entwickelt (1.7) und (1.8) ggfs. zu höherer Ordnung und setzt dann neben $\delta = \Delta x$ auch $\delta = 2\,\Delta x$ etc. ein und löst schließlich nach den gewünschten Größen auf, bzw. versucht, Korrekturterme höherer Ordnung zu eliminieren.

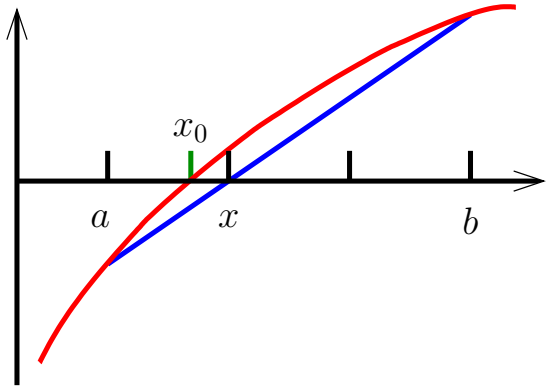


Abbildung 1.6: Zur Arbeitsweise der Regula Falsi.

1.4.2 Nullstellensuche

Ein häufig auftretendes Problem der numerischen Analysis ist die Suche von Nullstellen einer Funktion, d.h. die Lösung der Gleichung $f(x_0) = 0$. Das generelle Vorgehen ist dabei, sich zunächst ein Intervall $[a, b] \ni x_0$ zu suchen, für das $f(a) \cdot f(b) < 0$ gilt. Darauf aufbauend, gibt es dann folgende Verfahren:

1. **Bisektion:**
   In n-ten Schritt liegt im Intervall $[a, b]$ eine Nullstelle, d.h. $f(a) \cdot f(b) < 0$. Sei $x = (a+b)/2$. Ist dann $f(a) \cdot f(x) < 0$, so liegt die Nullstelle im Intervall $[a, x]$ und man setzt $b = x$, andernfalls $a = x$. Das Verfahren wird so lange durchgeführt, bis $\max\{|f(a)|, |f(b)|\} < \epsilon$ mit einer vorgegebenen Schranke ϵ (Konvergenzkriterium).

   Das Verfahren zeichnet sich durch hohe Stabilität aus, allerdings nicht unbedingt durch Effizienz.

2. **Regula falsi:**
   Man benutzt die lineare Interpolationsformel

   $$0 = f(x) \approx f(a) + \frac{f(b) - f(a)}{b - a}(x - a)$$

   um

   $$\boxed{x \approx a - \frac{b - a}{f(b) - f(a)} f(a)}$$

   abzuschätzen. Gilt dann $f(a) \cdot f(x) < 0$, so setzt man wieder $b = x$, ansonsten $a = x$ (siehe Abb. 1.6). Auch dieses Verfahren ist relativ stabil und schneller als das Intervallschachtelungsverfahren. Probleme können jedoch auftreten, wenn die Ableitung von $f(x)$ in der Nähe der Nullstelle sehr klein wird.

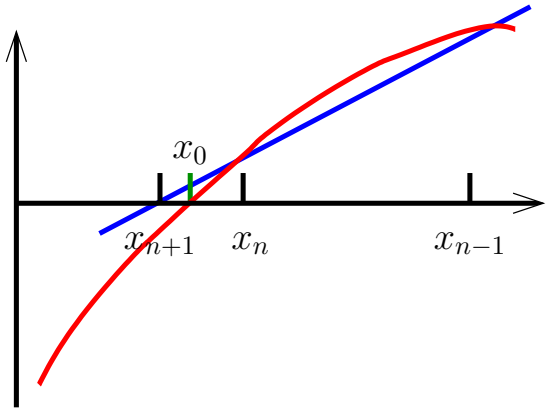


Abbildung 1.7: Zur Arbeitsweise des Sekantenverfahrens.

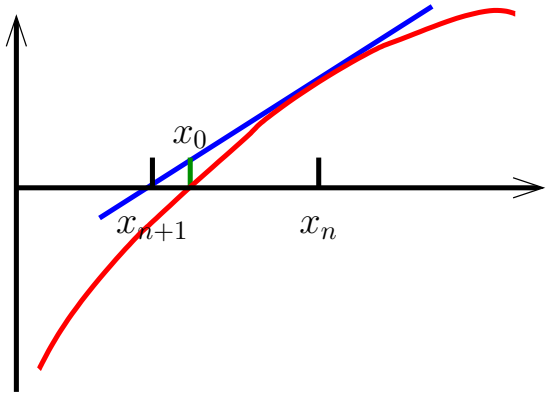


Abbildung 1.8: Zur Arbeitsweise des Newton-Raphson-Verfahrens.

3. **Sekantenmethode:**
   Das Verfahren funktioniert ähnlich zur Regula falsi, allerdings ersetzt man das Intervall immer durch die vorangegangenen Punkte, d.h. $[x_{n-1}, x_n]$, auch wenn in diesem Intervall keine Nullstelle liegt, wie in Abb. 1.7. gezeigt. Die neue Näherung für die gesuchte Nullstelle erhält man dann aus

$$x_{n+1} \approx x_{n-1} - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_{n-1})$$

   Das Verfahren ist sehr schnell, allerdings kann es passieren, dass man eine Nullstelle „verliert“.

4. **Newton-Raphson:**
   Diese Methode benutzt die exakte Ableitung $f'(x)$ der Funktion und berechnet eine neue Abschätzung der Nullstelle aus

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad .$$

Das Verfahren konvergiert sehr schnell, aber auch hier ist Vorsicht angebracht, wenn die Ableitung $f'(x)$ ebenfalls Nullstellen im untersuchten Intervall hat.

Kapitel 2

Differentialgleichungen

Differentialgleichungen sind das A und O der Naturwissenschaften. Sie begegnen uns in den Newton'schen Gleichungen

$$m_i \vec{a}_i = m_i \ddot{\vec{r}}_i = \vec{F}_i(\{\vec{r}_j\}, \{\vec{v}_j\}, \ldots, t) = \vec{F}_i(\{\vec{r}_j\}, \{\dot{\vec{r}}_j\}, \ldots, t) \qquad (2.1)$$

in der Physik, Ratengleichungen in Reaktionen in Chemie oder Populationsdynamiken in Biologie und vielen anderen Gebieten. Die erste grobe Unterscheidung ist zwischen *gewöhnlichen* Differentialgleichungen (*ordinary differential equations*, ODE) und *partiellen* Differentialgleichungen (*partial differential equations*, PDE) zu machen.

ODEs sind von der allgemeinen Form

$$f(y^{(n)}(x), y^{(n-1)}(x), \ldots, y'(x), y(x), x) = 0 \ ,$$

mit einer bislang völlig frei zu wählenden Funktion $f(\ldots)$. Die höchste auftretende Ableitung, n, bezeichnet man als *Ordnung* der DGL. Für unser Beispiel aus der Populationsdynamik (1.2) würde man die Ersetzung $y \leftrightarrow n$, $x \leftrightarrow t$ und $f(y', y, x) = y' - r_0\,(1 - K \cdot y)\,y$ benützen; die Ordnung der DGL wäre dann 1. Eine andere, häufig auftretender Gleichungstyp sind Schwingungsgleichungen

$$\frac{d^2y}{dt^2} + \Gamma \frac{dy}{dt} + \omega_0^2\, y = g(t) \ . \qquad (2.2)$$

Wie viele DGLn, die sich aus Aufgabenstellungen der Mechanik ergeben, ist dies eine DGL 2. Ordnung. Die Schwingungsgleichung hat noch ein paar andere Besonderheiten. Zum Einen treten die gesuchte Funktion und ihre Ableitungen nur *linear* auf. Entsprechend spricht man von einer *linearen DGL*. Zum Anderen sind die Koeffizienten konstant, so dass man es mit einer *linearen DGL mit konstanten Koeffizienten* zu tun hat. Schließlich hat diese DGL noch eine von Null verschiedene rechte Seite $g(t)$, so dass also eine *inhomogene lineare DGL mit konstanten Koeffizienten* vorliegt.

Wie Sie in der Mathematik noch lernen werden, sind die *linearen DGL mit konstanten Koeffizienten* besonders schön, da man sie analytisch lösen kann. Man muss eigentlich nur die Nullstellen eines Polynoms bestimmen.

PDEs treten im Zusammenhang mit Schwingungen, Strömung von Flüssigkeiten oder körnigen Materialien, Aerodynamik, Elektrodynamik, Quantenmechanik etc. auf. Auch hier bezeichnet die höchste auftretende (diesmal partielle) Ableitung die Ordnung der DGL. Auch hier ist ein Prototyp die Schwingungsgleichung[1]

$$\frac{\partial^2 G(x,y,z,t)}{\partial x^2} + \frac{\partial^2 G(x,y,z,t)}{\partial y^2} + \frac{\partial^2 G(x,y,z,t)}{\partial z^2} - \frac{1}{c^2}\frac{\partial^2 G(x,y,z,t)}{\partial t^2} = 0 \ .$$

[1] Wenn sie das Symbol ∂ und die Bedeutung der Schreibweise noch nicht kennen, dann ignorieren Sie das Folgende einfach im Moment.

In beiden Fällen kann man noch unterscheiden zwischen *Anfangswertproblemen* (treten üblicherweise in der Newton'schen Mechanik auf), *Randwertproblemen* (Elektro- und Magnetostatik) und *Eigenwertproblemen* (Quantenmechanik). Bei den Ersteren gibt man zur DGL noch Funktions- und Ableitungswerte (bis zur Ordnung $n-1$) zu einem bestimmten x_0 (oder t_0) vor und bei den Zweiten die Werte von der gesuchten Größe „auf dem Rand". Die dritte Klasse ist typischerweise ein Randwertproblem, und die DGL von der Form

$$f(y^{(n)}(x), y^{(n-1)}(x), \ldots, y'(x), y(x), x) = \lambda y(x)$$

mit zu bestimmendem λ. Typischerweise erlauben die Randbedingungen nur bestimmte Werte von λ, die man dann als Eigenwerte der DGL bezeichnet. Diesen Typus werden Sie bis zur Ermüdung in der Quantenmechanik kennen lernen.

2.1 Gewöhnliche Differentialgleichungen

ODEs der Ordnung $n > 1$ kann man mit einem Trick wieder als DGL 1. Ordnung schreiben. Natürlich hat das keinen Einfluss auf die Lösbarkeit, aber für die numerische Behandlung kann man sich damit formal auf Verfahren für ODEs erster Ordnung beschränken. Dazu definieren wir

$$\begin{aligned} y_1(x) &\equiv y(x) \\ y_2(x) &\equiv \frac{dy(x)}{dx} \\ y_3(x) &\equiv \frac{d^2y(x)}{dx^2} \\ &\vdots \\ y_n(x) &\equiv \frac{d^{n-1}y(x)}{dx^{n-1}} \end{aligned}$$

und führen für die Gleichung

$$f\left(\frac{d}{dx}y_n(x), y_n(x), \ldots, y_1(x), x\right) = 0$$

die explizite Form[2]

$$\frac{d}{dx}y_n(x) - g(y_n(x), \ldots, y_1(x), x) = 0$$

[2] OBdA nehmen wir an, dass die Gleichung $f = 0$ (zumindest lokal) nach der ersten Variablen aufgelöst werde kann.

ein. Die DGL nimmt dann folgende Form an:

$$\begin{aligned}
\frac{dy_n(x)}{dx} &= g\left(y_n(x), \ldots, y_1(x), x\right) \\
\frac{dy_{n-1}(x)}{dx} &= y_n(x) \\
\frac{dy_{n-2}(x)}{dx} &= y_{n-1}(x) \\
&\vdots \\
\frac{dy_1(x)}{dx} &= y_2(x)
\end{aligned}$$

Zuletzt führen wir die Vektorschreibweise

$$\boxed{\begin{aligned}
\vec{y}(x) &:= (y_1(x), y_2(x), \ldots, y_n(x)) \quad , \\
\vec{F}(\vec{y}(x), x) &:= (y_2(x), y_3(x), \ldots, y_n(x), g(\vec{y}(x), x))
\end{aligned}}$$

ein, und erhalten so sehr kompakt

$$\boxed{\frac{d\vec{y}(x)}{dx} = \vec{F}(\vec{y}(x), x) \quad . \qquad (2.3)}$$

Für die Newtonschen Bewegungsgleichungen (2.1) bedeutet dies, dass wir das System aus $3\,N$ gekoppelten Gleichungen *zweiter* Ordnung für die Orte $\vec{r}_i$ durch ein System von $6\,N$ gekoppelten Gleichungen *erster* Ordnung für die Orte $\vec{r}_i$ und Geschwindigkeiten $\vec{v}_i$ ersetzen.
Als konkretes Beispiel nehmen wir unsere Schwingungsgleichung (2.2) mit Inhomogenität $g = 0$. Wir definieren

$$\begin{aligned}
y_1(t) &\equiv y(t) \\
y_2(t) &\equiv \frac{dy(t)}{dt} = \frac{dy_1(t)}{dt}
\end{aligned}$$

und erhalten das System von DGLn

$$\frac{d}{dt}\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} y_2(t) \\ -\omega_0^2 y_1(t) - \Gamma y_2(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -\Gamma \end{pmatrix} \cdot \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$$

Die allgemeine Funktion $\vec{F}$ in (2.3) ist für diesen konkreten Fall also einfach eine lineare Abbildung, dargestellt durch eine Matrix.
Je nach Art der DGL und Anwendungsgebiet benutzt man sehr unterschiedliche numerische Verfahren zur Lösung. Ich werde hier nur die gebräuchlichsten vorstellen. Wenn Sie das Thema insgesamt interessiert, dann schauen Sie am besten in die mathematische Literatur oder in der Referenz [2] für einen Überblick mit weiteren Literaturangaben.

2.1.1 Euler-Cauchy- und Leapfrog-Algorithmus für Anfangswertprobleme

Nachdem wir im Abschnitt 1.4.1 verschiedene Formeln für die numerische Differentiation kennengelernt haben, kann man diese natürlich auch zur Näherungslösung der DGL (2.3) heranziehen. Daraus ergeben sich verschiedene Algorithmen. Generell werden wir jetzt ein Intervall $[x_0, x]$ annehmen und auf diesem ein diskretes Netz $x_i = x_0 + i\Delta x$, $i = 0, \ldots, N$, $\Delta x = (x - x_0)/N$ definieren. Unsere DGL hat die Form $y' = F(y, x)$ und der Anfangswert sei $y(x_0) = y_0$.

Euler-Cauchy

Diesen Algorithmus haben wir bereits kennengelernt. Man erhält ihn, wenn man die Newton-Gregory-Vorwärtsableitung (1.9) einsetzt, d.h.[3]

$$F(y(x_i), x_i) = \left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y(x_{i+1}) - y(x_i)}{\Delta x} \quad .$$

Wir schreiben noch $y(x_i) \equiv y_i$ und erhalten

$$\boxed{y_{i+1} = y_i + F(y_i, x_i) \cdot \Delta x + \mathcal{O}\left[(\Delta x)^2\right] \quad ,} \qquad (2.4)$$

d.h. abgesehen von der Bezeichnungsweise dieselbe Form wie (1.5).

Leapfrog

Einen anderen Algorithmus erhält man, wenn man anstelle der Newton-Gregory-Vorwärtsableitung die Stirlingableitung (1.11) einsetzt. Der Algorithmus lautet dann

$$\boxed{y_{i+1} = y_{i-1} + 2\,F(y_i, x_i) \cdot \Delta x + \mathcal{O}\left[(\Delta x)^3\right] \quad .} \qquad (2.5)$$

Der Algorithmus benötigt für die Berechnung von y_{i+1} also die explizite Kenntnis von y_{i-1} sowie die rechte Seite $F(y_i, x_i)$ ausgewertet bei x_i. Den dazu benötigen Wert y_i muss man aus dem vorangegangenen Schritt nehmen. Dieses Zweischrittverfahren wird auch anschaulich „leapfrog“ (Bocksprung) genannt. Ein Problem bei diesem Verfahren ist, dass man keinen Wert y_{-1} oder alternativ nicht beide Werten y_0 und y_{-1} zur Verfügung hat. Ein möglicher Ausweg besteht zum beispiel darin, sich aus y_0 eine Abschätzung von y_1 aus einem Euler-Cauchy-Schritt zu beschaffen.

2.1.2 Stabilität von Algorithmen

Im Abschnitt 1.2.3 haben wir gelernt, dass der Euler-Cauchy-Algorithmus – wie jeder Algorithmus – unter Umständen instabil werden kann. Wie lässt sich diese Beobachtung quantifizieren?

[3] Wir lassen jetzt die explizite Vektorschreibweise der Bequemlichkeit halber weg.

Die folgende Stabilitätsanalyse kann man für einen beliebigen iterativen Algorithmus durchführen. Wir bezeichnen die (unbekannte) exakte Lösung am Punkt x_i mit y_i und den Fehler, den wir durch unser Verfahren – einschließlich der numerischen Ungenauigkeiten infolge der endlichen Zahlendarstellung im Computer – erhalten, mit e_i. Die von unserem Algorithmus erzeugte Näherungslösung am Punkt x_i lautet dann $\tilde{y}_i = y_i + e_i$, und die am Punkt x_{i+1} ergibt sich aus dem Euler-Cauchy-Schritt zu $y_{i+1} + e_{i+1} = y_i + e_i + F(y_i, x_i) \cdot \Delta x$. Allgemein kann man sagen, dass $y_{i+1} + e_{i+1} = T(y_i, e_i)$, mit geeigneter Abbildungsvorschrift T. Da wir den Fehler klein haben wollen, können wir $T(y_i, e_i) \approx T(y_i) + T'(y_i) \cdot e_i$ schreiben, so dass

$$e_{i+1} \approx T'(y_i) \cdot e_i$$

folgt. Offensichtlich kann man garantieren, dass der Fehler beschränkt bleibt, wenn $|T'(y_i)| \leq 1$ für alle i. Falls die DGL ein System darstellt, ist entsprechend die „Ableitung“ des Euler-Schrittes eine Matrix und die vorherige Aussage gilt für deren Eigenwerte, d.h. der betragsmäßig größte Eigenwert muss kleiner oder gleich eins sein.
Betrachten wir ein einfaches Beispiel, nämlich die DGL

$$\frac{dy}{dt} = \lambda y(t)\,, \qquad \lambda \in \mathbb{R} \ .$$

Hier gilt

$$T(y) = y + \lambda\,\Delta t\, y$$

und mithin

$$T'(y) = 1 + \lambda\,\Delta t \ .$$

Das Stabilitätskriterium lautet also $|1 + \lambda\,\Delta t| \leq 1$ und ist z.B. für $\lambda > 0$ niemals erfüllt. Andererseits stört uns das hier wenig, da das dadurch vorhergesagte exponentielle Anwachsen ja genau die Lösung für $\lambda > 0$ ist. Für $\lambda < 0$ gibt es ein $(\Delta t)_c = 2/|\lambda|$, so dass für $\Delta t \geq (\Delta t)_c$ der Euler-Cauchy-Algorithmus keine exponentiell abklingende Lösung liefert. Probieren Sie es aus!
Für unsere Populationsdynamik ist

$$T'(n_i) = 1 + r_0\Delta t - 2Kr_0n_i\Delta t$$

und somit unser Kriterium

$$|T'(n_i)| = |1 + r_0\Delta t - 2Kr_0n_i\Delta t| = 4\mu\,|1 - 2x_i| \leq 1 \ .$$

Dabei haben wir die Definitionen von Gleichung (1.6) eingesetzt. Mit ein wenig Manipulation der exakten Lösung (1.3) findet unter Beachtung von $4\mu > 1$ für $i \to \infty$

$$x_\infty = \frac{4\mu - 1}{4\mu}$$

und das Stabilitätskriterium lautet $1 \geq 2\,|2\mu - 1|$ bzw. $4\mu \leq 3$, wie es auch die numerische Lösung zeigt.
Beachten Sie, dass wir hier keine Aussage über die *Genauigkeit* des Algorithmus erhalten haben, sondern nur über das asymptotische Verhalten wenn $t \to \infty$. Für endliche Zeiten kann der Algorithmus durchaus noch Instabilitäten wie Oszillationen zeigen (was er ja auch tut).
Bei der Schwingungsgleichung als letztes Beispiel hat $T(y)$ die Form

$$T(y) = [1 + L\Delta t] \cdot y$$

und mithin

$$T'(y) = 1 + L\Delta t \ ,$$

wobei

$$L = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -\Gamma \end{pmatrix} \ .$$

Wir brauchen die Beträge der Eigenwerte von $T'(y)$. Die Eigenwerte selber ergeben sich zu

$$l_{1,2} = 1 - \frac{\Gamma\Delta t}{2} \pm \frac{1}{2}\sqrt{(\Gamma\Delta t)^2 - (2\omega_0\Delta t)^2} \ .$$

Für die Beträge müssen wir eine Fallunterscheidung treffen, nämlich ob $\Gamma < 2\omega_0$ oder nicht. Dadurch erhalten wir

$$|l_{1,2}| = \begin{cases} \sqrt{1 - \Gamma\Delta t + (\omega_0\Delta t)^2} \text{ für } \Gamma < 2\omega_0 \\ \left|1 - \frac{\Gamma\Delta t}{2} \pm \frac{1}{2}\sqrt{(\Gamma\Delta t)^2 - (2\omega_0\Delta t)^2}\right| \text{ für } \Gamma \geq 2\omega_0 \end{cases}$$

Im ersten Fall ist der Algorithmus stabil für $\omega_0^2\Delta t < \Gamma < 2\omega_0$. Insbesondere muss man immer $\Delta t < 2/\omega_0$ wählen. Im anderen Fall erhalten wir die Bedingung $2\Gamma\Delta t < 4 + (\omega_0\Delta t)^2$.

2.1.3 Runge-Kutta-Verfahren

Runge-Kutta Algorithmen, insbesondere der Algorithmus 4. Ordnung haben sich in der Praxis als robust und vielseitig verwendbar erwiesen. Wenn man „mal schnell“ eine ODE, über die nichts Nachteiliges bekannt ist, integrieren will, ist dieser Algorithmus immer zu empfehlen. Erst wenn er scheitert, sei es wegen Gemeinheiten in der Struktur der ODE oder aus Zeitgründen (eigentlich fast unvorstellbar bei einer einzigen ODE) muss man sich tiefe Gedanken über einen speziellen, auf die ODE zugeschnittenen Algorithmus machen.
Zur Illustration der Runge-Kutta Verfahren betrachten wir das Runge-Kutta Verfahren 2. Ordnung. Die Idee ist es nun, einen Euler(-Cauchy)-Schritt $x_i \to x_i + \Delta x$

mit zwei aufeinanderfolenden Euler-Cauchy-Schritten $x_i \to x_i + \alpha\,\Delta x \to x_i + \Delta x$ zu kombinieren und durch Anpassung der Parameter die Ordnung des Verfahrens zu verbessern. In diesem Sinn machen wir zunächst den Ansatz

$$\begin{aligned} k_1 &= \Delta x \cdot F(y_i, x_i) \\ y_{i+1} &= y_i + \Delta x\,\left[w_1\,F\left(y_i, x_i\right) + w_2\,F\left(y_i + \alpha\,k_1, x_i + \alpha\,\Delta x\right)\right] \end{aligned} \tag{2.6}$$

mit freien Parametern α, w_1 und w_2. Dieser Ansatz soll nun die Taylor-Entwicklung für $y(x)$ bis zur zweiten Ordnung reproduzieren:

$$\begin{aligned} y(x+\Delta x) - y(x) &= \Delta x\,\frac{\mathrm{d}\,y}{\mathrm{d}x} + \frac{\Delta x^2}{2}\,\frac{\mathrm{d}^2\,y}{\mathrm{d}x^2} + \mathcal{O}(\Delta x^3) \\ &= \Delta x\,F + \frac{\Delta x^2}{2}\,\frac{\mathrm{d}\,F}{\mathrm{d}x} + \mathcal{O}(\Delta x^3)\,. \end{aligned} \tag{2.7}$$

Hierbei ist

$$\frac{\mathrm{d}\,F}{\mathrm{d}x} = \frac{\partial\,F}{\partial x} + \frac{\partial\,F}{\partial y}\,\frac{\mathrm{d}\,y}{\mathrm{d}x} = \frac{\partial\,F}{\partial x} + \frac{\partial\,F}{\partial y}\,F\,, \tag{2.8}$$

wobei wir im zweiten Schritt die Differentialgleichung (2.3) eingesetzt haben. Andererseits lautet die Taylor-Entwicklung von (2.6)

$$y_{i+1} - y_i = w_1\,\Delta x\,F + w_2\,\Delta x\,F + w_2\,\Delta x^2\,\alpha\,\left(\frac{\partial\,F}{\partial x} + \frac{\partial\,F}{\partial y}\,F\right) + \mathcal{O}(\Delta x^3)\,, \tag{2.9}$$

mit $F = F(y_i, x_i)$.
Durch Vergleich der Koeffizienten von (2.7) und (2.9) folgt

$$w_1 + w_2 = 1\,, \qquad w_2\,\alpha = \frac{1}{2}\,. \tag{2.10}$$

Diese Bedingungen besitzen eine einparametrige Schar von Lösungen, unter denen keine ausgezeichnet ist. Eine einfache Wahl ist

$$\alpha = \frac{1}{2}\,, \qquad w_2 = 1\,, \qquad w_1 = 0\,. \tag{2.11}$$

Wir erhalten damit das Runge-Kutta-Verfahren 2. Ordnung, das bis zur Ordnung Δx^2 genau ist:

$$\boxed{\begin{aligned} k_1 &= \Delta x \cdot F(y_i, x_i) \\ y_{i+1} &= y_i + \Delta x\,F\left(y_i + \frac{k_1}{2}, x_i + \frac{\Delta x}{2}\right) + \mathcal{O}(\Delta x^3) \end{aligned}} \tag{2.12}$$

Das Verfahren ist schematisch in Abbildung 2.1 gezeigt. Mit Hilfe des Inkrements k_1 schätzt man zunächst y in der Intervallmitte $x_i + \Delta x/2$ ab. Dies liefert dann eine Schätzung der Steigung in der Mitte des Intervalls, die man verwendet, um den

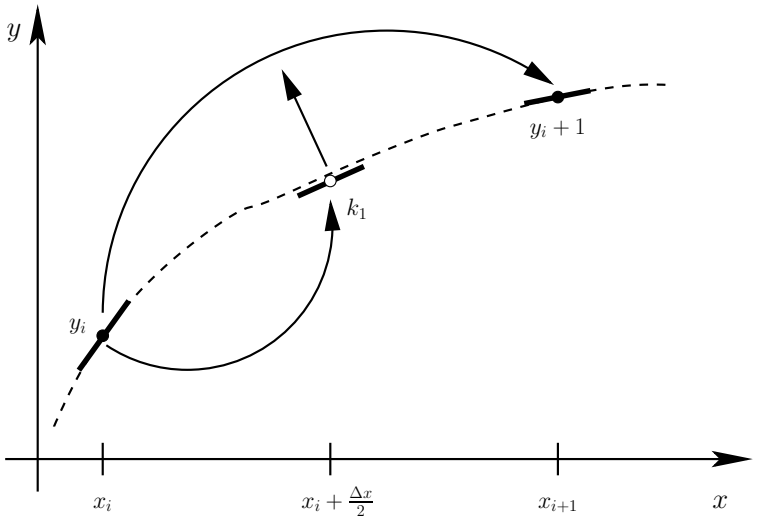


Abbildung 2.1: Schematische Darstellung des Runge-Kutta-Verfahrens 2. Ordnung

Schritt von x_i nach x_{i+1} auszuführen. Das resultierende Verfahren ist eine Ordnung in Δx besser als der Euler-Cauchy-Algorithmus.
Die Verallgemeinerung dieser Idee ist offensichtlich. Man berechnet eine „repräsentative Steigung“ im Einzelschritt

$$y_{i+1} = y_i + a_1\, k_1 + a_2\, k_2 + \cdots + a_n\, k_n \quad ,$$

wobei

$$k_1 = \Delta x\, F(y_i, x_i)$$

ist und die k_i für $i > 1$ weitere geeignete Funktionswerte von F sind,

$$\begin{aligned} k_2 &= \Delta x \cdot F\left(y_i + c_{1,1}\, k_1, x_i + \alpha_1\, \Delta x\right) \\ k_3 &= \Delta x \cdot F\left(y_i + c_{2,1}\, k_1 + c_{2,2} k_2, x_i + \alpha_2\, \Delta x\right) \\ &\vdots \\ k_n &= \Delta x \cdot F\left(y_i + \sum_{j=1}^{n-1} c_{n-1,j}\, k_j, x_i + \alpha_{n-1}\, \Delta x\right) . \end{aligned}$$

Das Entscheidende ist nun, dass man die Konstanten a_i, α_i und $c_{i,j}$ durch Vergleiche mit Taylorentwicklung von F so bestimmt, dass die Fehlerordnung des Verfahrens möglichst hoch ist. Man beachte aber, dass die Wahl dieser Konstanten nicht eindeutig ist und man daher noch weitere, nützliche Bedingungen berücksichtigen kann.
Das bekannteste dieser Verfahren ist der Runge-Kutta-Algorithmus 4. Ordnung. Der Algorithmus sieht wie folgt aus:

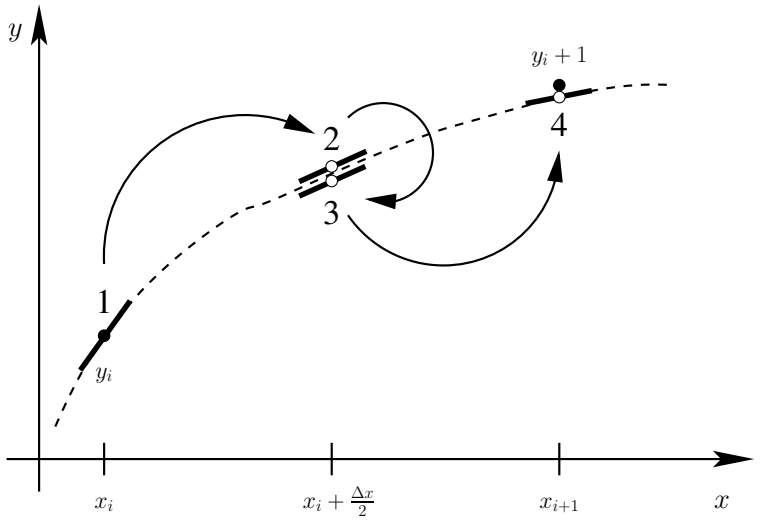


Abbildung 2.2: Schematische Darstellung des Runge-Kutta-Verfahrens 4. Ordnung

$$
\begin{aligned}
y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(\Delta x^5) \\
k_1 &= \Delta x \cdot F(y_i, x_i) \\
k_2 &= \Delta x \cdot F\left(y_i + \frac{k_1}{2}, x_i + \frac{1}{2}\Delta x\right) \\
k_3 &= \Delta x \cdot F\left(y_i + \frac{k_2}{2}, x_i + \frac{1}{2}\Delta x\right) \\
k_4 &= \Delta x \cdot F\left(y_i + k_3, x_i + \Delta x\right) . \qquad (2.13)
\end{aligned}
$$

Dieses Verfahren ist schematisch in Abbildung 2.2 gezeigt. Mit Hilfe des Inkrements k_1 schätzt man zunächst y in der Intervallmitte $x_i + \Delta x/2$ ab. Dies liefert genau wie beim Runge-Kutta-Verfahren 2. Ordnung eine Schätzung der Steigung in der Mitte des Intervalls. Diese verwendet man nun jedoch, um den Schritt von x_i nach $x_i + \Delta x/2$ zu wiederholen. Dies liefert eine verbesserte Schätzung der Steigung in der Intervallmitte, die man verwendet, um den Schritt von x_i nach x_{i+1} auszuführen. Zuletzt führt man noch einen Schritt am Punkt x_{i+1} aus.

Man braucht im Runge-Kutta-Verfahren 4. Ordnung offensichtlich vier Berechnungen der Funktion F pro Zeitschritt, während der Algorithmus 2. Ordnung mit zwei auskommt. Damit die 4. Ordnung also effizienter wird, muss man (bei gleicher Genauigkeit) die Schrittweite verdoppeln können. Geht das? Die Antwort ist: Oftmals ja, aber eine Garantie gibt es nicht.

Hier eine minimale Implementierung einer Runge-Kutta Routine 2. Ordnung in C++ für die Zerfallsgleichung $y' = -2y$. Anfangszeit, Endzeit, Schrittweite und

Anfangsbedingung werden im Programm festgelegt (nicht gerade empfehlenswert für numerische Untersuchungen, bei denen man diese Werte oft verändern muss). Die rk2-Routine führt einen RK2-Schritt mit dem Anfangswert y und der Schrittweite dx aus.

```
// Runge-Kutta-Verfahren 2. Ordnung fuer DGL y' = -2y
#include <iostream>
#include <fstream>
using namespace std;
// Prototypen des rk2 Schritts und der rechten Seite der ODE
void rk2(double x, double &y, double dx);
double F(double y , double x);
//
int main()
{
  ofstream aus("rk2.dat");
  double x = 0.0, dx = 5, xmax=100.0;
  double y = 0.1;
  aus << x << "\t" << y << endl;
// Start der eigentlichen Berechnungs-Schleife
   while (x<=xmax) {
    rk2(x,y,dx);
    x += dx;
    aus << x << "\t" << y << endl;
  }
// Ende der Berechnungsschleife
  cout << "data stored in rk2.dat" << endl;
  aus.close();
}
// Start der rk2 Prozedur
void rk2(double x, double &y, double dx )
{
  double k1;
  k1=dx*F(y,x);
  y += dx*F(y+0.5*k1,x+0.5*dx);
}
// Rechte Seite der DGL
double F(double y, double x)
{
  return -2.0*y;
}
```

Das Programm wird wie üblich mit `g++ -O -o rk2 rk2.cc` übersetzt. Die Ergebnisse für die DGL $y' = -2y$ mit der Anfangsbedingung $y(0) = 10$ für zwei Werte

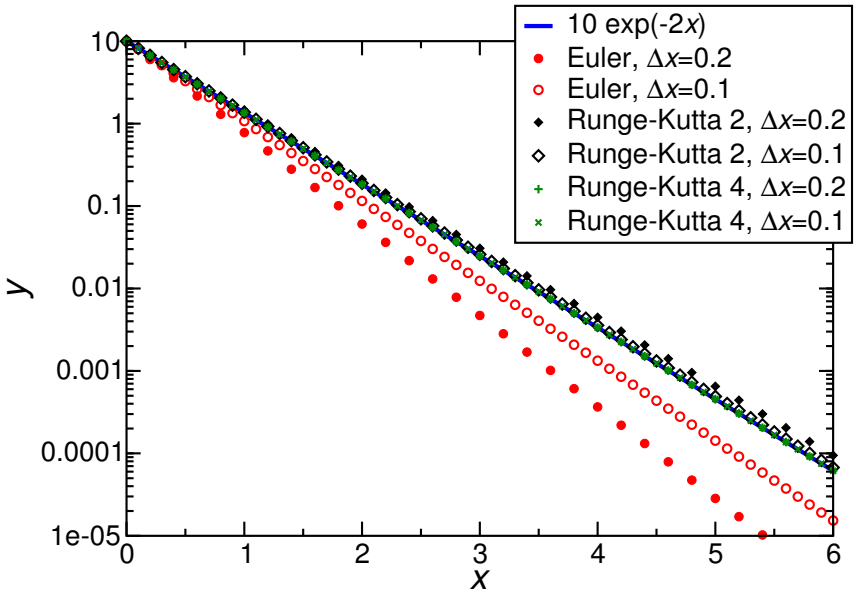


Abbildung 2.3: Vergleich von Euler-Cauchy, Runge-Kutta 2. Ordnung und Runge-Kutta 4. Ordnung.

von Δx finden Sie in der Abbildung 2.3, die außerdem auch einen Vergleich mit dem Euler-Cauchy- und dem Runge-Kutta-Algorithmus 4. Ordnung enthält. Man sieht deutlich, dass der Runge-Kutta-Algorithmus 2. Ordnung bei gleicher Schrittweite genauer als das Euler-Cauchy-Verfahren ist. Den Preis, den man zu zahlen hat, sind zwei Aufrufe der Funktion F anstelle von einem. Auch der Runge-Kutta-Algorithmus 4. Ordnung ist für $\Delta x = 0.2$ genauer als der Algorithmus 2. Ordnung mit $\Delta x = 0.1$, d.h. man erhält bei vergleichbarem Aufwand bessere Ergebnisse. Das spielt für die Anwendung auf kleine Systeme von ODEs keine wesentliche Rolle, kann aber zu einem Problem bei Simulationen von Vielteilchensystemen („Molekulardynamik“) werden, wo der DGL-Löser nicht nur viele hunderttausend Mal aufgerufen wird, sondern auch die Berechnung von F ggfs. teuer ist. Hier muss man sich dann einen „billigeren“ und für die Anwendung stabilen Algorithmus überlegen.

2.1.4 Schrittweitenanpassung und Fehlerkontrolle

Einerseits wollen wir den Verfahrensfehler abschätzen, den wir durch die endliche Schrittweite Δx induzieren (*Diskretisierungsfehler*). Andererseits gibt es Probleme, wie z.B. der Flug einer Rakete zum Mond, in der sowohl Phasen mit schnellen Änderungen (Raketenstart, Einschwenken in Mondumlaufbahn, ...) solchen mit einer recht übersichtlichen Zeitentwicklung (z.B. Flugphase der Rakete von der Erde zum Mond) gegenüberstehen. In solchen Fällen ist es sinnvoll, nicht das gesamte

Problem mit einer so kleinen Schrittweite Δx zu rechnen, dass zu allen Zeiten die gewünschte Genauigkeit erreicht wird, sondern eine Anpassung der Schrittweite im Verlauf des Verfahrens vorzunehmen.
Am elegantesten ist eine adaptive Steuerung der Schrittweite, die gleichzeitig auch eine Abschätzung des Fehlers erlaubt. Eine Möglichkeit, den Fehler zu kontrollieren erhält man, wenn man jeden Schritt doppelt ausführt, und zwar einmal direkt und einmal als zwei halbe Schritte[4]:

$$\begin{aligned} y_i \quad &\rightarrow \quad y_{i+1} =: Y_1 \,, \\ y_i \quad &\rightarrow \quad y\left(x_i + \frac{\Delta x}{2}\right) \quad \rightarrow \quad y_{i+1} =: Y_2 \,. \end{aligned} \tag{2.14}$$

Für ein Verfahren $m$ter Ordnung gilt nun

$$\begin{aligned} y\left(x_i + \Delta x\right) &= Y_1 + c\,\Delta x^{m+1} + \mathcal{O}\left(\Delta x^{m+2}\right) \\ &= Y_2 + 2\,c\left(\frac{\Delta x}{2}\right)^{m+1} + \mathcal{O}\left(\Delta x^{m+2}\right) \end{aligned} \tag{2.15}$$

(mit $c = \frac{1}{(m+1)!}\frac{\mathrm{d}^{m+1}y}{\mathrm{d}x^{m+1}}$). Der genaue Ausdruck für c spielt keine Rolle, entscheidend ist nur, dass in beiden Zeilen von (2.15) das gleiche c auftritt. Wir haben also

$$Y_2 - Y_1 \approx c\,\Delta x^{m+1}\left(1 - 2^{-m}\right) \,. \tag{2.16}$$

Wir können nun eine Schrittweite $\widetilde{\Delta x}$ so bestimmen, dass der Fehler mit dieser Schrittweite in einem Schritt nach (2.14) einen vorgegebenen Wert δ_0 annimmt

$$\left|\tilde{Y}_2 - \tilde{Y}_1\right| = \delta_0 \,. \tag{2.17}$$

Aus (2.16) folgt

$$\begin{aligned} \delta_0 &= |c|\,\widetilde{\Delta x}^{m+1}\left(1 - 2^{-m}\right) \,, \\ |Y_2 - Y_1| &= |c|\,\Delta x^{m+1}\left(1 - 2^{-m}\right) \,. \end{aligned} \tag{2.18}$$

Nach Division der beiden Gleichungen fallen die (unbekannten) Konstanten heraus:

$$\frac{\delta_0}{|Y_2 - Y_1|} = \left(\frac{\widetilde{\Delta x}}{\Delta x}\right)^{m+1} \,. \tag{2.19}$$

Dies kann nach $\widetilde{\Delta x}$ aufgelöst werden:

$$\boxed{\widetilde{\Delta x} = \Delta x \sqrt[m+1]{\frac{\delta_0}{|Y_2 - Y_1|}} \,.} \tag{2.20}$$

[4] Der einfache und doppelte Schritt teilen zumindest den Anfangspunkt. Durch Wiederverwenden solchen Überlapps kann der zusätzliche Rechenaufwand z.B. bei Runge-Kutta-Verfahren auf weniger als 50% reduziert werden.

War der Fehler größer als gewünscht, muß der letzte Schritt mit der Schrittweite (2.20) *wiederholt* werden. Ist der Fehler geringer als gefordert, kann im *nächsten* Schritt die größere Schrittweite (2.20) verwendet werden[5].

[5] Es empfiehlt sich, die Schrittweite Δx grundsätzlich etwas kleiner zu wählen, als die „genaue“ Formel nahelegt.

Kapitel 3

Anwendungen Teil I: Klassische Mechanik

Wir wollen nun das Gelernte auf Probleme der Mechanik anwenden. Mit Hilfe einer numerischen Lösung der Newtonschen Bewegungsgleichungen (2.1) kann man viele Probleme behandeln, die anders nicht oder nur mit großem Aufwand zugänglich sind. Eine Problemklasse sind Reibungskräfte, die von der Geschwindigkeit abhängig sind. Für ausgedehnte Körper gehört hierzu z.B. die Magnus-Kraft, die z.B. beim Fußball eine Rolle spielt (siehe z.B. Kapitel 3 von [1]).
Wir wollen uns hier vor allem mit Mehrkörperproblemen beschäftigen. Bekanntlich lässt sich bei Zentralkräften das Zweikörperproblem zumindest formal lösen (eine explizite Lösung findet man jedoch nur für ganz ausgezeichnete Potentiale). Selbst eine solche Lösung ist ab $N \geq 3$ nicht mehr verfügbar. Tatsächlich kann bereits bei drei Körpern entsprechend komplexes Verhalten auftreten. Man beachte, dass das System Sonne-Erde-Mond ein solches Dreikörperproblem darstellt. Eine analoge Komplexität tritt selbstverständlich auch für Mehrkörperprobleme wie z.B. das komplette Sonnensystem auf. Erfahrungsgemäß setzen sich die Bewegungen in unserem Sonnensystem allerdings in guter Näherung aus Zweikörperbewegungen zusammen. Für Simulationen solcher astronomischer Probleme, die Sie in den Übungen durchführen sollen, ist es daher nützlich, zunächst einige grundlegende Fakten zu Zweikörper-Zentralproblemen zusammenzufassen.

3.1 Kepler-Probleme

Wir betrachten zunächst ein Zentralproblem in dem Gravitationspotential

$$V(r) = -\frac{G\,M\,m}{r}\,, \tag{3.1}$$

wobei M die Masse des Zentralkörpers ist. Aus dem Potential (3.1) ergibt sich eine Kraft

$$\vec{F}(\vec{r}) = -\frac{G\,M\,m}{r^3}\,\vec{r} \tag{3.2}$$

mit $r = |\vec{r}| = \sqrt{\vec{r}\cdot\vec{r}}$.
Für Energie $E < 0$ erhalten wir gebundene Bahnen. In dem Potential (3.1) handelt es sich hierbei um geschlossene Kurven, und zwar um Ellipsen. Wir fassen nun kurz einige bekannte Eigenschaften elliptischer Bahnen zusammen, die bei der Simulation von Kepler-Problemen nützlich sind. Dazu betrachten wir die in Abb. 3.1 dargestellte Ellipse. B_1 und B_2 sind die beiden Brennpunkte, a die große Halbachse und b die kleine Halbachse. $\epsilon \geq 0$ ist die sogenannte Exzentrizität. Für $\epsilon = 0$ findet man $a = b$ und die beiden Brennpunkte fallen zusammen, so dass sich ein Kreis ergibt. Eine mögliche Charakterisierung einer Ellipse ist, dass die Summe der Abstände von den beiden Brennpunkten B_1 und B_2 zu einem beliebigen Punkt auf ihr konstant ist. Mit den Bezeichnungen in Abb. 3.1 kann man daraus folgern, dass

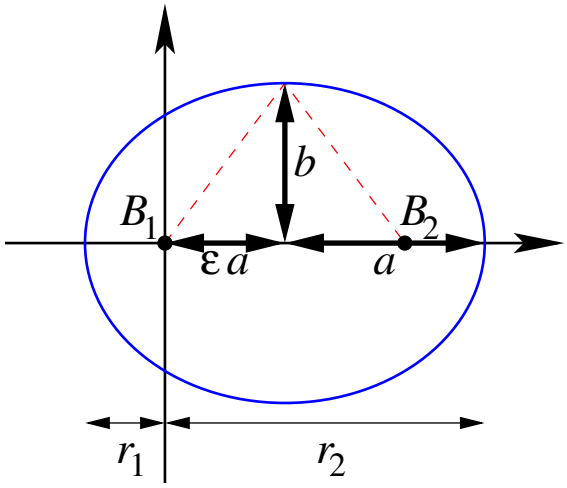


Abbildung 3.1: Eine Ellipse. Die Bezeichnungen sind im Text erläutert.

$$b = a\,\sqrt{1-\epsilon^2}\,. \tag{3.3}$$

Aus dieser Formel folgert man weiter, dass $\epsilon < 1$ für eine eine Ellipse.
Das Kraftzentrum (und damit auch der Koordinatenursprung $r = 0$) befindet sich in dem Brennpunkt, den wir B_1 genannt haben. Die Orte mit den minimalen und maximalen Bahnradien (r_1 bzw. r_2) werden Apsiden genannt. Aus obiger Skizze liest man ab:

$$r_1 = (1-\epsilon)\,a\,, \qquad r_2 = (1+\epsilon)\,a\,. \tag{3.4}$$

Dies ist übrigens auch konsistent mit der Definition der großen Halbachse $a = (r_1 + r_2)/2$.
Aus der Diskussion des Zentralpotentials (3.1) in der Physik I oder aber spätestens in der Analytischen Mechanik ist bekannt, dass

$$a = \frac{r_1 + r_2}{2} = -\frac{G\,M\,m}{2\,E}\,, \tag{3.5}$$

mit der Gesamtenergie

$$E = \frac{1}{2}\,m\,v^2 + V(r)\,. \tag{3.6}$$

Diese Definition von E kann man nun nach $v(r)$ auflösen. Man findet mit Hilfe von (3.1) und (3.5)

$$\boxed{v(r) = \sqrt{G\,M\,\left(\frac{2}{r} - \frac{1}{a}\right)}\,.} \tag{3.7}$$

Für eine Kreisbahn ist $r = a$. In diesem Fall reduziert sich (3.7) auf

$$v = \sqrt{\frac{G\,M}{r}}\,. \tag{3.8}$$

Letzteres Ergebnis folgt auch elementar durch Betrachtung des Gleichgewichts von „Fliehkraft“ und Gravitationskraft (3.2).
Da man außer dem Betrag der Geschwindigkeit auch ihre Richtung kennen muß, empfiehlt es sich, (3.7) auf Werte von r anzuwenden bei denen $\mathrm{d}\,r/\mathrm{d}t$ verschwindet, was $\vec{v} \perp \vec{r}$ bedeutet. Dies ist für eine Kreisbahn immer erfüllt, so dass für gegebenes r die zu einer Kreisbahn führende Geschwindigkeit aus (3.8) bestimmt werden kann. Im Fall einer Ellipse gilt $\vec{v} \perp \vec{r}$ insbesondere für die beiden Apsiden r_1 und r_2. Für eine gegebene elliptische Bahn kann man die Geschwindigkeit $\vec{v}$ daher mit Hilfe von (3.7) an den Apsiden besonders einfach bestimmen.
Das Potential (3.1) gilt auch für die relative Bewegung zweier Körper der Massen m und M. In diesem Fall ist $\vec{r}$ als Differenz der Ortsvektoren der beiden Körper zu interpretieren. Dies läßt sich leicht auf die gravitative Wechselwirkung von N Körpern der Massen m_i verallgemeinern:

$$V(\vec{r}_1, \ldots, \vec{r}_N) = -G \sum_{i<j} \frac{m_i\, m_j}{|\vec{r}_i - \vec{r}_j|}\,. \tag{3.9}$$

Hieraus folgen die Bewegungsgleichungen

$$\ddot{\vec{r}}_i = -G \sum_{\substack{j=1 \\ j\neq i}}^{N} m_j\, \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3}\,. \tag{3.10}$$

Bei der Simulation ist zu beachten, dass die schnellste Dynamik aufgelöst werden muß. Die Schrittweitenanpassung aus Kapitel 2.1.4 hilft hier leider nicht weiter, da die schnellen und langsamen Dynamiken im vorliegenden Fall parallel ablaufen. Im Fall des Sonnensystems ist die kürzeste Zeitskala z.B. durch den Merkur mit einer Umlaufteit von ca. 116 Tagen gegeben; bei Hinzunahme des Erdmonds reduziert sich die Zeitskala auf 27 Tage. Dies ist zu verhleichen mit der Umlaufdauer des Pluto von 248 Jahren (bzw. des Neptun von 165 Jahren, falls man Pluto nicht als Planet betrachten will). Man kann sich nun am Beispiel einer Kreisbahn in einem Zentralpotential überzeugen, dass as Euler-Cauchy-Verfahren selbst mit 1000 Schritten pro Umlauf keine stabile Bahn liefert. Beim Runge-Kutta-Verfahren 2. Ordnung kann man mit ca. 200 Schritten pro Umlauf eine stabile Kreisbewegung simulieren, beim Runge-Kutta-Verfahren 4. Ordnung erhält man mit ca. 50 Schritten pro Umlauf eine gute Qualität. Für Anwendungen wie die Planetenbewegungen sind somit die Algorithmen höherer Ordnung zu empfehlen.

3.1.1 Einheiten im Sonnensystem

Die bisherigen Bemerkungen gelten recht allgemein für ein Zweikörper-Zentralproblem, so z.B. auch das System Erde-Mond.
Für eine numerische Lösung empfiehlt es sich grundsätzlich, an das Problem angepaßte Einheiten zu verwenden. Bei einem Zentralkraftproblem sollten diese so gewählt werden, dass der Zahlenwert von $G\,M$ in der Nähe von 1 liegt. Zur Beschreibung des Sonnensystems verwendet man als Längeneinheit die astronomische Einheit (AU), die gleich der großen Halbachse der Erdumlaufbahn (d.h. im wesentlichen deren mittleren Radius) gesetzt wird:

$$1\,AU = 1.496 \cdot 10^{11}\mathrm{m}\,. \tag{3.11}$$

Als Zeiteinheit verwendet man das Jahr, $1\,yr \approx 3.15 \cdot 10^7$s, d.h. die Umlaufdauer der Erde, die in guter Näherung auf einer Kreisbahn umläuft. Nun gilt für die Umlaufdauer $T = \frac{2\pi\,r}{v}$ auf einer Kreisbahn mit Radius r nach (3.8)

$$T^2 = \frac{4\pi^2}{G\,M}\,r^3\,. \tag{3.12}$$

Diese Beziehung ist die Aussage des 3. Keplerschen Gesetzes für den speziellen Fall einer Kreisbahn. Die Beziehung (3.12) kann man leicht nach dem Produkt von Gravitationskonstante und Sonnenmasse auflösen, und erhält in astronomischen Einheiten ($r = 1\,AU$, $T = 1\,yr$) den folgenden Wert:

$$G\,M = 4\pi^2\,\frac{AU^3}{yr^2}\,. \tag{3.13}$$

3.2 Visualisierung

Die Darstellung der Ergebnisse mit einem Plotprogramm wie `gnuplot` oder `xmgrace` ist zwar nützlich, und mit Java kann man recht einfach Animationen realisieren. Aber manchmal wünscht man sich doch eine anspruchsvollere Grafik. Für solche Zwecke gibt es einige Tools, z.B. kann man die grafischen Fähigkeiten von `maple` oder `MatLab` ausnutzen. Wer darauf keinen Zugriff hat, sollte mal bei `gnuplot` auf die dreidimensionalen Darstellungsmöglichkeiten schauen. Hier wollen wir ein anderes Programm kennen lernen, nämlich `povray` („Persistence of Vision Raytracer“), einem sogenannten „Raytracer“. Mit `povray` können Sie geometrische Objekte erzeugen und räumlich anordnen, mit Oberflächenstrukturen belegen und mit verschiedenen Lichtquellen bescheinen. Das Ergebnis ist ein ziemlich realistisch aussehendes Szenario. Und das Schöne ist: `povray` ist kostenlos für Linux, Windows Mac OS X, etc. verfügbar.

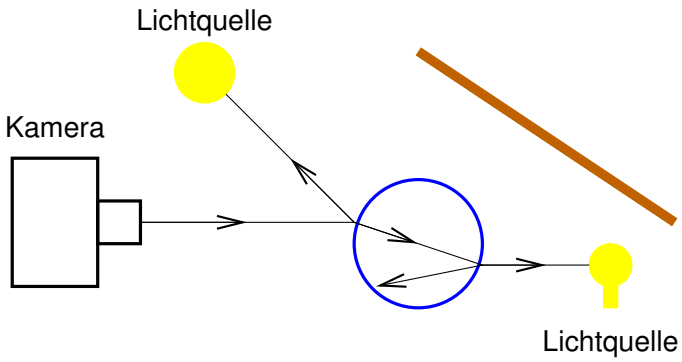


Abbildung 3.2: Das Grundprinzip eines Raytracers: Szene mit Kamera und Lichtquellen.

Die Grundidee stammt aus der geometrischen Optik und ist in Abb. 3.2 skizziert. Man definiert eine Szene, die aus verschiedenen Objekten mit unterschiedlichen Oberflächenbeschaffenheiten besteht. Hinzu kommen eine Kamera und eine oder mehrere Lichtquelle(n). `povray` verfolgt dann pixelweise die Lichtstrahlen von der Kamera zurück zu den Lichtquellen und fertig ist ein realistisches Bild.
Zur Erzeugung solcher Bilder muß man (wie bei `gnuplot` eigentlich auch) allerdings die „Sprache“ von `povray` lernen. Die ist jedoch nicht sehr kompliziert. Man muß nur an einer Stelle etwas aufpassen: In `povray` wird mit einem *linkshändigen* Koordinatensystem gearbeitet, wobei die x-Achse nach rechts zeigt, die y-Achse nach oben und die z-Achse nach hinten.
Eine ganz nette Webseite, mit der man einen kleinen Eindruck über die Möglichkeiten von `povray` gewinnen kann, ist

`http://www.f-lohmueller.de/pov_tut/pov__ger.htm`

Diese Seite stellt auch einige Beispiele und Dokumentation der Syntax zur Verfügung. Einige wichtige Sprachelemente von `povray` sind:

- Kamera: `camera { location` <Standort> `look_at` <Blickrichtung> `}`
- Lichtquelle: `light_source { <Ort> color <Farbe> }`
- Ebene: `plane { <Normale>, Abstand Eigenschaften }`
- Quader: `box { <Ecke1>, <Ecke2> Eigenschaften }`
  Die Konventionen für die Ecke-Koordinaten sind in Abb. 3.3 illustriert.
- Kugel: `sphere { <Mittelpunkt>, Radius, Eigenschaften }`

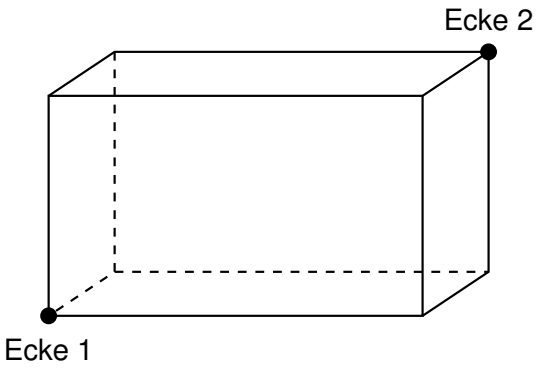


Abbildung 3.3: Definition eines Quaders („box“) in povray.

sowie weitere Objekte analoger Typen (wie z.B. Zylinder). Zu den Eigenschaften der Oberfläche gehören die Farbe („pigment“) sowie eine Modulation senkrecht zu der Oberfläche („normal“). Hinzu kommen Makros, mit deren Hilfe zahlreiche komplexere Objekte definiert werden können bzw. vordefiniert sind sowie Direktiven, ähnlich wie auch in anderen Sprachen. Nützlich ist auch die clock-Variable, die povray für animierte Szenen verwendet. Die Ausgabe besteht aus einer Bild-Datei (im Fall einer Szene) bzw. mehreren Bilddateien, die zu einer Animation zusammengefügt werden können.
Nach den vorhergehenden Kapiteln interessieren uns insbesondere Planeten-Szenen. Eine vorgefertigte eine einfache Sonne-Erde-Mond-Konfiguration finden Sie unter

http://www.f-lohmueller.de/pov_tut/animate/anim112d.htm

bzw. mit kleinen Modifikationen in Stud.IP. Die Szenen-Beschreibung, d.h. Kamera-Position, Lichtquelle und Objekte befindet sich in der Datei planet_00ani.pov. Das Kommando

```
povray planet_00ani.pov
```

erzeugt eine Bilddatei mit der Szene.
Hinzu kommt noch die Steuerdatei planet_00ani.ini. Hier ist insbesondere der Ablauf der clock-Variablen definiert und zwar so, dass diese in 120 Schritten von 0 bis 1 läuft. In dieser Datei kann man auch mittels der Variablen Width und Height die Bildgröße in Pixeln. definieren. Das Kommando

```
povray planet_00ani.ini
```

erzeugt insgesamt 120 Bilddateien. Die Szene 76 (Datei planet_00ani076.png) ist in Abbildung 3.4 gezeigt. Offensichtlich erhält man mit vergleichsweise geringem Aufwand realistische Effekte, wie z.B. Schatten auf den der Sonne abgwandten Seiten von Erde und Mond sowie einen Schattenwurf von dem Mond auf die Erde. Ist

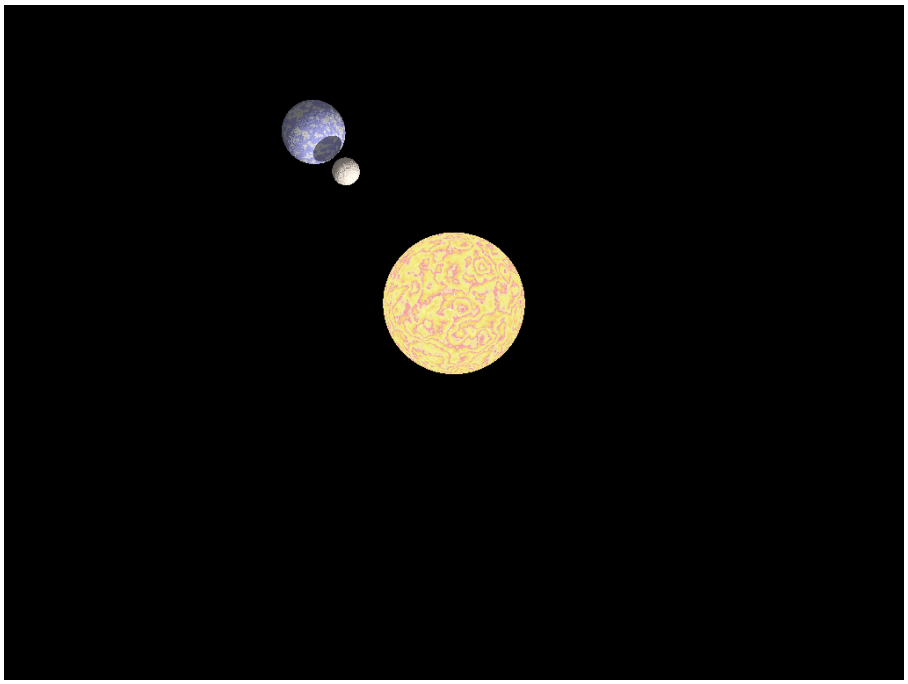


Abbildung 3.4: Mit povray gerenderte Szene bestehend aus Sonne, Erde und Mond.

man hiermit nicht zufrieden[1], kann man aus den 120 Szenen mit dem convert-Kommando aus dem imagemagick-Programmpaket mit Hilfe des folgendes Kommandos ein kleines Video erzeugen:

```
convert -delay 10 -loop 0 planet_00ani???.png Planet.mpg
```

Die delay-Option definiert den Abstand zwischen Bildern und -loop 0 erzeugt eine Endlosschleife, d.h. der Ablauf des Umlaufs wird nach dem Ende jeweils wiederholt (funktioniert leider mit MPEG nicht). Die Erweiterung „.mpg“ an Planet.mpg spezifiziert, dass wir ein MPEG-Video erzeugen möchten. Eine Erweiterung „.gif“ würde hingegen eine animierte Gif-Datei liefern, die sich z.B. gut auf Web-Seiten einsetzen läßt.

Als ein weiteres Beispiel wollen wir die Visualisierung der Flugbahn eines Fußballs diskutieren. Eine vollständige Diskussion dieses Problems einschließlich eines Programms für die Simulation finden Sie in Kapitel 3 von [1]. Hier wollen wir uns primär die Definition der Szene mit Rasen, Tor und Ball in der Datei fussball.pov ansehen:

```
// Persistence Of Vision raytracer: Fussball
```

[1] Realistische Oberflächen von Mond, Erde und Planeten findet man im Internet. Gute Ausgangspunkte sind z.B. http://earthview.sourceforge.net/ oder http://www.midnightkite.com/render.html.

```
// Ein paar nuetzliche vordefinierte Dinge
#include "colors.inc"
#include "textures.inc"

// Kamera = Standpunkt des Beobachters
// <.,.,.> ist ein Vektor im linkshaendigen Koordinatensystem
// Achtung: y-Achse zeigt nach oben!!
camera {location <-3.00 , 1.5 , 0.5>
             look_at  <7 , 0, 0.0>}
// Lichtquelle: Position und Lichtfarbe (aus colors.inc)
light_source{<-15000,25000,5000> color White}
//

// Der Boden des Fussballplatzes ist ein gruener Rasen mit ein
// wenig Textur, dargestellt durch "bumps"
plane { y, 0
           texture {
                 pigment {color Green }
                 normal  {bumps 0.75 scale 0.025 }
                   }
} // end of plane

// Das Tor, 21m weg vom Abschusspunkt:
// Linker Pfostem
box {<21,0,-3.91>,<21.25,2.44,-3.66>
             texture {
                   pigment {color White}
                   finish { brilliance 0.0 phong 0.0}
                      }
}
// Rechter Pfosten
box {<21,0,3.91>,<21.25,2.44,3.66>
            texture {
                  pigment {color White}
                  finish { brilliance 0.0 phong 0.0}
                     }
}
// Latte
```

```
box {<21,2.44,-3.91>,<21.25,2.69,3.91>
            texture {
                  pigment {color White}
                  finish { brilliance 0.0 phong 0.0}
                    }
}

// Der Himmel
sky_sphere {
    pigment {
        gradient <0,1,0>
        color_map { [0.00 rgb <0.6,0.7,1.0>]
                    [0.35 rgb <0.0,0.1,0.8>]
                    [0.65 rgb <0.0,0.1,0.8>]
                    [1.00 rgb <0.6,0.7,1.0>]
                  }
                scale 2
            } // end of pigment
           } //end of sky_sphere

// Einlesen und Darstellen der Daten der Fussballbahn
// Format der Datei: Eine Zeile mit kommaseparierten Vektoren
// <.,.,.> , <.,.,.> , <.,.,.> , ...
// Beim Rausschreiben darauf achten, dass y- und z-Wert der Bahn
// vertauscht werden.
#fopen data "fussball_pov.dat" read
// defined(data) prueft nach, ob das Ende der Datei erreicht ist
#while (defined(data))
#read (data,Position)
// Das erzeugt eine kleine rote Kugel mit schwarzen Sechsecken,
// also fast ein richtiger Fussball ...
sphere {Position, 0.1 texture { pigment { hexagon
                   Red, White,Red
                   scale <1,1,1>*0.025
}
          finish { phong 0.3}
}}
#end
// Das wars!
```

Achten Sie insbesondere darauf, wie hier die `Position` des Balles aus der Datei `fussball_pov.dat` eingelesen wird. Steht in dieser Datei ein Ort, wird der Ball hierhin gesetzt; bei mehreren Einträgen enthält die Szene entsprechend mehrere Bälle.
Eine Möglichkeit, aus Ihren Simulationsdaten ein Video zu erzeugen ist es, `povray` für jede Konfiguration aus dem Programm heraus aufgerufen. Folgendes C++-Fragment bewirkt genau dies für die Simulation der Flugbahn des Fußballs, unter der Annahme, dass diese zuvor berechnet und in `r(t)` gespeichert wurde (insbesondere der Aufruf von `povray` über `system()` ist nicht unbedingt als Stil-Empfehlung zu verstehen, aber es erfüllt seinen Zweck):

```
#include <string>
#include <sstream>
#include <cstdio>
#include <cstdlib>
⋮

for (int i=0;i<t.size();i++) {
  ofstream aus_pov("fussball_pov.dat");       // Koordinate in Datei
  aus_pov << "<" << r[i].x << ","             // ausgeben
          << r[i].z << ","
          << r[i].y << ">";
  aus_pov.close();
  system("povray -W1024 -H768 fussball.pov");
  // Bilddatei so umbenennen, dass sie mit genau 3 Dezimalstellen
  // durchnummeriert wird (nuetzlich um korrekte Reihenfolge der
  // Szenen sicherzustellen)
  stringstream fname;
  fname << "fussball" << setw(3) << setfill ('0') << i << ".png";
  rename("fussball.png", fname.str().c_str());
}
```

Ein Bild, das `povray` auf diese Weise für eine bestimmte Position des Fussballs erzeugt hat, ist in Abbildung 3.5 zu sehen.
Auch hier können wir aus allen Bilddateien mit folgenem Kommando wieder ein kleines Video erzeugen:

```
convert -delay 4 -loop 1 fussball???.png Fussball.mpg
```

Im Vergleich zu den Planeten haben wir uns hier für eine höheren Bildrate entschieden und wollen die Flugballbahn nur einmal abspielen (ein Torschuß ist ja ein

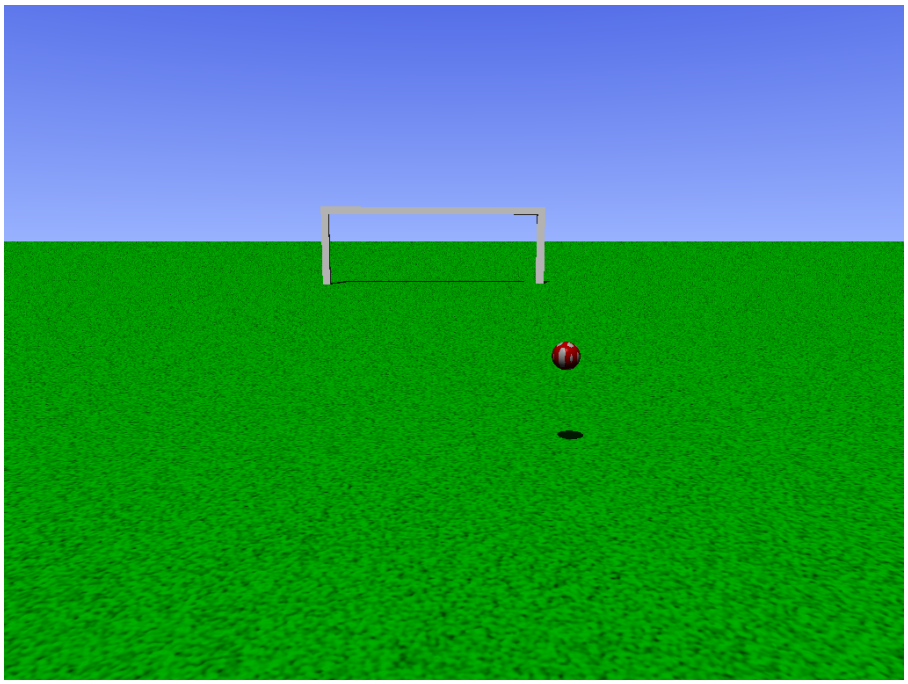


Abbildung 3.5: Mit `povray` gerenderte Szene aus der Flugbahn eines Fußballs.

einmaliges Ereignis, während es sich beim Planeten-Umlauf um eine periodische Bewegung handelt).
Wie immer gilt: Lassen Sie Ihrem Spieltrieb freien Lauf!

Kapitel 4

Prinzipien der Molekulardynamik

Gegeben die Newton'schen Bewegungsgleichungen (2.1), können wir die Eigenschaften von Gasen, Flüssigkeiten, Festkörpern oder noch komplexeren Systemen wie Polymeren und Proteinen berechnen? Mal davon abgesehen, dass es einem Physiker einfach Spaß macht, solche Dinge zu untersuchen, stehen dahinter natürlich auch handfeste praktische, ja sogar kommerzielle Ziele. Von Standpunkt der Grundlagenforschung helfen solche „Computerexperimente", die Ergebnisse vom Messungen zu interpretieren und Vorschläge für weitere Experimente zu machen. Auf der anderen Seite sollten industriell eingesetzte Materialien bestimmte Eigenschaften haben bzw. sich unter bestimmten äußeren Bedingungen (Druck, Temperatur, mechanische Spannungen usw.) in vorherbestimmbarer Weise verhalten. Je weniger reale Versuche man zu diesem Zweck durchzuführen hat, umso kostengünstiger ist die Entwicklung eines Produktes.

Es ist klar, dass man sich mit einer solchen Fragestellung eine formidable Aufgabe gestellt hat. In einem typischen System gibt es Größenordnung 10^{23} Teilchen und in zwei Realisierungen unter gleichen äußeren Bedingungen werden die individuellen Bahnen dieser Teilchen in einem gegebenen Zeitintervall durchaus sehr verschieden sein. Dennoch können die für uns beobachtbaren Größen wie Temperatur, Druck, Aggregatzustand, etc. identisch sein oder sich doch nur sehr wenig unterscheiden.

Ein erster Zugang zur Berechnung der Eigenschaften von Vielteilchensystemen basiert auf der direkten Lösung der Newton'schen Bewegungsgleichungen. Dieses Verfahren wird als *Molekulardynamik* bezeichnet. In modernen Anwendungen kann man hier Systeme bis zu 10^9 Teilchen simulieren. Das ist zwar noch weit entfernt von den 10^{23} Teilchen in realen Gasen u.ä., aber durchaus schon genug, um Aussagen über die Thermodynamik zu erhalten. Die Frage ist hier, wie sich die Kenntnis von einer solchen Unzahl von Daten, wie es die Bahnen von auch nur 10^9 Teilchen über einen gewissen Zeitraum darstellen, sinnvoll in experimentell zugängliche Größen überführen lassen.

Im Folgenden werden wir uns mit den Prinzipien der Molekulardynamik bekannt machen und einige grundlegende Verfahren in diesem Zugang diskutieren.

4.1 Vorbemerkungen

Wie bereits bei den einleitenden Bemerkungen erwähnt, besteht der Molekulardynamikzugang darin, die Newton'schen Bewegungsgleichungen (2.1) für N Teilchen direkt numerisch zu lösen. Hier fängt bereits das Problem an, denn im Allgemeinen wissen wir nicht, wie die Wechselwirkungen zwischen Atomen oder Molekülen eigentlich aussehen. Da diese elektrisch neutral sind, könnte man zunächst argwöhnen, dass es sich eventuell um Gravitationskräfte handeln könnte. Nähme man diese Idee ernst, dann müssten Sie feststellen, dass keiner von uns existieren

würde, da die Gravitation eine viel zu schwache Kraft ist, um die Stabilität von kondensierter Materie zu erklären.
Die eigentliche Ursache für die intermolekularen Kräfte liegt in der Quantenmechanik der Elektronenhülle begründet. Zum Einen sind Elektronen – wie Sie vielleich bereits wissen – Fermionen, d.h. sie mögen es nicht, gleiche Zustände im Atom einzunehmen. Das führt dazu, dass Atome auf kurze Distanzen im Prinzip wie harte Kugeln wirken. Auf der anderen Seite ist das Atom zwar elektrisch neutral, d.h. die Ladungsdichte verschwindet, aber die Bewegung der Elektronen um den Kern führt dazu, dass im zeitlichen Mittel höhere Multipolmomente nicht verschwinden, die dann zu einer entsprechenden, typischerweise kurzreichweitigen attraktiven Wechselwirkung führen.
Eine genaue Berücksichtigung dieser Effekte und damit eine Berechnung von Wechselwirkungen von Molekülen ist im Prinzip mit quantenchemischen Verfahren möglich. Für die Anwendung benutzt man aber häufig ein phänomenologisches Potential. Oftmals nimmt man dabei zunächst an, dass die gesamte potentielle Energie als Summe von Paarpotentialen geschrieben werden kann, d.h.

$$U = \frac{1}{2} \sum_{i \neq j} u(\vec{r}_i - \vec{r}_j) \quad .$$

Beachten Sie, dass diese eigentlich recht intuitiv aussehende Form im Allgemeinen nur eine Näherung darstellt.
Für die Paarpotentiale $u(\vec{r})$ ist ein üblicher Ansatz das Lennard-Jones-Potential

$$u(\vec{r}) = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \quad . \quad (4.1)$$

Es wird charakterisiert durch zwei Parameter, eine Energieskala ε und eine Längenskala σ. Sein Verlauf ist in Abb. 4.1 dargestellt. Die starke Abstoßung für kleine Abstände wird durch den ersten Term simuliert, während der zweite Term die Anziehung auf mittleren Längenskalen beschreibt. Für praktische Anwendungen ist es hilfreich, dass $u(r) \approx 0$ für $r \geq 3\sigma$. Beachten Sie, dass das Lennard-Jones-Potential bei weitem nicht das Einzige ist, welches für Molekulardynamik eingesetzt wird. Wenn es Sie interessiert, was man für welchen Typ von Atom bzw. Molekül benutzt, dann finden Sie eine kleine Übersicht z.B. in [2].

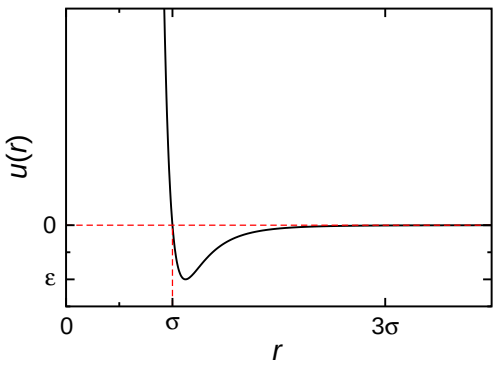


Abbildung 4.1: Verlauf des Lennard-Jones-Potentials.

Eine weitere wichtige Frage betrifft die eigentlichen physikalischen Skalen, über die wir bei einer solchen Simulation reden. Energie und Länge werden durch ε und

Größe	Einheit	Werte für Argon
Länge	σ	$3,4 \cdot 10^{-10}$m
Zeit	$\sigma(m/\varepsilon)^{1/2}$	$2,2 \cdot 10^{-12}$s
Masse	m	$6,7 \cdot 10^{-26}$kg
Geschwindigkeit	$(\varepsilon/m)^{1/2}$	$1,6 \cdot 10^{2}$m/s
Energie	ε	$1,7 \cdot 10^{-21}$J
Kraft	ε/σ	$4,9 \cdot 10^{-12}$N
Druck	ε/σ^3	$4,2 \cdot 10^{7}$N/m^2

Tabelle 4.1: Physikalische Einheiten für Argon.

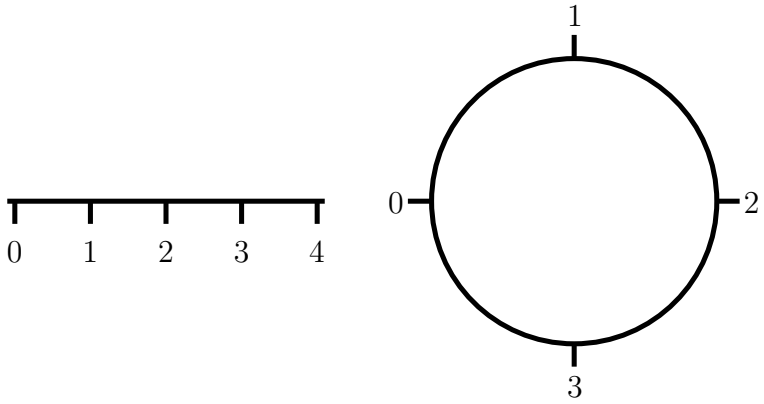


Abbildung 4.2: Periodische Randbedingungen.

σ festgelegt, eine dritte Skala erhalten wir durch die Masse m der Teilchen. Ein typisches Beispiel für die sich daraus ergebenden Skalen zeigt die nebenstehende Tabelle 4.1. Schreibt man eine Simulation, dann werden im Programm alle Größen in sog. reduzierten Einheiten $m = \sigma = \varepsilon = 1$ eingesetzt. Die Rückrechnnung in reale Einheiten geschieht dann am Ende mit Hilfe einer solchen Tabelle. Zum Beispiel erhalten wir für eine Simulation von Argon mit einem Zeitschritt von $\Delta t = 0.01$ bei 10^5 Zeitschritten eine Gesamtdauer der Simulation von 10^3 reduzierten Zeiteinheiten, entsprechend etwa 2ns!

4.2 Definition des Systems

Verglichen mit realen, makroskopischen Systemen lassen sich in einer Computersimulation nur extrem kleine Systeme untersuchen. Es besteht die Gefahr, dass bei so kleinen Systemen die Eigenschaften im wesentlichen von den Rändern bestimmt werden. Daher ist die richtige Wahl der *Randbedingungen* entscheidend für eine Simulation. Um Einflüsse von den veränderten Verhältnissen an realen Rändern zu

minimieren wählt man *periodische Randbedingungen*. Für ein eindimensionales System bedeutet dies wie in Abb. 4.2 dargestellt, dass linker („0“) und rechter („4“) Rand „zusammengebunden“ werden, so dass ein Kreis entsteht. Für zweidimensionale Systeme wird dann entsprechend die Oberfläche eines Torus daraus und für dreidimensionale Systeme ...; na ja, stellen Sie es sich vor.

Allerdings bringen periodische Randbedingungen auch ein Problem mit sich: Jedes Teilchen hat gemäß Abb. 4.3 jetzt unendliche viele „periodische Bilder“ und damit ist der Abstand zwischen zwei Teilchen, der ja die Kräfte bestimmt, nicht mehr eindeutig. Genau genommen wechselwirkt jedes Teilchen sogar mit dem „Urbild“ und den unendlich vielen periodischen Replika jedes anderen Teilchens. Sind die Wechselwirkungen nur kurzreichweitig, so definiert man üblicherweise einen *Abschneideabstand* (z.B. 3σ für das Lennard-Jones-Potential), jenseits dessen die Wechselwirkungskräfte auf Null gesetzt werden. Da dieser Abschneideabstand viel kleiner als die Ausdehung der Simulationsbox sein sollte, nimmt man unter allen möglichen Wechselwirkungspartnern eines Teilchens nur dasjenige Bild mit dem kleinsten Abstand (*minimal image convention*). Dieses Verfahren ist in Abb. 4.3 gezeigt.

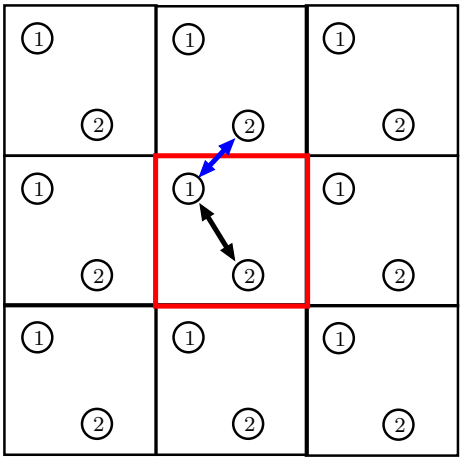


Abbildung 4.3: Zur Minimum-Image-Konvention.

4.3 Anfangsbedingungen

Die Wahl geeigneter Anfangsbedingungen erfordert bei MD-Simulationen erheblichen Aufwand. Bei manchen MD-Simulationen ist die Erzeugung einer richtigen Gleichgewichts-Anfangsbedingung sogar der rechenaufwendigste Teil der Simulation.

Das erste Problem besteht darin, eine geeignete *Startkonfiguration* (also die Orte zur Anfangszeit) zu bestimmen, wenn das System eine hohe Dichte besitzt, wie es bei Flüssigkeiten oder Festkörpern der Fall ist. Es ist eine *schlechte Idee*, diese Ort „auszuwürfeln“, denn bei hoher Dichte produziert eine Gleichverteilung von Orten typischerweise Konfigurationen, bei denen sich Teilchenpaare sehr nahe kommen. Dadurch erhält man eine Startkonfiguration mit enorm großen Anfangskräften und als Konsequenz ein numerisch instabiles System. Eine gängige Methode zur Erzeugung einer Anfangskonfiguration besteht darin, die Teilchen

auf einem Gitter zu platzieren, wobei die Gitterkonstante aus der vorgegebenen, mittleren Dichte $N/(L_x L_y L_z)$ bestimmt wird. Für harte Kugeln oder auch das Lennard-Jones-Potential erwartet man als Grundzustandskonfiguration eine *dichteste Kugelpackung* (*hexagonales Gitter*). Konsequenterweise verwendet man dann auch gerne ein solches Gitter als Startkonfiguration. Im Prinzip tut es aber auch jedes andere Gitter, z.B. die Ecken eines Würfels (*einfach-kubisches Gitter*). Wichtig bei der Auswahl der Abstände im Gitter ist aber, dass sich zwei Teilchen nicht zu nahe kommen. Ein guter Wert für diese „minimale Gitterkonstante“ für das Lennard-Jones-Potential ist $2^{1/6}\sigma$.
Natürlich ist diese so erzeugte Anfangskonfiguration i.a. nicht die gesuchte Gleichgewichtskonfiguration. Man läßt daher die Startkonfiguration sich als Funktion der Zeit entwickeln. Dabei wird die anfänglich rein potentielle Energie zum Teil in kinetische Energie umgewandelt. Manchmal ist es sinnvoll, dem System bereits zu Beginn durch Vorgabe von Anfangsgeschwindigkeiten eine kinetische Energie aufzuprägen. Allerdings sollte man dann darauf achten, dass

$$\sum_{i=1}^{N} \vec{v}_i = \vec{0}$$

gilt, da sich ansonsten unser System mit konstanter Geschwindigkeit durch das „Labor“ bewegen würde, was man immer gerne vermeidet. In der Natur verhindert dies die Existenz von Wänden. Auf jeden Fall ist es immer eine gute Idee, sich davon zu überzeugen, dass die Ergebnisse nicht signifikant von den Anfangsbedingungen abhängen.

4.4 Algorithmus

Der Kern des Algorithmus besteht aus der Integration der Newton'schen Bewegungsgleichungen (2.1). Es ist klar, dass wir für die hier anvisierten langen Zeitskalen der Simulation ein zuverlässiges Lösungsverfahren benötigen, das aber gleichzeitig nicht zu komplex ist. Im Kontext der Molekulardynamik häufig verwendete Algorithmen sind die sogenannten *Verlet-Algorithmen*. Wir schreiben zunächst (2.1) in der Form

$$\ddot{x} := \frac{d^2 x}{dt^2} = \frac{F(x, \dot{x}, t)}{m} =: a(x, \dot{x}, t)\,. \tag{4.2}$$

Die „naive“ Anwendung von z.B. der Näherung (1.12) für die zweite zeitliche Ableitung auf der linken Seite von (4.2) führt nun auf den einfachsten Verlet-Algorithmus

$$x_{n+1} \;=\; 2\,x_n - x_{n-1} + a(x_n, \dot{x}_n, t_n)\cdot(\Delta t)^2 + \mathcal{O}\left[(\Delta t)^4\right] \quad . \tag{4.3}$$

Diese Algorithmus ist genau bis zur dritten Ordnung in Δt, kommt also bei geringerem Aufwand in die Nähe des Runge-Kutta-Algorithmus 4. Ordnung (2.13). Dies liegt primär daran, dass wir das ursprüngliche Problem nicht auf ein System von ODEs erster Ordnung zurückgeführt haben, sondern direkt die zweite zeitliche Ableitung nähern.
Das Zweischrittverfahren (4.3) benötigt genau wie das Leapfrog-Verfahren (2.5) die Vorgabe von $x(t_0)$ und $x(t_0 + \Delta t)$. Den zweiten Wert kann man sich z.B. aus der Taylorentwicklung

$$x(t_0 + \Delta t) \approx x(t_0) + \dot{x}(t_0) \cdot \Delta t + a(x, \dot{x}, t_0) \cdot \frac{(\Delta t)^2}{2}$$

beschaffen.
Zudem erhält man im Verlet-Algorithmus erst einmal keine brauchbare Näherung für die Geschwindigkeit $v_{n+1} := \dot{x}_{n+1}$. Abhilfe schafft hier z.B. der *Velocity-Verlet Algorithmus*, d.h. der Verlet-Integrationsschritt für x_n aus Glg. (4.3) erweitert um eine verbesserte Näherung

$$v_{n+1} = v_n + \frac{1}{2}\left[a(x_n, t_n) + a(x_{n+1}, t_{n+1})\right] \cdot \Delta t \qquad (4.4)$$

für die Geschwindigkeit.
Sowohl in (4.3) als auch in (4.4) benötigt man die Beschleunigung $a(x, t)$, die über eine Kraftberechnung zu bestimmen ist. Aus Kostengründen sollte man eine Wiederholung dieser Rechnung vermeiden und stattdessen das einmal berechnete Ergebnis zwischenspeichern. In der bisherigen Formulierung benötigt man damit insgesamt 5 Vektoren: x_{n-1}, x_n, v_{n-1}, $a(x_n, t_n)$ und $a(x_{n+1}, t_{n+1})$.
Die folgende Formulierung hingegen mit 3 Vektoren x_n, v_n und $a_n := a(x_n, t_n)$ aus und ist somit speichereffizienter:

$$\begin{aligned}
x_{n+1} &= x_n + v_n\,\Delta t + \frac{1}{2}\,a_n\,\Delta t^2 && (4.5a)\\
v(t_n + \Delta t/2) &= v_n + \frac{1}{2}\,a_n\,\Delta t && (4.5b)\\
a_{n+1} &= \frac{1}{m}\,F(x_{n+1}, t_{n+1}) && (4.5c)\\
v_{n+1} &= v(t_n + \Delta t/2) + \frac{1}{2}\,a_{n+1}\,\Delta t\,. && (4.5d)
\end{aligned}$$

Zunächst einmal reproduziert man durch Einsetzen von (4.5b) in (4.5d) Glg. (4.4). Glg. (4.5a) sieht zunächst nach einer einfachen Taylorentwicklung 2. Ordnung aus. Man kann jedoch zunächst die Differenz von (4.5a) mit n und $n-1$ betrachten, um

$$x_{n+1} - 2\,x_n + x_{n-1} = (v_n - v_{n-1})\,\Delta t + \frac{1}{2}(a_n - a_{n+1})\,\Delta t^2 \qquad (4.6)$$

zu zeigen. Einsetzen von $v_{n-1} = v_n - \frac{1}{2}(a_{n-1} + a_n)$ gemäß (4.4) führt auf $x_{n+1} - 2\,x_n + x_{n-1} = a_n\,\Delta t^2$, womit (4.3) reproduziert wäre. Damit folgt nebenbei, dass der Korrekturterm für den Algorithmus (4.5a-4.5d) nicht wie nach dem ersten Augenschein von der Ordnung Δt^3, sondern von der Ordnung Δt^4 ist. Vorzugeben sind ferner die Anfangsbedingungen x_0 und v_0 (aus denen a_0 berechnet werden kann), d.h. man vermeidet das Initialisierungsproblem der Zweischritt-Formulierung. Ein kleiner Nachteil ist, dass die Kraft $F(x,t)$ nicht von der Geschwindigkeit abhängen sollte, da die Geschwindigkeit v_{n+1} bei der Berechnung von a_{n+1} in (4.5c) noch nicht bekannt ist.

4.5 Equilibrierung

Vom Standpunkt der Physik aus gesehen liegt ein System vor, auf das keine äußeren Kräfte wirken, ein sogenanntes *abgeschlossenes System*. Daher wissen wir, dass die Gesamtenergie als Funktion der Zeit konstant ist oder mit anderen Worten die Zeitentwicklung bei *konstanter Gesamtenergie* stattfindet (bei Vernachlässigung numerischer Fehler). Diese Situation bezeichnet man in der statistischen Physik als *mikrokanonisch*.
Ein Vielteilchensystem entwickelt sich bei „generischen“ Anfangsbedingungen gegen einen thermischen Gleichgewichtszustand und für genau diese Zustände interessiert man sich auch im Normalfall. Um sie zu erzeugen, braucht man also im Prinzip nur zu warten, während der Computer rechnet. Bleiben allerdings zwei Fragen:

- Wie lange muss man warten?
- Wie produziert man nicht irgendeinen Gleichgewichtszustand, sondern einen bestimmten, also z.B. einen mit vorgegebener Temperatur?

Die Antwort auf beide Fragen erfordert die Betrachtung *makroskopischer Observablen*. Beispiele hierfür sind z.B. die Temperatur oder den Druck. Ein Gleichgewichtszustand ist dann dadurch charakterisiert, dass sich makroskopische Observablen in der Zeit nicht verändern. Kann man sie also aus den Computerdaten bestimmen, so weiss man (im Prinzip), wie lange man *equilibrieren* muss.
Zum Beispiel liefert eine Simulation für 64 Teilchen das Ergebnis in Abb. 4.4 für die kinetische und potentielle Energie des Systems. Offentsichtlich „wackeln“ beide Größen ordentlich hin und her, aber ebenso offensichtlich um einen wohldefinierten Mittelwert. Außerdem ist durch Inspektion klar, dass die Gesamtenergie schön konstant ist. Ein ähnliches Verhalten beobachtet man auch für andere Größen. Die *Equilibrierung* ist hier etwa nach einer Zeit $t = 10$ erfolgt, d.h. nach dieser Zeit kann man anfangen, am System „Messungen“ durchzuführen.

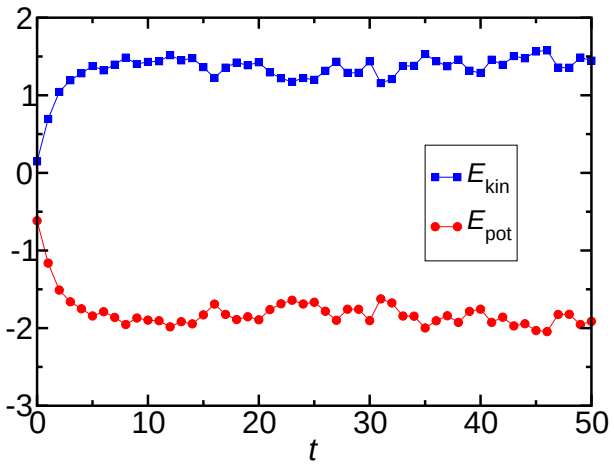


Abbildung 4.4: Beispiel für die Zeitabhängigkeit makroskopischer Größen in einer Molekulardynamik-Simulation.

Übrigens sind in den „Fluktuationen“ um den Mittelwert durchaus auch noch interessante, physikalische Informationen enthalten.
Es gibt übrigens noch eine weitere, wichtige Methode zur Erzeugung von Anfangsbedingungen. Hat man nämlich einmal ein System equilibriert, so kann man den Zustand des equilibrierten Systems (Orte und Geschwindigkeiten) in einer Datei abspeichern und für weitere Simulationen als Anfangsbedingung benutzen. Das spart of eine Menge Arbeit und Rechenzeit. Will man eine Anfangsbedingung für ein System mit einer anderen Dichte als der im abgespeicherten Zustand, so kann man einfach alle Teilchenabstände *reskalieren*. Obwohl man so i.a. noch keinen Gleichgewichtszustand erzeugt, kann man häufig die *Equilibirierungszeit* gegenüber anderen Anfangzuständen erheblich abkürzen.

4.6 Festlegung der Temperatur

Bislang haben wir uns keine Gedanken darüber gemacht, wie man für eine Molekulardynamik-Simulation die Temperatur einstellt. Dies ist eine nichttriviale Aufgabe, für die es verschiedene Methoden gibt. Wir wollen hier die einfachste diskutieren.
Aus der klassischen Thermodynamik ist uns der sogenannte *Gleichverteilungssatz* bekannt, der besagt, dass jeder Freiheitsgrad die Energie $k_{\mathrm{B}}T/2$ beiträgt. Für unser Modellsystem sind die einzigen Freiheitsgrade die der Bewegung, und konsequenterweise gilt zu einem gegebenen Zeitpunkt t formal

$$E_{\mathrm{kin}}(t) = \frac{m}{2}\sum_{i=1}^{N}\vec{v}_i^{\,2} =: \frac{Nd}{2}k_{\mathrm{B}}T(t) \quad ,$$

wobei d die Dimension unseres Systems darstellt.

Natürlich macht eine zeitabhängige Temperatur keinen direkten Sinn, sie zeigt aber einen Weg, den Begriff „Temperatur“ für unser System zu definieren. Dazu erinnern wir uns, dass die kinetische Energie nach der Equilibrierung zufällig um einen wohldefinierten Mittelwert schwankt. Es macht also durchaus Sinn, für den *Mittelwert* der kinetischen Energie

$$\overline{E_{\text{kin}}} = \frac{m}{2} \sum_{i=1}^{N} \overline{\vec{v}_i \cdot \vec{v}_i} =: \frac{Nd}{2} k_{\text{B}} T \tag{4.7}$$

zu definieren und somit bei bekannter kinetischer Energie die Temperatur unseres Systems festzulegen. Auch hier ist wieder wichtig, dass

$$\sum_{i=1}^{N} \vec{v}_i = \vec{0}$$

gilt, da die Schwerpunktsbewegung eines makroskopischen Körpers normalerweise nicht in Verbindung mit seiner Temperatur gebracht wird.
Nachdem wir jetzt also eine der für uns messbaren und damit kontrollierbaren makroskopischen Systemgrößen mit der Temperatur in Verbindung gebracht haben, können wir uns der Frage zuwenden, wie man jetzt umgekehrt die Temperatur des Systems vorgeben kann. Dazu drehen wir den Spieß einfach um: Wir benutzen den Zusammenhang (4.7), um *in jedem Schritt* die Geschwindigkeiten aller Teilchen so zu *reskalieren*, dass immer

$$E_{\text{kin}}(t) = \frac{m}{2} \sum_{i=1}^{N} \vec{v}_i(t) \cdot \vec{v}_i(t) = \frac{Nd}{2} k_{\text{B}} T \tag{4.8}$$

gilt, d.h. anstelle der Gesamtenergie fixieren wir jetzt nur die kinetische Energie. Dieselbe Simulation für 64 Teilchen aus Abb. 4.4 ergibt jetzt für eine Temperatur $T = 0.1$ das Ergebnis in Abb. 4.5. Die kinetische Energie ist jetzt konstant, und die potentielle enthält alle Informationen über die Dynamik des Systems. Es ist auch ersichtlich, dass jetzt die Equilibrierung deutlich länger braucht, nämlich $t \approx 30$. Dass die Equilibrierung überhaupt stattfindet, zeigt uns, dass das Konzept in der Tat funktioniert, wir also nach genügend langer Wartezeit ein System vorliegen haben, das die von uns vorgegebene Temperatur hat.

4.7 Messgrößen

Was fehlt, ist die Angabe von sinnvollen Meßgrößen für die Simulation. Die Energie selber ist nicht messbar – wohl aber deren Schwankung in Form der spezifischen Wärme – und somit zwar eine Größe, um das Programm zu testen, aber keine, um

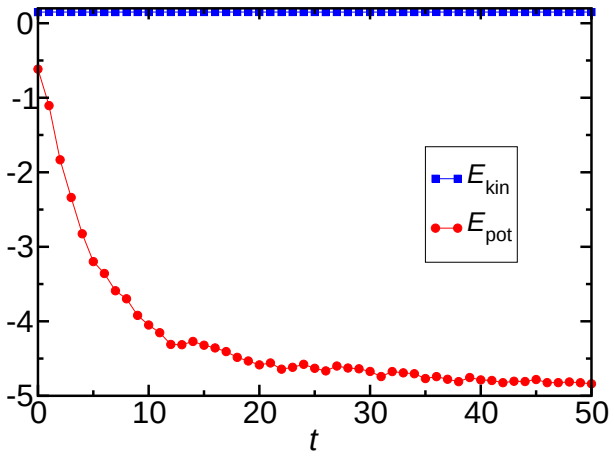


Abbildung 4.5: Beispiel für die Zeitabhängigkeit der Energie in einer Molekulardynamik-Simulation bei fester kinetischer Energie entsprechend einer Temperatur $T = 0.1$.

einen Kollegen aus der experimentellen Chemie oder Physik irgendwie hinter dem Ofen hervorzulocken.

Interessanter sind da Größen wie z.B. der Druck. Dieser ist nicht so ganz einfach zu erhalten, dazu muß man ein wenig in die Trickkiste der statistischen Physik greifen. Das Ergebnis ist aber relativ einfach, und zwar erhält man

$$\frac{pV}{Nk_{\mathrm{B}}T} - 1 = \frac{1}{dNk_{\mathrm{B}}T} \sum_{i,j=1}^{N} \overline{\vec{r}_{ij} \cdot \vec{F}_{ij}} \quad , \tag{4.9}$$

wobei $\vec{F}_{ij}$ die Paarkräfte bezeichnet und wir die Konvention $\vec{F}_{ii} = \vec{0}$ benutzen. Den Term in der Summe nennt man auch das *Virial* des Systems.

Eine weitere Messgröße (z.B. durch Neutronenstreuung o.ä. zugänglich) ist die sogenannte *Paarkorrelationsfunktion*

$$g(r) = \frac{V}{4\pi r^2 N(N-1)} \overline{\sum_i \sum_{j \neq i} \delta(r - r_{ij})} \quad ; \tag{4.10}$$

sie gibt die Wahrscheinlichkeitsdichte an, im Abstandsintervall $[r, r+dr]$ von einem Teilchen ein weiteres zu finden. Speziell für ein ideales Gas findet man $g(r) = 1$, und jede Abweichung von diesem Verhalten impliziert, dass die Teilchen echte Wechselwirkungseffekte zeigen. Ein Beispiel für $N = 216$ Teilchen bei einer Temperatur $T = 1$ in einem L^3-Kasten mit Kantenlänge $L \approx 11.37$ sehen Sie in Abb. 4.6. Man beachte, dass man sich bei der Darstellung aufgrund der periodischen Randbedingungen auf Abstände $r \leq L/2$ beschränken sollte.

Interessant ist noch, dass man mit der Paarkorrelationsfunktion $g(r)$ andere Mess-

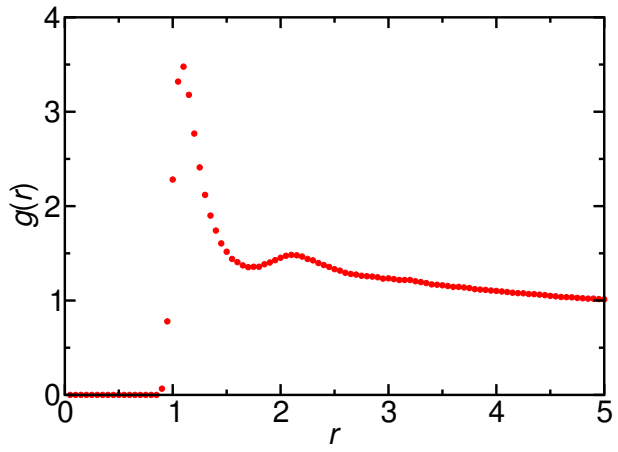


Abbildung 4.6: Beispiel für die Paarkorrelationsfunktion $g(r)$ für $N = 216$, $T = 1$ und Kantenlänge $L \approx 11.37$.

größen berechnen kann, z.B.

$$\begin{aligned} \frac{1}{N}\overline{E_{\text{pot}}} &= \frac{N}{2V}\int_V \mathrm{d}V\, g(r)\, u(r) \\ \frac{PV}{Nk_{\mathrm{B}}T} &= 1 - \frac{N}{2Vdk_{\mathrm{B}}T}\int_V \mathrm{d}V\, g(r)\, r\, \frac{du(r)}{dr} \quad . \end{aligned}$$

Kapitel 5

Zufallszahlen

5.1 Numerische Integration

Betrachten wir ein einfaches Beispiel. Gegeben sei eine Funktion $f(x)$ und die Aufgabe, das Integral

$$I = \int_a^b \mathrm{d}x\, f(x)$$

numerisch zu berechnen. Wie man an so etwas herangeht, haben wir jetzt ja schon mehrfach praktiziert: Wir diskretisieren das Intervall $[a, b]$ mit Schrittweite Δx und greifen auf die Riemann'sche Integraldefinition

$$I = \lim_{N\to\infty} \sum_{i=0}^{N} f(x_i)\, \Delta x$$

zurück. Die einfachste numerische Integrationsformel erhält man dann, indem man einfach den Limes vergisst, d.h.

$$I \approx \sum_{i=0}^{N} f(x_i)\, \Delta x \quad . \tag{5.1}$$

Diese einfache Formel kann man durch Einsetzen von polynomialen Interpolationen verbessert werden. Dies ist ein wichtiges Ergebnis der numerischen Mathematik, soll jedoch hier nicht weiter diskutiert werden.
Eine etwas andere, auf den ersten Blick merkwürdige Art, das Integral zu berechnen, basiert auf der Erkenntnis, dass das bestimmte Integral nichts anderes ist als die Fläche zwischen der Kurve und der x-Achse. Wir nehmen in Folgenden oBdA an, dass $f(x) \geq 0$ gilt und bezeichnen mit f_1 das Maximum von $f(x)$ auf dem Intervall $[a, b]$.

1. Wir wählen eine Zahl $C \in \mathbb{R}$ so, dass $f_1 \in [0, C]$ und definieren einen Zähler $count$, der zu Beginn den Wert $count = 0$ haben soll.
2. Wir wählen jetzt zufällig eine Zahl $\xi \in [a, b]$ und eine zweite Zahl $\phi \in [0, C]$. Wenn $\phi \leq f(\xi)$, dann erhöhen wir $count$ um Eins.
3. Wir wiederholen Schritt 2 M mal, wobei M groß genug sein sollte.
4. Die Näherung für I ist dann

$$I \approx \frac{count}{M}\,(b-a)\cdot C \quad .$$

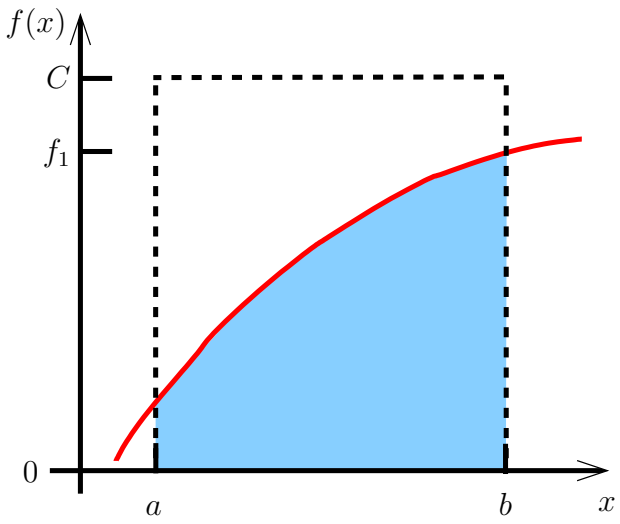


Abbildung 5.1: Zur Monte-Carlo-Integration.

Ist das Minimum von $f(x)$, f_0, auf $[a, b]$ kleiner Null, dann schreiben wir das Integral um gemäß

$$I = \int_a^b \mathrm{d}x \ (f(x) - f_0) + f_0 \cdot (b - a)$$

und wenden die Methode auf den ersten Term an.

Wie und vor allem warum funktioniert dieses Verfahren? Dazu betrachten wir die Skizze in Abbildung 5.1. Die zu berechnende Fläche ist blau eingefärbt. Was tut nun der Algorithmus? Wir ziehen zwei zufällig ausgewählte Zahlen, eine aus dem Intervall entlang der x-Achse, die zweite aus ein Intervall entlang der y-Achse, welches den Wertebereich der Funktion über dem Integrationsintervall umfasst. Beide Zahlen zusammen ergeben die Koordinaten eines Punktes innerhalb des durch die gestrichelten Linien definierten Rechteckes. Die zweite Bedingung fragt ab, ob dieser Punkt nun *unterhalb* der Kurve $f(x)$ liegt. Tut er das, so zählen wir ihn (erhöhen also unseren Zähler $count$), andernfalls verwerfen wir den Zug. Dann "ziehen" wir zwei neue, zufällige Zahlen etc., bis wir eine genügend große Stichprobe genommen haben. Schließlich sind die Züge, die unterhalb der Kurve zu liegen kamen, ein Maß für die Fläche unterhalb der Kurve. Normiert man noch auf die Gesamtzahl der Züge und die Fläche $(b - a) \cdot C$ des Rechteckes, dann landen wir bei der Näherung für unser Integral.

Merken Sie, dass ich ganz allmählich die Sprache verändert habe? Wir "ziehen" zwei "Zufallszahlen", nehmen eine Stichprobe und berechnen die Häufigkeitsverteilung für ein bestimmtes Ereignis "der durch die beiden Zahlen definierte Punkt liegt unterhalb der Kurve $f(x)$". Das sind alles Begriffe aus der Wahrscheinlichkeitstheorie.

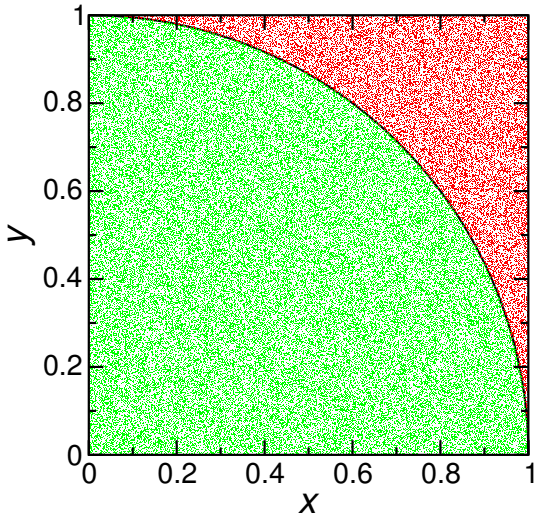


Abbildung 5.2: 100 000 Punkte, die im Einheitsquadrat gleichverteilt sind. 78 538 Punkte liegen im Einheitskreis.

In der Tat haben Sie soeben ein sogenanntes *stochastisches* Verfahren der numerischen Analysis kennengelernt, den *Monte-Carlo-Algorithmus* zur Berechnung eines bestimmten Integrals.
Das Verfahren lässt sich leicht auf höhere Dimensionen verallgemeinern. Als ein einfaches Beispiel betrachten wir zufällig ausgewählte Punkte (x, y) im Einheitsquadrat $0 \leq x, y \leq 1$. Der Anteil der Punkte, die $x^2 + y^2 \leq 1$ erfüllen, ist $\pi/4$. Durch Umkehrung der Formel kann man π abschätzen. Abb. 5.2 zeigt 100 000 Punkte (x, y) die mit einer Gleichverteilung im Einheitsquadrat erzeugt wurden. Man kann den Computer beim Erzeugen der Zufallszahlen leicht nachrechnen lassen, dass 78 538 dieser Punkte die Bedingung $x^2 + y^2 \leq 1$ erfüllen. Man findet in diesem Beispiel als Schätzwert für π: $4 \cdot 78\,538/100\,000 = 3.14152$, was mit dem exakten Ergebnis bis auf 4 Nachkommastellen übereinstimmt.
Der geschilderte Monte-Carlo-Algorithmus zur Berechnung eines bestimmten Integrals hat zwar eine schöne anschauliche Bedeutung, aber man kann die Effizienz leicht verbessern, indem man sich die Berechnung einer Zufallszahl $\phi \in [0, C]$ spart und direkt die Funktionswerte $f(\xi_i)$ für die N Zufallszahlen ξ_i aufsummiert:

$$I \approx \frac{b-a}{N} \sum_{i=1}^{N} f(\xi_i) \quad . \tag{5.2}$$

Der Vorfaktor $(b-a)/N$ entspricht Δx in (5.1). Anstelle von Nullen und Einsen summieren wir also die Funktionswerte $f(\xi_i)$, so dass sich auch die Forderung erübrigt, dass f positiv sein soll. Als einfachste Monte-Carlo-Integrationsformel für den praktischen Einsatz sei (5.2) empfohlen.

5.2 Grundzüge der Wahrscheinlichkeitstheorie

Stochastische Verfahren spielen eine zunehmend wichtige Rolle in der numerischen Untersuchung von komplexen Systemen, sei es in der Physik, der Chemie oder Biologie, den Wirtschafts- oder Gesellschaftswissenschaften oder auch für die Börse und Firmen. Die Grundlagen sind immer dieselben, nämlich man untersucht nicht die vollen Details des Systems, sondern nach bestimmten Kriterien ausgewählte, zufällige Ereignisse, die in genügend großer Zahl gesammelt, das System beschreiben.
Um mit diesen Methoden richtig umgehen zu können, muss man ein klein wenig über Wahrscheinlichkeitstheorie lernen.

5.2.1 Wahrscheinlichkeit, Zufallsvariable und Verteilungsfunktionen

Der mathematische Begriff von Wahrscheinlichkeit legt "klassische" Experimente mit "Zuständen" eines Systems zugrunde. Wir wollen hier nicht die mathematische Theorie entwickeln, sondern gehen von einer operativen Definition aus, die allerdings direkt der mathematischen Definition nachgebildet ist, so dass man einen Einblick in die mathematische Idee gewinnen kann. Dieser Abschnitt ist also eine leicht "schlampige" Version der mathematischen (axiomatischen) Wahrscheinlichkeitsdefinition.
Die wichtigste Grundvoraussetzung ist, dass es Experimente gibt, deren Ergebnis E_i *zufällig* in dem Sinne sind, dass für *identisch präparierte Ausgangszustände* des betrachteten Systems *dasselbe Experiment zu verschiedenen Endzuständen führen kann*. Alle möglichen Ergebnisse (d.h. Endzustände) bilden einen Raum Ω, den man in der Wahrscheinlichkeitstheorie als *Stichprobenraum* oder *Menge der Elementarereignisse* bezeichnet. Diese Menge kann diskret sein (z.B. beim Würfel $\Omega = \{1, 2, 3, 4, 5, 6\}$) oder auch überabzählbar (z.B. die Lebensdauer einer Glühbirne $\Omega = \{t \in \mathbb{R} | t > 0\}$). *Ereignisse* schließlich sind gewisse Teilmengen des Stichprobenraums.
Wahrscheinlichkeiten sind nun einfach Zahlen

$$0 \leq P(E_i) \leq 1,$$

die man den Ereignissen zuordnet und zwar so, dass für *disjunkte* Ereignisse E_i und E_j mit

$$E_i \cap E_j = \emptyset$$

die *Additivitätseigenschaft* gilt

$$P(E_i \cup E_j) = P(E_i) + P(E_j)$$

und dass die *Normierung*

$$\sum_i P(E_i) = 1$$

gewährleistet ist.
Die "messbaren Ereignisse" eines Experiments sind Ereignisse, die noch nicht unbedingt *Zahlen* entsprechen müssen (beim 2-farbigen Würfel sind die Ereignisse z.B. "rot" oder "grün"). Eine Abbildung aus dem Raum der Ereignisse Ω

$$X : \Omega \to S \subseteq \mathbb{R}$$

von Ausgangszuständen auf Messskalen, definiert eine *Zufallsvariable* X, wenn jedes Messergebnis auf der *Skala* S auch einem messbaren Ereignis des Experiments entspricht (mathematisch: die Funktion X ist messbar).
Jede Zufallsvariable definiert über[1]

$$F_X(x) := P(X < x)$$

eine Funktion, die die Eigenschaften von X vollständig charakterisiert. Diese Funktion heißt *Verteilungsfunktion* und ist offenbar einfach die Wahrscheinlichkeit, dass X Werte kleiner als die vorgegebene Zahl x annimmt. Ist $F_X(x)$ glatt genug,[2] so kann man sie ableiten und dadurch eine *Wahrscheinlichkeitsdichte der Zufallsvariablen X*

$$p_X(x) = \frac{d}{dx} F_X(x)$$

definieren. Damit ist $p_X(x)dx$ also die Wahrscheinlichkeit, dass X Werte im Intervall $[x, x + dx]$ annimmt.
Die *Momente* der Wahrscheinlichkeitsdichte sind wichtige, dem Experiment zugängliche Kenngrößen einer Verteilungsfunktion

$$\langle X \rangle := \int_{-\infty}^{\infty} \mathrm{d}x \, x \, p_X(x) \quad \text{Erwartungswert bzw.}$$

$$\langle X^r \rangle := \int_{-\infty}^{\infty} \mathrm{d}x \, x^r \, p_X(x) \quad r\text{-tes Moment.}$$

Von speziellem Interesse ist die *Varianz* einer Zufallsvariablen, definiert durch

$$\sigma_X^2 := \int_{-\infty}^{\infty} \mathrm{d}x \, (x - \langle X \rangle)^2 \, p_X(x) \ .$$

[1] Das ist die Physikerschreibweise. Mathematisch korrekt muss es $F_X(x) := P(\{\omega | X(\omega) < x\})$ lauten. Ich werde im Folgenden immer die "schlampigen" Physikerschreibweisen benutzen, solange es keine Missverständnisse gibt.

[2] Falls die Zufallsvariable diskret ist, z.B. beim Würfel, dann muss man ein wenig aufpassen. Diesen Stress wollen wir uns hier aber nicht machen.

Haben wir eine Zufallsvariable X über die Dichte $p_X(x)$ gegeben, so können wir über

$$Y = f(X)$$

neue Zufallsvariablen bilden. Wie bestimmt man die Dichte $p_Y(y)$? Offenbar bildet f das Intervall dx auf

$$dy = \frac{df}{dx}dx$$

ab. Die Wahrscheinlichkeit X in $[x, x + dx]$ zu finden – also $p_X(x)dx$ – ist die gleiche, wie Y in $[y, y + dy]$ zu finden, vorausgesetzt f ist eineindeutig. Also gilt

$$p_Y(y = f(x))dy = p_X(x)dx$$

bzw.

$$p_Y(y = f(x)) = p_X(x) \left|\frac{df}{dx}\right|^{-1} . \qquad (5.3)$$

Hierbei trägt der Betrag der Tatsache Rechnung, dass Wahrscheinlichkeiten positiv sind. Mathematisch ist er darauf zurückzuführen, dass wir bei einer negativen Steigung von f ggfs. die Integrationsgrenzen umdrehen müssen. Die Transformationsregel (5.3) werden wir in der Numerik benutzen, um aus bekannten Verteilungen von Zufallszahlen neue zu generieren.

5.2.2 Gesetz der großen Zahlen, zentraler Grenzwertsatz

Betrachten wir nun N Zufallsvariable, z.B. $X_1, X_2, \ldots, X_N$. Dazu stelle man sich Experimente vor, bei denen z.B. mit N Würfeln gleichzeitig gewürfelt wird oder man die Lebensdauern von N gleichzeitig eingeschalteten Glühbirnen beobachtet. Wie für eine Zufallsvariable ist

$$p(x_1, \ldots, x_N)dx_1dx_2 \cdots dx_N$$

die Wahrscheinlichkeit, Realisierungen der N Zufallsvariablen bei einem Einzelexperiment in den Intervallen $x_i < X_i < x_i + dx_i$, $i = 1, \cdots, N$ zu finden (*gemeinsame Wahrscheinlichkeit*). Fasst man die X_i als kartesische Koordinaten im $\mathbb{R}^N$ auf, so spricht man auch von einem N-dimensionalen *Zufallsvektor* $\vec{X} = (X_1, \cdots, X_n)$. Eine wichtige Klasse von Zufallsvektoren sind solche, die *statistisch unabhängige, aber identisch verteilte Komponenten* haben, d.h.

$$p(x_1, \cdots, x_n) = p(x_1) \cdots p(x_n) .$$

Ein Beispiel sind N Würfel. Man kann eine einzige Realisierung solcher Vektoren auch als die Ergebnisse von N Einzelexperimenten auffassen, z.B. einmal mit N

Würfeln oder N-mal mit einem Würfel! Durch diesen Trick gelingt es, mit Hilfe der Wahrscheinlichkeitstheorie Aussagen über das Verhältnis von *Erwartungswerten* zu *Mittelwerten über endlich viele Experiment zu machen*.

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{5.4}$$

bezeichnet man auch als *empirischen Mittelwert*, gewonnen aus N Messergebnissen. Man beachte, dass $\bar{X}$ selber eine Zufallsvariable ist, d.h. N weitere Messungen liefern i.A. einen verschiedenen empirischen Mittelwert. Der Erwartungswert des empirischen Mittels ist

$$\langle \bar{X} \rangle = \frac{1}{N}\sum_{i=1}^{N} \langle X_i \rangle \quad . \tag{5.5}$$

Wegen der identischen Verteilung gilt $\langle X_i \rangle = \langle X \rangle$, d.h. im Erwartungswert stimmen X und $\bar{X}$ überein. Wie groß ist nun die Varianz von $\bar{X}$? Nach dem bisher gesagten gilt

$$\begin{aligned} \sigma^2_{\bar{X}} &= \langle \bar{X}^2 \rangle - \langle \bar{X} \rangle^2 \\ &= \frac{1}{N^2}\sum_{i,j=1}^{N} \left(\langle X_i\, X_j \rangle - \langle X_i \rangle \langle X_j \rangle \right) \end{aligned} \tag{5.6}$$

In der Summe sind nun zwei Fälle zu betrachten. Für $j = i$ gilt

$$\langle X_i^2 \rangle - \langle X_i \rangle^2 = \langle X^2 \rangle - \langle X \rangle^2 = \sigma^2_X \,,$$

wobei wir wieder verwendet haben, dass die Verteilungen identisch sind. Andererseits gilt aufgrund der statistischen Unabhängigkeit für $j \neq i$

$$\langle X_i\, X_j \rangle = \langle X_i \rangle \langle X_j \rangle \,,$$

so dass die Terme mit $i \neq j$ in der Summe (5.6) wegfallen. Wir haben also

$$\sigma^2_{\bar{X}} = \frac{1}{N^2}\sum_{i}^{N} \left(\langle X_i^2 \rangle - \langle X_i \rangle^2 \right) = \frac{1}{N^2} N \sigma^2_X$$

bzw.

$$\boxed{\sigma^2_{\bar{X}} = \frac{1}{N}\,\sigma^2_X \,.} \tag{5.7}$$

Die wichtige Aussage ist nun, dass für *große N* die Varianz immer kleiner wird, d.h. die Schwankungen $\sigma_{\bar{X}}$ um den Mittelwert verschwinden mit $N \to \infty$ wie $1/\sqrt{N}$.

Dies bezeichnet man als das *Gesetz der großen Zahlen*, es ist grundlegend wichtig für unser Vertrauen in reproduzierbare Messungen.
Es gibt noch eine Reihe weitergehender Aussagen über das Grenzverhalten der Wahrscheinlichkeitsverteilung des empirischen Mittelwerts. Die wohl wichtigste ist der sogenannte *zentrale Grenzwertsatz*, den ich hier nur angeben möchte. Dahinter steckt die Aufgabe, $p_{\bar{X}}(\bar{x})$ zu berechnen. Das Ergebnis ist

$$p_{\bar{X}}(x) = \frac{1}{\sqrt{2\pi\,\sigma_X^2/N}} \exp\left(-\frac{(x-\langle X\rangle)^2}{2\,\sigma_X^2/N}\right)\left(1+\mathcal{O}(N^{-1/2})\right)\ , \qquad (5.8)$$

d.h. eine Gauß-Funktion. Mit zunehmendem N wird die Gauß-Glocke immer schärfer um $\langle X\rangle$ lokalisiert. Dies ist eine wesentliche Verschärfung unserer obigen Aussage zum Gesetz der großen Zahlen.
Eine wichtige Konsequenz ist, dass wir den Fehler des empirischen Mittelwertes (5.4) über seine *Standardabweichung*

$$\boxed{\bar{\sigma} = \sqrt{\frac{1}{N\,(N-1)}\sum_{i=1}^{N}\left(X_i-\bar{X}\right)^2}} \qquad (5.9)$$

schätzen können. Der Korrekturtem -1 in dem Faktor $N-1$ trägt dabei der Tatsache Rechnung, dass wir aus den N Daten X_i bereits eine Größe, nämlich $\bar{X}$ geschätzt haben. Durch Integration der Gaußverteilung kann man leicht schließen, dass die Abweichung des empirischen Mittelwerts $\bar{X}$ vom wahren Mittelwert in 68% aller Fälle maximal $\bar{\sigma}$ ist, in 95% aller Fälle ist die Abweichung kleiner $2\,\bar{\sigma}$ und nur in höchstem einem aus 10^4 Fällen ist sie größer als $4\,\bar{\sigma}$. Es hat sich daher eingebürgert, $\bar{\sigma}$ als Maß für den Fehler zu verwenden und das Ergebnis wie folgt anzugeben:

$$\bar{X} \pm \bar{\sigma}\,.$$

5.3 Erzeugung von Zufallszahlen auf dem Computer

Wir haben bereits angenommen, dass uns Zufallszahlen zur Verfügung stehen. Tatsächlich ist es nicht trivial, „Zufallszahlen“ zu erzeugen. Später werden wir zeigen, dass sich allgemeine Verteilungen $p(x)$ von Zufallszahlen x aus Gleichverteilungen herleiten lassen, so dass wir uns zunächst auf Gleichverteilungen in einem Intervall konzentrieren.
Wir suchen also Folgen von „Zufallszahlen“ x_i mit folgenden Eigenschaften:

(i) Die Zufallszahlen sollen auf einem gegebenen Intervall gleichverteilt sein. Für Fließkommazahlen verwendet man meistens eine Gleichverteilung auf dem

Intervall $[0,1]$, bei ganzen Zahlen ein Intervall $[0,n]$ mit einer festen ganzen Zahl n. Offensichtlich lassen sich Intervalle leicht umrechnen.
Hat man z.B. gleichverteilte ganzzahlige Zufallszahlen $x \in [0,n]$, so ist x/n auf den realisierten Werten in $[0,1]$ gleichverteilt. Es reicht also, sich mit der Erzeugung ganzzahliger Zufallszahlen zu beschäftigen, zumindest solange n hinreichend groß gewählt ist (möglichst in der Gegend des auf dem Rechner darstellbaren Zahlenbereichs).

(ii) Zwischen den einzelnen x_i sollen keine nachweisbaren Korrelationen bestehen, d.h. sind x_0, ..., x_{i-1} bereits bekannt, soll keine Vorhersage für das Ergebnis der Zahl x_i möglich sein.

Bekannte Verfahren zur Erzeugung von „Zufallszahlen“ sind z.B. Würfel oder Roulette. Bei physikalischen Realisierungen mag man u.a. an (radioaktive) Zerfallsprozesse und thermisches Rauschen in elektronischen Schaltkreisen denken. Nun gilt es allerdings zu bedenken, dass moderne Computer Taktfrequenzen im GHz-Bereich aufweisen, so dass man durchaus bei einem Bedarf von $\geq 10^9$ zufälligen Bits in der Sekunde landet. Solche Raten sind schwer zu realisieren, wenn dabei gleichzeitig die Eigenschaften (i) und (ii) erfüllt sein sollen (d.h. Gleichverteilung und fehlende Korrelationen – beides mit hoher Genauigkeit).
Daher beschreitet man meistens einen anderen Weg und verwendet eine „chaotische“ (also möglichst nicht vorhersagbare), aber deterministische Berechnungsvorschrift. Derart erzeugte Zahlenfolgen sind natürlich nicht mehr zufällig, so dass man auch von „Pseudo-Zufallszahlen“ spricht.

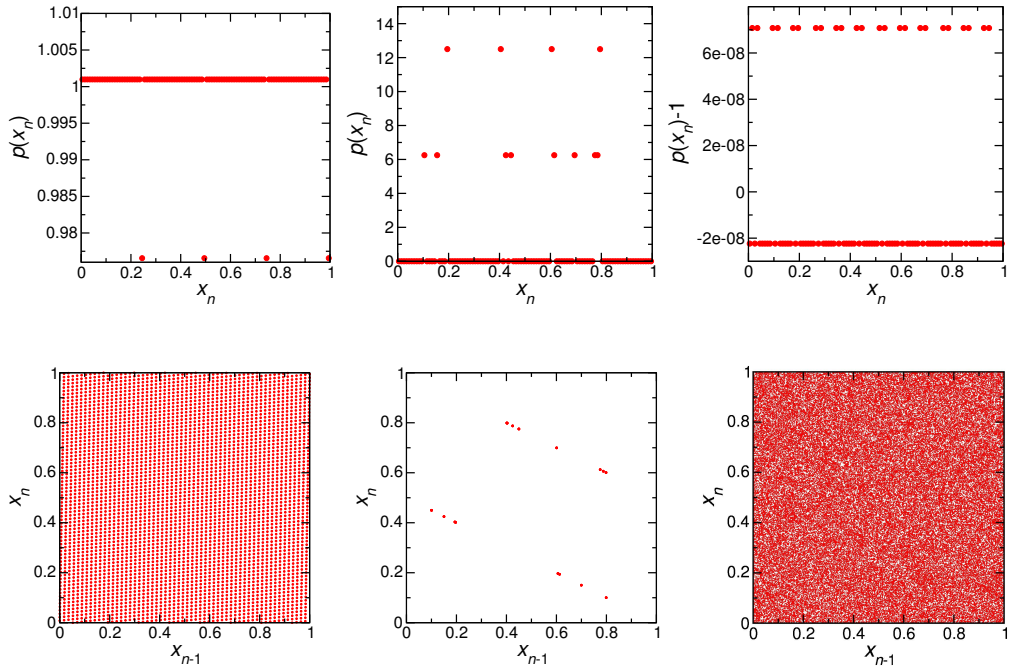
5.3.1 Pseudo-Zufallsgeneratoren

Wir wollen nun Beispiele für Pseudo-Zufallszahlsgeneratoren diskutieren. Wie bereits erwähnt, sind diese durch eine deterministische Berechnungsvorschrift charakterisiert, die ggfs. mit einem Gedächtnis arbeitet.
Dieser Zugang hat natürlich verschiedene Nachteile. Zunächst ist jedes x_i prinzipiell vorhersagbar (insbesondere dann, wenn man die Berechnungsvorschrift kennt). Ferner ist der Informationsgehalt der Berechnungsvorschrift zusammen mit dem Gedächtnis klein gegen eine Folge von echten Zufallszahlen mit einer typisch benötigten Länge. Genauer gesagt ist die einzige Möglichkeit, eine Folge echter Zufallszahlen zu reproduzieren das Speichern der kompletten Folge, d.h. aufgrund der Unvorhersagbarkeit der jeweils nächsten Zufallszahl existiert keine Berechnungsvorschrift, die (deutlich) kürzer ist als die Folge der Zufallszahhlen selbst. Hingegen benötigt man typischerweise nicht besonders viel Speicher, um die Berechnungsvorschrift und das Gedächtnis eines Pseudo-Zufallszahlsgenerators zu speichern. Pseudo-Zufallszahlszahlen sind aus diesen Gründen prinzipiell nicht wirklich zufällig,

so dass man grundsätzlich immer testen muß, ob ein bestimmter Zufallsgenerator für einen gegebenen Zweck brauchbar ist.
Schließlich erzeugt jeder Pseudo-Zufallsgenerator die gleichen Folgenglieder x_i, wenn er sich im gleichen Zustand befindet. Da die Zahl der möglichen Zustände immer endlich ist, sind alle Pseudo-Zufallsgeneratoren periodisch. Für praktische Anwendungen sollte man darauf achten, dass die Periode lang genug ist, d.h. groß gegen die Anzahl der benötigten Zufallszahlen. Auf keinen Fall sollte man mehr Pseudo-Zufallszahlen verwenden als die Periodenlänge des Generators.
Ein positiver Randeffekt ist andererseits, dass die Ergebnisse reproduzierbar sind. Typischerweise befindet sich ein Pseudo-Zufallsgenerator beim Programmstart jeweils im gleichen Zustand, so dass das Verhalten eines Programms reproduzierbar sein sollte. Ein reproduzierbarer Programmablauf ist z.B. wichtig für die Fehlersuche.
Wir wollen nun Beispiele für Pseudo-Zufallsgeneratoren vorstellen. Diese haben eher historischen Charakter und dienen zur Illustration von Problemen dieses Zugangs, sind also nicht als Empfehlung für den praktischen Einsatz zu verstehen.
Als erstes Beispiel wollen wir *linear kongruente Zufallsgeneratoren* diskutieren. Diese Generatoren verwenden eine Rekursionsrelation für ganze Zahlen I_j:

$$I_{n+1} = a\, I_n + c \pmod m \tag{5.10}$$

mit geeignet gewählten ganzzahligen a, c, m. Das „chaotische“ Verhalten wird in dieser Generatorklasse durch die modulo-Operation erzeugt. Je nach Wahl der Parameter können linear kongruente Generatoren gut oder schlecht sein. Erfahrung zeigt, dass die Wahl $c = 0$ bei guter Wahl von a und m keine schlechteren Generatoren liefert als $c \neq 0$. Wir werden uns daher auf den Fall $c = 0$ konzentrieren.
Der Rekursionsrelation (5.10) sieht man leicht an, dass die Zahlenfolge periodisch ist, d.h. es existiert eine Periodenlänge k, so dass $I_{n+k} = I_n$ für alle n ist. Da I_n maximal m verschiedene Werte annehmen kann und I_n ferner alle weiteren Folgenglieder eindeutig bestimmt, ist die Periodenlänge eines linear kongruenten Generators maximal m, d.h. es gilt $k \leq m$. m sollte also möglichst groß gewählt werden.
Wie bereits erwähnt, ist die Güte der Zufallsgeneratoren zu testen. Zunächst sollte man die Gleichverteilung der erzeugten Zufallszahlen überprüfen; kritischer sind jedoch meist Korrelationen in der Folge der erzeugten Zufallszahlen.
Die Gleichverteilung und die Paarkorrelationen $\langle x_n\, x_{n-r} \rangle$ (x_n ist das auf $[0, 1]$ normierte I_n) kann man einem einfachen geometrischen Test unterwerfen: Man zeichnet Punkte, deren x- bzw. y-Koordindaten durch zwei aufeinanderfolgende Zufallszahlen x_{n-1} und x_n gegeben ist. In diesem Test sollte die Ebene gleichmäßig, d.h. ohne erkennbares Muster, ausgefüllt werden.



$a = 57,\ m = 4096,\ c = 33$ | $a = 2^{15} - 1 = 32\,767,\ m = 2^{16} - 1 = 65\,535,\ c = 0$ | $a = 665\,539,\ m = 2^{32},\ c = 0$

Abbildung 5.3: Von dem linear kongruenten Zufallsgenerator erzeugte normierte Zufallszahlen $x_n = I_n/(m-1)$ für drei Wahlen der Parameter: $a = 57$, $m = 4096$, $c = 33$ (linke Spalte), $a = 2^{15} - 1 = 32\ 767$, $m = 2^{16} - 1 = 65\ 535$, $c = 0$ (mittlere Spalte) und $a = 665\ 539$, $m = 2^{32}$, $c = 0$ (rechte Spalte). Die obere Zeile zeigt Histogramme der Wahrscheinlichkeitdichte $p(x_n)$, wobei insgesamt m Zufallszahlen ausgewertet wurden (d.h. eine odere mehrere Perioden). Die untere Zeile zeigt Paare aufeinanderfolgender Zufallszahlen (x_{n-1}, x_n).

Höhere Korrelationen können ebenfalls relevant sein. Die nächste Korrelationsfunktion ist die Tripletkorrelation $\langle x_n\, x_{n-r}\, x_{n-s}\rangle$. Für diese gilt in dem Fall, dass die Folgenglieder unkorreliert sind:

$$\langle x_n\, x_{n-r}\, x_{n-s}\rangle = \langle x_n\rangle\, \langle x_{n-r}\rangle\, \langle x_{n-s}\rangle = \langle x_n\rangle^3 = \left(\frac{1}{2}\right)^3 = \frac{1}{8}\,. \tag{5.11}$$

Man beachte, dass die Annahme von Unkorreliertheit an dieser Stelle insbesondere bedeutet, dass n, $n-r$ sowie $n-s$ paarweise verschieden sind, d.h. (5.11) gilt nur für $r \neq 0$, $s \neq 0$ und $r \neq s$.
Wir wollen nun ein paar Beispiele linear kongruenter Zufallsgeneratoren (5.10) diskutieren:

1. $a = 57$, $m = 4096$, $c = 33$. Das Histogramm der erzeugten Zufallszahlen

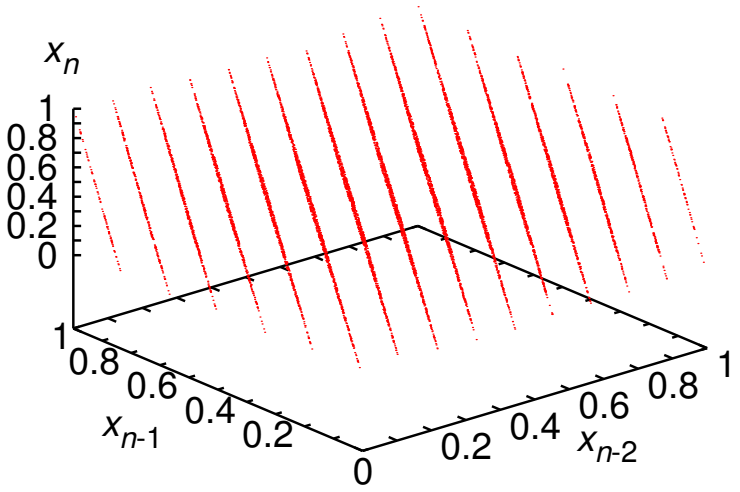


Abbildung 5.4: 10^4 von dem linear kongruenten Zufallsgenerator mit $a = 65\ 539$, $m = 2^{31} = 2\ 147\ 483\ 648$, $c = 0$ erzeugte Triplets (x_{n-2}, x_{n-1}, x_n).

(linke Spalte von Abb. 5.3 oben) sieht zunächst nicht schlecht aus, die Paare aufeinanderfolgender Zufallszahlen (x_{n-1}, x_n) (linke Spalte von Abb. 5.3 unten) liegen jedoch offensichtlich zu regulär in der Ebene.
Nun mag man gegen diesen Generator einwenden, dass nicht nur m vielleicht zu klein ist, sondern auch a noch einmal deutlich zu klein im Vergleich zu m ist. Also versuchen wir es mit einer größeren Wahl von m und einem Wert von a, der näher bei m liegt:

2. $a = 2^{15} - 1 = 32\ 767$, $m = 2^{16} - 1 = 65\ 535$, $c = 0$. Man beachte, dass a und m teilerfremd sind.
   Sowohl das Histogramm (mittlere Spalte von Abb. 5.3 oben) als auch die Paarkorrelation (mittlere Spalte von Abb. 5.3 unten) zeigen, dass dies ein sehr schlechter Generator ist – noch einmal deutlich schlechter als der erste.

3. $a = 65\ 539$, $m = 2^{31} = 2\ 147\ 483\ 648$, $c = 0$ (dieses a ist eine Primzahl).
   Dieser Generator war zeitweise auf IBM-Großrechnern weit verbreitet und die Histogramme bzw. Paarkorrelationen (rechte Spalte von Abb. 5.3) sehen tatsächlich auf den ersten Blick gut aus. Ein Defizit dieses Generators ist allerdings leicht zu sehen: Aufgrund der Mulitplikation mit einer ungeraden Zahl mit anschließender modulo-Bildung mit einer geraden Zahl bleibt das unterste Bit der Folge I_n erhalten, d.h. ein ungerades I_0 führt zu einer Folge, deren Glieder I_n alle ungerade sind, ein gerades I_0 führt hingegen zu einer Folge, deren Glieder I_n ausschließlich gerade sind. Das Hauptproblem

dieses Generators sieht man jedoch in Abb. 5.4: Faßt man drei aufeinander folgende Zufallszahlen x_{n-2}, x_{n-1}, x_n als Koordinaten eines Punktes in drei Dimensionen auf, so findet man nur eine sehr geringe Anzahl von Ebenen. Im allgemeinen gilt, dass die k-Tupel von k aufeinander folgenden mit einem linear kongruenten Generator erzeugten Zufallszahlen auf $(k-1)$-dimensionalen Hyperebenen liegen, wobei es höchstens $\sqrt[k]{m}$ solcher Hyperebenen gibt[3]. Im hier vorliegenden Fall $m = 2^{31}$, $k = 3$ wären im Prinzip etwa $1\,290$ Ebenen möglich – wie man Abb. 5.4 entnimmt, sind es tatsächlich deutlich weniger.

Auch ein guter linear kongruenter Generator hat Probleme. Zunächst sind die niedrigen Bits der I_j weniger zufällig als die hohen (ein besonders deutliches Beispiel hierfür war unser zweites Beispiel). Wird nur ein kleinerer Wertebereich benötigt, sollte man also *nie* die unteren, sondern immer nur die oberen Bits verwenden. Insbesondere sollte man auf die Zufallszahlen keine modulo-Operation anwenden. Ferner ist die Periode vergleichsweise kurz, d.h. man erhält typischerweise maximal von der Ordnung 10^8 verschiedene I_j, danach wiederholt sich die Zahlenfolge.
Schieberegister-Zufallsgeneratoren stellen eine zweite Generator-Klasse dar. Diese Generatoren verwenden ein Register und folgende Rekursionsrelation

$$I_n = I_{n-p} \hat{} I_{n-q}\,. \tag{5.12}$$

Das Symbol ^ kann für unterschiedliche Verknüpfungen stehen; oft wird ein bitweises exklusives oder verwendet.
Schieberegister-Zufallsgeneratoren besitzen verschiedene Vorteile. Zunächst stellt ein größeres Gedächtnis sehr lange Perioden sicher. Ferner sind alle Bits gleich zufällig, wenn die Anfangsbelegung des Registers gut gewählt ist, d.h. wenn die ersten $\max(p, q)$ Zahlen I_n geeignet gewählt werden.
Ein populärer Schieberegister-Generator ist unter dem Namen „R250“ bekannt. R250 ist durch die Parameter $p = 250$ und $q = 147$ definiert. Bei diesem Generator sind die Paarkorrelationen $\langle x_n\, x_{n-1}\rangle$ in Ordnung. Abb. 5.5 zeigt einen Ausschnitt der Tripletkorrelation $\langle x_n\, x_{n-r}\, x_{n-s}\rangle$ für R250. In den meisten Fällen beobachtet man den nach (5.11) erwarteten Wert von 0.125. Allerdings beobachtet man ein deutliches Signal bei $r = p$, $s = q$ (bzw. dem in Abb. 5.5 nicht gezeigten, aber äquivalenten Fall $r = q$, $s = p$): Für R250 findet man einen deutlich erniedrigten Wert $\langle x_n\, x_{n-250}\, x_{n-147}\rangle \approx 0.107$. Aufgrund des Bildungsgesetzes (5.12) ist diese Korrelation bei $r = p$ und $s = q$ nicht überraschend, sie kann aber zu systematischen Abweichungen im Ergebnis einer Simulation führen. Ein besonderes drastisches Beispiel findet sich in F. Schmid, N.B. Wilding, Int. J. Mod. Phys. **C6** (1995) 781-787; hier wurden durch R250 verursachte systematische Fehler im Ergebnis von bis zu 20% beobachtet.

[3] G. Marsaglia, PNAS **61** (1968) 25-28.

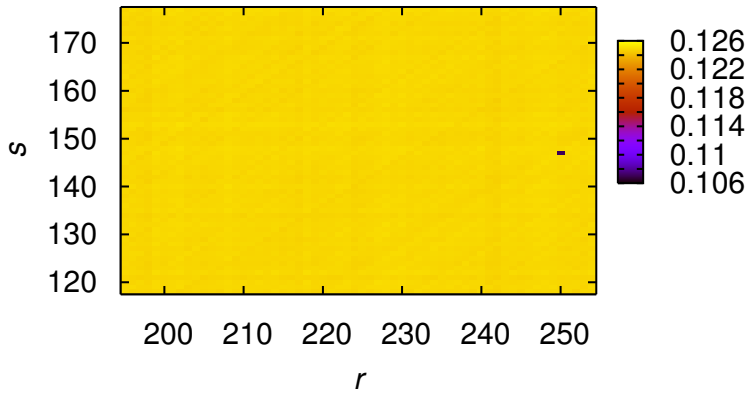


Abbildung 5.5: Werte der Tripletkorrelation $\langle x_n\, x_{n-r}\, x_{n-s}\rangle$ in dem Schieberegister-Zufallsgenerator R250.

Im allgemeinen reicht auch kein „Durcheinanderwirbeln“ der Zufallszahlen durch den Algorithmus, um ein korrektes Ergebnis sicherzustellen. Dies ist spätestens seit der „Ferrenberg-Landau-Affäre“[4] bekannt. In dieser Arbeit wurden Simulationen mit einem Algorithmus durchgeführt, der dem Augenschein nach nicht empfindlich auf vereinzelte Korrelationen reagieren sollte. Dennoch wurden bei verschiedenen zu dieser Zeit beliebten Zufallsgeneratoren statistisch signifikante systematische Fehler im Ergebnis nachgewiesen; auch für R250 wurden bereits in der Arbeit von Ferrenberg, Landau und Wong Probleme berichtet, wobei der Grund für das Versagen des Generators weniger offensichtlich ist als in der späteren Arbeit von Schmid und Wilding.

Es gibt viele andere Pseudo-Zufallsgeneratoren sowie Methoden um Eigenschaften von Zufallsgeneratoren zu „verbesseren“. Man sollte sich dennoch immer daran erinnern, dass kein Pseudo-Zufallsgenerator perfekt sein kann, da er grundsätzlich immer weniger Information enthält als eine (hinreichend lange) Folge echter Zufallszahlen. Für einfache Anwendungen ist meistens der `random()`-Generator aus der C-Bibliothek ausreichend; für anspruchsvollere Anwendungen gibt es Generatoren, die in verschiedenen Bibliotheken implementiert sind, wie z.B. den „Mersenne Twister“.

Ferner sei daran erinnert, dass Pseudo-Zufallsgeneratoren für gewöhnlich nach jedem Programmstart die gleiche „Zufalls“folge liefern. Will man also durch Wiederholung der Simulation die Statistik verbessern, so ist darauf zu achten, dass der Zufallsgenerator beim erneuten Programmstart geeignet initialisiert wird. Dies kann von Hand geschehen, oder z.B. über die Systemzeit[5]. Darüber hinaus stellt

[4] A.M. Ferrenberg, D.P. Landau, Y.J. Wong, Phys. Rev. Lett. **69** (1992) 3382-3384.

[5] Bei einer Abfrage der Systemzeit ist natürlich darauf zu achten, dass evtl. mehrere Abfragen

z.B. Linux unter `/dev/random` einen Pool von Zufallszahlen zur Verfügung. Diese Verwaltung von Zufallszahlen mit Hilfe des Betriebssystems ist in erster Linie für kryptographische Zwecke gedacht, sie kann aber natürlich auch zur Initialisierung eines Pseudo-Zufallsgenerators verwendet werden – für eine direkte Verwendung in Monte-Carlo-Simulationen liefert das Betriebssystem Zufallszahlen allerdings nur mit einer deutlich zu niedrigen Geschweindigkeit, so dass wir hier auf Pseudo-Zufallsgeneratoren angewiesen sind.

5.4 Nichtgleichverteilte Zufallszahlen

Standardisierte Generatoren liefern gleichverteilte Zufallszahlen auf dem Intervall $[0, 1]$. Daher ist eine erste, offensichtliche Aufgabe, andere Verteilungen von Zufallsvariablen zu generieren, die in der Physik (und anderswo) häufig vorkommen. Zwei Methoden sind numerisch brauchbar: Die *Transformationsmethode* und der *von Neumann'sche Rejektionsalgorithmus* („Verwerfungs"-Algorithmus).

5.4.1 Transformationsverfahren

Die Transformationsmethode benutzt die Transformationsformel (5.3) zwischen Wahrscheinlichkeitsdichten. Ist $y = f(x)$ eine eineindeutige Funktion – z.B. monoton – so gilt

$$p_Y(y = f(x)) = \left|\frac{df}{dx}\right|^{-1} p_X(x) \ .$$

Als Beispiel wollen wir *exponentiell* verteilte Zufallszahlen

$$p(t) = \begin{cases} \frac{1}{\tau}\mathrm{e}^{-t/\tau} \text{ für } t \geq 0 \\ 0 \text{ sonst} \end{cases}$$

aus gleichverteilten Zufallszahlen $0 < x < 1$ erzeugen. Es ist also

$$p_X(x) = \begin{cases} 1 \text{ für } 0 < x \leq 1 \\ 0 \text{ sonst} \end{cases}$$

während wir

$$p_Y(y) = \frac{1}{\tau}\mathrm{e}^{-y/\tau}$$

erreichen wollen. Um f zu bestimmen, betrachten wir die Umkehrfunktion

$$x = f^{-1}(y) \ ,$$

so dass

$$p_Y(y) = \left|\frac{df^{-1}}{dy}\right| p_X(x = f^{-1}(y))$$

deutlich länger auseinander liegen als die Zeitauflösung der Uhr.

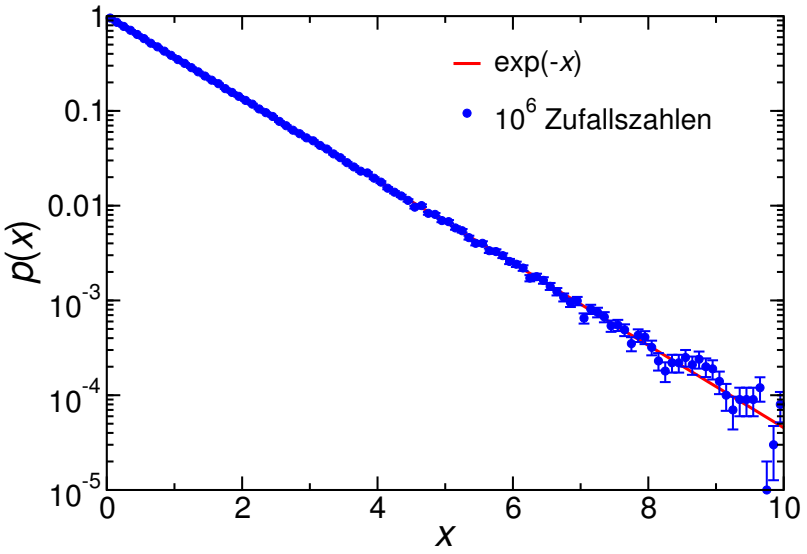


Abbildung 5.6: 10^6 Exponentiell verteilte Zufallszahlen. Man beachte die logarithmische Darstellung der vertikalen Skala.

Also muss

$$\frac{df^{-1}}{dy} = \pm\frac{1}{\tau}\,\mathrm{e}^{-y/\tau}$$

sein, d.h.

$$f^{-1}(y) = \mathrm{e}^{-y/\tau}$$

und somit

$$y = f(x) = -\tau\ln(x) \quad .$$

Der Algorithmus lautet also einfach

1. Ziehe Zufallszahl $0 < x \leq 1$, x gleichverteilt.
2. Bilde $y = -\tau\ln(x)$ für exponentiell verteilte Zufallszahl.
3. Gehe zu 1

Das Histogramm von 10^6 exponential verteilten Zufallszahlen, die nach diesem Algorithmus erzeugt wurden, zeigt Abbildung 5.6. Um genau zu sein, haben wir ein Histogramm erzeugt und den Fehler in jeder Box nach (5.9) geschätzt. Wie man insbesondere bei den größeren Werten von x (d.h. kleinere $p(x)$ und damit auch größerer relativer Fehler) sieht, ist die Abweichung bei grob einem Drittel der Simulationsdaten die Abweichung vom exakten Ergebnis (etwas) größer als der geschätzte Fehlerbalken – genau wie nach dem zentralen Grenzwertsatz erwartet.

Wie man sieht, funktioniert der Transformationsalgorithmus (5.3) immer dann gut, wenn die Umkehrfunktion f^{-1} eine einfache, analytisch handhabbare Funktion ist. Wenn das nicht der Fall ist, dann muß man zum von Neumann'sche Rejektionsalgorithmus greifen.

5.4.2 Box-Muller Algorithmus

Bevor wir den Neumann'schen Rejektionsalgorithmus näher diskutieren, soll der wichtige Spezialfall *Gauß'scher Zufallszahlen*

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-x_0)^2}{2\sigma^2}\right)$$

untersucht werden. Da die gesuchte Verteilung nicht monoton ist, kann der Transformationsalgorithmus nicht direkt implementiert werden. Hier hilft ein kleiner Trick weiter, der darin besteht, *zwei statistisch unabhängige Gauß'sche Zufallsvariablen* zu betrachten. Deren gemeinsame Wahrscheinlichkeitsdichte ist

$$p(x,y)\,dxdy = \frac{1}{2\pi\sigma^2}\,\exp\left(\frac{x^2+y^2}{2\sigma^2}\right)\,dxdy = p(x)\,dx\,p(y)\,dy$$

Führen wir nun *ebene Polarkoordinaten*

$$x = \sqrt{2\rho}\cos\theta \tag{5.13a}$$
$$y = \sqrt{2\rho}\sin\theta \tag{5.13b}$$

mit etwas ungewöhnlicher Radialvariable (*Box-Muller Algorithmus*[6]) ein, so wird die gemeinsame Wahrscheinlichkeitsdichte in diesen Polarkoordinaten zu

$$p(\rho,\theta)d\rho d\theta = \frac{1}{2\pi\sigma^2} = \mathrm{e}^{-\rho/\sigma^2} d\rho d\theta \quad .$$

> Erzeugen wir also mit einer in $(0,1]$ gleichverteilten Zufallszahl α eine exponentiell verteilte Zufallszahl $\rho = -\sigma^2 \ln\alpha$ und ziehen noch eine auf dem Intervall $[0,2\pi]$ gleichverteilte Zufallszahl θ, so transformiert der Box-Muller Algorithmus (5.13a,b) diese in *zwei* gaußverteilte Zufallszahlen der gewünschten Varianz.

Mit 10^6 gezogenen Zufallszahlen findet man die Verteilung in Abbildung 5.7. Man beachte, dass wir auch hier wieder ein Histogramm erzeugt und den Fehler in jeder Box nach (5.9) geschätzt haben. Auch hier sieht man an der im Einsatz von Abb. 5.7 gezeigten Vergrößerung des Bereichs um das Maximum, dass die Abweichung bei grob einem Drittel der Simulationsdaten die Abweichung vom exakten Ergebnis (etwas) größer als der geschätzte Fehlerbalken ist.

[6] Die Namensgebung erinnert an die Autoren der Orignalpublikation: G.E.P. Box, M.E. Muller, Ann. Math. Statist. **29** (1958) 610-611.

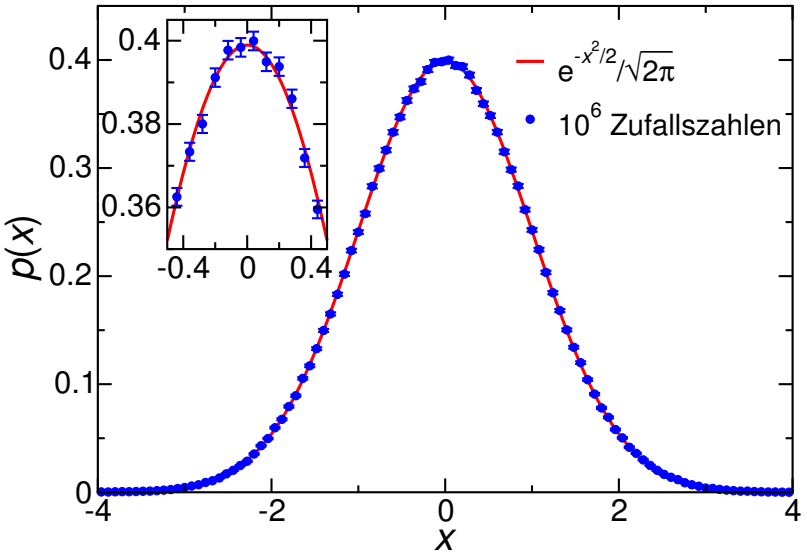


Abbildung 5.7: 10^6 Gauß'sch verteilte Zufallszahlen nach Box-Muller Algorithmus mit Varianz $\sigma^2 = 1$.

5.4.3 Neumann'scher Rejektionsalgorithmus

Sind Tricks wie der Box-Muller Algorithmus nicht in Sicht, bleibt nur noch der *von Neumann'sche Rejektionsalgorithmus*. Er beruht auf der geometrischen Interpretation des $p_X(x)$-Graphen. Die Fläche unter der Kurve $y = p_X(x)$ über einem Intervall $[x - \Delta x/2, x + \Delta x/2]$ ist die Wahrscheinlichkeitsdichte, X in dem betrachteten Intervall anzutreffen. Wir beschränken uns hier auf den einfachsten Fall von Wahrscheinlichkeitsdichten $p_X(x)$, die in einen rechteckigen Kasten eingeschlossen werden können[7]. Sie sind charakterisiert durch $p(x) = 0$ für $x \notin [x_{\min}, x_{\max}]$ und $p(x) \leq p_{\max} < \infty$.

Das Vorgehensweise ist nun in Abb. 5.8 skizziert. Zunächst zieht man im Intervall $[x_{\min}, x_{\max}]$ gleichverteilte Zufallszahlen x_0. Die Anpassung von der Gleichverteilung auf die gewünschte Verteilung $p_X(x)$ erfolgt mit Hilfe einer weiteren im Intervall $[0, 1]$ gleichverteilten Zufallszahl r_2. Die Punkte $(x, r_2\, p_{\max})$ bilden eine gleichverteilte Punktwolke im Kasten $x_{\min} \leq x \leq x_{\max}$, $0 \leq y \leq p_{\max}$. Um Zufallszahlen ξ zu erzeugen, die nach $p_X(x)$ verteilt sind, müssen wir einfach nur die Punkte *akzeptieren* für die $r_2\, p_{\max} < p_X(x)$ ist. Die anderen werden *zurückgewiesen*. Hier der Algorithmus nochmals zusammengefaßt:

[7] Hat die Verteilungsfunktion keinen beschränkten Träger, so muß sie aus Normierungsgründen für $x \to \pm\infty$ hinreichend schnell abfallen. Zur Einschließung kann dann z.B. eine Exponential- oder Gauß-Verteilung nützlich sein.

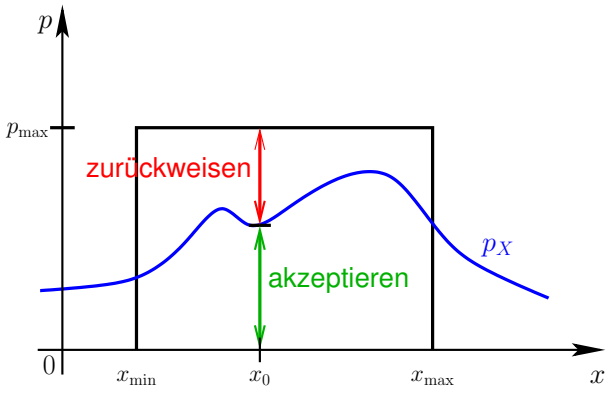


Abbildung 5.8: Illustration der Neumann'schen Verwerfungs- („Rejektions"-) Methode zur Erzeugung einer beliebigen gegebenen Zufallsverteilung $p_X(x)$.

1. Ziehe Standardzufallszahl $0 \leq r_1 \leq 1$ und bilde $x_0 = x_{\min} + r_1\,(x_{\max} - x_{\min})$
2. Ziehe Standardzufallszahl $0 \leq r_2 \leq 1$ und bilde $r_2\,p_{\max}$
3. Wenn $r_2\,p_{\max} \leq p_X(x_0)$, akzeptiere x_0.

Als ein Beispiel betrachten wir eine ganz willkürlich gebastelte Wahrscheinlichkeitsdichte

$$p_X(x) = \begin{cases} 0 & \text{für } x < 0 \text{ und } x > 1.5, \\ 2/3 & \text{für } 0 \leq x < 0.5, \\ 1/3 & \text{für } 0.5 \leq x < 1, \\ 4(x-1) & \text{für } 1 \leq x \leq 1.5. \end{cases} \tag{5.14}$$

Diese können wir mit $x_{\min} = 0$, $x_{\max} = 1.5$, $p_{\max} = 2$ einschließen.
Für dieses Beispiel geben wir auch einen Beispiels-Quellcode an. Zunächst kann man den Neumannschen Rejektions-Algorithmus in C++ wie folgt implementieren:

```
#include <cstdlib>
⋮
double rejectVerteilung(double (&P)(double),
                        double xmin, double xmax, double pmax) {
  while(true) {                          // Enflosschleife
   double r1=random()/double(RAND_MAX);  // Zufallszahl 0<=r1<=1
   double x0 = xmin + r1*(xmax-xmin);    // liefert zufaelliges xmin<=x0<=xmax
   double r2=random()/double(RAND_MAX);  // Zufallszahl 0<=r2<=1
   if(r2*pmax <= P(x0))                  // Mit Zielfunktion vergleichen
```

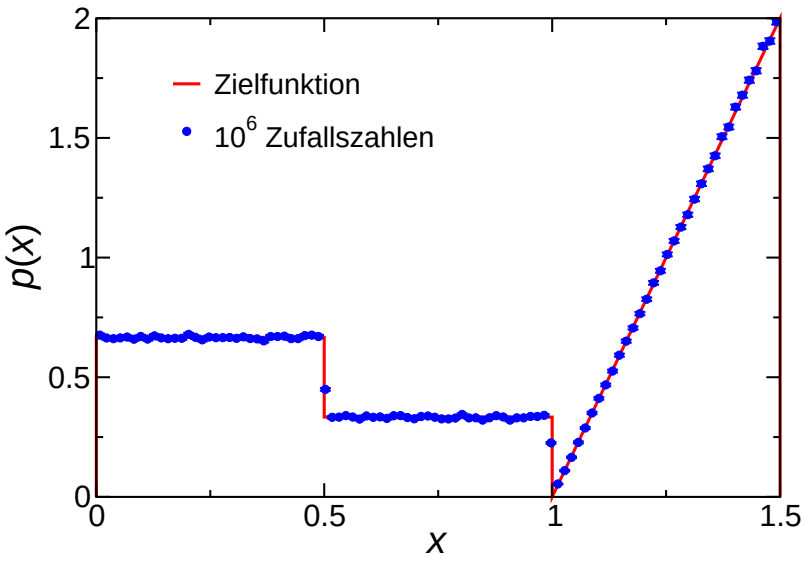


Abbildung 5.9: Beispiel für den Rejektionsalgorithmus.

```
    return(x0);                          // und ggfs. akzeptieren
  }
}
```

(`random()` ist der Zufallsgenerator aus der C-Bibliothek – natürlich kann man hier auch einen anderen einsetzen). Unsere Zielfunktion (5.14) implementieren wir z.B. wie folgt:

```
double ZielFunktion(double x) {
 if((x<0) || (x>1.5))    return(0);
 if(x<0.5)               return(2.0/3.0);
 if(x<1)                 return(1.0/3.0);
 return(4*(x-1));
}
```

Nun erhalten wir mit

```
rejectVerteilung(ZielFunktion, 0, 1.5, 2)
```

eine Zufallszahl, die der gewünschten Verteilung genügt. Nach 10^6 gezogenen Zufallzahlen sieht die Häufigkeitsverteilung so aus, wie in Abbildung 5.9 gezeigt.
Man beachte, dass wir im vorliegenden Beispiel im Mittel 6 gleichverteilte Zufallszahlen ziehen müssen, um eine mit der gewünschten Verteilung zu erhalten. Die Effizienz des Neumann'schen Rejektions-Algorithmus kann also ein Problem werden. Die in 3 Abschnitten definierte Verteilungsfunktion (5.14) hätte man natürlich auch mit einer (zufälligen) Fallunterscheidung per Hand erzeugen können. Ich hoffe jedoch, dass die einfache und generische Implementierung des Rejektions-Algorithmus (mit Übergabe der Zielfunktion und einschließendem Kasten) auch Sie beeindruckt.

Ein zweites Beispiel für einen Rejektionsalgorithmus haben wir im Prinzip bereits in Kapitel 5.1 behandelt als wir in Abb. 5.2 im *Einheitskreis* gleichverteilte Zufallspunkte erzeugt haben.
Mit Verfeinerungen des Rejektionsalgorithmus kann man insbesondere auch die Gamma-, *Poisson-*

$$P(X = k \in \mathbb{N}) = \frac{\lambda^k}{k!}\,\mathrm{e}^{-\lambda}$$

und *Binomial-Verteilung*

$$P^{(N)}(k) = \binom{N}{k} p^k\,(1-p)^{N-k}$$

erzeugen. Zu den Verfeinerungen gehört unter anderem, dass man bei der Poisson- und Binomial-Verteilung eine ursprünglich kontinuierliche Verteilung in eine Verteilung ganzer Zahl konvertieren muß.

5.5 Monte-Carlo-Integration

Die Monte-Carlo- (MC) Integration ist von zentraler Bedeutung für die Physik. Dies liegt nicht nur an ihrer Bedeutung für höherdimensionale Integrale, die z.B. in der Hochenergie-Physik auftreten, sondern auch daran, dass Mittelungen über Konfigurationen in Mehr- bzw. Vielteilchensystemen als hochdimensionale Summen bzw. Integrale interpretiert werden können.

5.5.1 Einfache Monte-Carlo-Integration

Wir wollen noch einmal einen genaueren Blick auf die einfache Monte-Carlo-Integration werfen, die wir eingangs als Motivation für stochastische Methoden diskutiert haben. Glg. (5.2) kann leicht auf ein d-dimesionales Integral verallgemeinert werden:

$$I = \int \mathrm{d}V\, f \approx \frac{V}{N} \sum_{i=1}^{N} f(\vec{\xi}_i)\,, \tag{5.15}$$

wobei die Punkte $\vec{\xi}_i$ zufällig und gleichverteilt in V gewählt werden. Nun wissen wir, dass wir den Fehler von (5.15) nach (5.9) abschätzen können durch (wir nehmen hier an, dass $N \gg 1$)

$$\delta I = \bar{\sigma}_I = V \sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}\,. \tag{5.16}$$

(Zur Erinnerung: $I = V\,\langle f\rangle$). Mit diesem Ergebnis können wir durch Betrachtung des asymptotischen Verhalten für große N diskutieren, wann eine MC-Integration den bekannteren Gitterverfahren überlegen ist.

Betrachten wir zunächst ein *Gitter-Verfahren* $n$ter Ordnung[8] in d Dimensionen. Haben wir insgesamt N Stützstellen zur Verfügung, so erzwingt eine reguläre Anordnung in d Dimensionen einen mittleren Abstand

$$\Delta x \propto N^{-\frac{1}{d}} . \tag{5.17}$$

Für den Fehler des Integrals I folgt nach Definition und mit der Überlegung (5.17)

$$\delta I \propto \Delta x^{n+1} \propto N^{-\frac{n+1}{d}} . \tag{5.18}$$

Andererseits gilt für den Fehler des *einfachen Monte-Carlo-Verfahrens* mit N Stützstellen gemäß (5.16)

$$\delta I \propto N^{-\frac{1}{2}} . \tag{5.19}$$

Man beachte, dass der Fehler in diesem Fall unabhängig von der Dimension d des Integrationsgebiets ist.
Durch Vergleich von (5.18) und (5.19) findet man, dass der MC-Fehler schneller abfällt als der des Gitter-Verfahrens, wenn

$$\frac{1}{2} > \frac{n+1}{d} \qquad \Leftrightarrow \qquad d > 2\,(n+1) \tag{5.20}$$

ist. Die MC-Integration ist also vor allem in hohen Dimensionen effizienter.
Neben der asymptotischen Betrachtung gilt zu beachten, dass in hohen Dimensionen bei einem Gitterverfahren nur sehr wenige Gitterpunkte je Richtung zur Verfügung stehen. Betrachten wir z.B. $d = 10$ und fordern, dass insgesamt höchstens $N \leq 10^9$ Stützstellen verwendet werden sollen, so sind maximal 7 Gitterpunkte je Richtung realisierbar. Bei einem enstpechend großen Δx ist man normalerweise nicht im asymptotischen Bereich und das Gitterverfahren verhält sich schlechter als aufgrund der asymptotischen Überlegung (5.18) zu erwarten wäre. Deswegen sind MC-Verfahren oft auch in niedrigeren Dimensionen überlegen als die Abschätzung (5.20) nahe legt.
Zusammenfassend ergeben sich als Anwendungsgebiete für die MC-Integration vor allem hohe Dimensionen. Dies entspricht in der Physik vielen Freiheitsgraden.
Zur Illustration betrachten wir folgendes D-dimensionales Integral

$$I(D) := \int_{-1}^{1} \mathrm{d}x_1 \cdots \int_{-1}^{1} \mathrm{d}x_D \exp\left\{ -\sum_{i=1}^{D} x_i^2 \right\} .$$

Natürlich ist dieses Integral trivial, da es offensichtlich in ein Produkt von D Gauß-Integralen faktorisiert:

$$I(D) = I(1)^D \approx 1.493648266^D .$$

[8] Für die naive Riemann-Formel (5.1) gilt $n = 0$, die Simpson-Formel –eine verbreitete numerische Integrationsformel– ist charakterisiert durch $n = 3$.

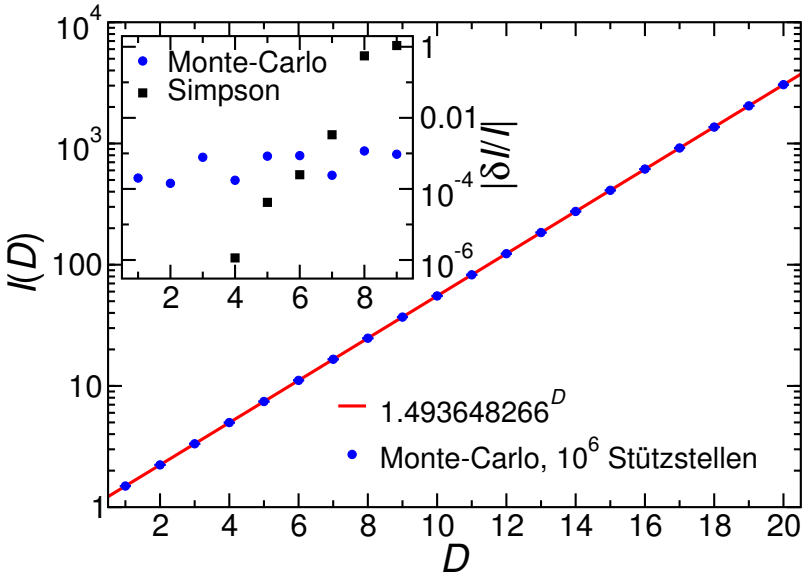


Abbildung 5.10: Monte-Carlo-Integration für D-dimensionale Gaußintegrale mit $N = 10^6$ Stützstellen. Der Einsatz zeigt den tatsächlichen Monte-Carlo Fehler $|\delta I/I|$ im Vergleich zum Fehler des Simpson-Verfahrens auf einem regulären D-dimensionalen Gitten mit $N \leq 10^6$ Stützstellen. Man beachte die logarithmischen Skalen der vertikalen Achsen, insbesondere beim Einsatz.

Abbildung 5.10 zeigt das Ergebnis für $I(D)$ bis $D = 20$, berechnet mit jeweils 10^6 Zufallsvektoren. Wie man z.B. am Einsatz von Abb. 5.10 sieht, liegt der relative Monte-Carlo Fehler zwischen $\approx 2 \cdot 10^{-4}$ (für $D = 1$) und $\approx 2 \cdot 10^{-3}$ (für $D = 20$); die Fehlerschätzung nach (5.16) ist konsistent mit den beobachteten Abweichungen.

Der Einsatz von Abb. 5.10 zeigt ferner den Fehler, den man bei der Auswertung mit der Simpson-Formel erhält, wenn man ein reguläres D-dimensionales Gitter mit maximal 10^6 Punkten verwendet. Für $D \leq 3$ ist das Ergebnis der Simpson-Regel sehr genau und der Monte-Carlo-Integration deutlich überlegen. Allerdings wächst der Fehler des Gitter-Verfahrens schnell mit der Dimension, so dass das Monte-Carlo-Ergebnis für $D > 6$ genauer ist. Man beachte, dass wir mit dem Simpson-Verfahren ein Verfahren hoher Ordnung eingesetzt haben ($n = 3$). Die asymptotische Analyse (5.20) zeigt, dass das Monte-Carlo-Verfahren dem Simpson-Verfahren für $D > 8$ überlegen sein sollte; tatsächlich gilt dies im vorliegenden Fall bereits ab $D = 7$. Ferner gilt zu beachten, dass das Simpson-Verfahren mindestens 4 Stützstellen je Dimension benötigt, um sinnvoll zu arbeiten, so dass für $N \leq 10^6$ nur $D < 10$ zugänglich ist. Aus diesem Grund haben wir uns im Einsatz von Abb. 5.10 auf $D \leq 9$ beschränkt. Im Vergleich mit der einfachen Riemann-Formel (5.1) würde sich der Bereich, in dem die MC-Methode überlegen ist, zu kleineren

D verschieben. Ferner sei angemerkt, dass die Rechenzeit für die MC-Methode bei fester Zahl der Stützstellen N nur langsam mit D zunimmt.
Es sei noch eine Bemerkung ergänzt, nämlich dass die MC-Integration für ein konstantes $f(\vec{x})$ exakt ist. Formal folgt dies z.B. aus der für konstantes f gültigen Identität $\langle f^2\rangle = \langle f\rangle^2$, die mit (5.16) auf $\delta I = 0$ führt. Diese Beobachtung kann zur Fehlerreduktion verwendet werden. Kann z.B. eine Näherung an f analytisch integriert werden, so kann ein verbleibender Korrekturterm mit einem deutlich kleineren MC-Fehler berechnet werden. Dies kann z.B. im Sinne einer Variablensubstitution, Sampling relativ zu einer Wahrscheinlichkeitsverteilung $p(\vec{x})$, die $f(\vec{x})$ gut nähert, oder gemäß dem im nächsten Unterkapitel diskutierten Verfahren umgesetzt werden.

5.5.2 Verbesserte Monte-Carlo-Integration

Wir haben gesehen, dass der Fehler einer MC-Integration einer Funktion f durch die Varianz $\sigma^2(f) = \langle f^2\rangle - \langle f\rangle^2$ kontrolliert wird. Jede Verbesserung einer MC-Integration zielt daher auf eine Minimierung von $\sigma^2(f)$. Dies kann mit verschiedenen Strategien erzielt werden. Wir wollen als ein erstes Beispiel zunächst eine als „Stratified Sampling“ bekannte Strategie vorstellen.
Man teilt zunächst das Integrationsvolumen V in zwei *gleich große* Teilvolumina A und B auf. Die Mittelwerte sind dann

$$\langle f\rangle = \frac{1}{V}\int \mathrm{d}V\, f\,, \qquad \langle f\rangle_{A/B} = \frac{2}{V}\int\limits_{A/B} \mathrm{d}V\, f\,, \tag{5.21}$$

und die Varianzen lauten

$$\sigma^2(f) = \langle f^2\rangle - \langle f\rangle^2\,, \qquad \sigma^2_{A/B}(f) = \langle f^2\rangle_{A/B} - \langle f\rangle^2_{A/B}\,. \tag{5.22}$$

Durch Einsetzen und Umformen findet man

$$\begin{aligned}
\sigma^2(f) &= \langle f^2\rangle - \langle f\rangle^2 = \frac{1}{2}\left(\langle f^2\rangle_A + \langle f^2\rangle_B\right) - \frac{1}{4}\left(\langle f\rangle_A + \langle f\rangle_B\right)^2 \\
&= \frac{1}{2}\left(\sigma^2_A(f) + \sigma^2_B(f)\right) + \frac{1}{2}\langle f\rangle^2_A + \frac{1}{2}\langle f\rangle^2_B - \frac{1}{4}\left(\langle f\rangle_A + \langle f\rangle_B\right)^2 \\
&= \frac{1}{2}\left(\sigma^2_A(f) + \sigma^2_B(f)\right) + \frac{1}{4}\left(\langle f\rangle_A - \langle f\rangle_B\right)^2\,.
\end{aligned} \tag{5.23}$$

Die Varianz bei der Mittelung über das Gesamtvolumen V unterscheidet sich also von der Summe der Varianzen über die Teilvolumina A und B lediglich durch einen positiven Korrekturterm!
Man kann nun zur Berechnung des Integrals zwei Strategieen betrachten. Erstens kann man N zufällige $\vec{\xi}_i \in V$ zum Schätzen des Integrals wählen. Zweitens kann man je $N/2$ zufällige $\vec{\xi}_i \in A$ und $\vec{\xi}_i \in B$ verwenden. Das Ergebnis (5.23) zeigt,

dass die Summe der Varianzen der zweiten Strategie nie größer ist als die Varianz der ersten Methode.
Es lohnt sich also im Zweifelsfall, das Integrationsgebiet in Teilgebiete aufzuteilen. Qualitativ ist dies auch einsichtig: Nach der Aufteilung wird f im allgemeinen in den Teilgebieten A und B jeweils weniger stark variieren als im Gesamtgebiet V, so dass man näher an dem idealen Fall eines konstanten f liegt, für das die MC-Integration exakt wird.
Die Unterteilung in Teilvolumina kann weiter geführt werden, solange in jedem Teilvolumen hinreichend viele Meßpunkte für eine sinnvolle Statistik verbleiben. Ferner kann man die Zahl der Meßpunkte N_U in einem Teilvolumen U abhängig von U wählen. Man kann zeigen, dass die *optimale Wahl*

$$N_U \propto \sigma_U(f) \tag{5.24}$$

ist. Dies bedeutet insbesondere, dass man mehr Meßpunkte in den Bereichen verwendet, in denen sich f sich am stärksten ändert. In Bereichen stark variierenden f wird der Fehler somit durch besonders viele Meßpunkte reduziert, in Bereichen mit schwachen Variationen von f führt hingegen auch eine geringere Anzahl von Meßpunkten zu einem hinreichend kleinen Fehler.

Kapitel 6

Anwendungen Teil II: Monte-Carlo

Das Ziel einer statistischen Beschreibung eines Problems ist recht allgemein die Berechnung von Erwartungswerten

$$\langle f \rangle = \frac{\int \mathrm{d}^d x \, p(\vec{x}) \, f(\vec{x})}{\int \mathrm{d}^d x \, p(\vec{x})} \, . \tag{6.1}$$

Die Vektoren $\vec{x}$ stehen symbolisch für Elemente eines „Konfigurationsraums“, der durchaus (insbesondere bei einem Vielteilchenproblem) eine hohe Dimension haben kann; $p(\vec{x}) \geq 0$ ist eine gegebene (nicht notwendig normierte) „Wahrscheinlichkeitsdichte“ und die Funktion $f(\vec{x})$ steht für eine „Observable“ in der Konfiguration $\vec{x}$. Bei (6.1) handelt es sich im Prinzip um ein hochdimensionales Integral und folglich sind gemäß Unterkapitel 5.5 Monte-Carlo (MC) Methoden ideal, um solche Probleme anzugehen.
Genauer können wir zwei Typen unterscheiden:

1. $p(\vec{x})$ ist flach oder gar konstant ($p(\vec{x}) = p_0$). In diesem Fall ist die typische Herausforderung entweder die Erzeugung der Konfigurationen $\vec{x}$ oder die Auswertung der Observablen $f(\vec{x})$. Beispiele für diese Problemklasse sind geometrische Probleme, insbesondere die Zufallswege, die Sie in den Übungen behandelt haben, oder die Perkolation, auf die wir gleich eingehen wollen. Im Fall der Zufallswege haben Sie gesehen, dass die effiziente Erzeugung von Konfigurationen schwierig wird, wenn man Selbstüberschneidungen verbietet; im Fall der Perkolation wird die Berechnung der Observablen f die Herausforderung darstellen.

2. $p(\vec{x})$ variiert stark. Dies ist der typische Fall bei Anwendungen in der Thermodynamik. Hier ist die typische Herausforderung die effiziente Summation der Beiträge *wichtiger* $\vec{x}$. Auf diesen Problemtypus werden wir etwas später eingehen.

6.1 Perkolation

Bei Perkolations-Problemen[1] handelt es sich ganz allgemeinen um Modelle für das Eindringen einer Flüssigkeit in ein poröses Medium. In den Übungen wollen wir den Fall untersuchen, dass die Flüssigkeit entlang einer Vorzugsrichtung fließt („gerichtete Perkolation“ – siehe auch Anhang B), während wir uns hier mit dem komplizierteren Fall beschäftigen, dass es keine Vorzugsrichtung gibt.

[1] Das Wort „Perkolation“ leitet sich vom lateinischen „percolare“ ab, was durchseihen bzw. durchsickern bedeutet. Heutzutage eher bekannt ist wahrscheinlich der angelsächsische „percolator“ – die Kaffeemaschine.

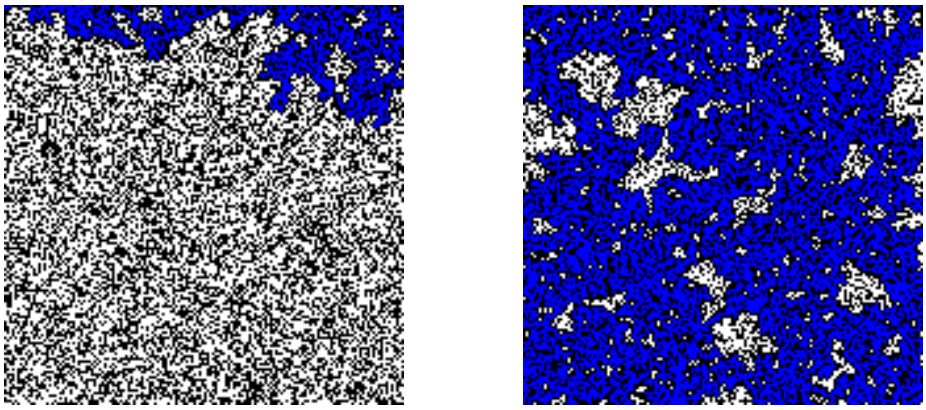


Abbildung 6.1: Zwei Konfigurationen von Platzperkolation auf einem 152×152 Quadratgitter. Weisse Plätze sind durchlässig, schwarze hingegen nicht. Die Dichte durchlässiger Plätze ist $p = 0.55$ (links) bzw. $p = 0.6$ (rechts). Flüssigkeit (blau) dringt vom oberen Rand in das System ein.

Wir betrachten als Modell für das poröse Medium ein reguläres Gitter, z.B. ein Quadratgitter. Den Plätzen dieses Gitters werden Zellen zugeordnet, die Verbindungen zu ihren Nachbarn besitzen.
Die Porösität des Mediums wird dadurch abgebildet, dass mögliche Wege nur mit einer gewissen Wahrscheinlichkeit tatsächlich durchlässig sind. Es gibt zwei Varianten des Modells:

- Platz („Site“) Perkolation: Die Plätze des Gitters sind mit Wahrscheinlichkeit p durchlässig (die Verbindungen immer).
- Verbindungs („Bond“) Perkolation: Die Verbindungen zwischen den Gitterplätzen sind mit Wahrscheinlichkeit p durchlässig (die Plätze selbst hingegen immer).

Bei dem hier diskutierten Fall isotroper Perkolation kann die Flüssigkeit von einem nassen Gitterpunkt entlang durchlässiger Wege zu *allen* Nachbarn vordingen.
Ist p klein, so kann die Flüssigkeit nicht besonders tief in das Medium eindringen. Die linke Tafel von Abb. 6.1 illustiert diesen Fall auf dem Quadratgitter für ein bereits relativ großes $p = 0.55$. Im Grenzfall $p \to 1$ sind hingegen alle Wege offen, so dass das Medium auf jeden Fall durchlässig wird. Obwohl in der rechten Tafel von Abb. 6.1 $p = 0.6$ nur geringfügig größer ist als bei der linken Tafel, so ist diese Realisierung von Platzperkolation auf dem Quadratgitter offensichtlich durchlässig. Perkolationstheorie beschäftigt sich u.a. mit der Frage, mit welcher Wahrscheinlichkeit das System durchlässig ist.

Das Erzeugen von Konfigurationen ist nicht besonder schwer: man muß einfach für jeden Platz oder jede Verbindung entscheiden, ob dieser oder diese durchlässig oder blockiert ist. Dies geschieht durch Auswürfen einer Zufallszahl $r \in [0, 1]$. Ist $r \leq p$, so wird der Platz bzw. die Verbindung als durchlässig markiert, andernfalls als blockiert. Fragt man hingegen, ob eine Verbindung von oben nach unten existiert, so ist dies eine Frage nach der Konnektivität. Dies ist ein kompliziertes geometrisches Problem. Natürlich kann man dieses Problem zumindest für kleine Gitter lösen; schließlich ist es mit dem Ausfüllproblem der Computergrafik verwandt.

6.1.1 Hoshen-Kopelman-Algorithmus

Ein effizienter Algorithmus zur Lösung des Perkolationsproblems wurde von Hoshen und Kopelman vorgeschlagen[2]. Dieser Algorithmus identifiziert alle Zusammenhangskomponenten, sogenannte „Cluster“ in *einem* zeilweisen Durchgang durch das System. Dies ermöglicht es auch, sich auf die Speicherung einzelner Zeilen zu beschränken. Der Algorithmus basiert auf der Verwendung von Labels für die Cluster, die wir mit M bezeichnen wollen, und einer Liste $N(M)$.

Betrachte zeilenweise alle Plätze sowie deren erreichbaren Nachbarn unter den bereits besuchten Plätzen. Geht man in zwei Dimensionen von oben nach unten sowie von links nach rechts durch, sind also der linke und obere Nachbar zu betrachten. Bei Platz-Perkolation muß man die Betrachtung auf die durchlässigen Plätze beschränken. Nun sind drei Fälle zu unterscheiden:

1. Ist kein Platz erreichbar, erzeuge ein neues Label für den aktuellen Platz.
2. Ist *ein* Platz erreichbar, so übernimm dessen Label.
3. Sind zwei (oder mehr) Plätze erreichbar, so übernimm das *kleinste* Label und setze die größeren Labels gleich dem kleinsten.

Das Gleichsetzen der Labels im dritten Schritt sorgt dafür, dass zwei Teile eines Clusters zusammengefügt werden, sobald eine Verbindung zwischen ihnen gefunden wird und löst auf diese Weise das nicht-lokale Problem, die Konnektivität zu bestimmen. Dieser dritte Schritt kann z.B. wie folgt implementiert werden:

- Ist $N(M) = M$, so handelt es sich bei M um ein gültiges Label.

[2] J. Hoshen, R. Kopelman, Phys. Rev. **B14** (1976) 3438-3445.

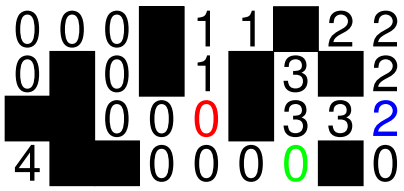


Abbildung 6.2: Illustration des Hoshen-Kopelman-Algorithmus: Labels werden bei einem Zeilenweisen Durchgang zugewiesen, wobei man oben anfängt und dann die Zeilen jeweils von links durchgeht.

- Ist das Label M mit einem Label M' zu identifizieren, so setzt man $N(M) = M'$. Das zugehörige gültige Label findet man, indem man $M'' = N(M')$ bestimmt und solange weiter in der Liste nachsieht, bis $N(M^{(n)}) = M^{(n)}$ gilt. $M^{(n)}$ ist dann das zu M gehörige gültige Label.

Der Perkolationsfrage ist nun leicht zu beantworten. Eine gegebene Realisierung ist nämlich genau dann durchlässig, wenn ein Label aus der obersten Zeile des Systems auch in der untersten Zeile auftritt.
Zur Verdeutlichung betrachten wir das Beispiel in Abb. 6.2 für Platzperkolation auf dem Quadratgitter. Labels werden jeweils zeilenweise den Plätzen zugewiesen. Hat ein Platz keinen Vorgänger (Fall 1), so erzeugen wir ein neues Label. Ist genau ein bereits besuchter Nachbar erreichbar (Fall 2), so übernehmen wir dessen *gültiges* Label. Manchmal, an den farbig markierten Stellen, finden wir zwei Nachbarn (Fall 3), die wir bisher unterschiedlichen Zusammenhangskomponenten zugeordnet hatten. Hier müssen wir das kleinere gültige Label übernehmen, und das größere Label gleich dem kleineren setzen. Im konkreten Fall, haben wir hierzu nacheinander in der Liste die Setzungen $N(1) = 0$, $N(3) = 2$ und $N(2) = 0$ vorgenommen.
Für viele Anwendungen reicht es, die vorhergende sowie die aktuelle Zeile zu speichern, d.h. man muß nicht den kompletten Systemzustand speichern.
Wir geben eine Beispiels-Implementierung für Platzperkolation auf einem $L \times L$ Quadratgitter an:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int *N;                          // Labels muessen global sichtbar sein
const int blocked=-1;            // Markierung fuer blockiert

// Bestimme ein gueltiges Label und komprimiere gleichzeitig den Pfad
int get_valid(int M) {
   if(N[M] == M)                 // gueltiges Label ?
```

```
    return(M);                  // => zurueckgeben
  N[M] = get_valid(N[M]);       // gueltiges Label suchen, merken
  return(N[M]);                 // und zurueckgeben
 }

// Hoshen-Kopelman-Algorithmus: gibt 1 zurueck, wenn System perkoliert,
// 0 sonst
// Argumente: Besetzungswahrscheinlichkeit p, Kantenlaenge L
int hk(double pval, int L) {
  // 2 Zeilen erzeugen
  int *zeile[2];
  zeile[0] = new int[L];
  zeile[1] = new int[L];
  // Liste fuer Label-Zuweisungen erzeugen
  N = new int[L*long(L)];       // Grosszuegige Groesse

  int nlabels = 0;              // Zaehler fuer Labels
  int lastzero;                 // Letztes Label in 1. Zeile

  // Vorgaengerzeile auf blockiert setzen
  for(int col=0; col<L; col++)
    zeile[0][col] = blocked;
  int alt=0;                    // Alte Zeile
  int neu=1;                    // Neue Zeile

  // Zeilenweise durchgehen
  for(int row=0; row<L; row++) {       // Zeile
   for(int col=0; col<L; col++) {      // Spalte
      int oben = zeile[alt][col];
     int links = -1;                           // Linker Nachbar
     if(col)                                   // wenn vorhanden
        links = zeile[neu][col-1];             // uebernehmen
     if(random()/double(RAND_MAX) <= pval) {   // aktueller Platz leitfaehig ?
       if(oben >= 0) {                         // oben nass ?
         if(links >= 0) {                      // beide nass ?
           int M1=get_valid(oben);
           int M2=get_valid(links);
           if(M1 < M2)                         // Kleineres Label uebernehmen
             zeile[neu][col] = N[M2] = M1;
           else
             zeile[neu][col] = N[M1] = M2;
          }
         else
           zeile[neu][col] = get_valid(oben);
```

```
        }
       else {
         if(links >= 0)                        // Nur links erreichbar
           zeile[neu][col] = get_valid(links);
         else {                                // Neues Label erzeugen
           zeile[neu][col] = N[nlabels] = nlabels;
           nlabels++;
          }
        }
      }
     else
       zeile[neu][col] = blocked;              // blockiert
    }
   if(! row)
     lastzero = nlabels-1;  // Letztes gueltiges Label in erster Zeile merken
   alt = 1-alt;             // alt und neu
   neu = 1-neu;             // vertauschen
  }

 // Teste, ob ein Label aus der ersten Zeile in letzter auftaucht
 int percolates = 0;
 for(int col=0; col<L; col++)
  if(zeile[alt][col] >= 0)            // Besetzt ?
   if(zeile[alt][col] <= lastzero)    // und im Bereich 1. Zeile ?
      percolates = 1;

 delete[] N;                   // Labels freigeben
 delete[] zeile[1];            // Zeile 1 freigeben
 delete[] zeile[0];            // Zeile 0 freigeben

 return(percolates);
}

int main(int argc, const char *argv[]) {
 if(argc != 3) {
   cerr << "Verwendung:\n\t" << argv[0]
        << " p L" << endl;
   exit(13);
  }
 double p = atof(argv[1]);
 int L = atoi(argv[2]);
 int flag = hk(p, L);
 if(flag == 1)
   cout << "Realisierung mit p=" << p
```

```
          << ", L=" << L << " leitet" << endl;
  else
    cout << "Realisierung mit p=" << p
          << ", L=" << L << " ist blockiert" << endl;
}
```

Bemerkungen zum Programmbeispiel:

- Die Implementierung ist CPU-Zeit- und Speicher-effizient. In der vorliegenden Form kann man sie problemlos für $L \leq 10\ 000$ (d.h. bis zu 10^8 Plätze) verwenden.
- Eine rekursive „Pfadkomprimierung“ in `get_valid()` sorgt dafür, dass die Liste `N[]` möglichst schnell auf das gültige Label verweist, man also nicht lange suchen muß.
- Die Dimensionierung der Liste `N[]` am Anfang von `hk()` ist großzügig ausgelegt. Optimierung auf die tatsächlich benötigte Anzahl von Labels erlaubt eine Verwendung des Codes für Systeme mit $L = 60\ 000$ (also $3.6 \cdot 10^9$ Plätze).
- Für ernsthafte Simulationen sollte man den `random()`-Generator aus der C-Bibliothek durch einen „besseren“ Generator ersetzen, der insbesondere eine größere Periode aufweist.

6.1.2 Ergebnisse

Wir wollen nun den Hoshen-Kopelman-Algorithmus anwenden, um Eigenschaften von Platzperkolation auf dem Quadratgitter zu untersuchen. Hierzu formalisieren wir zunächst die Aussage, ob das System leitfähig ist:

> Definition: In einem endlichen System der Kantenlänge L gibt es mit *Wahrscheinlichkeit* $P_L(p)$ eine Verbindung von einem (z.B. dem oberen) zum gegenüberliegenden (z.B. dem unteren) Rand.

Jede einzelne Realisierung mit Besetzungswahrscheinlichkeit p ist natürlich entweder leitfähig oder blockiert. Durch Mittelung über hinreichend viele zufällige Realisierungen kann man jedoch $P_L(p)$ schätzen. Abbildung 6.3 zeigt das Ergebnis für Platzperkolation auf dem Quadratgitter. Die Beschränkung auf Kantenlängen $L \leq 1024$ ergibt sich hier primär durch die Anzahl der Realisierungen (wir haben jeweils mindestens 1000 Realisierungen betrachtet), um ein hinreichend genaues rgebnis für $P_L(p)$ zu erhalten.

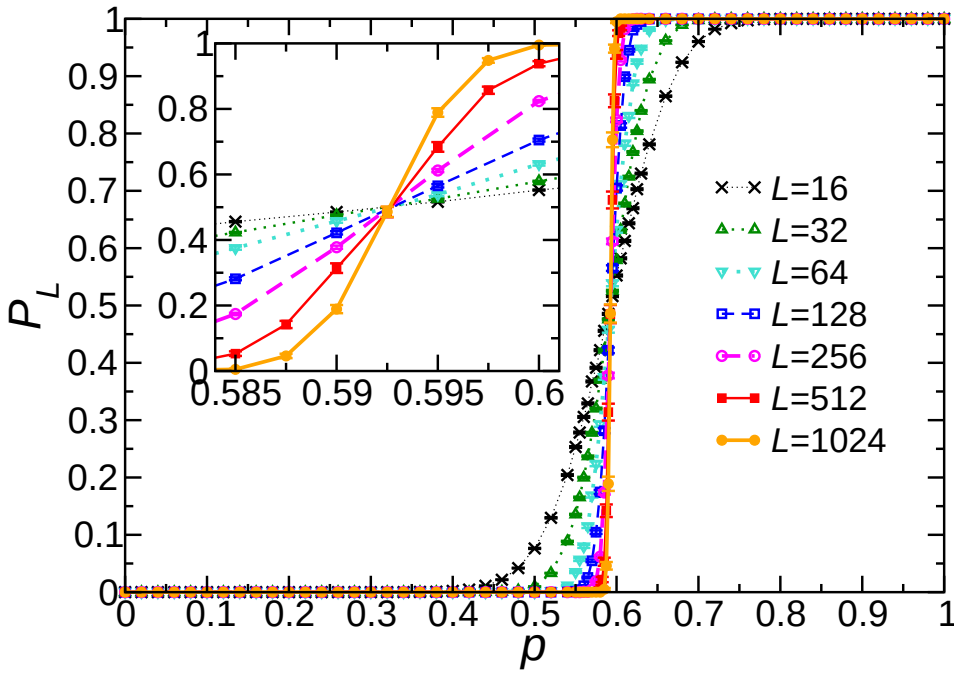


Abbildung 6.3: Wahrscheinlichkeit, dass gegenüberliegende Ränder bei Platzperkolation auf einem $L \times L$ Quadratgitter verbunden sind, für verschiedene Kantenlängen L und als Funktion der Platz-Besetzungswahrscheinlichkeit p.

Wir beobachten, dass sich die Kurven für wachsende Systemgrößen immer mehr einer Stufenfunktion annähernd. Dies liegt die Existenz einer „Perkolationsschwelle“ p_c nahe, so dass

$$\begin{aligned} \lim_{L\to\infty} P_L(p) &= 0 \qquad \text{für } p < p_c\,. \\ \lim_{L\to\infty} P_L(p) &= 1 \qquad \text{für } p > p_c\,. \end{aligned} \tag{6.2}$$

Dies bedeutet insbesondere, dass für $p < p_c$ (bzw. $p > p_c$) und hinreichend großes L eine *einzelne* Realisierung mit an Sicherheit grenzender Wahrscheinlichkeit leitfähig (bzw. blockiert) ist[3].

Da die Kurven $P_L(p)$ mit zunehmendem L immer steiler werden, müssen sich Kurven für zwei verschiedene Kantenlängen L_1 und L_2 in der Nähe von p_c schneiden, d.h. $P_{L_1}(p_{1,2}) = P_{L_2}(p_{1,2})$ mit $p_{1,2} \approx p_c$. Dem Einsatz der Abbildung 6.3 entnimmt man, dass sich tatsächlich alle Kurven innerhalb der numerischen Genauigkeit in *einem* Punkt schneiden. Dies kann man verwenden, um

[3] Die Aussage, dass eine Perkolationsschwelle p_c existiert, so dass (6.2) gilt, kann mathematisch rigoros bewiesen werden, siehe z.B. R. Langlands, P. Pouliot, Y. Saint-Aubin, Bull. Amer. Math. Soc. **30** (1994) 1-61.

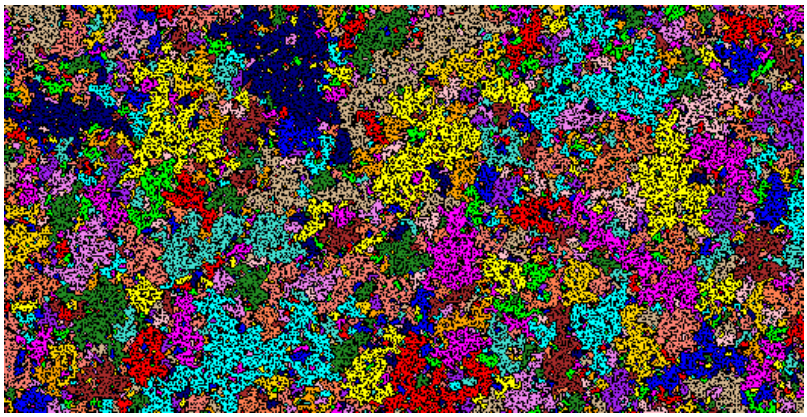


Abbildung 6.4: Ausschnitt einer Konfiguration von Platzperkolation mit $p = 0.55$ auf einem Quadratgitter. Verschiedene Zusammenhangskomponenten („Cluster“) sind mit unterschiedliche Farben gekennzeichnet.

$$p_c = 0.5926 \pm 0.0002 \qquad (6.3)$$

für Platzperkolation auf dem Quadratgitter zu schätzen. Der Fehlerbalken ist hierbei nicht rigoros bestimmt, sondern eher eine grobe Abschätzung für die durch statische Fehler bedingte Ungenauigkeit[4].

Werfen wir noch einmal einen genaueren Blick auf das Verhalten für $p < p_c$ und betrachten hierzu die in Abbildung 6.4 gezeigt Beispielskonfiguration mit $p = 0.55$ für Platzperkolation auf einem Quadratgitter. Augenscheinlich existieren keine sehr großen Cluster, d.h. die Cluster besitzen für $p < p_c$ eine typische Skala $\xi(p)$. Für eine quantitative Bestimmung dieser Skala, ist es nützlich, analog zu Kapitel 4.7 (vgl. (4.10)) eine Korrelationsfunktion einzuführen:

Definition: Die *Korrelationsfunktion* $g(r)$ ist die Wahrscheinlichkeit, dass zwei Plätze im Abstand r zu demselben Cluster gehören.

Die Korrelationsfunktion $g(r)$ läßt sich ebenfalls mit dem Hoshen-Kopelman-Algorithmus messen. Allerdings muß für diesen Zweck der Systemzustand einschließlich Labels gespeichert werden. Außerdem kann die Messung der Korrelationsfunktion die CPU-Zeit dominieren. Ferner ist es günstig, nun *periodische* anstelle der bisherigen offenen Randbedingungen zu verwenden, denn bei offenen Randbedingungen

[4] Zum Vergleich: Der vermutlich genaueste verfügbare Literaturwert ist $p_c = 0.59274621 \pm 0.00000013$, siehe M.E.J. Newman, R.M. Ziff, Phys. Rev. Lett. **85** (2000) 4104-4107. Man beachte, dass auch die Genauigkeit von (6.3) leicht durch mehr Einsatz von CPU-Zeit zu verbessern wäre; allerdings nutzt das Literaturergebnis auch einen effizienteren Algorithmus.

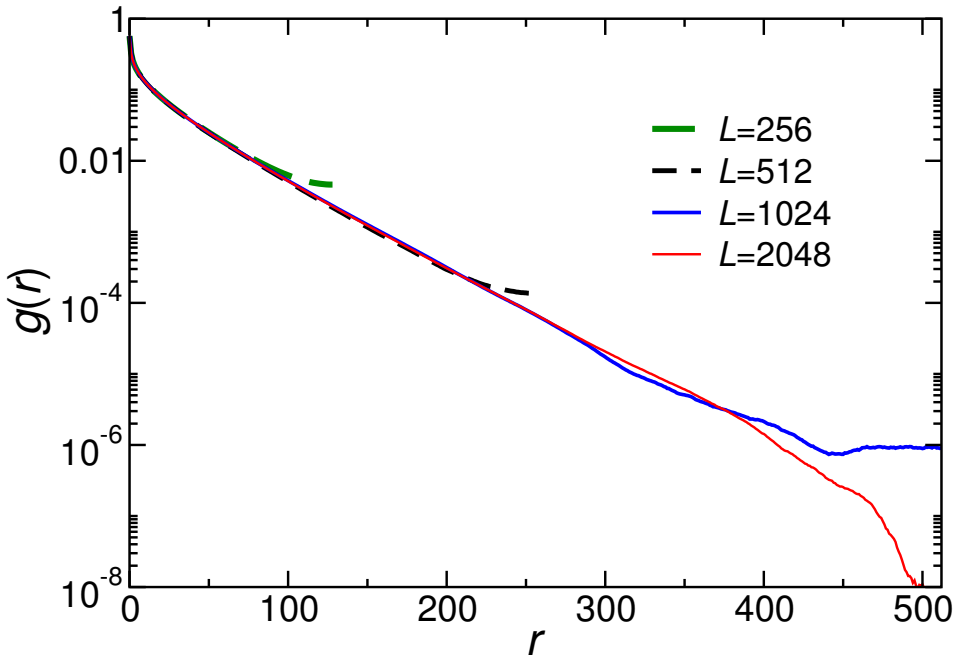


Abbildung 6.5: Korrelationsfunktion bei $p = 0.57$ für Platzperkolation auf einem $L \times L$ Quadratgitter mit periodischen Randbedingungen. Man beachte die logarithmische vertikale Skala.

würde die Korrelationsfunktion zwischen zwei Punkten nicht nur von deren Abstand, sondern auch vom Abstand zum Rand abhängen. Periodische Randbedingungen lassen sich ebenfalls leicht im Hoshen-Kopelman-Algorithmus implementieren, insbesondere wenn man den gelabelten Systemzustand speichert.

Abbildung 6.5 zeigt Ergebnisse für die Korrelationsfunktion im Beispiel $p = 0.57$. Zunächst gilt zu beachten, dass aufgrund der periodischen Randbedingungen der maximal erreichbare Abstand (z.B. entlang der horizontalen Richtung) $r = L/2$ ist. Bei diesen Abständen erwartet man dann auch Einflüsse der endlichen Systemgröße, wie sie tatsächlich für $L = 256$ und 512 in Abbildung 6.5 beobachtet werden. In diesem Fall haben wir keine saubere Fehleranalyse durchgeführt. Dennoch kann man die Schwankungen für $r \gtrsim 400$ in den $L = 1024$ und 2048 Daten auf statistische Fehler zurückführen. Für $p = 0.57$ können somit die Ergebnisse für $g(r)$ und $L \gtrsim 1024$ im Rahmen der statistischen Fehler als repräsentativ für ein unendlich großes System betrachtet werden.

Beachtet man die logarithmische vertikale Skala in Abbildung 6.5, so fällt $g(r)$ offensichtlich exponentiell ab

$$g(r) \propto \exp(-r/\xi)\,, \tag{6.4}$$

wobei ξ die gesuchte charakteristische Längenskala darstellt. Wir könnten die Daten nun an (6.4) anpassen und würden $\xi \approx 36$ für $p = 0.57$ aus den Daten in Abbildung 6.5 finden.

Eleganter ist es jedoch, über das zweite Moment von $g(r)$ eine *integrierte Korrelationslänge* einzuführen:

$$\xi_{\text{int}}^2 := \frac{\sum_{r=0}^{r_{\max}} r^2\, g(r)}{\sum_{r=0}^{r_{\max}} g(r)}\,. \tag{6.5}$$

Zur Motivation dieser Definition nehmen wir an, dass die Korrelationsfunktion exakt durch (6.4) gegeben ist, d.h. $g(r) = c\,\mathrm{e}^{-r/\xi}$. Diese funktionale Form kann man in (6.5) einsetzen und findet, wenn man ferner die Summe durch ein Integral nähert:

$$\xi_{\text{int}}^2 \approx \frac{\int_{r=0}^{\infty} \mathrm{d}r\, r^2\, g(r)}{\int_{r=0}^{\infty} \mathrm{d}r\, g(r)} = 2\,\xi^2\,. \tag{6.6}$$

Wir finden also, dass die integrierte Korrelationslänge ξ_{int} bis auf einen konstanten Faktor gleich der Korrelationslänge ξ ist. Eine sauberere Analayse zeigt, dass der Proportionalitätsfaktor von der $\sqrt{2}$ aus (6.6) abweichen kann, aber die zentrale Aussage, dass ξ_{int}/ξ eine von p unabhängige Konstante ist, bleibt gültig.

Abbildung 6.6 zeigt das Verhalten der integrierten Korrelationslänge ξ_{int} als Funktion von p für $p < p_c$. Man beobachtet, dass ξ_{int} für $p \nearrow p_c$ divergiert (siehe obere Tafel und beachtet die logarithmische vertikale Skala). Natürlich ist die Länge ξ_{int} durch die lineare Systemausdehnung L beschränkt, so dass eine echte Divergenz nur für $L \to \infty$ auftritt. Umgekehrt hängen die Schätzwerte von ξ_{int} insbesondere für Werte von p in der Nähe von p_c von L ab, so dass man eine gute Näherung an das unendliche System nur für hinreichend große L findet. Die Divergenz von ξ_{int} ist charakteristisch für einen *Phasenübergang* (hier zwischen einer nicht-durchlässigen „Phase“ für $p < p_c$ und einer durchlässigen „Phase“ für $p > p_c$).

Wie die doppelt logarithmische Auftragung von ξ_{int} versus $p_c - p$ in der unteren Tafel von Abbildung 6.6 zeigt, wird die Divergenz von ξ_{int} durch ein asymptotisches Potenzgesetz beschrieben

$$\xi_{\text{int}} \propto \frac{1}{(p_c - p)^{\nu}}\,. \tag{6.7}$$

Bei ν handelt es sich um einen sogenannten *kritischen Exponenten*.

Zum Abschluß der Diskussion der Perkolation wollen wir ν schätzen. Unsere Schätzwerte für ξ_{int} seien in folgender Datei `xiInt.dat` gespeichert:

```
# p      xi_int
0.4      1.95785764043    256
0.45     3.02291158441    256
0.5      5.59417064271    512
0.53     9.69238012209    512
```

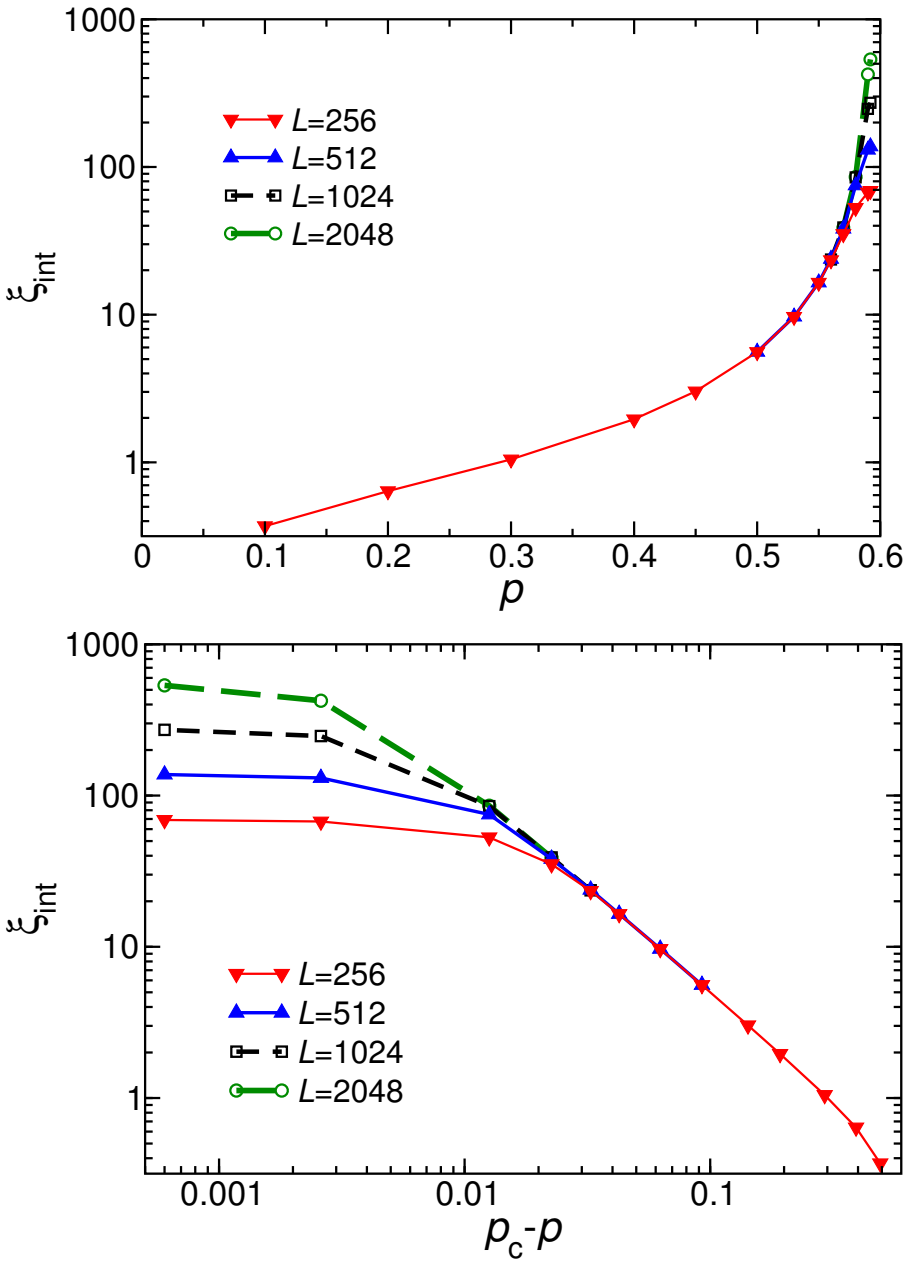


Abbildung 6.6: Schätzwerte für die integrierte Korrelationslänge ξ_{int} bei Platz-Perkolation als Funktion von p. Die obere Tafel zeigt eine logarithmische Skalierung der vertikalen Achse mit einer linearen Darstellung von p auf der horizontalen Achse. Die untere Tafel zeigt eine doppelt-logarithmische Darstellung der gleichen Daten, wobei auf der horizontalen Achse $p_c - p = 0.5926 - p$ dargestellt ist (vgl. (6.3)).

```
0.55      16.4999143601     512
0.56      23.620772038      1024
```

```
0.57    38.7661860854    2048
0.58    85.6048190328    2048
```

Hierbei haben wir einerseits die Werte für kleine p herausgenommen, da sie zuweit weg von p_c und somit nicht im Gültigkeitsbereich des asymptotischen Potenzgesetzes (6.7) liegen (vgl. untere Tafel von Abbildung 6.6). Andererseits haben wir auch Werte in der Nähe von p_c weggelassen, da die verfügbaren Systemgrößen hier keine zuverlässige Bestimmung von ξ_{int} gestatten.
Man kann obige Daten an die Form $\Xi(p) = a/(p_c - p)^\nu$ mit drei Fit-Parametern a, p_c und ν anpassen. Dies funktioniert im vorliegenden Fall erstaunlich gut: man findet $p_c = 0.5929 \pm 0.0002$ und $\nu = 1.379 \pm 0.013$. Im allgemeinen ist jedoch folgende Vorgehensweise stabiler: man fixiert p_c auf den zuvor bestimmten Wert (6.3), logarithmiert beide Achsen und passt eine Gerade an. Dies kann man z.B. mit `gnuplot` und folgenden Eingaben bewerkstelligen:

```
lnXi(lnpdiff) = lna - nu*lnpdiff
lna = 0.1
nu = 1.0
pc = 0.5926
fit lnXi(x) "xiInt.dat" using (log(pc-$1)):(log($2)) via lna, nu
```

Die erste Zeile definiert die Fit-Funktion, die zweite und die dritte Zeile setzen Startwerte für die Parameter. Die vierte Eingabezeile enthält den Wert von p_c gemäß (6.3) und die fünfte Zeile logarithmiert schließlich die Daten und führt den Fit aus.
Es stellt sich heraus, dass der systematische Fehler von p_c mindestens genauso wichtig ist wie der statistische Fehler bei der Anpassung für ein festes p_c. Diesen Fehler kann man abschätzen, indem man Anpassungen an der unteren Grenze des Konfidenzintervalls (6.3), also $p_c = 0.5924$ sowie an der oberen Grenze $p_c = 0.5928$ durchführt. Diese Analyse führt für die gezeigten Daten auf:

$$\nu = 1.38 \pm 0.03\,. \qquad (6.8)$$

Im vorliegenden Fall von zwei Dimensionen ist der exakte Wert für diesen kritischen Exponenten bekannt[5]: $\nu = 4/3 = 1.3333\ldots$. Angesichts der Tatsache, dass wir uns zwar schon ein wenig Mühe gegeben haben, aber durchaus noch mehr CPU-Zeit und theoretische Verfeinerungen investieren könnten, liegen wir mit unserem Ergebnis (6.8) gar nicht schlecht.

[5] Siehe z.B. D. Stauffer, A. Aharony, *Perkolationstheorie: Eine Einführung*, VCH Verlagsgesellschaft mbH, Weinheim (1995).

6.2 Importance Sampling

Wir wollen uns nun der statistischen Beschreibung eines System bei einer Temperatur T zuwenden. In diesem Fall sind die Wahrscheinlichkeiten durch die *Boltzmann-Gewichte* von Konfigurationen $\vec{x}$ mit Energie $E(\vec{x})$ gegeben

$$p(\vec{x}) = \frac{\mathrm{e}^{-E(\vec{x})/(k_B\,T)}}{Z}\,. \tag{6.9}$$

Hierbei ist k_B die Boltzmann-Konstante.
Nun treten folgende Probleme auf:

1. Bei Vielteilchensystemen haben wir es mit hohen Dimensionen zu tun. Das ist allerdings nicht neu – aus diesem Grund wollen wir schließlich Monte-Carlo-Verfahren anwenden.

2. Lokale Energiedifferenzen addieren sich auf, so dass man Energiedifferenzen $\Delta E \propto N$ findet, wobei N die Teilchenzahl ist. Für Quotienten der die Boltzmann-Gewichte zu Konfigurationen $\vec{x}$ und $\tilde{\vec{x}}$ mit einer Energiedifferenz ΔE gilt
$$p(\vec{x})/p(\tilde{\vec{x}}) = \exp(-\Delta E/(k_B\,T))\,.$$
Die Gewichte können also *exponentiell* in N bzw. $1/T$ verschieden sein.

3. Die Normierungskonstante in (6.9) $Z = \sum_{\vec{x}} \mathrm{e}^{-E(\vec{x})/(k_B\,T)}$ ist im allgemeinen *unbekannt*[6].

Im allgemeinen ist das Ziel, Erwartungswerte einer (oder mehrer) Observablen f berechnen (eine spezielle meßbare Größe ist die Energie E). Für die Gewichte (6.9) führt (6.1) auf

$$\langle f\rangle = \frac{1}{Z}\sum_{\vec{x}} f(\vec{x})\,\mathrm{e}^{-E(\vec{x})/(k_B\,T)} = \frac{\sum_{\vec{x}} f(\vec{x})\,\mathrm{e}^{-E(\vec{x})/(k_B\,T)}}{\sum_{\vec{x}} \mathrm{e}^{-E(\vec{x})/(k_B\,T)}}\,. \tag{6.10}$$

Aufgrund der oben genannten Eigenschaft 2 haben wir es mit einem Problem des am Anfang dieses Kapitels genannten zweiten Problemtyps mit einem stark variierenden $p(\vec{x})$ zu tun. Nach den Bemerkungen aus Unterkapitel 5.5.2 sollten wir bei einer Monte-Carlo-Auswertung von von (6.10) tunlichst bevorzugt wichtige Konfigurationen auswerten. Dies werden wir implementieren, indem wir einen Zufallsweg auf dem Konfigurationsraum konstruieren, so dass eine Konfiguration $\vec{x}$ mit Wahrscheinlichkeit $P(\vec{x}) \propto \exp(-E(\vec{x})/(k_B\,T))$ besucht wird. Wenn uns dies gelingt,

[6] Tatsächlich hat Z in der statistischen Mechanik die Bedeutung der Zustandssumme. Wie Sie in einem späteren Theorie-Kurs lernen werden, erlaubt allein die Kenntnis von Z bereits im wesentlichen die Ableitung der thermodynamischen Eigenschaften des Systems.

können wir den Erwartungswert (6.10) während m Schritten eines solchen Zufallsweges über

$$\langle f \rangle \approx \frac{1}{m} \sum_{i=1}^{m} f(\vec{x}_i)\,. \tag{6.11}$$

schätzen.
Um dieses Ziel zu erreichen, müssen wir allerdings etwas ausholen.

6.2.1 Markov-Prozesse und Mastergleichung

Ein Zufallsweg auf dem Konfigurationsraum, wird allgemein durch die Wahrscheinlichkeiten

$$P_k(\vec{x}_1, \ldots, \vec{x}_k) \tag{6.12}$$

charakterisiert, dass nacheinander die Punkte $\vec{x}_1$, ..., $\vec{x}_k$ besucht werden.
Von besonderem Interesse sind nun die sogenannten *Markov-Prozesse*. Ein Markov-Prozess ist dadurch definiert, dass die Wahrscheinlichkeit dafür, im $k$ten Schritt $\vec{x}_k$ zu finden, nur von $\vec{x}_{k-1}$ abhängt, aber eben nicht von der ganzen Geschichte des Weges. Für die Wahrscheinlichkeit, $\vec{x}_k$ in einem Weg $\vec{x}_1$, ..., $\vec{x}_k$ zu beobachten, gilt also

$$\boxed{P_k(\vec{x}_k|\vec{x}_1, \ldots, \vec{x}_k) =: p_k(\vec{x}_k|\vec{x}_{k-1})\,.} \tag{6.13}$$

Die $p_k(\vec{x}_k|\vec{x}_{k-1})$ werden auch als *Übergangswahrscheinlichkeiten* bezeichnet.
Die Markov-Eigenschaft (6.13) kann man wiederholt in (6.12) einsetzen und findet, dass für die Wahrscheinlichkeit, dass in einem Markov-Prozess der Weg $\vec{x}_1$, ..., $\vec{x}_k$ auftritt, gilt:

$$\begin{aligned} P_k(\vec{x}_1, \ldots, \vec{x}_k) &= p_k(\vec{x}_k|\vec{x}_{k-1})\, p_{k-1}(\vec{x}_1, \ldots, \vec{x}_{k-1}) \\ &= p_k(\vec{x}_k|\vec{x}_{k-1})\, p_{k-1}(\vec{x}_{k-1}|\vec{x}_{k-2}) \cdots p_2(\vec{x}_2|\vec{x}_1)\, P_1(\vec{x}_1)\,. \end{aligned} \tag{6.14}$$

Im allgemeinen hängen die Übergangswahrscheinlichkeiten $p_k(\vec{y}|\vec{x})$ in einem Markov-Prozess vom Schritt k ab. Häufig interessiert man sich jedoch für solche Prozesse, bei denen die Übergangswahrscheinlichkeiten nicht explizit vom Schritt abhängen. Prozesse, bei denen die Übergangswahrscheinlichkeiten unabhängig von k sind, werden *stationäre Prozesse* genannt[7]. Für einen stationären Prozess bezeichnen wir die Übergangswahrscheinlichkeiten mit

$$\boxed{p_k(\vec{y}|\vec{x}) = T(\vec{x} \to \vec{y})\,.} \tag{6.15}$$

[7] Über die Wahrscheinlichkeiten P_k haben wir an dieser Stelle noch keine Aussage gemacht: sie hängen auch bei einem stationären Markov-Prozess üblicherweise von k ab.

Wir wollen nun eine Gleichung für die „Zeitentwicklung“ der Wahrscheinlichkeit $P(\vec{x},k)$ angeben, dass im $k$ten Schritt eines stationären Markov-Prozesses die Konfiguration $\vec{x}$ auftritt. Zunächst drücken wir $P(\vec{x},k+1)$ durch die Wahrscheinlichkeit $P(\vec{x},k+1;\vec{y},k)$ aus, im Schritt k die Konfiguration $\vec{y}$ zu finden, und im Schritt $k+1$ dann $\vec{x}$:

$$P(\vec{x},k+1)=\sum_{\vec{y}}P(\vec{x},k+1;\vec{y},k)=\sum_{\vec{y}}T(\vec{y}\to\vec{x})\,P(\vec{y},k)\,, \tag{6.16}$$

wobei wir die Markov-Eigenschaft (6.13) verwendet haben. Wir subtrahieren nun auf beiden Seiten

$$P(\vec{x},k)=\sum_{\vec{y}}P(\vec{y},k+1;\vec{x},k)=\sum_{\vec{y}}T(\vec{x}\to\vec{y})\,P(\vec{x},k) \tag{6.17}$$

und erhalten eine *diskrete Mastergleichung*

$$P(\vec{x},k+1)-P(\vec{x},k)=\sum_{\vec{y}}\Big(T(\vec{y}\to\vec{x})\,P(\vec{y},k)-T(\vec{x}\to\vec{y})\,P(\vec{x},k)\Big)\,. \tag{6.18}$$

Diese Mastergleichung läßt sich wie folgt interpretieren: auf der linken Seite steht die Änderung der Wahrscheinlichkeit, $\vec{x}$ zu finden während des Schrittes von k zu $k+1$. Der erste Term auf der rechten Seite beschreibt den *Gewinn* von Konfiguration $\vec{x}$, dadurch dass ein Schritt von Konfiguration $\vec{y}$ nach $\vec{x}$ stattfindet. Der zweite Term auf der rechten Seite beschreibt den *Verlust* durch einen Schritt aus Konfiguration $\vec{x}$ in eine Konfiguration $\vec{y}$.
Wir wollen diese abstrakten Konzepte nun am Beispiel eines Teilchens illustrieren, das einen *Zufallsweg auf einem eindimensionalen Gitter* ausführt. Anfangs (d.h. für $k=0$) soll das Teilchen bei $x=0$ sitzen. In jedem Schritt geht das Teilchen dann entweder nach links (d.h. $x\to x-1$) oder nach rechts (also $x\to x+1$), und zwar jeweils mit Wahrscheinlichkeit $1/2$. Anstelle, wie in den Übungen, dieses Problem zu simulieren, wollen wir hier direkt die Zeitentwicklung für die Aufenthaltswahrscheinlichkeit berechnen. Zu diesem Zweck verwenden wir die Mastergleichung (6.18), die konkret für unser Beispiel

$$P(x,k+1)-P(x,k)=\frac{1}{2}\,P(x-1,k)+\frac{1}{2}\,P(x+1,k)-P(x,k) \tag{6.19}$$

lautet. Die beiden ersten Terme auf der rechten Seite stehen für den Gewinn dadurch, dass ein Teilchen Platz x entweder von links oder von rechts erreicht. Der letzte Term auf der rechten Seite von (6.19) beschreibt den Verlust dadurch, dass

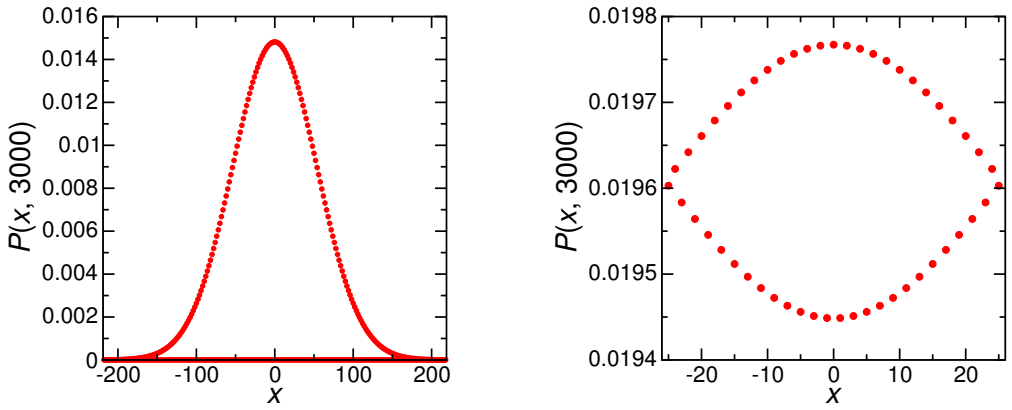


Abbildung 6.7: Aufenthaltswahrscheinlichkeit eines Teilchens, das ausgehend von $x = 0$ bei $k = 0$ einen Zufallsweg mit $k = 3000$ Schritten auf einem eindimensionalen Gitter durchlaufen hat. Die linke Tafel zeigt das Ergebnis für ein unendlich ausgedehntes System, die rechte ein System mit $N = 51$ Plätzen und periodischen Randbedingungen.

das Teilchen entsprechend unseren Annahmen den aktuellen Platz x auf jeden Fall verlassen muß. Glg. (6.19) ist äquivalent zu

$$P(x, k+1) = \frac{1}{2}\, P(x-1, k) + \frac{1}{2}\, P(x+1, k)\,. \tag{6.20}$$

Hinzu kommt die Anfangsbedingung, dass das Teilchen anfangs bei $x = 0$ sitzt, was gleichbedeutend ist mit

$$P(x, 0) = \delta_{x,0}\,. \tag{6.21}$$

Die Rekursionsrelation (6.20) mit den Anfangsbedingungen (6.21) kann man z.B. numerisch auswerten. Das Ergebnis von k Iterationen liefert die *exakte* Wahrscheinlichkeitsdichte $P(x, k)$. Diese ist in Abbildung 6.7 nach $k = 3000$ Schritten für zwei Beispiele gezeigt: für ein unendliches System (links) und für ein endliches System mit $N = 51$ Plätzen und periodischen Randbedingungen (d.h. Indentifikationen von x und $x \pm 51$ – rechts). Im Fall des unendlichen Systems geht die Einhüllende der Wahrscheinlichkeitsdichte gegen eine Gauß-Kurve[8], die im Verlauf der Zeit immer weiter zerfließt (Abbildung 6.7 links). Im Fall der periodischen Randbedingungen und einer ungeraden Platzzahl N geht die Aufenthaltswahrscheinlichkeit für große

[8] Aufgrund unserer Annahmen sitzt das Teilchen in einem geraden (ungeraden) Schritt k immer auf einem geraden (ungeraden) Platz x. Diese Untergitterentkopplung ist ein Artefakt unserer speziellen Dynamik und würde verschwinden, wenn das Teilchen mit einer endlichen Wahrscheinlichkeit auf dem aktuellen Platz sitzen bleiben dürfte. In diesem Fall würde die Aufenthaltswahrscheinlichkeit $P(x, k)$ gegen eine glatte Kurve gehen. Für weitere Details vgl. Anhang A.2.

k gegen eine Konstante

$$P(x,k) \underset{k\to\infty}{\to} \frac{1}{N}\,. \tag{6.22}$$

Dies ist in der rechten Tafeln von Abbildung 6.7 für $k = 3000$ illustriert (man beachte, dass alle Werte von $P(x, 3000)$ nahe an $1/51 = 0.0196\ldots$ liegen)[9]. Dies bedeutet, dass ein Teilchen nach einer großen Anzahl von Schritten auf einem Ring an jedem Platz mit gleicher Wahrscheinlichkeit anzutreffen ist, und zwar unabhängig von dem Anfangspunkt seines Zufallsweges.
Letzterer Fall ist ein Beispiel für einen stationären Zustand. Ein *stationärer Zustand* eines stationären Markovprozesses ist allgemein dadurch charakterisiert, dass die Wahrscheinlichkeit unabhängig vom Schritt k wird, d.h.

$$P(\vec{x}, k+1) = P(\vec{x}, k) =: P(\vec{x})\,. \tag{6.23}$$

Die Wahrscheinlichkeitsverteilung in einem stationären Zustand bezeichen wir mit $P(\vec{x})$. Aus der Mastergleichung (6.18) folgt für einen stationären Zustand

$$0 = \sum_{\vec{y}} \Big(T(\vec{y}\to\vec{x})\,P(\vec{y}) - T(\vec{x}\to\vec{y})\,P(\vec{x}) \Big)\,. \tag{6.24}$$

6.2.2 Metropolis-Algorithmus

Nach den Vorarbeiten können wir nun endlich einen allgemeinen Algorithmus vorstellen, der von Metropolis und Koautoren 1953 vorgeschlagen wurde[10], und daher als „Metropolis-Algorithmus“ bezeichnet wird.
Für eine gegebene (nicht notwendig normierte) Wahrscheinlichkeitsdichte $p(\vec{x})$ werden wir einen stationären Markov-Prozess auf dem Konfigurationsraum konstruieren, so dass der *stationäre Zustand* der Markov-Kette die gesucht Verteilung $p(\vec{x})$ bis auf eine Normierungskonstante C reproduziert:

$$P(\vec{x}) = C\,p(\vec{x})\,. \tag{6.25}$$

Dies bedeutet, dass im stationären Zustand des Markov-Prozesses eine Konfiguration $\vec{x}$ mit einer Wahrscheinlichkeit proptional zu $p(\vec{x})$ besucht wird.
Der Markov-Prozess sollte so konstruiert werden, dass sein stationärer Zustand eindeutig ist. Dann wird dieser typischerweise zumindest näherungsweise nach einer

[9] In diesem Fall sind die Unergitter aufgrund der periodischen Platzzahl und der periodischen Randbedingungen gekoppelt.

[10] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, J. Chem. Phys. **21** (1953) 1087-1092.

hinreichend großen Anzahl von Schritten $k \gg 1$ angenommen, d.h. $P(\vec{x}, k \gg 1) \approx P(\vec{x})$.

Den Zusammenhang zu unserem ursprünglichen Problem, d.h. dem vorgegebenen $p(\vec{x})$ stellt eine Bedingung an die Übergangswahrscheinlichkeiten $T(\vec{x} \to \vec{y})$ her, die als *detailliertes Gleichgewicht* bekannt ist

$$p(\vec{x})\, T(\vec{x} \to \vec{y}) = p(\vec{y})\, T(\vec{y} \to \vec{x})\,. \tag{6.26}$$

Im allgemeinen hat man eine große Wahlfreiheit bei der Lösung dieser Bedingung. Für nicht allzu pathologische Wahlen von $T(\vec{x} \to \vec{y})$[11] stellt (6.26) sicher, dass (6.25) gilt.

Zur Begründung dieses Ergebnisses betrachten wir die Stationaritätsbedingung (6.24)

$$\begin{aligned} 0 &\overset{(6.24)}{=} \sum_{\vec{y}} \left(T(\vec{y} \to \vec{x})\, P(\vec{y}) - T(\vec{x} \to \vec{y})\, P(\vec{x})\right) \\ &= \sum_{\vec{y}} P(\vec{y})\, T(\vec{x} \to \vec{y}) \left(\frac{T(\vec{y} \to \vec{x})}{T(\vec{x} \to \vec{y})} - \frac{P(\vec{x})}{P(\vec{y})}\right) \\ &\overset{(6.26)}{=} \sum_{\vec{y}} P(\vec{y})\, T(\vec{x} \to \vec{y}) \left(\frac{p(\vec{x})}{p(\vec{y})} - \frac{P(\vec{x})}{P(\vec{y})}\right)\,. \end{aligned} \tag{6.27}$$

Analog kann man n Schritte im Markov-Prozess betrachten. Für die Übergangswahrscheinlichkeit mit n Schritten von $\vec{x}$ nach $\vec{y}$ zu kommen, gilt bei einem Markov-Prozess

$$T_n(\vec{x} \to \vec{y}) = \sum_{\vec{x}_1, \ldots, \vec{x}_{n-1}} T(\vec{x} \to \vec{x}_1)\, T(\vec{x}_1 \to \vec{x}_2) \cdots T(\vec{x}_{n-2} \to \vec{x}_{n-1})\, T(\vec{x}_{n-1} \to \vec{y})\,. \tag{6.28}$$

Dies ist im wesentlichen (6.14). Da uns die $n-1$ Zwischenkonfigurationen nicht interessieren, müssen wir über alle Möglichkeiten summieren.

Man überzeugt sich leicht, dass auch T_n das detaillierte Gleichgewicht (6.26) erfüllt, wenn bereits T diese Bedingung erfüllt. Analog zu (6.27) folgt dann

$$0 = \sum_{\vec{y}} P(\vec{y})\, T_n(\vec{x} \to \vec{y}) \left(\frac{p(\vec{x})}{p(\vec{y})} - \frac{P(\vec{x})}{P(\vec{y})}\right)\,. \tag{6.29}$$

Sind die Übergangswahrscheinlichkeiten $T_n(\vec{x} \to \vec{y})$ hinreichend verschieden, so kann man deren Koeffizienten in (6.29) vergleichen, womit

$$0 = P(\vec{y}) \left(\frac{p(\vec{x})}{p(\vec{y})} - \frac{P(\vec{x})}{P(\vec{y})}\right) \tag{6.30}$$

[11] Ein triviale Lösung von (6.26) ist z.B. $T(\vec{x} \to \vec{y}) = \delta_{\vec{x},\vec{y}}$. Der Markov-Prozess bleibt dann an einem einmal angenommenen Ort $\vec{x}$ stecken (d.h. $\vec{x}_k = \vec{x}$ für alle k) und enthält offensichtlich keine Information über $p(\vec{x})$.

und schließlich (6.25) folgt. An dieser Stelle wird normalerweise *Ergodizität* des Markov-Prozesses gefordert, d.h. dass für alle Anfangs- bzw. Endzustände $\vec{x}$ bzw. $\vec{y}$ ein n existiert, so dass $T_n(\vec{x} \to \vec{y}) \neq 0$ ist. Ist der Markov-Prozess nämlich nicht ergodisch, d.h. existieren disjunkte Untermengen, so gelangt man offensichtlich je nach Startbedingung $\vec{x}_0$ zu unterschiedlichen stationären Zuständen. Die gesuchte Identität (6.25) gilt in einem solchen nicht-ergodischen Fall allenfalls lokal, aber nicht global.

Die *Metropolis-Wahl* für die Übergangswahrscheinlichkeiten ist zunächst eine Faktorisierung in eine Vorschlags- und eine Akzeptanzwahrscheinlichkeit[12]

$$T(\vec{x} \to \vec{y}) = \pi(\vec{x}, \vec{y})\, W(\vec{x} \to \vec{y})\,. \tag{6.31}$$

Hierbei wird die Vorschlagswahrscheinlichkeit symmetrisch $\pi(\vec{x}, \vec{y}) = \pi(\vec{y}, \vec{x})$ und ohne Berücksichtigung der vorgegebenen Verteilungsfunktion $p(\vec{x})$ gewählt. Bei der Wahl der Vorschlagswahrscheinlichkeit ist darauf zu achten, dass Ergodizität sichergestellt wird.

Die Akzeptanzwahrscheinlichkeit ist bei der Metropolis-Wahl gemäß

$$W(\vec{x} \to \vec{y}) = \min\left(1, \frac{p(\vec{y})}{p(\vec{x})}\right) \tag{6.32}$$

aus dem gegebenen $p(\vec{x})$ zu bestimmen.

Die Wahl (6.31) mit (6.32) stellt sicher, dass das detaillierte Gleichgewicht (6.26) erfüllt ist.

Der Markov-Prozess wird nun über folgende Vorschrift für eine Metropolis-Monte-Carlo-Simulation realisiert:

1. Schlage im Schritt i ein neues $\vec{x}_j = \vec{x}_i + \delta\vec{x}$ vor, wobei zufällig eine (kleine) Änderung $\delta\vec{x}$ gewählt wird.
2. Berechne $w = \dfrac{p(\vec{x}_j)}{p(\vec{x}_i)}$.
3. Ist $w \geq 1$, so akzeptiere das vorgeschlagene $\vec{x}_j$, d.h. setze $\vec{x}_{i+1} = \vec{x}_j$.
4. Ist $w < 1$, erzeuge eine Zufallszahl r, die in $[0, 1]$ gleichverteilt ist.
   (a) Im Fall $r \leq w$ akzeptiere ebenfalls das vorgeschlagene $\vec{x}_j$, d.h. setze $\vec{x}_{i+1} = \vec{x}_j$.
   (b) Für $r > w$ ist der Vorschlag zu verwerfen, d.h. $\vec{x}_{i+1} = \vec{x}_i$ zu setzen.

[12] Ganz so formal wurde dies in der Original-Arbeit von Metropolis und Koautoren allerdings nicht formuliert. Interessant ist in diesem Zusammenhang daher auch W.K. Hastings, Biometrika **57** (1970) 97.

Wir erinnern daran, dass die Änderungen $\delta\vec{x}$ im ersten Schritt so zu wählen sind, dass der Algorithmus erstens ergodisch wird und zweitens symmetrisch, d.h. dass für den Zustand $\vec{x}_j$ die Änderung $-\delta\vec{x}$ mit der gleichen Wahrscheinlichkeit auftritt wie eine Änderung $\delta\vec{x}$ im Zustand $\vec{x}_i$.
Da wir im stationären Zustand Vektoren $\vec{x}_i$ mit Wahrscheinlichkeit $P(\vec{x}_i) = C\,p(\vec{x}_i)$ erzeugen, gilt in diesem Zustand

$$\langle f \rangle = \int \mathrm{d}^d x\, P(\vec{x})\, f(\vec{x}) \approx \frac{1}{m} \sum_{i=1}^{m} f(\vec{x}_i)\,. \tag{6.33}$$

Das ursprüngliche Problem (6.1) wird also gelöst, indem man bei hinreichend späten Zeiten der Metropolis-Simulation eine gewünschte Anzahl m von Messungen durchführt. An dieser Stelle gilt zu beachten, dass die Folge der $\vec{x}_i$ per Konstruktion korreliert ist. Der Fehler darf deswegen *nicht* einfach über (5.9) geschätzt werden. Für eine gegebene Starkonfiguration $\vec{x}_0$ ist man normalerweise zu Beginn nicht im stationären Zustand. Man muß daher bei einer Metropolis-Monte-Carlo-Simulation zunächst eine hinreichende Anzahl Schritte „wegwerfen“, bis man sich dem stationären Zustand bis auf statistische Fehler angenähert hat.

6.3 Lennard-Jones-Potential

Das Verhalten eines über das Lennard-Jones-Potential (4.1) wechselwirkendes Vielteilchensystem haben wir ja bereits in Kapitel 4 kennengelernt. Im Gegensatz zu der dort betrachteten „mikroskopischen“ Dynamik sind wir aber oft gar nicht an den individuellen Trajektorien interessiert. Unsere „Beobachtungen“ sind typischerweise nicht instantan, sondern umfassen Zeitintervalle (typischerweise $> 10^{-6}$s), die sehr viel größer sind als die typischen Zeitskalen ($\lesssim 10^{-12}$s) der mikroskopischen Dynamik. Daher sind die konkreten Zustände, in denen man das System zum Zeitpunkt solcher „Beobachtungen“ antrifft, im Prinzip zufällig aus der Vielzahl der möglichen Zustände herausgegriffen, im Prinzip so wie bei einem stoboskopischen Bild der Tanzenden in einer Diskothek. Dies legt eine statistische Beschreibung nahe, so dass man das Metropolis-Monte-Carlo-Verfahren anwenden kann.

6.3.1 Simulation im kanonischen Ensemble

Im Folgenden wollen wir das Programm, welches die MD-Simulation durchführt so abändern, dass man damit eine MC-Simulation erzeugen kann. Schauen wir uns dazu noch einmal den Kernteil des MD-Programmes an:

```
⋮
int main(int argc, char *argv[])
{
```

```
  ⋮
  // **** Starte Simulation
  time = 0;
  energy_pot = forces(&system, atom);
  energy_kin = kinetic_energy(&system, atom);
  rescale_kin_energy(&system, atom);

  cout << time << "\t" << system.T << "\t";
  cout << energy_pot/system.N << "\t";
  cout << energy_kin/system.N << "\t";
  cout << (energy_pot+energy_kin)/system.N << "\t";
  cout << system.virial/volume << endl;

  // **** Anfang Hauptschleife
  for(int step=1; step<=num_steps; step++) {
    energy_pot = verlet_step(&system, atom);
    energy_kin = kinetic_energy(&system, atom);
    rescale_kin_energy(&system, atom);
    time += system.tau;
    if(step>steps_eq) {                // equilibriert ?
      pair_correlations(&system,atom);
      pc += system.pc;
      N_PC_EVAL++;
    }
  }
  ⋮
}
```

Die wesentlichen Elemente der MD-Rechnung sind in der Hauptschleife die drei Anweisungen `verlet_step`, `kinetic_energy` und `rescale_kin_energy`. Für die Monte-Carlo-Simulation muss man diese drei Schritte durch einen entsprechenden Metropoliszyklus ersetzen. Zudem sind wir an der Thermodynamik interessiert, d.h. an Observablen, die von der Position, aber nicht der Geschwindigkeit der Teilchen abhängen. Damit folgt für die Erwartungswerte

$$\begin{aligned}
\langle A\rangle &= \frac{1}{Z}\int \mathrm{d}^{3N}\int \dot{x}\mathrm{d}^{3N}x\, A(\{\vec{r}_i\})\,\mathrm{e}^{-(E_{\mathsf{kin}}(\{\dot{\vec{r}}_i\})+E_{\mathsf{Pot}}(\{\vec{r}_i\}))/k_{\mathrm{B}}T} \\
&= \frac{1}{Z}\int \mathrm{d}^{3N}\dot{x}\,\mathrm{e}^{-E_{\mathsf{kin}}(\{\dot{\vec{r}}_i\})/k_{\mathrm{B}}T}\int \mathrm{d}^{3N}x\, A(\{\vec{r}_i\})\,\mathrm{e}^{-E_{\mathsf{Pot}}(\{\vec{r}_i\})/k_{\mathrm{B}}T} \\
&= \frac{1}{\tilde{Z}}\int \mathrm{d}^{3N}x\, A(\{\vec{r}_i\})\mathrm{e}^{-E_{\mathsf{Pot}}(\{\vec{r}_i\})/k_{\mathrm{B}}T}\quad ,
\end{aligned}$$

wobei

$$\begin{aligned} Z &= \int \mathrm{d}^{3N}\dot{x} \int \mathrm{d}^{3N}x \, \mathrm{e}^{-(E_{\mathsf{kin}}(\{\dot{\vec{r}}_i\})+E_{\mathsf{Pot}}(\{\vec{r}_i\}))/k_{\mathrm{B}}T} \\ &= \int \mathrm{d}^{3N}\dot{x} \, \mathrm{e}^{-E_{\mathsf{kin}}(\{\dot{\vec{r}}_i\})/k_{\mathrm{B}}T} \int \mathrm{d}^{3N}x \, \mathrm{e}^{-E_{\mathsf{Pot}}(\{\vec{r}_i\})/k_{\mathrm{B}}T} \\ &= \tilde{Z} \int \mathrm{d}^{3N}\dot{x} \, \mathrm{e}^{-E_{\mathsf{kin}}(\{\dot{\vec{r}}_i\})/k_{\mathrm{B}}T} \quad . \end{aligned}$$

Anders ausgedrückt: Die Geschwindigkeiten interessieren für die Monte-Carlo nicht die Bohne, sondern alleine die potentielle Energie ist interessant.
Will man die Gesamtenergie dennoch wissen, so erhält man diese als

$$E = \langle E_{\mathsf{kin}} + E_{\mathsf{Pot}} \rangle = \frac{d}{2} N \, k_{\mathrm{B}} \, T + \langle E_{\mathsf{Pot}} \rangle \quad ,$$

wobei im lezten Schritt wieder der Gleichverteilungssatz der Thermodynamik angewandt auf die kinetische Energie benutzt wurde[13].
Damit können wir jetzt die Spezialisierung des *Metropolis Monte-Carlo-Algorithmus* auf unser Problem angeben:

1. Erzeuge einen beliebigen Anfangszustand und berechne dessen Energie E.
2. Erzeuge einen neuen zufälligen Mikrozustand und berechne dessen Energie E' sowie $\Delta E = E' - E$.
3. Ist $\Delta E \leq 0$, dann akzeptiere den neuen Zustand und gehe zu Schritt 6.
4. Wenn $\Delta E > 0$, dann berechne $w = \mathrm{e}^{-\Delta E/k_{\mathrm{B}}T}$ und ziehe eine gleichverteilte Zufallszahl $r \in [0, 1]$.
5. Ist $r \leq w$, dann akzeptiere den neuen Mikrozustand, ansonsten verwerfe ihn.
6. Berechne die gewünschten Messgrößen.
7. Wiederhole ab Schritt 2 bis eine hinreichend große Zahl von Mikrozuständen erzeugt wurden.

Für den Metropolis-Algorithmus müssen wir also in Punkt 2 die Differenz der potentiellen Energie

$$\Delta E = E_{\mathsf{Pot}}(\{\vec{r}_i{}'\}) - E_{\mathsf{Pot}}(\{\vec{r}_i\}) \tag{6.34}$$

[13] Man kan die Gauß-Integrale auch einfach ausrechnen!

zwischen vorgeschlagener und alter Konfiguration berechnen. Typischerweise schlägt man pro Monte-Carlo-Schritt eine Änderung $\vec{r}_{i_0} \to \vec{r}_{i_0} + \vec{\zeta}$ für *ein* Atom vor, wobei $\vec{\zeta}$ eine zufällig gewählte Verschiebung ist. Damit vereinfacht sich die Berechung von ΔE auf (alle anderen Beiträge in (6.34) heben sich weg):

$$\Delta E = \sum_{\substack{j=1 \\ j \neq i_0}}^{N} \left[u(|\vec{r}_{i_0}{}' - \vec{r}_j|) - u(|\vec{r}_{i_0} - \vec{r}_j|) \right] \quad .$$

Im Programm berechnet man also nur jeweils die potentielle Energie für das ausgewählte Teilchen i_0, wobei man wie in der MD-Simulation auch periodische Randbedingungen sowie die Minimum-Image-Konvention einführt. Beachten Sie, dass wir hier nur *eine* Summe anstelle der Doppelsumme bei der Berechnung der Kräfte in der MD-Simulation zu berechnen haben.

Die Vorschläge für $\vec{\zeta}$ sind symmetrisch und aus Ergodizitätsgründen so zu wählen, dass auch beliebig kleine Änderungsvorschläge gemacht werden. Für unsere Anwendung kann man das dadurch erreichen, dass man $\vec{\zeta}$ aus einer um 0 symmetrischen Verteilung zieht, also z.B.

$$\zeta_i \in [-\alpha, \alpha] \,,$$

wobei $\alpha \in \mathbb{R}^+$ geeignet zu wählen ist. α kontrolliert die *Akzeptanzrate*, d.h. die relative Häufigkeit, mit der in Schritt 5 der neue Mikrozustand akzeptiert wird. Ihr Wert sollte in der Größenordnung $1/3 \ldots 2/3$ liegen, damit der Random Walk auch wirklich durchgeführt wird. Ist nämlich die Akzeptanzrate zu klein (α zu groß) oder zu groß (α zu klein), dann kommt das System nicht richtig „vorwärts“; im ersten Fall, weil keine Updates durchgeführt werden, und im zweiten Fall, weil man dann typischerweise keine relevanten Änderungen vorschlägt. Typischerweise kann man α im Verlauf der Simulation (z.B. in der Thermalisierungsphase) so anpassen, dass in der Messperiode die Akzeptanzrate im gewünschten Bereich liegt.

Um Ergodizität sicher zu stellen, müssen ferner alle Atom i_0 gleich häufig „besucht“ werden. Offensichtlich kann sich der Gesamtzustand des Systems erst dann ändern, wenn man für alle Atome eine Änderung vorgschlagen hat. Als natürliche MC-Zeiteinheit ergibt sich damit der *Sweep*: ein Sweep besteht genau aus N elementaren MC-Schritten, so dass jedes Atom genau einmal besucht wird.

Schließlich gilt es, der Versuchung zu widerstehen, in Schritt 6 auf die Messung zu verzichten, wenn man keine Änderung durchgeführt hat. Tatsächlich ist das Sitzenbleiben in einer Konfiguration ein Zeichen für ihr hohes Gewicht und man muß die Messung auf jeden Fall wiederholen, um die richtige Verteilungsfunktion auszuwerten. Natürlich kann man sich entscheiden, nur in festen Abständen zu messen, aber dies muß immer unabhängig von den Metropolis-Schritten geschehen.

Insgesamt sieht eine Implementierung des Metropolis-Verfahrens für das Lennard-Jones-Potential so aus:

```
/* Beitrag von Atom j zu potentieller Energie berechnen */

double pot_energy(int j,
                  valarray<double> xnew,
                  system_type *system,
                  atom_type *atom)
{
  double energy = 0;
  valarray<double> r(system->dim);      // Differenzvektor
  double r2, rm6;                       // Abstand^2, Abstand^-6

  for(int t=0; t<system->N; t++)        // Schleife ueber alle Atome
    if(t != j) {                        // ausser aktuelles Atom j
      for(int d=0; d<system->dim; d++) {
        r[d] = atom[t].x[d] - xnew[d];
        if(r[d] < -system->lh[d])       // "Minimum Image" Konvention
          r[d] += floor(-r[d]/system->l[d]+0.5)*system->l[d];
        if(r[d] > system->lh[d])
          r[d] -= floor(r[d]/system->l[d]+0.5)*system->l[d];
      }
      r *= r;
      r2 = r.sum();
      rm6 = 1.0/(r2*r2*r2);
      energy += 4.0*(rm6*(rm6-1.0));     // Energie berechnen
    }
  return(energy);
}

/* Ein "Sweep": jedes Atom kommt einmal dran */

void mc_sweep(system_type *system,
              atom_type *atom,
              int &ntry,
              int &nacc)
{
  static double alpha = 0.1;            // Intervall fuer neue Position
  valarray<double> xnew(system->dim);   // Neue Position
```

```
  for(int t=0; t<system->N; t++) {      // Schleife ueber alle Atome
    double e_old = pot_energy(t,        // alter Energie-Beitrag
          valarray<double> (atom[t].x, system->dim),
          system, atom);
    for(int d=0; d<system->dim; d++) {
      double r = random()/double(RAND_MAX);
      // Neue Position von Atom t
      xnew[d] = atom[t].x[d] + alpha*(r-0.5);
    }
    ntry++;                             // Ein neuer Metropolis-Schritt
    // Berechne Epot fuer Atom t
    double e_new  = pot_energy(t, xnew, system, atom);
    // Energie kleiner ?
    if (e_new<e_old) {                  // Akzeptiere auf jeden Fall
      for(int d=0; d<system->dim; d++)
        atom[t].x[d] = xnew[d];         // Ersetze alte Position
      nacc++;                           // Akzeptiert!
    }
    else {
      double w = exp(-(e_new-e_old)/system->T);
      double r = random()/double(RAND_MAX);
      if(r <= w) {                      // Akzeptiere mit Wahrscheinlichkeit w
      for(int d=0; d<system->dim; d++)
        atom[t].x[d] = xnew[d];
      nacc++;
     }
    }
  }
  // Anpassen des Intervalls fuer neue Positionen
  if (ntry>100) {
    double acrat = ((double) nacc)/((double) ntry);
    alpha *= pow(sqrt(2.0),acrat-0.5);  // Nur ein Beispiel!!!
  }
}
```

Der wesentliche Teil des Hauptprogrammes lautet damit

```
int main(int argc, char *argv[])
{
  ⋮
```

```
  valarray<double> energy_pot(system.N);

  // **** Starte Simulation
  mc_time = 0;
  E = 0;
  for(int t=0; t<system.N; t++) {
    energy_pot[t] = pot_energy(t,
          valarray<double> (atom[t].x, system.dim),
          &system, atom);
    E += energy_pot[t];
  }

  // **** Anfang Hauptschleife
  for(int step=1; step<=num_steps; step++) {
    mc_sweep(&system, atom, ntry, nacc);
    mc_time += system.tau;
    ⋮
  }
  ⋮
}
```

6.3.2 Ergebnisse

Im Folgenden zeige ich Ihnen Ergebnisse der MC-Simulation für 216 Atome bei einer Temperatur $T = 1$ in einem Quader mit Kantenlängen $6.734\ldots$, $5.832\ldots$, $5.498\ldots$. Wie in der MD-Simulation auch, wurden die Atome anfangs auf ein hexagonales Gitter gesetzt.

Der erste interessante Punkt ist die Anzahl der MC-Schritte bis zur Thermalisierung. Abbildung 6.8 zeigt die Entwicklung der potentiellen Energie als Funktion der MC-Sweeps. Man sieht sehr schön, dass nach grob 500 MC-Sweeps diese Größe um den in blau eingetragenen Mittelwert schwankt, d.h. ab dieser MC-Zeit das System im thermodynamischen Gleichgewicht ist. In anderen Worten: erst nach ca. 500 Sweeps hat man sich dem stationären Zustand des Markov-Prozesse innerhalb statistischer Fluktuationen angenähert und kann anfangen, zu messen.

Die Thermalisierung ist nicht spezifisch für Monte-Carlo-Simulationen, sondern gilt in gleicher Weise für die Molekulardynamik. Der Einsatz von 6.8 zeigt die Entwicklung der potentiellen Energie während der MD-Simulation als Funktion der Verlet-Schritte (vom Aufwand her entsprechend den Sweeps in der MC Simulation). Man sieht, dass die MD tatsächlich grob 10 mal länger braucht zum Thermalisieren als die MC. Insbesondere ist der Potential-Berg augenfällig, der erst nach grob 1000

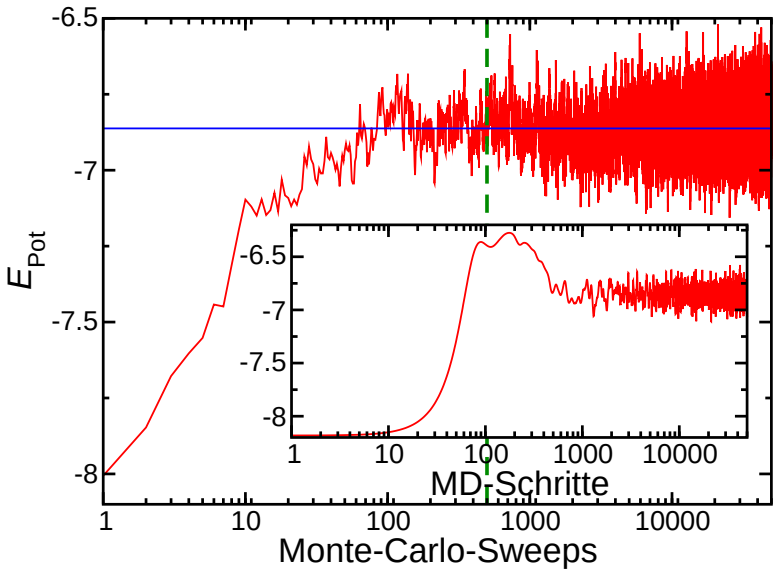


Abbildung 6.8: Potentielle Energie als Funktion der MC-Sweeps für $N = 216$ Teilchen bei $T = 1$ in einem Quader mit Kantenlängen $6.734\ldots$, $5.832\ldots$, $5.498\ldots$. Der Einsatz zeigt das analoge Verhalten für eine *Molekulardynamik*-Simulation mit den gleichen Parametern.

MD-Schritten überwunden wird. Diese Zahlen deuten bereits an, dass eine MC-Simulation effizienter sein kann (aber durchaus nicht sein muß!). Zudem erlaubt der Metropolis-Schritt mit einer endlichen Wahrscheinlichkeit, dass eine Simulation aus einem energetisch ähnlichen Nebenminimum schließlich doch in das das thermische Gleichgewicht definierende absolute Minimum wechselt. Nochmal zur Erinnerung: In der MD-Simulation ist infolge der mikrokanonischen Statistik ein Wechsel zwischen Mikrozuständen mit deutlich unterschiedlicher Energie extrem unwahrscheinlich, d.h. die Überwindung eines großen Potentialberges wird in der MD-Simulation schwerer zu realisieren sein als in der MC-Simulation.

Abgesehen von diesen eher formalen Argumenten ist natürlich die wichtigere Frage, ob die Physik in beiden Verfahren identisch ist. Dies wird in Abb. 6.9 gezeigt, wo die Paarkorrelationsfunktion für eine MD-Simulation mit 216 Teilchen für eine Temperatur $T = 1$ mit der einer MC-Simulation für das identische System verglichen wird. Die Übereinstimmung zeigt zunächst, dass die beiden deutlich verschiedenen Algorithmen tatsächlich die gleichen Ergebnisse liefern.

Der Unterschied zu Abb. 4.6 ist, dass der Kasten bei Abb. 6.9 kleiner ist, so dass das System unter deutlich höherem „Druck“ steht und ein fester (kristalliner) Zustand angenommen wird. Dies führt zu scharfen Strukturen in der Paarkorrelationsfunktion, wie man sie in Abb. 6.9 beobachtet.

Wenn die MC-Simulationen effizienter sind, warum macht man dann überhaupt

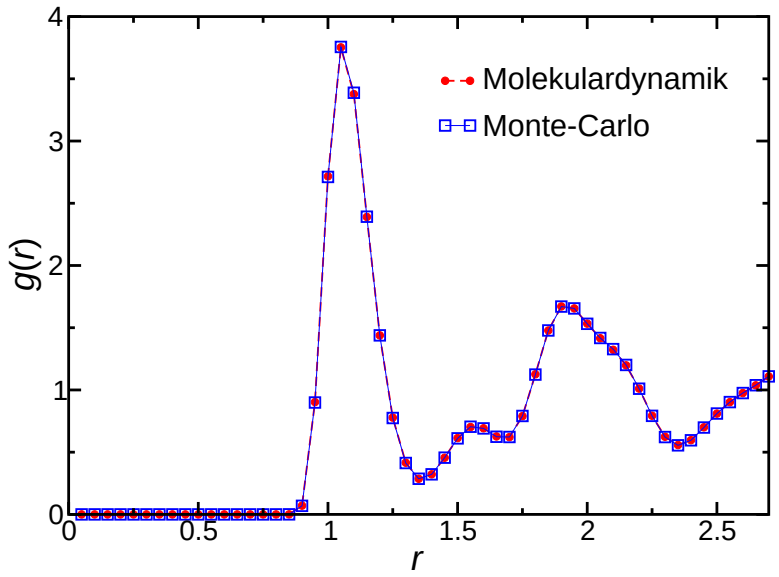


Abbildung 6.9: Paarkorrelationsfunktion für MD- und MC-Simulation von $N = 216$ Teilchen bei $T = 1$ in einem Quader mit Kantenlängen $6.734\ldots$, $5.832\ldots$, $5.498\ldots$.

MD-Simulationen? Wie bereits erwähnt, hat man mit dieser Technik die reale Zeitentwicklung in der Hand. Diesen Vorteil kann man speziell bei Wachstumsproblemen (Oberflächen, Aktienmarktentwicklung, ...) ausspielen. Eine andere Klasse von Systemen sind Gläser. Dort ist genau diese Vielfalt an Energieminima und das „Einfrieren“ in einem Zustand weg vom thermischen Gleichgewicht mit einer langen Relaxationszeitskala die interessante Physik!

6.3.3 Korrelationen in Daten

Bei einer Metropolis-MC-Simulation sind die Konfigurationen nicht statistisch unabhängig, sondern per Konstruktion korreliert. Somit sind auch die Werte der Observablen in einer Zeitreihe *statistisch korreliert*. Man darf daher den Fehler nicht einfach über die Varianz der Zeitreihe schätzen. Der einfachste Ausweg, den Fehler einer Metropolis-MC-Simulation zu schätzen, ist es eine hinreichende Anzahl (typischerweise 10-100) *unabhängiger* Simulationen durchzuführen, und dann die Fehler mit Hilfe von (5.9) aus den Ergebnissen der Einzelsimulationen zu schätzen.
Auch ist zu beachten, dass man bei häufigeren Messungen nicht unbedingt zusätzliche Information gewinnt. Insbesondere wenn eine Messung teuer ist (manche Messungen können durchaus die Rechenzeit dominieren), sollte man allein schon aus Effizienz-Gründen Messungen nur in hinreichend großen Abständen vornehmen.
Ein genaueres Maß für die Korrelationen in der Zeitreihe ist die sogenannte *Autokorrelationszeit* τ_A, die man aus der *Autokorrelationsfunktion*

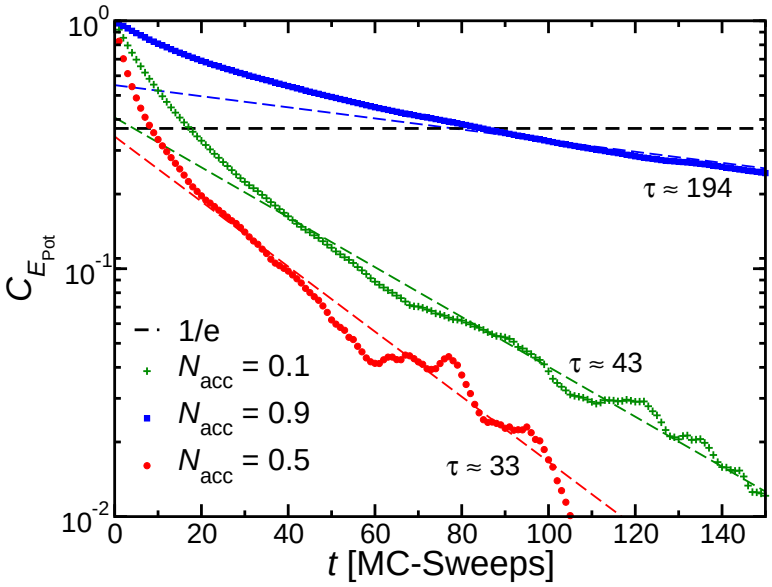


Abbildung 6.10: Autokorrelationsfunktion in Abhängigkeit von der Akzeptanzrate für den Metropolis-Schritt einer MC-Simulation von $N = 216$ Teilchen bei $T = 1$ in einem Quader mit Kantenlängen $6.734\ldots$, $5.832\ldots$, $5.498\ldots$. Die Autokorrelationsfunktion wurde aus einem Simulationslauf bestimmt; Schwankungen bei kleinen Werten der Autokorrelationsfunktion sind auf die hier nicht bestimmten statistischen Fehler zurückzuführen.

$$C_A(t) = \frac{\langle A(t+t_0)A(t_0)\rangle - \langle A\rangle^2}{\langle A^2\rangle - \langle A\rangle^2} \qquad (6.35)$$

der Observablen A bestimmen kann. Hierbei ist t die Monte-Carlo-Zeit, die wir wie zuvor geschildert in Einheiten von Sweeps messen. Die Erwartungswerte in (6.35) sind über den stationären Zustand des Markov-Prozesses zu bilden. Da dieser invariant unter Zeit-Translationen ist, hängen die Erwartungswerte von $A(t_0)$ und $A(t_0)^2$ nicht mehr von dem Zeitpunkt der Messung ab, so dass wir das Zeit-Argument weggelassen haben. Der erste Summand im Zähler von (6.35) ist ebenfalls unabhängig von t_0, hängt jedoch vom Zeitabstand t ab. Die additive Konstante im Zähler von (6.35) sorgt dafür, dass $C_A(t) \to 0$ für $t \to \infty$; der Nenner bewirkt die Normierung $C_A(0) = 1$.

Typischerweise fällt die Autokorrelationsfunktion für große Zeiten exponentiell ab

$$C_A(t \to \infty) \propto \mathrm{e}^{-t/\tau_A}\,, \qquad (6.36)$$

wobei die Autokorrelationszeit τ_A als Zeitkonstante auftritt.

Für unser Beispiel des Lennard-Jones-Systems zeigt Abb. 6.10 die Autokorrelati-

onsfunktion der potentiellen Energie für drei Akzeptanzraten, nämlich $N_{\text{acc}} = 0.1$, $N_{\text{acc}} = 0.5$, und $N_{\text{acc}} = 0.9$. Die Darstellung ist halblogarithmisch und man sieht sehr schön, dass für große Zeiten tatsächlich (6.36) gilt. Durch einen Fit unter Vernachlässigung der Daten bei kurzen Zeiten, findet man $\tau_{E_{\text{Pot}}} \approx 43$, 33 bzw. 194 bei $N_{\text{acc}} = 0.1$, 0.5 bzw. 0.9. Offensichtlich schneidet bei diesem Vergleich wie erwartet $N_{\text{acc}} = 0.5$ mit $\tau_{E_{\text{Pot}}} \approx 33$ am besten ab. Insbesondere die Simulation mit $N_{\text{acc}} = 0.9$ ist hingegen sehr ineffizient: man macht zwar viele Schritte, diese sind aber so klein, dass man trotzdem nicht richtig vorwärts kommt. Ein analoger Trend spiegelt sich in der Zeit, die die Autokorrelation benötigt, um auf $1/\mathrm{e}$ abzufallen (horizontale Linie in Abb. 6.10). Aufgrund des anfangs steilen Abfalls von $C_{E_{\text{Pot}}}(t)$ sind diese Zeiten zwar etwas kleiner als die entsprechende Autokorrelationszeiten, aber durchaus von der gleichen Größenordnung. Auch hier schneidet $N_{\text{acc}} = 0.5$ wieder am besten ab.

Übrigens: ein Abstand von τ_A zwischen Messungen der Observablen A stellt nicht sicher, dass man diese als statistisch unabhängig betrachten darf. Eher im Gegenteil: Messungen in Abständen kleiner als τ_A sind nicht unabhängig, sondern führen mit großer Wahrscheinlichkeit zu ähnlichen Ergebnissen.

6.4 Ising-Modell

Bislang haben wir uns mit der Bewegung von Atomen bzw. Molekülen beschäftigt und zum Beispiel den Übergang Gas $\leftrightarrow$ Flüssig $\leftrightarrow$ Fest. Nun hat ein Festkörper aber auch noch andere Eigenschaften, z.B. kennen Sie das Phänomen des Magnetismus: Eisen oder andere Materialien zeigen unterhalb einer sog. *kritischen Temperatur* T_c (Curie-Temperatur) ein spontanes magnetisches Moment. Die vollständige mikroskopische Beschreibung des Phänomenes Magnetismus erfordert die Quantenmechanik. Letztlich läuf es darauf hinaus, dass die magnetischen Momente („Spins“) der Elektronen eines magnetisch ordnenden Materials stark genug miteinander wechselwirken, um diese auf makroskopischer Skala auszurichten. Auch die Ursache dieser Wechselwirkungen ist rein quantenmechanischer Natur, ihre Struktur ist aber die einer Dipolwechselwirkung, nur um etliche Größenordnungen stärker.

Ernst Ising hat im Rahmen seiner Doktorarbeit bei Wilhem Lenz in der 1920er Jahren ein Modell untersucht, das als einfaches Modell für einen Ferromagneten gedacht war. Ernst Ising fand im exakt lösbaren Fall einer Dimension[14], dass keine ferromagnetische Ordnung auftritt, so dass er dieses Modell verwarf. Dessen ungeachtet hat sich dieses Modell zu einem wahren Arbeitspferd der statistischen Physik entwickelt und wird als *Ising-Modell* bezeichnet. Damit handelt es sich bei diesem Modell auch um eine der bekanntesten Anwendungen des Metropolis-Algorithmus.

[14] E. Ising, Z. Phys. **31** (1925) 253.

s_i	Magnet	Besetzung
$+1$	$\uparrow$	besetzt
-1	$\downarrow$	leer

Tabelle 6.1: Mögliche physikalische Interpretationen für Ising-Variable $s_i = \pm 1$.

Das Ising-Modell wird mit Hilfe von N Spin-Variablen s_i definiert, die die Werte $s_i = \pm 1$ annehmen können. Mögliche physikalische Interpretationen dieser beiden Werte im Rahmen eines uniaxialen Magneten oder Besetzungen von Gitterplätzen, die höchstens einfach besetzt werden können, sind in Tabelle 6.1 angegeben.
Das ferromagnetische Ising-Modell ist auf einem Gitter über die Energie

$$E(s_1, \ldots, s_N) = -\sum_{i,j} J_{i,j}\, s_i\, s_j \qquad (6.37)$$

definiert. Ein parallel eingestelltes Paar von Spins $(s_i, s_j) = (\uparrow, \uparrow)$ oder $(\downarrow, \downarrow)$ trägt zu der Summe in (6.37) einen Beitrag $E = -J_{i,j}$ bei, ein antiparallel eingestelltes Paar von Spins $(s_i, s_j) = (\uparrow, \downarrow)$ oder $(\downarrow, \uparrow)$ ergibt hingegen einen Beitrag $E = +J_{i,j}$. Somit favorisiert die Energie (6.37) für $J_{i,j} > 0$ parallele Einstellungen des Spins; thermische Fluktuationen erzeugen für $T > 0$ allerdings auch Beiträge von antiparallelen Spin-Einstellungen.
Das Ising-Modell wird auch jenseits der Grenzen der Physik eingesetzt. Beispielsweise kann man das Zusammenspiel von Neuronen im Gehirn beschreiben: $s_i = 1$ steht für ein inaktives Neuron, $s_i = -1$ für ein aktives und $J_{i,j}$ beschreibt die Kopplung (Synapsen) zwischen einem gegebenen Paar von Neuronen. Durch Vorgabe der $J_{i,j}$ kann man dem System z.B. Muster aufprägen („lernen“), die dann aus einem verrauschten Anfangszustand durch *Simulated Annealing* herausgefiltert werden („erkennen“). Andere Anwendungen sind binäre Legierungen von Metallen, Optimierungsprobleme der Mathematik, Beschreibung von sozialen, politischen und ökonomischen Strukturen („Econophysics“) und vieles mehr.
Wir wollen uns nun genauer mit dem Modell (6.37) beschäftigen, wobei wir uns auf nächste-Nachbar-Wechselwirkung beschränken wollen, d.h. $J_{i,j} = 0$, wenn i und j keine nächsten Nachbarn auf dem Gitter sind. Ferner wollen wir uns auf den Fall uniformer Kopplungen beschränken, d.h.

$$J_{i,j} = \begin{cases} J & \text{für } i,\ j \text{ nächste Nachbarn,} \\ 0 & \text{sonst.} \end{cases} \qquad (6.38)$$

Soweit sieht alles ganz einfach aus. Man beachte allerdings, dass die Anzahl der Konfigurationen von N Spins $s_1, \ldots, s_N$ gleich 2^N ist. Betrachten wir z.B. ein 16×16 Quadratgitter, d.h. $N = 256$, so treten in der Summe (6.10) $2^{256} \approx 10^{77}$ Terme auf, was offensichtlich nicht explizit durchführbar ist (bereits ein 8×8 Quadratgitter oder ein $4 \times 4 \times 4$ einfach kubisches Gitter, d.h. $N = 64$, führen auf $2^{64} > 10^{19}$ Summanden, was auch praktisch nicht durchführbar ist). Hinzu kommt, dass vor allen bei niedrigen Temperaturen durch die Exponentialfunktion in (6.10) viele Beiträge stark unterdrückt sind.

Wir haben also wieder ein Problem des zweiten auf S. 86 definierten Typs vorliegen. Somit kommt wieder Importance-Sampling in der Form des Metropolis-Algorithmus zum Einsatz.

6.4.1 Algorithmus und Messgrößen

Die allgemeine Form des Algorithmus kennen wir bereits aus Kapitel 6.2.2. Bei der Wahl der Änderungsvorschläge hat man viel Freiheit, die man zur Effizienz-Steigerung verwenden kann. Wir stellen hier lediglich die Einzel-Spin-Flip-Variante des Metropolis-Algorithmus für das Ising-Modell vor:

1. Wähle einen Anfangszustand.
2. Wähle einen Platz j aus und schlage vor, den Spin an diesem Platz zu „flippen“, d.h. $s_j \to -s_j$ zu ersetzen (s_i mit $i \neq j$ bleibt unverändert).
3. Berechne $\Delta E = E_{\text{neu}} - E_{\text{alt}}$.
4. Ist $\Delta E \leq 0$, so behalte den geflippten Spin und fahre bei 7 fort.
5. Ist $\Delta E > 0$, so berechne $w = \exp(-\Delta E/(k_B\, T))$.
6. Erzeuge eine Zufallszahl r, die in $[0, 1]$ gleichverteilt ist.
   (a) Im Fall $r \leq w$ akzeptiere den Spin-Flip.
   (b) Für $r > w$ ist der Spin-Flip zu verwerfen, d.h. man muß man zum Ausgangzustand für s_j zurückkehren.
7. Fahre bei 2 fort, solange bis hinreichend viele Konfigurationen erzeugt sind.

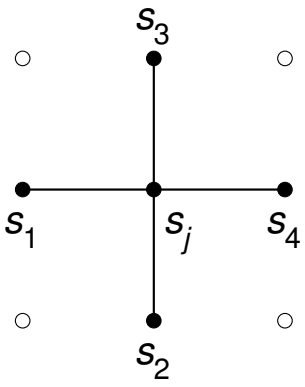


Abbildung 6.11: Illustration der 4 Nachbarn eines Spins s_j im Ising-Modell auf dem Quadratgitter.

Zunächst stellen wir fest, dass die Schritte 3–6 die Metropolis-Wahl (6.32) realisieren, die in diesem Fall

$$W(\text{alt} \to \text{neu}) = \min\left(1, \frac{p_{\text{neu}}}{p_{\text{alt}}}\right) = \min\left(1, \mathrm{e}^{-\Delta E/(k_B\,T)}\right) \tag{6.39}$$

lautet.

Genaugenommen hat man immer noch ein wenig Freiheit. So kann man die Gitterplätze im 2. Schritt zufällig auswählen (dies sollen Sie in den Übungen so machen). Alternativ könnte man die Plätze auch in einer bestimmten Reihenfolge (z.B. von oben links nach unten rechts) durchgehen. Die zweite Vorhgehensweise erfordert etwas weniger Operationen (insbes. Zufallszahlen), so dass wir uns für die geordnete Vorgehensweise in der unten angegebenen Beispiels-Implementierung entschieden haben. Damit der Algorithmus ergodisch wird, ist in jedem Fall darauf zu achten, dass alle Gitterplätze besucht werden.

Die Energie-Differenz ΔE im 3. Schritt sollte lokal berechnet werden. So hat der geflippte Spin s_j auf dem Quadratgitter nur 4 Nachbarn $s_1, \ldots, s_4$ (siehe Abb. 6.11). Bei dem Spin-Flip ändert sich nur die Wechselwirkung dieser vier Spins mit s_j, so dass sich mit (6.37) und $s_j^{\text{alt}} = -s_j^{\text{neu}}$

$$\Delta E = -2\,J\,\left(s_1 + s_2 + s_3 + s_4\right)\,s_j^{\text{neu}} \tag{6.40}$$

ergibt. Man kann noch eine weitere Optimierung durchführen: $(s_1+s_2+s_3+s_4)\,s_j^{\text{neu}}$ kann für ein Quadratgitter lediglich die 5 verschiedenen Werte -4, -2, 0, 2 und 4 annehmen. Für diese ganzzahligen Werte kann man leicht $w = \exp(-\Delta E/(k_B\,T))$ vorab berechnen und tabellieren, so dass die vergleichsweise teure Exponentialfunktion im 5. Schritt entfällt. Analoge Betrachtungen gelten für andere Gitter, allerdings ggfs. mit einer abweichenden Anzahl von Nachbarplätzen.

Die gewünschten Messungen (6.33) kann man vor dem 7. Schritt durchführen. Allerdings sollte man dies nicht zu oft tun, da ein einzelner Schritt nur wenig an der Konfiguration ändert und Messungen meistens teuer sind. Typischerweise sollte man einen „Sweep“ abwarten, d.h. N elementare MC-Schritte (bzw. im Mittel einen Flip-Versuch je Spin).
Auf folgende technische Details bei der Durchführung einer Simulation sei noch hingewiesen:

- Man verwendet gewöhnlich *periodische Randbedingungen*, um Randeffekte zu minimieren, die für endliche (insbesondere kleine) Systeme auftreten.
- Zu Beginn der Simulation benötigt man einige Zeit, um in den stationären Zustand zu gelangen. Daher muß die Simulation „thermalisiert“ werden, d.h. man muß sie einige Zeit laufen lassen, damit sie den willkürlich gewählten Anfangszustand vergißt. Erst nach dieser Thermalisierungs-Phase darf man Messungen von Gleichgewichts-Eigenschaften des Ising-Modells durchführen.

Eine vollständige Implementierung des Metropolis-Monte-Carlo-Algorithmus für das Ising-Modell auf einem dreidimensionalen kubischen Gitter mit $N = L^3$ Plätzen sieht dann z.B. wie folgt aus (dieser Code ist kein besonders elegantes C++, sollte aber eine recht CPU-effiziente Implementierung darstellen):

```
/* Metropolis-Monte-Carlo fuer das 3D Ising-Modell */

#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
typedef signed char BYTE;       // kleiner Datentyp (aber mit Vorzeichen)
const int L=64;                 // Kantenlaenge
const int N=L*L*L;              // Anzahl Spins
BYTE spins[L][L][L];            // Spin-Zustaende
BYTE field[L][L][L];            // Lokale Felder
int magn;                       // Magnetisierung
double weights[7];              // Boltzmann-Gewichte

/*  Initialisiere Boltzmann-Gewichte */

void init_weights(double ToverJ) {
  for(int r=0; r<=6; r++)
     weights[r] = exp(-2*r/ToverJ);
 }

/* Initialisiere Spin-Zustand und lokale Felder */
```

```
void init_field() {
  for(int x=0; x<L; x++)
   for(int y=0; y<L; y++)       // CPU-Cache-Optimierung:
    for(int z=0; z<L; z++)      // Schleife ueber 3. Argument innen
       spins[x][y][z] = 1;      // ferromagnetischer Zustand

  magn = 0;
  for(int x=0; x<L; x++)
   for(int y=0; y<L; y++)
    for(int z=0; z<L; z++) {
      field[x][y][z] =  spins[(x+1) % L][y][z] + spins[(x+L-1) % L][y][z]
                      + spins[x][(y+1) % L][z] + spins[x][(y+L-1) % L][z]
                      + spins[x][y][(z+1) % L] + spins[x][y][(z+L-1) % L];
      magn += spins[x][y][z];
     }
 }

/* Flippe Spin an Platz x,y,z */

inline void flip(int x, int y, int z) {
  BYTE s = spins[x][y][z];
  BYTE m2s = -s-s;
  magn += m2s;                          // Aktualisiere Magnetisierung
  // Lokale Felder der 6 Nachbarn aktualisieren
  field[(x+1) % L][y][z]   += m2s;      // Periodische Randbedingungen
  field[(x+L-1) % L][y][z] += m2s;      // werden ueber modulo L
  field[x][(y+1) % L][z]   += m2s;      // (% L) implementiert
  field[x][(y+L-1) % L][z] += m2s;      // und statt -1 addieren wir L-1
  field[x][y][(z+1) % L]   += m2s;
  field[x][y][(z+L-1) % L] += m2s;
  spins[x][y][z] = -s;                  // Zuletzt den Spin aktualisieren
 }


/* Einzelner Metropolis-Update-Schritt */

inline void metropolis(int x, int y, int z) {
  int DeltaE = spins[x][y][z]*field[x][y][z];
  if(DeltaE <= 0)
    flip(x, y, z);
  else  // fuer ernsthaften Einsatz: random() ersetzen (laengere Periode)
    if(random()/double(RAND_MAX) < weights[DeltaE])
       flip(x, y, z);
```

```
 }

/* Energie berechnen */

inline double energy()
 {
  double E = 0;
  for(int x=0; x<L; x++)
   for(int y=0; y<L; y++)       // CPU-Cache-Optimierung:
    for(int z=0; z<L; z++)      // Schleife ueber 3. Argument innen
      E += spins[x][y][z]*field[x][y][z];
  return(-E/2);
 }

/* Hauptprogramm */

int main(int argc, const char *argv[]) {
  if(argc != 2) {
    cerr << "Verwendung:\n\t" << argv[0] << " T/J" << endl;
    exit(13);
   }
  double ToJ = atof(argv[1]);
  cout << "# T/J = " << ToJ << endl;
  cout << "# L = " << L << endl;
  init_weights(ToJ);            // Boltzmann-Gewichte initialisieren
  init_field();                 // Systemzustand initialisieren

  cout << "# Sweep\tEnergie/Spin\tMagnetisierung/Spin\n";
  cout << 0 << "\t" << energy()/double(N)
            << "\t" << double(magn)/double(N) << endl;
   for(int sweep=0; sweep<100000; sweep++) {
    for(int x=0; x<L; x++)
     for(int y=0; y<L; y++)     // CPU-Cache-Optimierung:
      for(int z=0; z<L; z++)    // Schleife ueber 3. Argument innen
       metropolis(x, y, z);     // Einzelner Metropolis-Schritt
    cout << sweep+1 << "\t" << energy()/double(N)
                    << "\t" << double(magn)/double(N) << endl;
   }
 }
```

In einer solchen Simulation beobachtet man typischerweise einen (magnetischen) Ordnungübergang bei eine kritischen Temperatur T_c. Abbildung 6.12 illustriert dies für ein Quadratgitter, d.h. in zwei Dimensionen: Für $T > T_c$ liegt ein (magnetisch) *un*geordneter Zustand vor (vgl. Abb. 6.12 links), für $T < T_c$ hingegen ein (magne-

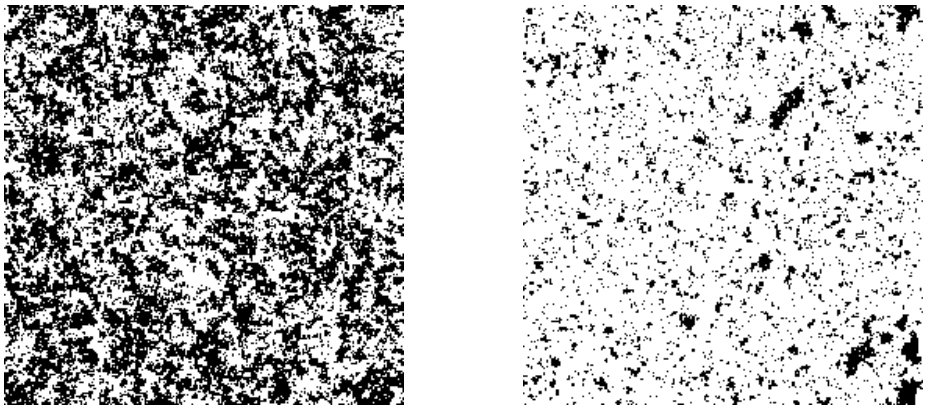


Abbildung 6.12: Momentane Zustände während einer Metropolis-Monte-Carlo-Simulation des Ising-Modells auf einem 256×256 Quadratgitter in der ungeordneten Hochtemperatur-Phase $T > T_c$ (links) und der geordneten Tieftemperatur-Phase $T < T_c$ (rechts).

tisch) geordneter Zustand (vgl. Abb. 6.12 rechts)[15]. Zur quantitativen Beschreibung dieser Physik führt man die *Magnetisierung*

$$M = \frac{1}{N} \sum_{i=1}^{N} s_i \qquad (6.41)$$

ein. Bei periodischen Randbedingungen gilt aufgrund von Translationsinvarianz

$$\langle s_i \rangle = \langle s_j \rangle = \langle M \rangle \qquad \forall i, j \,. \qquad (6.42)$$

Aus Symmetrie-Gründen (gleichzeitige Inversion aller Spins, d.h. $s_i \to -s_i$) ist $\langle M \rangle = 0$ für ein endliches System und alle Temperaturen. Man betrachtet daher M^2. Für diese Größe gilt

$$\lim_{N \to \infty} \langle M^2 \rangle = \begin{cases} 0 & \text{für } T > T_c, \\ M_0^2 > 0 & \text{für } T < T_c. \end{cases} \qquad (6.43)$$

M_0 hat die Bedeutung einer „spontanen Magnetisierung“, die ein sehr großes System annimmt, wenn es nur auf kurzen Zeitskalen betrachtet wird. Die Spin-Inversions-Symmetrie ist also im thermodynamischen Limes $N \to \infty$ im geordneten Zustand ($T < T_c$) spontan gebrochen.

[15] Eine gewisse Ähnlichkeit von Abb. 6.12 mit Abb. 6.1 ist kein Zufall. Allerdings würde eine Diskussion der Perkolations-Darstellung des Ising-Modells hier zu weit gehen.

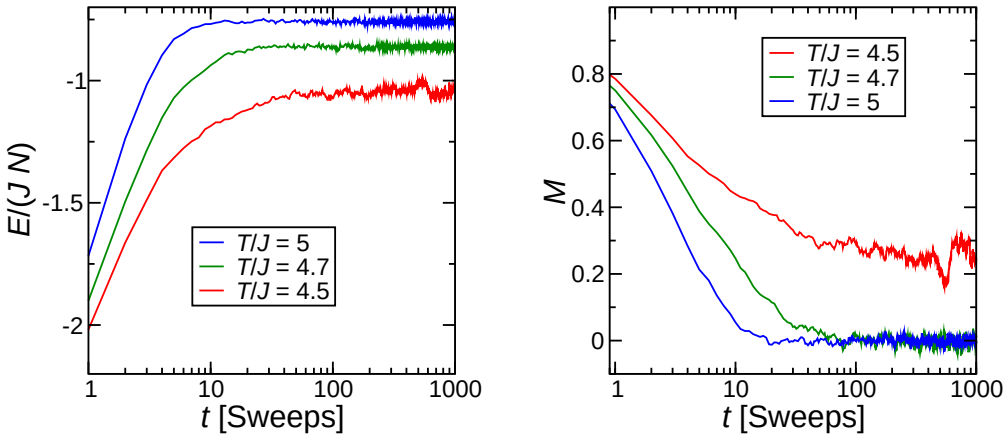


Abbildung 6.13: Relaxation von Energie E pro Platz N (linke Tafel) und Magnetisierung M (rechte Tafel) aus dem ferromagnetischen Zustand für eine Single-Spin-Flip Metropolis-Monte-Carlo Simulation des Ising-Modells auf einem $64 \times 64 \times 64$ einfach kubischen Gitter.

6.4.2 Ergebnisse

Wie bereits vom Lennard-Jones-Potential bekannt, ist es natürlich, die Monte-Carlo-„Zeit“ in Einheiten von sogenannten „Sweeps“ zu messen. Ein Sweep besteht per Definition aus N elementaren Schritten und entspricht genau der Anzahl Monte-Carlo Schritte, die erforderlich ist, um (im Mittel) für jeden Spin genau eine Änderung vorzuschlagen. Bei sehr hohen Temperaturen gilt für fast alle Zustände $E/T \approx 0$, so dass alle Zustände des Systems annähernd gleich wahrscheinlich sind. Folglich wird auch praktisch jeder vorgeschlagene Einzel-Spin-Flip akzeptiert. Ein einzelner Spin-Flip ändert natürlich den System-Zustand kaum. Flippt man jedoch Spins an N zufällig ausgewählten Plätzen, so erhält man einen Zustand, der nichts mehr mit dem Ausgangszustand zu tun hat. Bei sehr hohen Temperaturen wird man also grob einen Sweep benötigen, um einen komplett neuen Systemzustand zu erzeugen.

Auch hier wollen bzw. müssen wir uns mit den zeitlichen *Korrelationen* der Markov-Monte-Carlo-Simulation auseinandersetzen. Zunächst gilt auch bei dem Ising-Modell, dass wir zu Beginn der Simulation nicht im stationären Zustand sind, sondern erst eine ganze Reihe Schritte ausführen müssen, bevor wir Messungen vornehmen können, die der Gleichgewichts-Thermodynamik unterliegen.

Zur Illustration zeigt Abb. 6.13 die Relaxation von Energie E and Magnetisierung M aus einem *ferromagnetischen* Anfangszustand mit $s_i = 1$ für alle i. Gezeigt sind Daten einer Simulation, die auf einem $L \times L \times L$ einfach kubischen Gitter der Kantenlänge $L = 64$ bei Temperaturen $k_B\,T/J = 5$, 4.7 und 4.5 durchgeführt wurden.

$k_B\,T/J$	5	4.7	4.5
τ_E	1.25	2.8	120
τ_M	3.8	13	70000

Tabelle 6.2: Ungefähre Autokorrelationszeiten einer Single-Spin-Flip Metropolis-Monte-Carlo Simulation des Ising-Modells auf einem $64\times64\times64$ einfach kubischen Gitter.

Für kleine Zeiten t sieht man noch deutlich das Gedächtnis des Anfangszustandes, der mit $t=0$ bei der logarithmischen Skalierung der Zeitachse außerhalb von Abb. 6.13 liegt. Da das einfach kubische Gitter drei mal soviele Verbindungen wie Plätze hat, ist die Energie (6.37) des ferromagnetischen Zustandes gegeben durch $E(1,\ldots,1) = -3\,J\,N = -3\,J\,L^3$. Für die Magnetisierung (6.41) des ferromagnetischen Zustandes gilt $M = 1$. Der linken Tafel von Abb. 6.13 entnimmt man, dass die Energie bei diesen Temperaturen in einer Größenordnung von maximal 100 Sweeps thermalisiert. Dies ist bereits deutlich länger als der eine Sweep, den wir bei hohen Temperaturen abgeschätzt haben – im günstigsten dargestellten Fall $T = 5\,J/k_B$ benötigen wir immer noch ≈ 10 Sweeps für die Thermalisierung der Energie E. Die rechte Tafel von Abb. 6.13 zeigt jedoch, dass die Magnetisierung M noch einmal – zumindest bei $T = 4.5\,J/k_B$ deutlich – langsamer relaxiert.
Für eine genauere Charakterisierung des dynamischen Verhaltens betrachten wir die Autokorrelationsfunktion (6.35). Im Ising-Modell gilt für $A = s_i$ oder $A = M$, dass $\langle s_i\rangle = 0 = \langle M\rangle$. Die Autokorrelationsfunktion (6.35) vereinfacht sich daher für die Magnetisierung M zu

$$C_M(t) = \frac{\langle M(t_0+t)\,M(t_0)\rangle}{\langle M^2\rangle}\,. \tag{6.44}$$

Für eine einzelne Spin-Variable s_i gilt darüber hinaus $s_i^2 = 1$, also auch $\langle s_i^2\rangle = 1$, so dass sich die Autokorrelationsfunktion weiter vereinfacht:

$$C_{s_i}(t) = \langle s_i(t_0+t)\,s_i(t_0)\rangle\,. \tag{6.45}$$

Abbildung 6.14 zeigt Ergebnisse für die Autokorrelationsfunktion des Ising-Modells auf einem einfach kubischen Gitter der Kantenlänge $L = 64$. Für die Magnetisierung haben wir (6.44), d.h. $\langle M\rangle = 0$ verwendet. Die Fehler in Abb. 6.14 sind über die Auswertung der Autokorrelationsfunktion in 10 unabhängigen Läufen bestimmt worden.
Eine Anpassung der Daten in Abb. 6.14 an die exponentielle Form (6.36) führt auf die grobe Abschätzung der Autokorrelationszeiten in Tabelle 6.2. Man beobachtet, dass grundsätzlich $\tau_M > \tau_E$ (dies ist bereits aus Abb. 6.14 ersichtlich – beachte

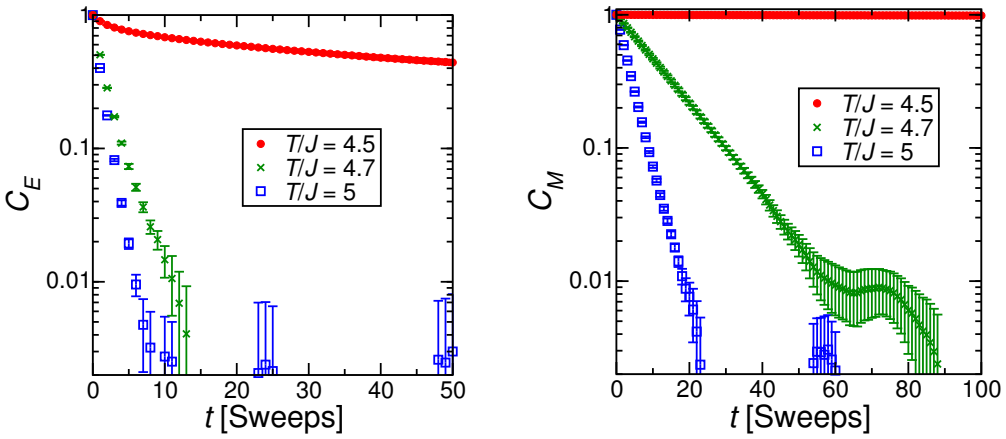


Abbildung 6.14: Autokorrelationsfuntion von Energie E (linke Tafel) und Magnetisierung M (rechte Tafel) im stationären Zustand einer Single-Spin-Flip Metropolis-Monte-Carlo Simulation des Ising-Modells auf einem $64 \times 64 \times 64$ einfach kubischen Gitter.

die unterschiedliche horizontale Skalierung der beiden Tafeln). Für $k_B T \gtrsim 4.7\, J$ findet man Autokorrelationszeiten, die 10 Sweeps nicht wesentlich überschreiten. Für $k_B T = 4.5\, J$ relaxiert die Energie langsam und die Magnetisierung praktisch gar nicht mehr. Dies liegt daran, dass wir (wie wir später sehen werden), hier knapp unterhalb von T_c liegen. Unter diesen Umständen muß man nicht nur mit der Thermalisierung aufpassen, sondern man sollte auch von häufigen Messungen Abstand nehmen, insbesondere wenn diese Rechenzeit-intensiv sind.

Oft ist man an dem Verhalten von Meßgrößen als Funktion der Temperatur T interessiert. In diesem Fall kann man zumindest dem Thermalisierungsproblem ein kleines Schnippchen schlagen, indem man bei hohen Temperaturen anfängt, schrittweise zu niedrigen Temperaturen geht und den gut thermalisierten Endzustand bei einer Temperatur jeweils als Anfangszustand bei der nächst niedrigeren Temperatur wählt. Ist der Temperatur-Schritt hinreichend klein, wird man anschließend bereits recht nah am Gleichgewichtszustand sein und kommt mit entsprechend wenigen Thermalisierungs-Schritten aus[16].

Abbildung 6.15 zeigt Ergebnisse von MC-Simulationen der Magnetisierung des Ising-Modells auf dem einfach kubischen Gitter. Im Sinn von (6.43) verwenden wir hier

$$\bar{M} = \sqrt{\langle M^2 \rangle}\,.$$

[16] Diese Art, über langsames Abkühlen zu thermalisieren, wird als *Annealing* bezeichnet. Man kann auch ein artifizielles „Abkühlen“ durchführen, um ein Energie-Minimum, d.h. den Grundzustand zu finden. Ein solches Verfahren wird dann *Simulated Annealing* genannt.

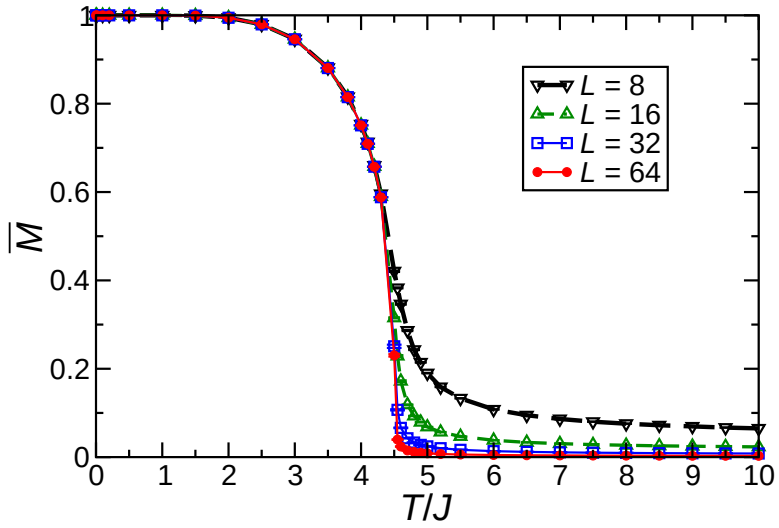


Abbildung 6.15: Magnetisierung des ferromagnetischen Ising-Modells auf einem einfach kubischen Gitter der Kantenlänge L als Funktion der Temperatur T.

Bei den Simulationen für Abb. 6.15 wurde die Temperaturachse entsprechend dem gerade geschilderten Protokoll von hohen zu tiefen Temperaturen abgetastest. Das Problem großer Autokorrelationszeiten bei bestimmten Temperaturen bleibt natürlich bestehen. In Abb. 6.15 sind die Fehlerbalken, die wir mit Hilfe von (5.9) aus 10 *unabhängigen* Simulationen bestimmt haben, trotzdem kaum zu sehen.
Abbildung 6.15 bestätigt zunächst (6.43). Zwar sind die Meßwerte von $\bar{M}$ auf einem endlicher System für alle Temperaturen endlich, jedoch werden sie bei hohen Temperaturen mit zunehmender Temperatur immer kleiner und gehen schließlich mit $L \to \infty$ (bzw. $N = L^3 \to \infty$) gegen Null. Unterhalb der kritischen bzw. Curie-Temperatur T_c gehen die Werte mit $N \to \infty$ hingegen gegen einen konstanten Wert $M_0 > 0$. Aus Abb. 6.15 liest man grob ab:

$$T_c \approx 4.5\, J/k_B\,. \tag{6.46}$$

Das Verhalten der Magnetisierung des Ising-Modells in Abb. 6.15 hat durchaus Ähnlichkeit mit der Perkolationswahrscheinlichkeit in Abb. 6.3. Während die Perkolationswahrscheinlichkeit $P_\infty(p)$ beim Phasenübergang springt, ist die Magnetisierung offensichtlich glatter. Tatsächlich genügt sie für $T \nearrow T_c$ einem Potenzgesetz

$$M_0 \propto (T_c - T)^\beta\,. \tag{6.47}$$

Bei β handelt es sich –genau wie bei ν im Perkolationsproblem– um einen *kritischen Exponenten*.
Eine genaue Bestimmung von T_c und anschließend β würde auch in methodischer Sicht deutlich zu weit führen. Wir beschränken uns daher darauf, Literaturwerte

anzugeben[17]:

$$T_c = (4.5115232 \pm 0.0000016)\, J/k_B \tag{6.48}$$

für das einfach kubische Gitter, und[18]

$$\beta = 0.32652 \pm 0.00009 \tag{6.49}$$

in *drei Dimensionen*.

Der Vollständigkeit halber seien auch die entsprechenden Ergebnisse in *zwei Dimensionen* angegeben. Das Ising-Modell ist auf dem Quadratgitter exakt lösbar, so dass man hier (im Gegensatz zu drei Dimensionen) keine numerischen Verfahren zur Berechnung der thermodynamischen Eigenschaften benötigt. Umgekehrt hat sich diese exakte Lösung sehr nützlich als Referenz zum Testen numerischer Verfahren erwiesen.

Die kritische Temperatur kann man einer berühmten Arbeit von Lars Onsager aus dem Jahr 1944 entnehmen[19]:

$$\frac{k_B\, T_c}{J} = \frac{2}{\ln\left(1+\sqrt{2}\right)} \approx 2.269 \tag{6.50}$$

für das Quadratgitter. Der Verlauf der spontanten Magnetisierung $M_0(T)$ (6.43) ist für das Quadratgitter ebenfalls exakt bekannt[20]. Insbesondere kennt man auch den Wert des kritischen Exponenten

$$\beta = \frac{1}{8} \tag{6.51}$$

in zwei Dimensionen exakt.

[17] Siehe z.B. M. Hasenbusch, Phys. Rev. B **82** (2010) 174433.

[18] Um diesen Wert aus den in der Referenz angegebenen Werten für η und ν zu extrahieren, muß man die Relation $\beta = \nu\,(2 - d + \eta)/2$ mit $d = 3$ verwenden.

[19] L. Onsager, Phys. Rev. **65** (1944) 117.

[20] C.N. Yang, Phys. Rev. **85** (1952) 808.

Kapitel 7

Schlußbemerkungen

Dieses Skript basiert auf der Vorlage von Thomas Pruchke [1], für deren Überlassung ich mich herzlich bedanke. Ich habe mich zwar dann doch entschlossen, insbesondere in der zweiten Hälfte von der Vorlage abzuweichen bzw. sie zu modifizieren, aber auch an vielen dieser Stellen ist die Vorlage immer noch zu erkennen. Zumindest Programmteile gehen letzten Endes auf Alexander Hartmann zurück. Weitere Beiträge in diesem Skript mögen ursprünglich Marcus Müller und Reiner Kree zuzuordnen zu sein.

Dank schulde ich ferner Steffen Klemer und Claus Heussinger für wichtige Korrekturhinweise. Nicht zuletzt möchte ich Reiner Kree für nützliche Hinweise und Claus Heussinger für die kompetente Betreuung des Übungsbetriebs herzlich danken.

Dieses Skript erhebt nicht den Anspruch eines Lehrbuches. Einige Themen (wie z.B. Zufallswege) wurden zwar in den Übungen behandelt, aber nicht in den Vorlesungen. Dementsprechend weist auch dieses Skript Lücken auf. Auch ist ein komplementärer Standpunkt oft erhellend. Ich möchte Sie daher ausdrücklich ermutigen, weitere Literatur zu konsultieren.

Vermutlich enthält dieses Skript noch einige (Tipp-) Fehler – entsprechende Hinweise nehme ich jederzeit gerne entgegen.

Göttingen, 15. Juli 2011

Andreas Honecker

Anhang

Dieser Anhang enthält ergänzende Informationen zu Themen, die zwar in den Übungsaufgaben, aber nicht in der Vorlesung behandelt wurden.

A Zufallswege

In den Übungsaufgaben haben Sie Zufallswege simuliert. Hier wollen wir noch einige Bemerkungen zu *freien Zufallswegen* ergänzen. Diese können als ein mikroskopisches Modell für Diffusionsprozesse angesehen werden. Zunächst diskretisieren wir das Problem, d.h. wir betrachten nur solche Orte, die auf einem kubischen Gitter liegen. Ein Zufallsweg besteht nun aus einer Folge von Punkten $\vec{x}_n$, wobei wir die Anfangsbedigung $\vec{x}_0 = \vec{0}$ wählen. Ein Schritt besteht dann aus

$$\vec{x}_{n+1} = \vec{x}_n \pm \vec{e}_r \,, \tag{A.1}$$

wobei wir die Schrittrichtung durch einen zufälligen Einheitsvektor $\vec{e}_r$ mit einem zufälligen Vorzeichen wählen.
Betrachten wir als Beispiel das Quadratgitter, d.h. zwei Dimensionen. Dann lauten die Anfangsbedingungen $x_0 = y_0 = 0$. Ein Schritt besteht in diesem Fall aus

$$\begin{array}{lll} \text{entweder} & x_{n+1} = x_n - 1\,, & y_{n+1} = y_n \\ \text{oder} & x_{n+1} = x_n + 1\,, & y_{n+1} = y_n \\ \text{oder} & x_{n+1} = x_n\,, & y_{n+1} = y_n - 1 \\ \text{oder} & x_{n+1} = x_n\,, & y_{n+1} = y_n + 1\,. \end{array} \tag{A.2}$$

Die Auswahl zwischen diesen vier Möglichkeiten ist zufällig, aber mit gleicher Wahrscheinlichkeit zu treffen. Dies kann z.B. über eine ganzzahlige Zufallszahl geschehen, die Werte im Intervall $[1, 4]$ (oder $[0, 3]$) annimmt und die Nummer der Richtung für den nächsten Schritt angibt.
Abb. A.1 zeigt drei auf diese Weise erzeugte Zufallswege in zwei Dimensionen. Interessant sind nun Eigenschaften im statistischen Mittel über viele Wege. Solche Mittelwerte wollen wir wie bereits zuvor mit eckigen Klammern „$\langle\cdot\rangle$“ bezeichnen. Mögliche Meßgrößen sind z.B.

$$\langle \Delta x_n^2 \rangle = \langle x_n^2 \rangle - \langle x_n \rangle^2 \,, \qquad \langle \Delta y_n^2 \rangle = \langle y_n^2 \rangle - \langle y_n \rangle^2 \,. \tag{A.3}$$

A.1 Mittlerer zurückgelegter Weg

Für die einfachen freien Wege ist es möglich, den im Mittel bis zu Schritt n zurückgelegten Weg elementar analytisch auszurechnen. In d Dimensionen ist der in n Schritten zurückgelegte Weg wie folgt definiert (beachte, dass wir den Ausgangspunkt zu $\vec{x}_0 = \vec{0}$ gewählt hatten):

$$\Delta R_n^2 = \vec{x}_n^2 = \sum_{i=1}^{d} \left(x_n^{(i)} \right)^2 \,. \tag{A.4}$$

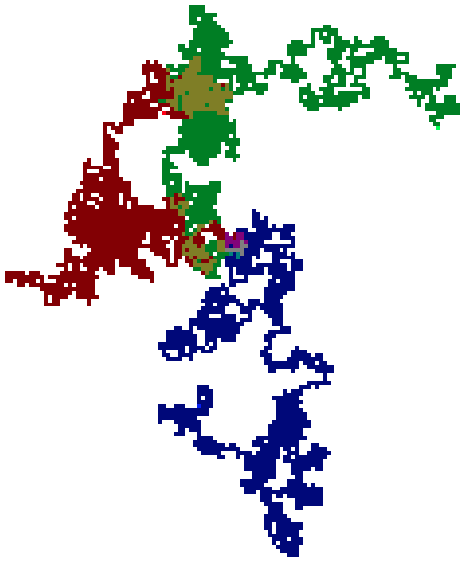


Abbildung A.1: Beispiel für freie Zufallswege: drei Läufer (gekennzeichnet durch die Farben Rot, Grün und Blau) sind auf einem Quadratgitter im Ursprung losgelaufen. Die besuchten Plätze sind mit der jeweiligen Farbe eingefärbt; die aktuelle Position des betreffenden Läufers ist durch einen hellen Punkt gekennzeichnet.

Führen wir nun die Schrittweite im Schritt n und Richtung i ein

$$\delta x_n^{(i)} = x_n^{(i)} - x_{n-1}^{(i)}\,, \tag{A.5}$$

so ergibt sich für (A.4)

$$\begin{aligned} \Delta R_n^2 &= \sum_{i=1}^{d} \left(x_n^{(i)}\right)^2 = \sum_{i=1}^{d} \left(\sum_{m=1}^{n} \delta x_m^{(i)}\right)^2 \\ &= \sum_{i=1}^{d} \left(\sum_{m=1}^{n} \delta x_m^{(i)}\right)^2 + 2\sum_{i=1}^{d}\sum_{k<l} \delta x_k^{(i)}\, \delta x_l^{(i)}\,. \end{aligned} \tag{A.6}$$

Nun ist der Mittelwert dieser Größe zu bilden. Für einen einzelnen Schritt in Richtung i gibt es die Möglichkeit, dass keine Veränderung stattfindet ($\delta x_k^{(i)} = 0$) oder eine Veränderung um ± 1, wobei beide Vorzeichen gleich wahrscheinlich sind. Im Mittel gilt somit

$$\langle \delta x_k^{(i)} \rangle = 0\,. \tag{A.7}$$

Für $k \neq l$ sind die Schrittrichtungen unabhängig, so dass der entsprechende Mittelwert faktorisiert. Nach Faktorisierung können wir (A.7) verwenden und finden somit:

$$\langle \delta x_k^{(i)}\, \delta x_l^{(i)} \rangle = \langle \delta x_k^{(i)} \rangle \, \langle \delta x_l^{(i)} \rangle = 0 \tag{A.8}$$

für $k \neq l$. Damit folgt, dass der Mittelwert des zweiten Terms im Ergebnis (A.6) verschwindet. Wir haben somit

$$\langle \Delta R_n^2 \rangle = \sum_{i=1}^{d} \sum_{m=1}^{n} \left\langle \left(\delta x_m^{(i)} \right)^2 \right\rangle = \sum_{m=1}^{n} \left\langle \sum_{i=1}^{d} \left(\delta x_m^{(i)} \right)^2 \right\rangle = n \,. \tag{A.9}$$

Im letzten Schritt haben wir verwendet, dass die Schrittlänge bei unserer Vorschrift immer eins ist, d.h. $\sum_{i=1}^{d} \left(\delta x_m^{(i)} \right)^2 = 1$. Wir haben damit das *exakte Ergebnis*, dass für den mittleren zurückgelegten Weg gilt:

$$R := \sqrt{\langle \Delta R_n^2 \rangle} = \sqrt{n} \,. \tag{A.10}$$

Man beachte, dass in diesem Ergbnis die Dimension d nicht mehr explizit auftritt. Glg. (A.10) ist ein wichtiges Ergebnis und als Diffusionsgesetz bekannt: der im Mittel zurückgelegte Weg R wächst mit der Wurzel der Schritte n bzw. bei einer gegebenen Zahl von Schritten pro Zeitintervall proportional zur Wurzel der Zeit t. Im allgemeinen gilt (A.10) nur asymptotisch und es tritt noch eine Proportionalitätskonstante auf, die Diffusionskonstante genannt wird. Aus (A.9) liest man leicht ab, dass diese Diffusionskonstante durch die mittlere Schrittlänge gegeben ist.

A.2 Mastergleichung

Weitergehende Aussagen kann man mit Hilfe der Mastergleichung (6.18) gewinnen. Zunächst lautet die diskrete Mastergleichung für den Zufallsweg (A.1)

$$P(\vec{x}, k+1) - P(\vec{x}, k) = \frac{1}{2\,d} \sum_{r=1}^{d} \Big(P(\vec{x} - \vec{e}_r, k) + P(\vec{x} + \vec{e}_r, k) \Big) - P(\vec{x}, k) \tag{A.11}$$

mit der Anfangsbedingung

$$P(\vec{x}, 0) = \delta_{\vec{x},\vec{0}} \,. \tag{A.12}$$

Für die analytische Behandlung ist es nützlich, zunächst einen zeitlichen Kontinuumslimes durchzuführen. Nehmen wir an, dass die Übergangswahrscheinlichkeiten mit der Länge des Zeitschritts τ skalieren, so können wir die *Übergangsraten*

$$\boxed{\gamma(\vec{x} \to \vec{y}) = \lim_{\tau \to 0} \frac{T(\vec{x} \to \vec{y})}{\tau}} \tag{A.13}$$

einführen. Aus der diskreten Mastergleichung (6.18) wird mit

$$t = \tau\, k \tag{A.14}$$

nun eine *kontinuierliche Mastergleichung* (oft auch einfach nur als „Mastergleichung“ bezeichnet)

$$\frac{\partial P(\vec{x},t)}{\partial t} = \sum_{\vec{y}} \Big(\gamma(\vec{y} \to \vec{x})\, P(\vec{y},t) - \gamma(\vec{x} \to \vec{y})\, P(\vec{x},t) \Big) \,. \tag{A.15}$$

Wir müssen also zunächst die Wahrscheinlichkeit, dass überhaupt ein Schritt stattfindet, auf $\alpha\,\tau < 1$ reduzieren[1]. Anstelle von (A.11) haben wir dann

$$P(\vec{x},k+1) - P(\vec{x},k) = \frac{\alpha\,\tau}{d} \sum_{r=1}^{d} \left(\frac{1}{2} \left(P(\vec{x} - a\,\vec{e}_r, k) + P(\vec{x} + a\,\vec{e}_r, k) \right) - P(\vec{x},k) \right) \tag{A.16}$$

Wir haben bei dieser Gelegenheit auch gleich einen Gitterabstand a eingeführt. Im Sinn von (A.15) lautet die kontinuierliche Variante von (A.16)

$$\frac{\partial P(\vec{x},t)}{\partial t} = \frac{\alpha}{d} \sum_{r=1}^{d} \left(\frac{1}{2} \left(P(\vec{x} - a\,\vec{e}_r, t) + P(\vec{x} + a\,\vec{e}_r, t) \right) - P(\vec{x},k) \right) \,. \tag{A.17}$$

Auf der rechten Seite erkennt man die Diskretisierung (1.12) der zweiten räumlichen Ableitung wieder. Man kann also auch einen räumlichen Kontinuumslimes durchführen und findet die *Diffusionsgleichung*

$$\frac{\partial P(\vec{x},t)}{\partial t} = D \sum_{r=1}^{d} \frac{\partial^2 P(\vec{x},t)}{\partial x_r^2} =: D\,\Delta\, P(\vec{x},t) \,. \tag{A.18}$$

Der Wert der Diffusionskonstanten folgt aus (A.17) zu

$$D = \lim_{a \to 0} \frac{a^2\,\alpha}{2\,d} \,. \tag{A.19}$$

Der Nutzen der ganzen Kontinuumsnäherungen ist, dass die Lösung von (A.18) leicht angegeben werden kann:

$$P(\vec{x},t) = \frac{1}{(4\,\pi\,D\,t)^{d/2}}\, \mathrm{e}^{-\vec{x}^2/(4\,D\,t)} \,. \tag{A.20}$$

Erstens möge man sich durch Einsetzen überzeugen, dass (A.20) die partielle Differentialgleichung (A.18) löst. Zweitens kann man nachrechnen, dass (A.20) normiert ist:

$$\int \mathrm{d}^d x\, P(\vec{x},t) = 1 \,. \tag{A.21}$$

Drittens gilt $\lim_{t\to 0} P(\vec{x},t) = 0$ für $\vec{x} \neq 0$. In Kombination mit der Normierungsbedingung (A.21) muß also

$$\lim_{t \to 0} P(\vec{x},t) = \delta^{(d)}(\vec{x}) \tag{A.22}$$

[1] Ein $\alpha\,\tau < 1$ beseitigt auch die artifizielle Untergitterentkopplung.

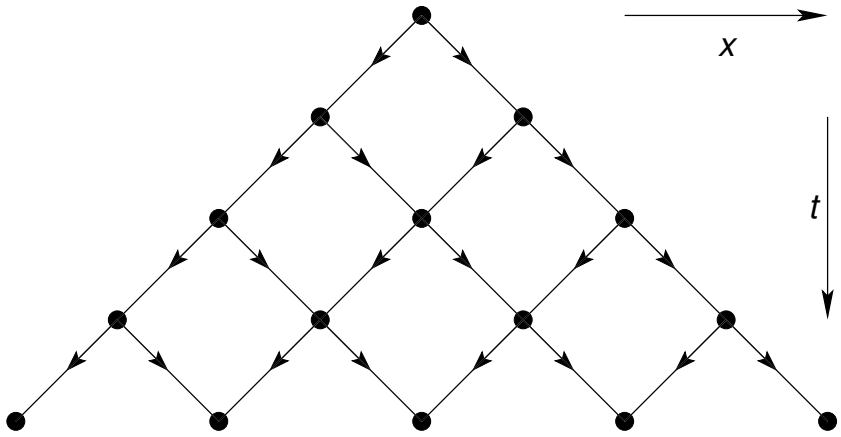


Abbildung A.2: Gerichtete Perkolation: Flüssigkeit kann entlang der Pfeile zwischen den Plätzen eines auf die Spitze gestellten Quadratgitters fließen.

gelten, mit $\delta^{(d)}(\vec{x})$ der d-dimensionalen δ-Distribution. Glg. (A.22) ist nichts anderes als der Kontinuumslimes unserer Anfangsbedingung (A.12).
Die Lösung (A.20) ist eine Gauß-Funktion, so dass wir hier in einem Spezialfall explizit den zentralen Grenzwertsatz (5.8) verifiziert haben.
Das zweite Moment der Lösung (A.20) ist

$$R(t)^2 := \int \mathrm{d}^d x \, \vec{x}^2 \, P(\vec{x}, t) = 2\, d\, D\, t\,. \tag{A.23}$$

Erinnert man sich an die Definitionen (A.14), (A.19) und eliminiert die ganzen Faktoren, die wir für die Kontinuums-Limites eingeführt hatten, so reproduziert (A.23) das Ergebnis (A.10), das wir zuvor mit elementaren Mitteln hergeleitet hatten.

B Gerichtete Perkolation

Sie haben sich in einer Übungsaufgabe mit gerichteter Perkolation beschäftigt. Hierzu möchte ich noch einige Hintergrundbemerkungen ergänzen. Im Gegensatz zu der in Kapitel 6.1 behandelten „normalen“ oder “isotropen“ Perkolation, nehmen wir nun an, dass eine Vorzugsrichtung vorliegt, die z.B. durch die Gewichtskraft oder einen Druckunterschied definiert sein kann[2].
Wir betrachten als Modell ein Quadratgitter, das um 45 Grad gedreht ist (vgl. Abb. A.2). Den Plätzen dieses Gitters werden Zellen zugeordnet, die Verbindungen zu ihren Nachbarn besitzen (gekennzeichnet durch die Linien in Abb. A.2). Die Flüssigkeit kann nun nur entlang einer Vorzugsrichtung vordringen, die durch die Pfeile in

[2]Eine Einführung in das Modell sowie eine Diskussion seiner Bedeutung findet man z.B. in H. Hinrichsen, Adv. Phys. **49** (2000) 815 und I. Jensen, J. Phys. A: Math. Gen. **29** (1996) 7013.

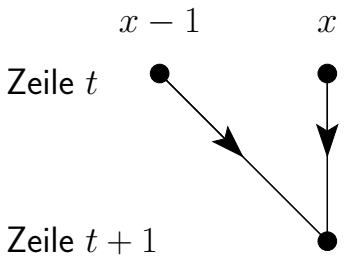


Abbildung A.3: Illustration eines lokalen Aktualisierungs-Schrittes bei der Simulation gerichteter Perkolation.

Abb. A.2 gekennzeichnet ist. Somit sind die beiden Richtungen des zweidimensionalen Gitters nicht äquivalent. Unter Berücksichtigung des zeitlichen Ablaufs der Flüssigkeitsausbreitung kann die horizontale Richtung als „Ort“ (bezeichnet mit x) und die vertikale Richtung als „Zeit“ (bezeichnet mit t) aufgefaßt werden.
Die Porösität des Mediums wird dadurch abgebildet, dass mögliche Wege nur mit einer gewissen Wahrscheinlichkeit tatsächlich durchlässig sind. Genau wie bei der isotropen Perkolation aus Kapitel 6.1 gibt es zwei Varianten des Modells:

- Platz („Site“) Perkolation: Die Plätze des Gitters sind mit Wahrscheinlichkeit p durchlässig (die Verbindungen immer).
- Verbindungs („Bond“) Perkolation: Die Verbindungen zwischen den Gitterplätzen sind mit Wahrscheinlichkeit p durchlässig (die Plätze selbst hingegen immer).

Ist p klein, so kann die Flüssigkeit nicht besonders tief in das Medium eindringen. Im Grenzfall $p \to 1$ sind hingegen alle Wege offen, so dass das Medium auf jeden Fall durchlässig wird. Dazwischen findet man auch hier wieder einen Phasen bei einer Perkolationsschwelle p_c.
Die Simulation gerichteter Perkolationen bereitet keine großen Schwierigkeiten. Für eine effiziente Simulation dieses Problems sollte man allerdings nicht den kompletten Systemzustand speichern. Tatsächlich hängt der Zustand einer Zeile nur von der direkt darüber ab. Man kann sich daher darauf beschränken, nur *eine Zeile* zu speichern. Die Beschränkung auf die Speicherung einer Zeile erlaubt es, diese sehr lang zu wählen. Es empfiehlt sich nun, die Zellen so zu versetzen, dass die beiden Nachbarn einer Zelle am Ort x zur Zeit $t+1$ bei $x-1$ und x zur Zeit t liegen (vgl. Abb. A.3). Der Inhalt der Zelle x zur Zeit $t+1$ kann nun lokal aus den Inhalten der Zellen $x-1$ und x zur Zeit t bestimmt werden. Zunächst kann die Zelle am Platz x nur dann nass werden, wenn mindestens eine der beiden Nachbarzellen im vorhergehenden Schritt nass war. Erst wenn diese Bedingung erfüllt ist, muß man

Zufallszahlen auswürfeln um zu bestimmen, ob Flüssigkeit in die Zelle fließen kann und diese somit nass wird. Bei Platz-Perkolation benötigt man eine Zufallszahl um zu entscheiden, ob der Platz durchlässig ist, aber eben nur, falls einer der beiden vorhergehenden Plätze nass war. Bei Verbindungs-Perkolation benötigt man je eine Zufallszahl, um zu entscheiden, ob eine Verbindung zu einem nassen Nachbarplatz offen ist. Stellt sich heraus, dass bereits die erste von zwei möglichen Verbindungen durchlässig ist, muß man tatsächlich auch die zweite Verbindung nicht mehr ansehen. Geht man von links nach rechts durch die Zeile, so muss man den Zustand der Zelle an Platz x zur Zeit t noch einmal zwischenspeichern, da man ihn zur Bestimmung des Zustands der Zelle an Platz $x+1$ zur Zeit $t+1$ benötigt. Unter Verwendung *eines* temporären Speicherplatzes kommt man aber trotzdem mit einer Zeile aus, die man jeweils beim Durchgang überschreibt.

Zweitens empfiehlt es sich, nur mit einem nassen Platz (z.B. bei $x = 0$) zu beginnen. Die Frage nach der Durchlässigkeit kann auch mit diesen Anfangsbedingungen beantwortet werden, nur wird bei einem einzigen nassen Anfangspunkt auch für $p > p_c$ nicht bei jeder Realisierung die Flüssigkeit beliebig weit in das System eindringen. Man muß daher viele Realisierungen betrachten und die Häufigkeiten analysieren, mit denen der Cluster tief in das System eindringt (als „Cluster“ bezeichnen wir eine Menge nasser Plätze, die miteinander verbunden sind).

Mit dieser Vorgehensweise simulieren Sie zwar nur einen Cluster, aber dafür in einer räumlich sehr großen Geometrie. Tatsächlich sieht ein Cluster, der zur Zeit $t = 0$ bei $x = 0$ anfängt, für lineare Ausdehnungen $L \geq t + 1$ nie den Rand. Man simuliert somit ein effektiv unendlich ausgedehntes System. Ferner benötigt man Zufallszahlen nur am (aktuellen) Rand des Clusters, und man muß auch nur diesen ansehen. Irrelevante Teile des Systems müssen hingegen gar nicht betrachtet werden.

Als Vergleichswert für die Perkolationsschwelle p_c zitieren wir einen der genausten Schätzwerte für kritischen Punkt p_c bei Platz-Perkolation, der auf einer Reihenentwicklung 158. Ordnung basiert[3]:

$$p_c = 0.70548522(4)\,. \tag{A.24}$$

Bei Annäherung an p_c beobachtet man sogenanntes „kritisches Verhalten“, d.h. die Größen der Cluster divergieren. Für eine quantitative Analyse betrachtet man die Häufigkeit einer Observablen, z.B. mit $N(x,p)$ eine (geeignet normierte) Anzahl von Clustern der Größe x bei einem gegebenen p. Eine Möglichkeit, die „Größe“ eines Clusters zu definieren, ist in Abb. A.4 illustriert: die extremalen räumlichen Koordinaten definieren eine Breite x, sowie die Differenz zwischen Anfangs- und

[3] I. Jensen, J. Phys. A: Math. Gen. **32** (1999) 5233.

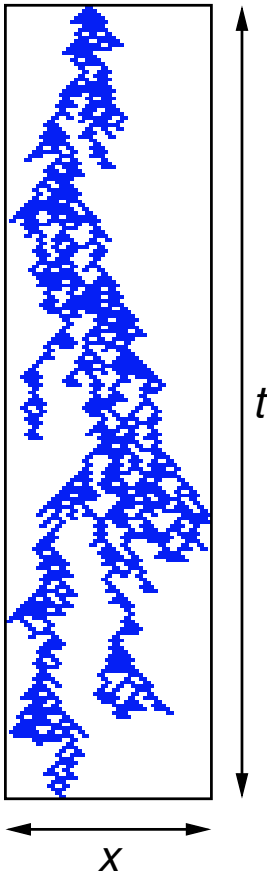


Abbildung A.4: Definition der Ausdehnung eines Clusters: der Cluster wird in ein Rechteck eingeschrieben, dessen Breite bzw. Höhe die räumliche Ausdehnung x bzw. die zeitliche Ausdehnung t definieren. Für ein gegebenes p haben Cluster eine charakteristische räumliche Ausdehnung $\xi_\perp$ sowie eine charakteristische zeitliche Länge $\xi_\parallel$. Man beachte, dass in der Abbildung ein Platz durch 2×1 Pixel dargestellt wird, so dass der Cluster in horizontaler Richtung um den Faktor 2 gestreckt erscheint.

Endzeitpunkt eine Länge t. Nebenbei beachte man die starke Asymmetrie des Clusters in Abb. A.4 (es handelt sich um einen für Platz-Perkolation mit $p = 0.6805$ ausgewürfelten Cluster). Dies spiegelt die deutlich verschiedenen Rollen wider, die die beiden Richtungen bei gerichteter Perkolation spielen. Für ein gegebene p haben die Cluster charakteristische Ausdehnungen, die wir mit $\xi_\perp$ (für den Ort) und $\xi_\parallel$ (für die Zeit) bezeichnen wollen.

Zur weiteren Analyse machen wir eine sogenannte „Skalenannahme“: Wir nehmen an, dass die Häufigkeitsverteilungen –zumindest in der Nähe des kritischen Punktes– nur von dem Argument bezogen auf die relevante Skala abhängen, und nicht direkt

von dem Systemparameter, der hier p ist. Für die Häufigkeit $N(x,p)$, einen Cluster der Größe x zu finden, machen wir z.B. die Annahme, dass

$$N(x,p) = N(x/\xi) \tag{A.25}$$

gilt. Die Abhängigkeit von p taucht nun nicht mehr explizit auf, sondern steckt in der Abhängigkeit der charakteristischen Skala $\xi(p)$ von p. Mit der Annahme (A.25) gilt nun für das zweite Moment der Verteilung $N(x,p)$:

$$\frac{\int \mathrm{d}x\, x^2\, N(x,p)}{\int \mathrm{d}x\, N(x,p)} = \frac{\int \mathrm{d}x\, x^2\, N(x/\xi)}{\int \mathrm{d}x\, N(x/\xi)} = \xi^2\, \frac{\int \mathrm{d}u\, u^2\, N(u)}{\int \mathrm{d}u\, N(u)}\,, \tag{A.26}$$

wobei wir im letzten Schritt $u = x/\xi$ substituiert haben. Der Quotient der Integrale im Ergebnis ist eine von p unabhängige Konstante. Wir können daher (A.26) zur Bestimmung von $\xi(p)$ verwenden. Solange uns Konstante nicht weitere interessieren, können wir

$$\xi^2 := \frac{\int \mathrm{d}x\, x^2\, N(x,p)}{\int \mathrm{d}x\, N(x,p)} \tag{A.27}$$

definieren. Dies entspricht im Prinzip genau der integrierten Korrelationslänge (6.5), allerdings haben wir diese Definition hier mit schwächeren Annahmen als in Kapitel 6.1.2 motiviert.

Es stellt sich nun heraus, dass $\xi_\perp$ und $\xi_\parallel$ für $p \to p_c$ ($p < p_c$) gemäß asymptotischen Potenzgesetzen divergieren:

$$\xi_\perp \propto \frac{1}{(p_c - p)^{\nu_\perp}}\,, \qquad \xi_\parallel \propto \frac{1}{(p_c - p)^{\nu_\parallel}}\,. \tag{A.28}$$

Die hier auftretenden Zahlen $\nu_\perp$ und $\nu_\parallel$ sind Beispiele für sogenannte „kritische Exponenten“.

Abbildung A.5 zeigt Simulationsergebnisse für die beiden Skalen $\xi_\perp$ und $\xi_\parallel$, die mit dem ab S. 137 beschriebenen Algorithmus erzeugt wurden[4]. Man beobachtet zunächst, dass $\xi_\parallel$ schneller divergiert als $\xi_\perp$ (d.h. $\nu_\parallel > \nu_\perp$). Daher kann man p_c genauer aus $\xi_\parallel$ schätzen. Anpassung der Daten in der oberen Tafel von Abb. A.5 mit $p \geq 0.67$ an (A.28) führt auf

$$p_c = 0.7054(3)\,. \tag{A.29}$$

Dieses Ergebnis stimmt innerhalb der angegebenen Fehlergrenzen mit (A.24) überein.

[4] Dies ist zwar ein vergleichsweise einfaches Problem, dennoch sind ein paar Details zu beachten. Erstens ist der in C-Bibliotheken implementierte `random()` Zufallsgenerator nicht ausreichend, zumindest wenn man sich p_c nähert. Zweitens muss man aufpassen, dass in der Nähe von p_c keine Bereichsüberläufe bei der Berechnung der zeitlichen Länge t^2 auftreten, zumindest wenn man die Multiplikation mit ganzzahliger Arithmetik ausführt.

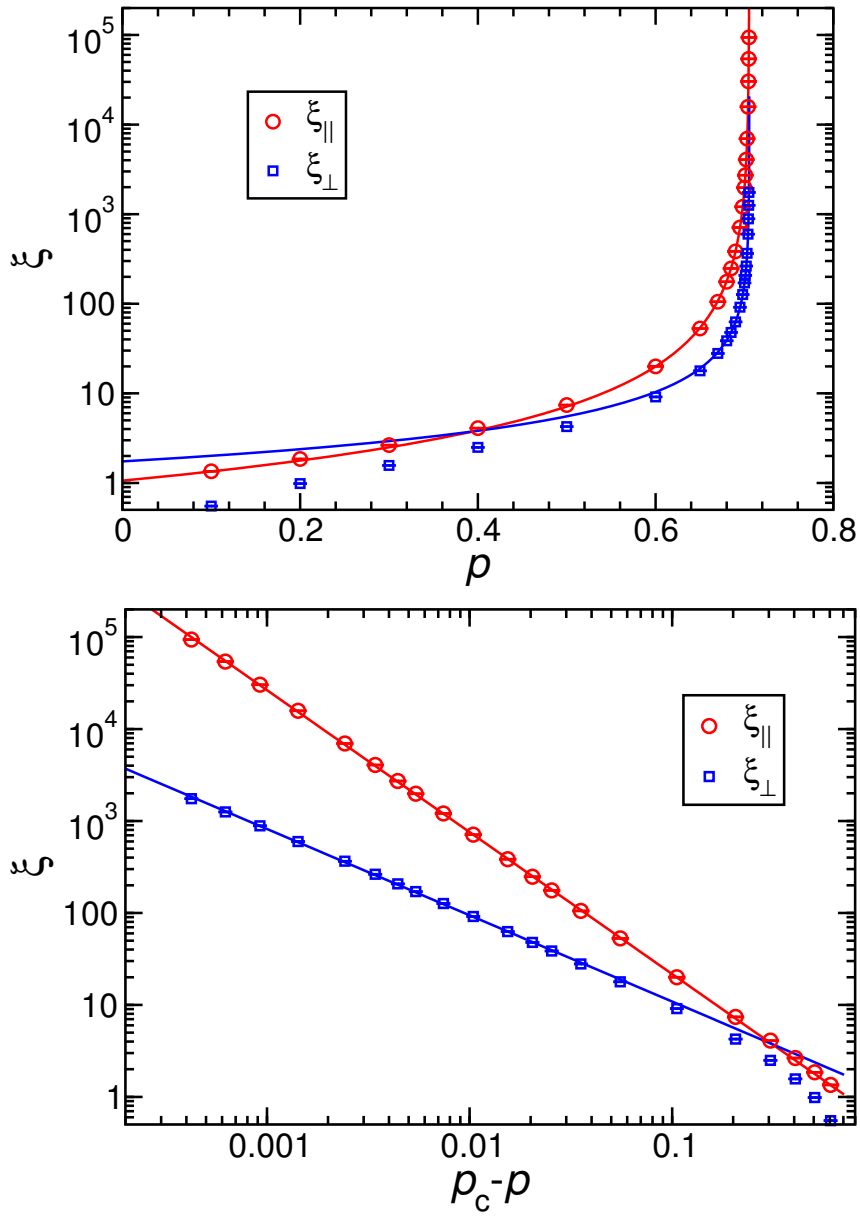


Abbildung A.5: Schätzwerte für die beiden Längenskalen $\xi_\perp$ und $\xi_\parallel$ bei gerichteter Platz-Perkolation als Funktion von p. Punkte zeigen Simulationsergebnisse, Linien eine Anpassung der Daten für $p \geq 0.67$ an die Potenzgesetze (A.28). Die obere Tafel zeigt eine logarithmische Skalierung der vertikalen Achse mit einer linearen Darstellung von p auf der horizontalen Achse. Die untere Tafel zeigt eine doppelt-logarithmische Darstellung der gleichen Daten, wobei auf der horizontalen Achse $p_c - p = 0.705423 - p$ dargestellt ist (vgl. (A.29)).

Der gleiche Fit liefert natürlich nicht nur p_c, sondern auch einen Schätzwert für $\nu_\parallel$. Allerdings ist es besser, nun zunächst einen doppelt-logarithmischen Plot von $\xi_\perp$ und $\xi_\parallel$ versus $p_c - p$ zu erstellen. Ein solcher Plot ist in der unteren Tafel von Abb. A.5 zu sehen. Gelten die asymptotischen Potenzgesetze (A.28) und ist p_c korrekt bestimmt, so sollten die Daten in einer solchen Darstellung auf einer Geraden liegen. Ferner ist eine Anpassung der Exponenten in einer solchen Darstellung stabiler. Aus den Daten mit $p \geq 0.67$ kann man nun die Exponenten zu

$$\nu_\parallel = 1.544\,, \qquad \nu_\perp = 0.938 \tag{A.30}$$

abschätzen. Wir geben an dieser Stelle zunächst keine Fehler an, da diese durch die Unsicherheit von p_c dominiert werden. Dies kann man z.B. durch Fits bei unterschiedlichen p_c abschätzen. Verwendet man $p_c = 0.7053$ und $p_c = 0.7056$ (bei diesen Werten sind bereits Abweichungen von der Geraden in der Darstellung der unteren Tafel von Abb. A.5 zu sehen), so variieren die Exponenten um bis zu $\delta\nu_\parallel = 0.1$ bzw. $\delta\nu_\perp = 0.07$ um die Schätzwerte (A.30).
Zum Vergleich seien die Ergebnisse aus der bereits erwähnten Arbeit I. Jensen, J. Phys. A: Math. Gen. **32** (1999) 5233 angegeben:

$$\nu_\parallel = 1.733847(6)\,, \qquad \nu_\perp = 1.096854(2)\,. \tag{A.31}$$

Wir zitieren hier die Werte für Verbindungsperkolation, da sie genauer sind als für Platzperkolation. Die Abweichungen zwischen unseren Schätzwerten (A.30) und (A.31) liegen etwas oberhalb unserer Fehlerschätzung (allerdings unterhalb der doppelten Fehlerschranke), so dass die Ergebnisse als konsistent betrachtet werden können. Die Literaturwerte (A.31) sind deutlich genauer als unsere Schätzwerte (A.30). Dennoch ist es auch mit den letzteren möglich, im vorliegenden $1+1$-dimensionalen Fall die Werte $\nu_\parallel = 1$, $\nu_\perp = 1/2$ auszuschließen, die eine einfache Molekularfeldtheorie unabhängig von der Dimension vorhersagt[5].
Man mag sich noch fragen, warum es überhaupt legitim ist, Ergebnisse für Platz- mit Verbindungs-Perkolation zu vergleichen. Hierhinter steckt das Konzept von „*Universalität*“: p_c hängt zwar von Details des Modells ab (wie z.B., ob wir Platz- oder Verbindungs-Perkolation betrachten oder der Geometrie des Gitters). Die kritischen Exponenten hingegen sind unabhängig von solchen Details und hängen z.B. nur von der Dimension und Symmetrie des Problems ab. Ein bestimmter Satz von Exponenten charakterisiert eine sogenannte „Universalitätsklasse“, der viele verschiedene mikroskopische Realisierungen angehören können.

[5] S. Redner, Phys. Rev. B **25** (1982) 3242.