Computational Physics II *

Ulli Wolff, Burkhard Bunk
mit
Francesco Knechtli

Institut für Physik der HU
Computational Physics

e-mail: uwolff@physik.hu-berlin.de
e-mail: knechtli@physik.hu-berlin.de

10. Juli 2003

Zusammenfassung

Die Skripten CP1 und CP2 aus früheren Semestern, die sich von Semester zu Semester allerdings geringfügig ändern (u. a. wegen software upgrades), finden sich im Web:
`http://www-com.physik.hu-berlin.de/comphys/comphys.html`
Bitte nicht unnötig drucken, das jeweils benötigte wird in aktualisierter Fassung zu Beginn der Vorlesung ausgeteilt!

*Kurs im Wahlpflichtfach *Wissenschaftliches Rechnen* SS 2003

Inhaltsverzeichnis

1 Diffusion

In diesem Abschnitt geht es um die eindimensionale Diffusion. Während in der Elektrostatik und Quantenmechanik (räumliche) Randwertaufgaben zu lösen waren, handelt es sich nun wieder um (zeitliche) Anfangswertprobleme, wie bei der Lösung der mechanischen Bewegungsgleichungen. Die räumliche Struktur stellt aber neue Anforderungen an die algorithmische Stabilität der diskreten Zeitentwicklung.

1.1 Diffusionsgleichung

Wir betrachten ein zeitabhängiges Feld in einer Raumdimension, das physikalisch z.B. eine Dichte oder Temperaturverteilung bedeuten mag, und bezeichnen es mit $\phi(x,t)$. Um die Diffusionsgleichung elementar herzuleiten, führen wir eine Diskretisierung in Raum ($x = nh$) und Zeit ($t = n\tau$) ein – für eine numerische Behandlung wird das früher oder später ohnehin nötig sein. Die Änderung des Feldes $\phi(x,t) \to \phi(x,t+\tau)$ ist nun gegeben durch "Übergänge" $x \leftrightarrow x+h$ mit "Wahrscheinlichkeit" $w(x+h/2)$ und $x \leftrightarrow x-h$ mit $w(x-h/2)$, dazu kommt eine lokale "Quelle" $q(x,t)$:

$$\begin{aligned} \phi(x,t+\tau) = \phi(x,t) &+ w(x+h/2)\,[\phi(x+h,t)-\phi(x,t)] \\ &+ w(x-h/2)\,[\phi(x-h,t)-\phi(x,t)] \\ &+ q(x,t). \end{aligned} \tag{1.1}$$

Hier erkennt man die genäherten Ableitungen

$$\phi(x,t+\tau) - \phi(x,t) = \tau \frac{\partial}{\partial t}\phi(x,t) + \mathrm{O}(\tau^2) \tag{1.2}$$

$$\phi(x \pm h,t) - \phi(x,t) = \pm h \frac{\partial}{\partial x}\phi(x \pm h/2,t) + \mathrm{O}(h^3) \tag{1.3}$$

und erhält

$$\begin{aligned} \tau\frac{\partial}{\partial t}\phi(x,t) &\approx w(x+h/2)\,h\frac{\partial}{\partial x}\phi(x+h/2,t) \\ &- w(x-h/2)\,h\frac{\partial}{\partial x}\phi(x-h/2,t) + \mathrm{O}(h^3) \\ &+ q(x,t) \\ &\approx h^2\frac{\partial}{\partial x}\left[w(x)\frac{\partial}{\partial x}\phi(x,t)\right] + q(x,t). \end{aligned} \tag{1.4}$$

So entsteht die Diffusionsgleichung

$$\frac{\partial}{\partial t}\phi(x,t) = \frac{\partial}{\partial x}\left[D(x)\frac{\partial}{\partial x}\phi(x,t)\right] + S(x,t) \tag{1.5}$$

mit (ortsabhängiger) Diffusionskonstante

$$D(x) = \frac{h^2}{\tau}w(x) \tag{1.6}$$

und Quellterm

$$S(x,t) = \frac{1}{\tau}q(x,t). \tag{1.7}$$

Für den Fall $D = \text{const}$, $S = 0$ und $x \in R$ läßt sich eine wichtige Lösung von Glg.(1.5) für alle $t > 0$ angeben:

$$G(x,t) = \frac{1}{\sigma(t)\sqrt{2\pi}}\exp\left[-\frac{x^2}{2\sigma(t)^2}\right]. \tag{1.8}$$

Dies ist eine (normierte) Gauß–Verteilung mit zeitabhängiger Breite

$$\sigma(t) = \sqrt{2Dt}. \tag{1.9}$$

Wegen

$$G(x,t) \to \delta(x) \quad \text{für} \quad t \to 0^+ \tag{1.10}$$

spielt $G(x,t)$ die Rolle einer Greensfunktion für die Diffusionsgleichnung: das Anfangswertproblem ($\phi(x,0)$ gegeben) wird allgemein gelöst durch

$$\phi(x,t) = \int_{-\infty}^{\infty} dy\, G(x-y,t)\phi(y,0) \quad \text{für} \quad t > 0. \tag{1.11}$$

Wenn nur ein endliches x-Intervall mit Randbedingungen zu betrachten ist, hilft eine räumliche Fourier–Entwicklung weiter. Bei ortsabhängigem $D(x)$ muß man auf numerische Methoden zurückgreifen.

1.2 FTCS–Diskretisierung

Eine diskretisierte Form der Diffusionsgleichung kennen wir schon: wir behalten die Konvention bei, die Diffusionskonstanten auf den Zwischengitterpunkten anzusiedeln, und schreiben Glg.(1.1) zusammen mit (1.6) und (1.7)

nun als

$$\begin{aligned}\phi(x,t+\tau) &= \phi(x,t) \\ &+ \frac{\tau}{h^2} D(x+h/2)\left[\phi(x+h,t)-\phi(x,t)\right] \\ &+ \frac{\tau}{h^2} D(x-h/2)\left[\phi(x-h,t)-\phi(x,t)\right] \\ &+ \tau S(x,t). \end{aligned} \tag{1.12}$$

Dies hat gerade die Form einer Rekursionsgleichung für die Zeitentwicklung (vgl. Euler-Methode in Kap. "Anfangswertprobleme" von CP1). Nach der Art der Ableitungsbildungen wird diese Version unter der Bezeichnung FTCS (Forward Time Centered Space) geführt.

Um den Einfluß der Diskretisierung abzuschätzen[1, Kap.7], nehmen wir wieder $D = const$ und $S = 0$ und setzen für die x-Abhängigkeit ebene Wellen an. Für ein Intervall $0 \le x \le L$ mit Randbedingungen $\phi(0,t) = \phi(L,t) = 0$ sei z.B.

$$\phi(x,t) = a(t)\sin kx \quad \text{mit} \quad k = \frac{\pi n}{L},\ n = 1,2,\ldots. \tag{1.13}$$

Ein beliebiges $\phi(x,0)$ kann aus solchen Sinuswellen überlagert werden; wegen der Linearität der Diffusionsgleichung kann ihre Zeitentwicklung einzeln untersucht werden. Die Kontinuums-Glg.(1.5) wird gelöst durch

$$a(t) = a(0)e^{-Dk^2 t}, \tag{1.14}$$

die diskrete Rekursion (1.12) lautet dagegen

$$a(t+\tau) = a(t) + \frac{\tau}{h^2} D(2\cos kh - 2)a(t) \tag{1.15}$$

$$\Rightarrow a(n\tau) = a(0)\left[1 - 4D\frac{\tau}{h^2}\sin^2(kh/2)\right]^n. \tag{1.16}$$

Es sollte also für alle in Frage kommenden k

$$1 - 4D\frac{\tau}{h^2}\sin^2(kh/2) \approx e^{-Dk^2\tau} \tag{1.17}$$

erfüllt sein. Das gilt nur für die langwelligen Anteile mit $kh \ll 1$, wo der Sinus durch sein Argument ersetzt werden kann, und mehr kann man auf einem diskreten Gitter nicht verlangen. Zur näherungsweisen Exponenzierung für diese Moden ist ausserdem ein hinreichend kleiner Zeitschritt τ erforderlich, so daß gilt

$$Dk^2\tau \ll 1. \tag{1.18}$$

Wenn jedoch

$$D\frac{1}{h^2}\,\tau > \frac{1}{2} \tag{1.19}$$

gilt, dann ist die Lösung der diskreten Gleichung für Moden mit $k \sim \pi/h$ nicht nur ungenau, sondern sogar instabil: es gibt Moden, die nach Glg.(1.16) nicht gedämpft werden, sondern (dem Betrag nach) exponentiell anwachsen! Das führt zu numerischen Problemen, selbst wenn diese Moden in der physikalischen Lösung gar nicht angeregt sind. Zumindest im "Rundungsfehlerrauschen" kommen diese Moden vor und "übernehmen" die Evolution in endlicher Zeit.

1.3 Nichtlineare Diffusion und Selbstorganisation

In Zusammenhang mit Fragen der Selbstorganisation (Strukturentstehung) und der Funktion biologischer Uhren hat die Gruppe um I. Prigogine (bio)chemische Reaktionszyklen untersucht. Dabei ist man auf folgendes Modell gestoßen: zwei Substanzen X und Y reagieren miteinander gemäß

$$\begin{aligned} A &\rightarrow X \\ B + X &\rightarrow Y + D \\ 2X + Y &\rightarrow 3X \\ X &\rightarrow E, \end{aligned} \tag{1.20}$$

wobei A und B aus einem Reservoir konstanter Dichte stammen sollen, D und E sind Abfallprodukte ohne weiteren Einfluß. Wenn man für X und Y auch noch berücksichtigt, daß sich ihre Konzentration durch Diffusion räumlich verteilt, dann erhält man für die Dichten $X(x,t)$ und $Y(x,t)$ das folgende System von Differentialgleichungen[1, S.189ff]:

$$\begin{aligned} \frac{\partial}{\partial t}X(x,t) &= D_X\nabla^2 X + k_a A - k_b BX + k_c X^2 Y - k_d X \\ \frac{\partial}{\partial t}Y(x,t) &= D_Y\nabla^2 Y + k_b BX - k_c X^2 Y. \end{aligned} \tag{1.21}$$

Die nichtlineare Kopplung, die das System dynamisch so interessant macht, kommt dadurch zustande, daß die lokale Rate einer chemischen Reaktion dem *Produkt* der Dichten der Reaktionspartner proportional ist.

Hier wollen wir nur den Fall einer Raumdimension betrachten, $0 \le x \le L$. An den Rändern sollen diesmal die Neumann–Bedingungen

$$\frac{\partial}{\partial x} \left\{ \begin{matrix} X \\ Y \end{matrix} \right\} (x,t) = 0 \quad \text{für} \quad x = 0, L \tag{1.22}$$

gelten, die gewährleisten, daß kein Material durch die "Wand" fließt.

Durch geeignete Skalierungen kann man die Systemgröße $L = 1$ setzen und $k_i = 1$ erreichen:

$$\begin{aligned} \frac{\partial}{\partial t} X(x,t) &= D_X \nabla^2 X + A - (B+1)X + X^2 Y \\ \frac{\partial}{\partial t} Y(x,t) &= D_Y \nabla^2 Y + BX - X^2 Y, \end{aligned} \tag{1.23}$$

so daß sich die Vielzahl von Parametern reduziert auf D_X, D_Y, A und B.

Bevor man sich an die numerische Lösung macht, zunächst noch ein paar analytische Vorbetrachtungen. Die Gleichungen (1.23) haben eine räumlich konstante, statische Lösung

$$X_0 = A \qquad Y_0 = B/A. \tag{1.24}$$

Zur Analyse ihrer Stabilität setzt man $X = X_0 + X'$, $Y = Y_0 + Y'$ und linearisiert:

$$\begin{aligned} \frac{\partial}{\partial t} X'(x,t) &= D_X \nabla^2 X' + (B-1)X' + A^2 Y' \\ \frac{\partial}{\partial t} Y'(x,t) &= D_Y \nabla^2 Y' - BX' - A^2 Y'. \end{aligned} \tag{1.25}$$

Die Fluktuationen werden als Eigenmoden der Form

$$\left\{ \begin{matrix} X' \\ Y' \end{matrix} \right\} (x,t) = \left\{ \begin{matrix} \xi \\ \eta \end{matrix} \right\} e^{\omega t} \cos kx \quad \text{mit} \quad k = \frac{\pi n}{L},\ n = 0, 1, 2, \ldots \tag{1.26}$$

angesetzt, ein Mode mit $\mathrm{Re}\omega > 0$ ist instabil. Einsetzen liefert das Gleichungssystem

$$\begin{aligned} \omega\, \xi &= (B - 1 - D_X k^2)\xi + A^2 \eta \\ \omega\, \eta &= -B\xi - (A^2 + D_Y k^2)\eta. \end{aligned} \tag{1.27}$$

Mit den Abkürzungen

$$\begin{aligned} \alpha &= B - 1 - D_X k^2 \\ \beta &= A^2 + D_Y k^2 \end{aligned}$$

ist dieses nichttrivial lösbar, wenn die quadratische Gleichung

$$\omega^2 - (\alpha - \beta)\omega + A^2 B - \alpha\beta = 0 \tag{1.28}$$

erfüllt ist. Ihre Lösungen lauten

$$\omega_\pm = \frac{1}{2}\left\{\alpha - \beta \pm [(\alpha+\beta)^2 - 4A^2 B]^{1/2}\right\}. \tag{1.29}$$

Um das Einsetzen von Instabilitäten systematisch zu diskutieren, nehmen wir D_X, D_Y und A jeweils fest an und variieren B. Bei $B = 0$ setzen wir $\omega_+ = \alpha = -1 - D_X k^2 < 0$ und $\omega_- = -\beta = -A^2 - D_Y k^2 < 0$. Mit zunehmendem B kann nun der Übergang zur Instabilität auf zwei verschiedene Weisen auftreten:

(a) Von zwei rellen Lösungen wird die grössere positiv. Nach Glg.(1.28) ist der Grenzfall dazu erreicht, wenn $\alpha\beta = A^2 B$, d.h. für Wellenzahl k bei

$$B = (1 + D_X k^2)(1 + \frac{A^2}{D_Y k^2}). \tag{1.30}$$

Die rechte Seite hat ein Minimum bei

$$k_c = \left[\frac{A}{\sqrt{D_X D_Y}}\right]^{1/2}, \tag{1.31}$$

mit dieser Wellenzahl (bzw. dem nächstliegenden $k = \pi n/L$) tritt der erste instabile Mode auf, und zwar bei

$$B = B_c = \left[1 + A\sqrt{\frac{D_X}{D_Y}}\right]^2. \tag{1.32}$$

(b) Bei einem Paar konjugiert komplexer Lösungen wird gerade $\mathrm{Re}\omega_\pm = 0$, d.h. $\alpha = \beta$ oder

$$B = 1 + A^2 + (D_X + D_Y)k^2. \tag{1.33}$$

Die Diskriminante $\mathcal{D}$ in Glg.(1.29) kann man schreiben als

$$\mathcal{D} = (\alpha+\beta)^2 - 4A^2 B = (A^2 + B - \Delta_k)^2 - 4A^2 B \tag{1.34}$$

mit

$$\Delta_k = 1 - (D_Y - D_X)k^2 \tag{1.35}$$

$\mathcal{D} < 0$ geht nur mit $\Delta_k > 0$. Weiter kann man umformen

$$\mathcal{D} = [B - (A - \sqrt{\Delta_k})^2][B - (A + \sqrt{\Delta_k})^2]. \tag{1.36}$$

Da der erste Faktor größer ist als der zweite, muss letzterer negativ und ersterer positiv sein. Die Bedingung für komplexe Wurzeln lautet also

$$\Delta_k > 0 \quad \text{und} \quad \left(A - \sqrt{\Delta_k}\right)^2 < B < \left(A + \sqrt{\Delta_k}\right)^2. \tag{1.37}$$

Das ist offenbar immer erfüllt für die kleinste Lösung von (1.33) die sich für $k = 0$ ergibt, $B = B_0 = 1 + A^2$.

Das Auftreten von Instabilität in den linearisierten Gleichungen (1.25) signalisiert, daß das volle (nichtlineare) System (1.23) langfristig nicht mehr in den Gleichgewichtszustand (1.24) übergeht, sondern ein Verhalten ähnlich den instabilen Moden annimmt. Wenn $B_c < B_0$, dann tritt bei $B \simeq B_c$ ein räumlich oszillierendes Muster auf, dessen Wellenzahl durch k_c aus Glg.(1.31) bestimmt ist. Wie ein Vergleich von B_c (1.32) und $B_0 = 1 + A^2$ zeigt, ist das genau dann der Fall, wenn

$$D_Y > D_X \quad \text{und} \quad A > \frac{2}{\sqrt{D_Y/D_X} - \sqrt{D_X/D_Y}}. \tag{1.38}$$

Im anderen Fall $B_0 < B_c$ entsteht ein Langzeit–Verhalten, das sogar räumlich *und* zeitlich oszilliert. Die Komplexität des räumlichen Musters hängt davon ab, für wieviele Wellenzahlen $k = \pi n/L$ die Dichte B die Instabilitätsschwelle überschreitet. Ein schönes Beispiel raum–zeitlicher Oszillation, das durch numerische Integration des Systems (1.23) mit Hilfe des einfachen FTCS–Verfahrens gewonnen wurde, ist in Fig.1 dargestellt. Eine Anleitung zur Implementierung wird in Form einer Übungsaufgabe ausgegeben.

1.4 Implizite Verfahren

Das Instabilitätsproblem der FTCS-Iteration kann man durch einen merkwürdigen Kunstgriff lösen: man nimmt sich vor, die räumlichen Differenzenterme in Glg.(1.12) zur "neuen" Zeit $t + \tau$ anstelle von t zu nehmen. Formal

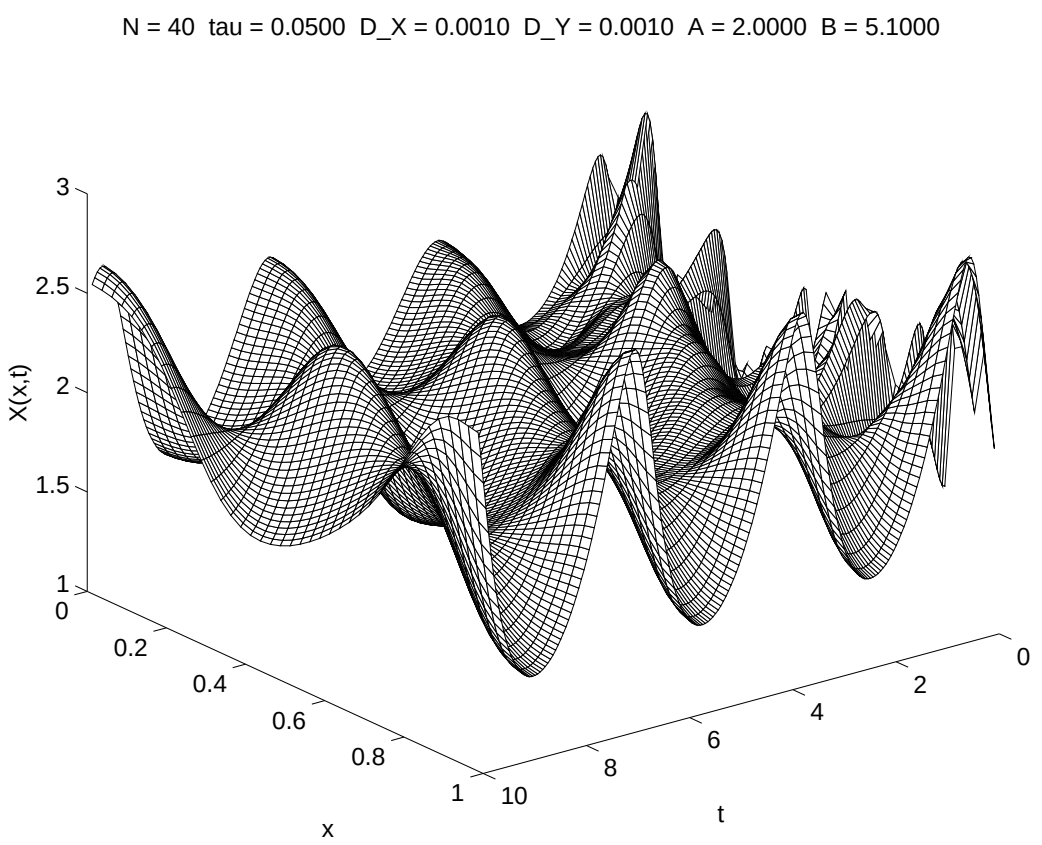

Abbildung 1: Oszillierende Lösung der Reaktionsgleichung

bedeutet das

$$\begin{aligned}\phi(x,t+\tau) &= \phi(x,t) \\ &+ \frac{\tau}{h^2} D(x+h/2)\left[\phi(x+h,t+\tau)-\phi(x,t+\tau)\right] \\ &+ \frac{\tau}{h^2} D(x-h/2)\left[\phi(x-h,t+\tau)-\phi(x,t+\tau)\right] \\ &+ \tau S(x,t). \end{aligned} \tag{1.39}$$

Damit wird bei der Stabilitätsanalyse in Glg.(1.16) der störende Term $\sim \sin^2(kh/2)$ auf die linke Seite gebracht:

$$a(t+\tau) \;+\; \frac{\tau}{h^2} 4D \sin^2(kh/2) a(t+\tau) = a(t) \tag{1.40}$$

$$\Rightarrow a(n\tau) \;=\; a(0)\left[1+4D\frac{\tau}{h^2}\sin^2(kh/2)\right]^{-n} \tag{1.41}$$

und alle Moden werden nun stabil iteriert. Natürlich hat dieses "Wunder" seinen Preis: die Differenzengleichung (1.39) ist nicht mehr nach $\phi(x,t+\tau)$ aufgelöst, sondern definiert die Iteration nur noch implizit (daher der Name). Man sollte also eher

$$\begin{aligned}\phi(x,t+\tau) \;&-\; \frac{\tau}{h^2}D(x+h/2)\left[\phi(x+h,t+\tau)-\phi(x,t+\tau)\right]\\ &-\; \frac{\tau}{h^2}D(x-h/2)\left[\phi(x-h,t+\tau)-\phi(x,t+\tau)\right]\\ &=\; \phi(x,t)+\tau S(x,t).\end{aligned} \tag{1.42}$$

schreiben und dies als lineares Gleichungssystem für $\phi(x,t+\tau)$ bei gegebenem $\phi(x,t)$ auffassen, dabei ist x der Vektorindex.

Eine andere Variante besteht darin, den Laplace–Term "zur Hälfte" auf die linke Seite zu bringen (Crank–Nic[h]olson–Verfahren):

$$\begin{aligned}\phi(x,t+\tau) \;&-\; \frac{\tau}{2h^2}D(x+h/2)\left[\phi(x+h,t+\tau)-\phi(x,t+\tau)\right]\\ &-\; \frac{\tau}{2h^2}D(x-h/2)\left[\phi(x-h,t+\tau)-\phi(x,t+\tau)\right]\\ &=\; \phi(x,t)\\ &+\; \frac{\tau}{2h^2}D(x+h/2)\left[\phi(x+h,t)-\phi(x,t)\right]\\ &+\; \frac{\tau}{2h^2}D(x-h/2)\left[\phi(x-h,t)-\phi(x,t)\right]\\ &+\; \tau S(x,t).\end{aligned} \tag{1.43}$$

Hiermit wird die Zeitableitung mit einer Genauigkeit der Ordnung τ^2 approximiert, denn abgesehen vom Quellterm $S(x,t)$ ist die Diskretisierung symmetrisch bezüglich $t+\tau/2$. Die Stabilitätsanalyse liefert diesmal eine Mischform von Glg.(1.16) und (1.41):

$$a(n\tau) = a(0)\left[\frac{1-2D\frac{\tau}{h^2}\sin^2(kh/2)}{1+2D\frac{\tau}{h^2}\sin^2(kh/2)}\right]^n \tag{1.44}$$

und zeigt wieder Stabilität für alle Moden k und beliebige τ.

Es ist anzumerken, daß das FTCS–Verfahren leicht auf Diffusion in mehreren Raumdimensionen auszudehnen ist, bei den impliziten Methoden sind dazu noch weitere Tricks nötig.

1.5 Lösung eines tridiagonalen Gleichungssystems

Die impliziten Rekursionsgleichungen (1.42) und (1.43) sind von der tridiagonalen Form

$$\begin{pmatrix} \beta_1 & \gamma_1 & & & & \\ \alpha_2 & \beta_2 & \gamma_2 & & & \\ & \alpha_3 & \beta_3 & \gamma_3 & & \\ & & \dots & \dots & \dots & \\ & & & \alpha_{N-1} & \beta_{N-1} & \gamma_{N-1} \\ & & & & \alpha_N & \beta_N \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \dots \\ \phi_{N-1} \\ \phi_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_{N-1} \\ b_N \end{pmatrix}, \qquad (1.45)$$

wobei die Unbekannten $\phi(x, t+\tau)$ hier kurz mit ϕ_i bezeichnet sind und die bekannten Größen auf der rechten Seite mit b_i.

In einem ersten Durchlauf (Gauß–Elimination) werden die unter der Diagonale stehenden α_i beseitigt. Dazu wird die mit α_2/β_1 multiplizierte erste Gleichung von der zweiten subtrahiert, d.h.

$$\begin{aligned} \alpha_2 &\to 0 \\ \beta_2 &\to \beta_2' = \beta_2 - \frac{\alpha_2}{\beta_1}\gamma_1 \\ b_2 &\to b_2' = b_2 - \frac{\alpha_2}{\beta_1} b_1, \end{aligned} \qquad (1.46)$$

dann analog (unter Benutzung von β_2' und b_2'!) die zweite Gleichung von der Dritten subtrahiert, so daß $\alpha_3 \to 0$ usw. Es entsteht

$$\begin{pmatrix} \beta_1' & \gamma_1 & & & & \\ & \beta_2' & \gamma_2 & & & \\ & & \beta_3' & \gamma_3 & & \\ & & & \dots & \dots & \\ & & & & \beta_{N-1}' & \gamma_{N-1} \\ & & & & & \beta_N' \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \dots \\ \phi_{N-1} \\ \phi_N \end{pmatrix} = \begin{pmatrix} b_1' \\ b_2' \\ b_3' \\ \dots \\ b_{N-1}' \\ b_N' \end{pmatrix}. \qquad (1.47)$$

Dieses System wird nun von unten nach oben gelöst:

$$\begin{aligned} \phi_N &= b_N'/\beta_N' \\ \phi_i &= (b_i' - \gamma_i \phi_{i+1})/\beta_i', \quad i = N-1, N-2, \dots \end{aligned} \qquad (1.48)$$

(Rückwärts–Substitution). Diese Prozeduren sind wegen ihres stark rekursiven Charakters tatsächlich leichter zu programmieren als mathematisch aufzuschreiben.

Auf diese Weise verlangen die Iterationen der impliziten Verfahren zwei zusätzliche, einfache Schleifen über die Gitterpunkte, der Aufwand für eine Iteration bleibt $\sim N$, wenn auch mit einem größeren Vorfaktor. Das ist der Preis der Stabilität.

2 Chaos in dynamischen Systemen

Die Existenz des Phänomens des Chaos in dynamischen Systemen ist schon lange bekannt, z. B. in Werken von Poincaré zur klassischen Mechanik. Erst in jüngerer Vergangenheit wurde es allerdings verstärkt untersucht. Diese neue Popularität hängt sicher auch zusammen mit der stark erweiterten Möglichkeit komplexere nichtlineare Systeme durch numerische Simulation zu untersuchen. Stark vereinfacht geht es dabei um Verhalten wie sehr irreguläre klassische Bahnen und extrem (exponentiell) empfindliche Abhängigkeit des Verhaltens nach endlicher Zeit von den Anfangsbedingungen. Eine genauere Definition wird erst später möglich sein. Es ist wichtig zu verstehen, daß es sich bei chaotischen Systemen nicht um hochspeziell konstruierte Ausnahmen handelt, sondern bei mehreren Freiheitsgraden um den Normalfall. Umgekehrt zeigen nur besonders einfache System *kein* Chaos. Dazu gehören gerade die geschlossen lösbaren wie z. B. der harmonische Oszillator, die Lehrbücher aber nicht (streng) die Natur dominieren.

Obwohl wir hier im Bereich der klassischen Physik bleiben, benötigen wir eine ganze Reihe neuer Begriffe, die wir einführen gemischt mit Beispielen und numerischen Experimenten. Referenzen zu diesem Abschnitt sind [2] und [3].

2.1 Dynamische Systeme, Phasenraum

Wir wollen hier dynamische Systeme betrachten, deren zeitliche Evolution durch ein differentielles Gesetz[1]

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{F}[\vec{x}(t)] \tag{2.1}$$

beschrieben wird mit Phasenraum Koordinaten $\vec{x} = (x_1, x_2, \ldots, x_N)$. Ein spezieller Fall ist die Hamiltonsche Mechanik mit der Hamilton Funktion $H(p_i, q_j)$

$$\vec{x} = (q_1, \ldots, q_n, p_1, \ldots, p_n), \quad N = 2n \tag{2.2}$$

$$\vec{F} = (\partial H/\partial p_1, \ldots, \partial H/\partial p_n, -\partial H/\partial q_1, \ldots, -\partial H/\partial q_n) \tag{2.3}$$

Klarerweise besitzt $\vec{F}$ hier aber spezielle Integrabilitätseigenschaften und (2.1) ist allgemeiner.

[1] Man beachte die Abwesenheit einer expliziten t-Abhängigkeit in $\vec{F}$; eine solche DGL heißt autonom

Jeder Punkt $\vec{x}$ im Phasenraum kann als Anfangsbedingung verwendet werden. Integriert man dann (2.1), so geht von ihm eine Trajektorie oder Bahn aus. Denkt man sich dies auf ein zusammenhängendes Gebiet von Punkten angewandt, so wird ein Fluß im Phasenraum erzeugt, bei dem sich das anfängliche Gebiet als Funktion der Zeit bewegt und deformiert ähnlich den Molekülgruppen einer Flüssigkeit. Eine wichtige Eigenschaft der Hamiltonschen Systeme ist das Liouville Theorem[2], das besagt, daß bei dem beschriebenen Fluß das Volumen im Phasenraum konstant bleibt. Ein Fall, der hierüber hinausgeht, ist offenbar z. B. dann gegeben, wenn $\vec{F}$ Reibungsterme enthält, die dafür sorgen, daß das System für jede Anfangsbedingung an einem bestimmten Punkt zur Ruhe kommt. Man kann etwa an ein Pendel mit Reibung denken.

2.2 Ein erstes Beispiel

Ein einfaches Beispiel für ein chaotisches System wird in [2] durch folgende Bewegungsgleichung in für das System in natürlichen Einheiten beschrieben,

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = g\sin(t) - \nu\frac{\mathrm{d}y}{\mathrm{d}t} - y^3 + y. \tag{2.4}$$

Der Term $\propto \nu$ beschreibt eine die Bewegung der reellen Koordinate y dämpfende Reibungskraft. Auf der rechten Seite sieht man eine antreibende periodische Kraft der Stärke g. Die anharmonische Kraft entspricht einem Potential $(y^2-1)^2/4$; es beschreibt eine für große $|y|>1$ rücktreibende Kraft mit zwei Gleichgewichtslagen bei $y=\pm 1$. Wählt man nun Parameter derart, daß die Antriebskraft von Zeit zu Zeit die gedämpften Oszillationen vom einen Minimum zum anderen kickt, dann erhält man typische Bilder wie in Fig.2. Hier wurde mit der MATLAB Routine `ode23` und `ode45` integriert mit den einfachen Programmen

```
% file chaos_ex.m
% Beispiel fuer Chaos: Ott S. 2/3
%
global g nu
g=0.32; nu=0.3;
```

[2] siehe z. B. [4] oder jedes andere Buch über Klassische Mechanik

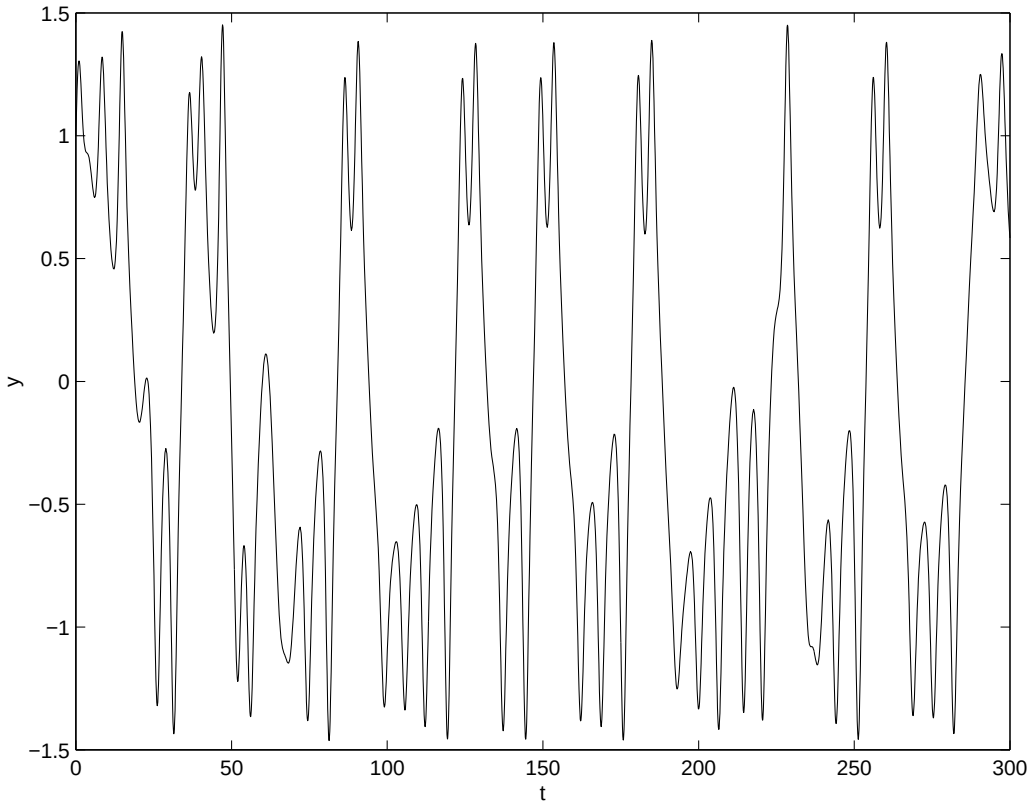

Abbildung 2: Gl. (2.4) mit $g = 0.32, \nu = 0.3$

```
% Anfangswerte:
y0=[1 ; 0.5];
t0=0;

% Integration:
tspan=[t0 300];
[t,y] = ode23('F_ex',tspan,y0);

plot(t,y(:,1))
xlabel('t'); ylabel('y')
```

mit dem Funktionsunterprogramm

```
% file F_ex.m
% Kraft fuer das 1. Beispiel Chaos
%
function yp = F_ex(t,y)
global g nu
c = g*sin(t) - nu*y(2) -y(1)*(y(1)*y(1)-1.0);
yp = [y(2) ; c];
```

Es ist sicher empfehlenswert, mit diesem oder einem ähnlichen Programm zu experimentieren, Parameter zu ändern etc.

In [2] ist ein physikalisches System beschrieben, das näherungsweise durch die obige Dynamik beschrieben werden soll. Man hat ein biegbares Stahlband über zwei Magneten. In den Ruhelagen zeigt das Band auf einen der Magneten. Die äußere Kraft entspricht einem periodischen Rütteln des Bandes. Tatsächlich beobachtet man qualitativ ähnliche Muster wie Fig.2.

Es soll noch angegeben werden, wie das System in die Form (2.1) gebracht werden kann. Mit

$$\vec{F} = (x_2\,,\; g\sin(x_3) - \nu x_2 - x_1^3 + x_1\,,\; 1) \tag{2.5}$$

erhalten wir die gleiche Dynamik wie vorher mit den Identifizierungen

$$x_1 = y \tag{2.6}$$

$$x_2 = \frac{\mathrm{d}y}{\mathrm{d}t} \tag{2.7}$$

$$x_3 = t \tag{2.8}$$

In die Form (2.1) gebracht liegen also 3 Freiheitsgrade vor, $\vec{x}$ ist ein 3-komponentiger Vektor. Allgemeine Untersuchungen [2] haben gezeigt, daß dies die Minimalzahl ist, damit Chaos eintreten kann (aber nicht muß).

2.3 Diskrete Dynamische Abbildungen

Im Zusammenhang mit Chaos ist es auch von Interesse, Systeme mit diskreter Zeit zu untersuchen. Statt Phasenraumtrajektorien $\vec{x}(t)$ hat man dann diskrete Folgen[3] $\vec{x}^{(0)}, \vec{x}^{(1)}, \ldots$, und das dynamische Gesetz ist durch eine Abbildung M gegeben, die den minimalen Zeitentwicklungsschritt liefert,

$$\vec{x}^{(n+1)} = M(\vec{x}^{(n)}), \tag{2.9}$$

[3]man beachte die von [2] abweichende Indexkonvention

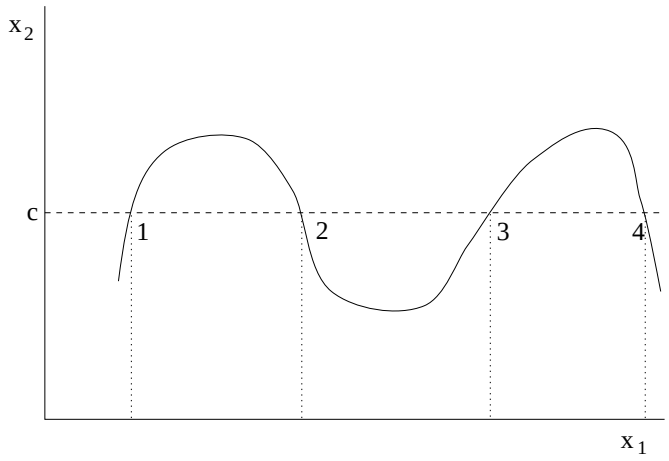

Abbildung 3: Beispiel Poincaré Schnitt von $N = 2$ auf 1 Dimension

und somit Evolution um eine Zeiteinheit beschreibt an Stelle eines infinitesimalen Schrittes in (2.1). Mit dem Anfangswert $\vec{x}^{(0)}$ kann man also durch wiederholte M-Anwendung alle künftigen Werte erzeugen. Eine Möglichkeit ist natürlich, ein System mit kontinuierlicher Zeit nur zu Zeitpunkten $t = n\Delta$ mit Zeitschritt Δ zu betrachten. Dann ist M durch Integration von (2.1) von t bis $t + \Delta$ zu konstruieren.

Eine andere Möglichkeit ist der Poincaré Schnitt. Hierbei wird eine Bahn im N-dimensionalen Phasenraum daraufhin analysiert, wann sie eine $N-1$-dimensionale Hyperebene, z. B. $x_N = c$, schneidet. Dies liefert in der Ebene diskret aufeinanderfolgende Punkte $(x_1^{(n)}, x_2^{(n)}, \ldots, x_{N-1}^{(n)}, c)$, die als ein $N-1$ dimensionales System mit diskreter (aber i. a. nicht äquidistanter) Zeit betrachtet werden können. Ein Schnittpunkt bestimmt den zeitlich nächsten, und in beiden hier betrachteten Fällen ist die Abbildung invertierbar durch Integration rückwärts in der Zeit. Das einfachste Beispiel eines Poincaré Schnitts ist in Abb. 3 zu sehen. Wir werden aber auch diskrete Abbildungen für sich definieren und im Sinne einer Dynamik untersuchen. Damit im diskreten Fall Chaos auftreten kann, benötigen wir zwei Dimensionen für invertierbare Abbildungen, was konsistent ist für aus kontinuierlichen Bahnen durch Poincaré Schnitt abgeleitete. Bei nichtinvertierbaren dynamischen Abbildungen gibt es sogar eindimensionales Chaos.

Ein einfaches stückweise lineares Beispiel ist gegeben durch die Drei-

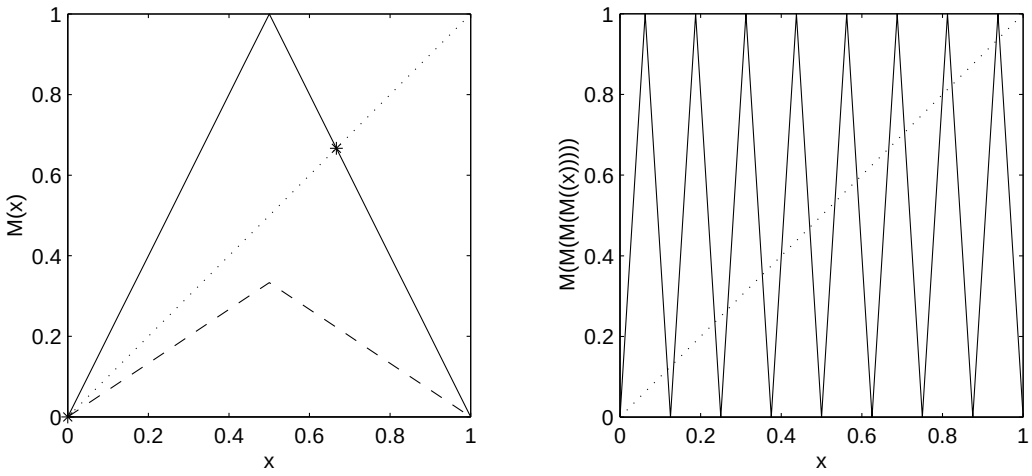

Abbildung 4: Dreiecksabbildung

ecksabbildung

$$M(x) = r\left(1 - 2\left|\frac{1}{2} - x\right|\right), \tag{2.10}$$

die das Intervall $[0,1]$ in $[0,r]$ abbildet. Sie besitzt für $r < 1/2$ einen stabilen Fixpunkt $x^* = 0$, in den man durch Iterieren von jedem Start aus reinläuft. Für $r > 1/2$ liegen zwei instabile Fixpunkte vor, von denen jede Iteration bei endlicher Genauigkeit weggetrieben wird. Nur der letztere Bereich ist von Interesse, und $r = 1$ ist hier repräsentativ und soll nur noch betrachtet werden. Die Fixpunkte liegen dann bei 0 und $2/3$ wie in Abb. 4 gezeigt. Die Abbildung hat fast überall eine Steigung mit Betrag 2. Kleine Differenzen beim Startwert werden also exponentiell in der Zeit vergrößert. Man spricht von einer streckenden Abbildung. Dazu gehört dann eine Rückfaltung (stretching and folding) um im Intervall $[0,1]$ zu bleiben, die auch die Nichtinvertierbarkeit mit sich bringt. Alle diese Eigenschaften sind typisch für chaotische Abbil-

dungen. Insbesondere die Streckung führt offenbar direkt zu empfindlicher Abhängigkeit von Anfangsbedingungen. Im rechten Teil von Abb. 4 ist die iterierte Abbildung M^4 gezeigt, die das Intervall schon gut verwirbelt.

Eine wichtige Frage bei der Analyse von Chaos ist die nach periodischen Bahnen. Eine periodische Bahn der Periode p besteht aus Punkten $x^{(1)}, x^{(2)}, \ldots, x^{(p)}$ mit $x^{(i)} \neq x^{(j)}, x^{(i+1)} = M(x^{(i)})$, die bei p-facher Anwendung der Abbildung invariant sind, $M^p(x^{(i)}) = x^{(i)}$. Speziell sind also Fixpunkte periodische Bahnen mit $p = 1$ und Punkte auf der periodischen Bahn Fixpunkte von M^p. Im rechten Teil der Abb. 4 ist die Abbildung M^4 gezeigt, und die Schnittpunkte mit der Geraden sind Fixpunkte. Es gibt 2^p solche. Teilweise gehören diese aber zu Bahnen mit niedrigerer Periode, die sich dann wiederholen. Dies kann natürlich für Primzahlen p nicht passieren. Wenn man sich die Abbildung für beliebige p vorstellt, dann sieht man leicht ein, daß es zu jedem x beliebig nahe Punkte gibt, die zu einer periodischen Bahn gehören. Die periodischen Bahnen sind also dicht im Phasenraum. Andererseits sind deren Punkte nach der geometrischen Konstruktion der Abb. 4 abzählbar und damit vom Maß Null im Phasenraum $[0, 1]$. Damit sind periodische Bahnen untypisch, die Bahn zu einem beliebig gewählten Startpunkt ist mit Wahrscheinlichkeit Eins nicht periodisch.

2.4 Charakterisierung chaotischer Dynamik

Wir wollen nun die Abhängigkeit von Anfangsbedingungen quantitativ beschreiben. Für engbenachbarte Startpunkte extrahieren wir exponentielles Auseinanderstreben der Form

$$|M^N(x) - M^N(x+\epsilon)| = \epsilon \exp(\lambda(x) N) \tag{2.11}$$

formal durch die Definition des Liapunov Exponenten

$$\lambda(x) = \lim_{N\to\infty} \lim_{\epsilon\to 0} \frac{1}{N} \log \left| \frac{M^N(x) - M^N(x+\epsilon)}{\epsilon} \right| . \tag{2.12}$$

Für die Dreiecksabbildung haben wir offenbar $\lambda = \log(2)$ unabhängig von x bzw. $\lambda = \log(2r)$ im allgemeinen Fall. Der chaotische Bereich $r > 1/2$ wird also durch positiven Lyapunov Exponenten charakterisiert, und bei $r = r_c = 1/2$ haben wir einen kritischen Punkt, in dessen Nähe $\lambda \propto (r - r_c)$ gilt.

Eine andere interessante Frage ist, welche Bereiche des Intervalls beim Iterieren der Abbildung überstrichen werden. Die Antwort gibt die invariante

Dichte oder das invariante Maß

$$\rho(x) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N} \delta[x - M^n(x^{(0)})]. \tag{2.13}$$

Hier wird mit der Notation die Unabhängigkeit des Grenzwertes von $x^{(0)}$ vorausgesetzt[4]. Solche Systeme heißen ergodisch. Wie in der Thermodynamik kann man dann Zeit- durch Koordinatenmittelwerte ersetzten, da für lange Zeiten unabhängig von Anfangsbedingungen der ganze Phasenraum überdeckt wird.

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} f(x^{(n)}) = \int_0^1 \mathrm{d}x \rho(x) f(x) \tag{2.14}$$

Wie in der statistischen Mechanik wollen wir nun statt von einem bestimmten Punkt $x^{(0)}$ aus von einer statistischen Verteilung von Anfangswerten $\rho^{(0)}(x)$ ausgehend ein Ensemble von Zeitentwicklungen betrachten, gegeben durch (Frobenius-Perron Gleichung)

$$\rho^{(n+1)}(x) = \int_0^1 \mathrm{d}y \rho^{(n)}(y) \delta[x - M(y)]. \tag{2.15}$$

Das invariante Maß muß Fixpunkt dieser Gleichung sein. Für die Dreiecksabbildung können wir explizit lösen[5]

$$\begin{aligned} \rho^{(n+1)}(x) &= \frac{1}{2}[\rho^{(n)}(x/2) + \rho^{(n)}(1 - x/2)] \qquad (2.16) \\ &= \frac{1}{2^{n+1}} \sum_{j=1}^{2^n} \left[\rho^{(0)} \left(\frac{j-1}{2^n} + \frac{x}{2^{n+1}} \right) + \rho^{(0)} \left(\frac{j}{2^n} - \frac{x}{2^{n+1}} \right) \right]. \end{aligned}$$

Für völlig beliebige glatte $\rho^{(0)}$ geht dies für große n in ein Integral über, das den Fixpunkt darstellt, und wir haben

$$\rho(x) = \lim_{n\to\infty} \rho^{(n)}(x) = \int_0^1 \mathrm{d}x \rho^{(0)}(x) = 1, \tag{2.17}$$

d. h. die invariante Dichte ist konstant und das System ergodisch. Daneben gibt es noch Fixpunkte vom Typ $\delta(x - x^*)$ mit einem (instabilen) Fixpunkt x^*. Diese sind jedoch unphysikalisch und spielen, wenn man die Frobenius Perron Gleichung numerisch (mit Rundungsfehlern) löst, keine Rolle [3].

[4] bis auf Ausnahmen vom Maß Null

[5] Vorzeichenfehler in [3]

2.5 Logistische Abbildung

Quadratische Abbildungen besitzen universelle Eigenschaften, und als typischer Stellverterter wird die logistische Abbildung (LA)

$$M(x) = rx(1-x), \quad x \in [0,1], \quad 0 < r \leq 4 \tag{2.18}$$

im Detail betrachtet. Diese Funktion erreicht ihr Maximum $r/4$ für $x = 1/2$. Die LA kann plausibel gemacht werden z. B. als Modell für die generationsweise Evolution einer Tierpopulation. Die Größe r ist dann die Vermehrungsrate pro Individuum, x die Population. Der Faktor $1-x$, ohne den rein exponentielles Wachstum vorläge, beschreibt den Rückgang der Vermehrung durch endliche Ressourcen wie Nahrung.

Für $r > 1$ ist der Fixpunkt $x = 0$ instabil, und nur dieser Bereich ist von Interesse. Für $r = 4$ kann man die LA durch eine trigonometrische Transformation $x = \sin^2(\alpha)$ auf die Dreiecksabbildung zurückführen. Bei diesem Wert liegt also Chaos vor, und wir können invariante Dichte und Liapunov Exponent exakt angeben. Für $1 < r < 3$ gibt es Fixpunkte $x = 0$ und $x = 1 - 1/r$. Durch Bilden der Ableitung sieht man, daß der erste instabil und der zweite stabil ist in diesem Bereich. Es zeigt sich, daß jeder Anfangswert in diesen Fixpunkt hineinläuft. Sein Anziehungsbereich (basin of attraction) ist also das ganze Intervall $[0,1]$. Es stellt sich nun die Frage wie, als Funktion von r zwischen 3 und 4, der Übergang zwischen diesem langweiligen Verhalten und vollem Chaos mit unendlich vielen instabilen periodischen Bahnen vonstatten geht. Wir wollen dies zunächst direkt numerisch untersuchen. Der folgende MATLAB code sollte inzwischen leicht zu analysieren sein:

```
% file logmapex1.m
% Experimente mit der logistischen Abbildung
global r
clf;
r1=2.8;r2=3.56;           % Grenzen in r
%r1=3.5;r2=4.0;             % Grenzen in r
r=[r1:(r2-r1)/100:r2];    % alle r-Werte
%r=[3.832 3.9 4]
rvals=size(r,2);
warmup=200;               % Anwendungen der Abb. vor Plot
points=1000;                % # x-Werte pro r-Wert
x0=0.12345;                 % Startwert
```

```
x=zeros(points,rvals);    % vordefinieren wg. Geschwindigkeit
x(1,1:rvals)=x0*ones(1,rvals); % Startwerte fuer alle r
for i=1:warmup,
 x(1,:)=logmap(x(1,:));        % Vorlauf
end
xp=zeros(1,rvals);
for i=2:points,
 xp=xp+logmapp(x(i-1,:));      % log(Ableitung) aufaddieren
 x(i,:)=logmap(x(i-1,:));      % Speichern fuer Plot
end
xp=xp/(points-1);                  % Liapunov Exponent

subplot(2,1,1),
plot(ones(points,1)*r,x,'.');hold on;
title('logistische Abbildung');
xlabel('r'); ylabel('x');
axis([r(1)-0.01 r(rvals)+0.01 0 1])

subplot(2,1,2),
plot(r,xp);hold on;
title('Lyapunov Exponent');
xlabel('r'); ylabel('lambda');
axis([r(1)-0.01 r(rvals)+0.01 -1 1]);
plot([r(1) r(rvals)],[0 0],':');  % Nullinie
```

Die hier vorkommenden Funktionen sind

```
% file logmap.m
% logistische Abbildung;
% vektorisiert ueber mehrfache r-Werte
function [m] = logmap(x)
global r
m=r.*x.*(1.0-x);
% file logmapp.m
% log der Ableitung der logistischen Abbildung;
% vektorisiert ueber mehrfache r-Werte
function [m] = logmapp(x)
global r
m=log(abs(r.*(1.0-(x+x))));
```

Hier wird für eine Serie von 100 r-Werten die LA jeweils ausgehend von $x = 0.12345$ iteriert. Wir haben die Funktion der LA so geschrieben, daß sie jeweils auf einen Vektor von x-Werten mit einem Vektor von verschiedenen r-Werten operiert. Die Evolutionen für verschiedene r werden also "nebeneinander" und nicht nacheinander ausgeführt. Die Benutzung solcher Vektoroperationen ist in MATLAB besonders effektiv. Die ersten 200 Werte werden jeweils weggelassen, dann wird zum Plotten gespeichert. Der Grund: soweit irgendwelche Fixpunkte genügend[6] anziehend wirken, wird der Weg dorthin, der vom Start abhing, "vergessen", und für den betreffenden r-Wert wird nur der typische Grenzzyklus gezeigt. Mit diesem Programm produzierten wir Abb. 5 und 6.

Für zwei r-Intervalle werden die charakteristischen Bahnen als Funktion von r gezeigt. Abb. 5 überstreicht den Bifurkationsbereich, wo sich bei gewissen Werten $r_1, r_2, \ldots$ jeweils die Periodizität verdoppelt auf $p = 2, 4, 8, \ldots$. In Abb. 6 wird bei $r_\infty = 3.5699...$ der Übergang zu Chaos erreicht. Bei noch größeren r gibt es wieder Zwischenbereiche, mit periodischen Bahnen niedriger Periode p. Obwohl das mit unserer Auflösung kaum sichtbar ist, bifurkieren diese wieder in Kaskden zum Chaos. Dies ist im Fall der LA (und vieler anderer Abbildung) der Mechanismus des Übergangs zum Chaos. In den Abbildungen wird jeweils auch der Liapunov Exponent gezeigt. Die Definition (2.12) haben wir genähert mit

$$\lambda = \frac{1}{N} \sum_{i=0}^{N-1} \log(M'(x^{(i)})), \tag{2.19}$$

wobei $x^{(i)}$ die gezeigten Punkte der Bahn sind. Diese Formel folgt aus (2.12), indem man für $\epsilon \to 0$ die N-fach iterierte Abbildung unter dem Logarithmus differenziert. Wir sehen negative λ, wo stabile periodisch Bahnen vorliegen, die bei den Bifurkationen jeweils verschwinden. Im chaotischen Bereich ist λ positiv entsprechend exponentiell auseinanderstrebenden Bahnen und erreicht bei $r = 4$ den Wert $\lambda = \log(2)$[7].

2.6 Theoretischer Ausblick

Das Gebiet Chaos ist umfangreich und könnte leicht eine Vorlesung für sich füllen. Wir wollen uns hier begnügen mit den in der Vorlesung und den

[6] $200|\lambda| \gg 1$ wäre ein einfaches Kriterium dafür.

[7] Der Lyapunov Exponent ist unabhängig von der Koordinatenwahl und daher hier gleich dem der Dreiecksabbildung

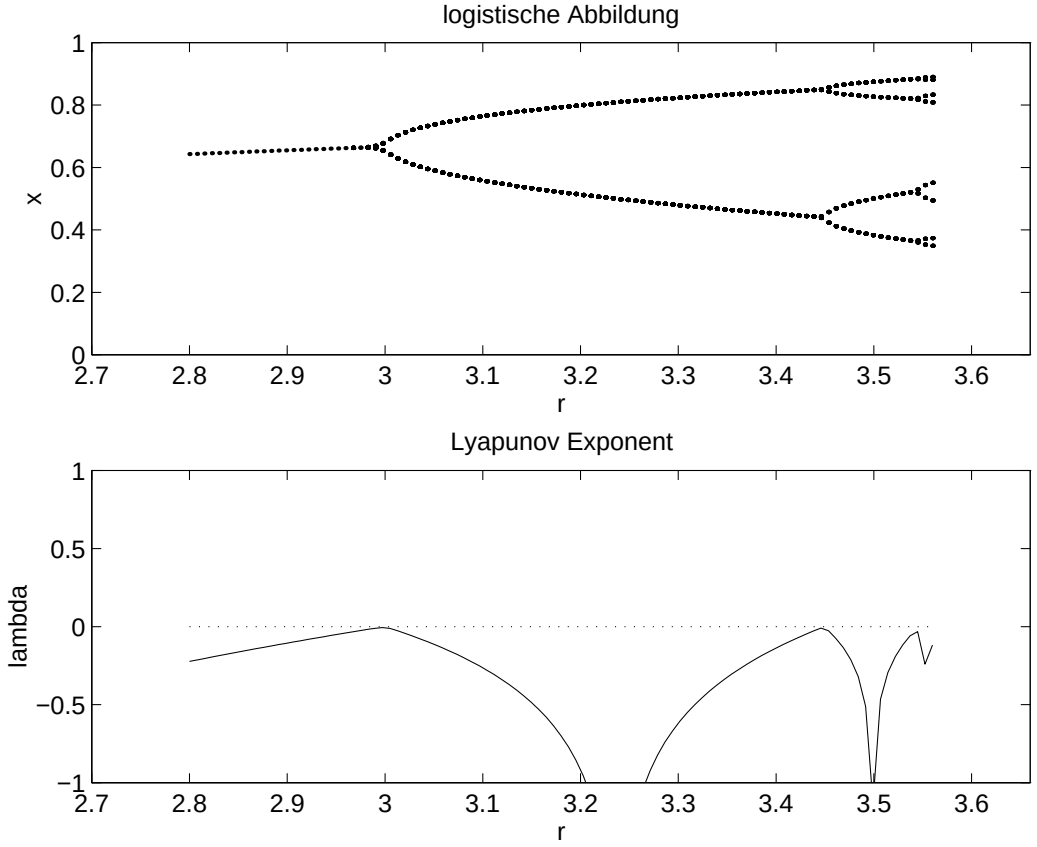

Abbildung 5: Logistische Abbildung, $2.8 \leq r \leq 3.56$

Übungen angerissenen Beispielen. Hier nur noch einige kurze Bemerkungen zur Theorie. Es ist Feigenbaum und anderen gelungen zahlreiche universelle Eigenschaften chaotischer Abbildungen sowohl theoretisch zu begründen als auch in numerischen Experimenten zu verifizieren. Dies betrifft sowohl den Übergang zum Chaos als auch das Verhalten im chaotischen Parameterbereich. Dabei gibt es sehr verblüffende Analogien zur Theorie der kritischen Phänomene (Phasenübergänge). In beiden Fällen versteht man unter universell, daß bestimmte Kenngrößen (kritische Exponenten) nur von qualitativen Dingen wie der Dimension des Phasenraumes und der Grobstruktur einer dynamischen Abbildung abhängen. Damit sind sie für ganze Klassen von Systemen gleich. Die Definition solcher Größen stellt allerdings eine gewisse

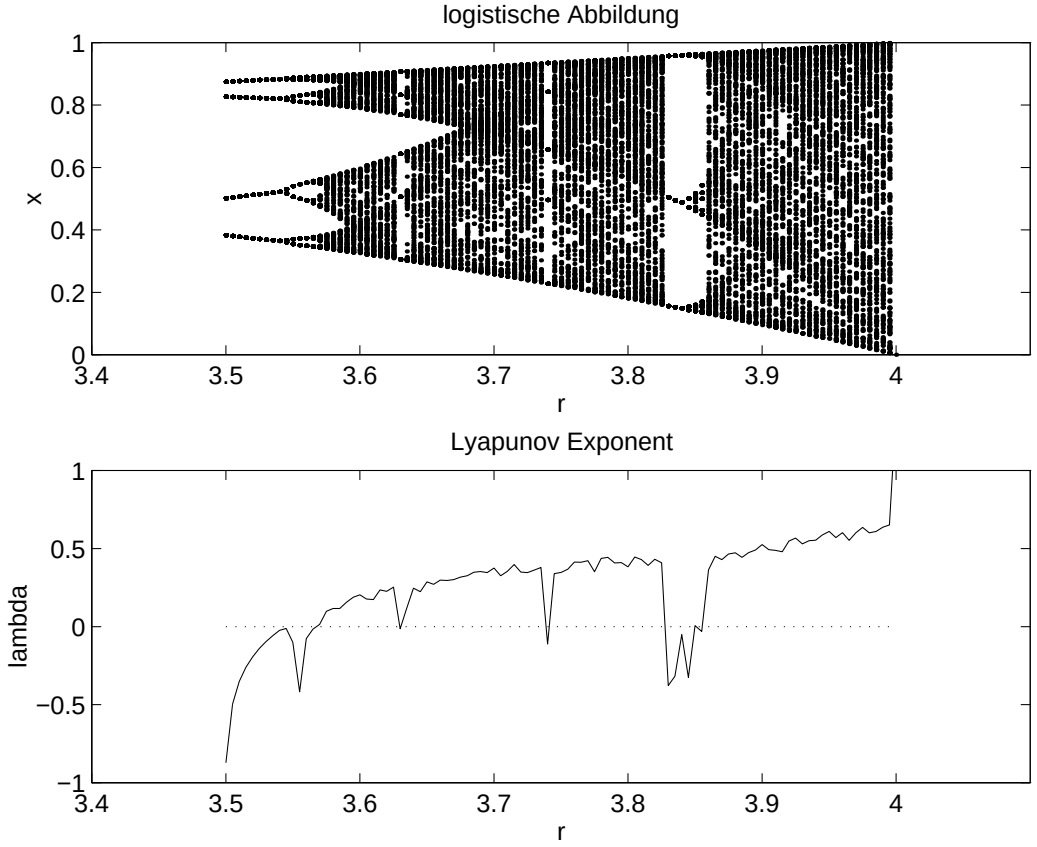

Abbildung 6: Logistische Abbildung, $3.5 \leq r \leq 4$

Kunst dar, die einer Theorie bedarf. Die universellen Eigenschaften sind es auch gerade, die es oft erlauben, Übereinstimmung mit realen Experimenten zu finden, da man dann nicht alle mikroskopischen Details "genau treffen" muß.

Eine naheliegende Frage ist auch, wieso man überhaupt numerisch Aussagen über chaotische Systeme machen kann, die per Definition sensitiv von Anfangsbedingungen abhängen, denn dies scheint zunächst sensitive Abhängigkeit auch von Rundungsfehlern zu implizieren und damit die Frage nach der Relevanz der Resultate. In [2] findet man dazu einige Bemerkungen. Ein Anfangswert im Rechner kann durch darstellbare Zahlen als exakt gegeben betrachtet werden. Bei einem Lyapunov Exponenten $\lambda > 0$ hat die gerechne-

te Bahn spätestens nach n Iterationen mit $10^{-15}\exp(\lambda n) \sim 1$ (z. B. typisch $n = 50$) nichts mehr mit der exakten zu tun. Nach [2] gibt es aber eine exakte Bahn in der Nähe der numerischen, die allerdings zu einem etwas anderen Anfangswert gehört. Somit erhält man also doch Beispiele von Bahnen des Systems, allerdings ohne beliebig genau kontrollieren zu können welche.

3 Monte Carlo Integration

Bei vielen bedeutenden numerischen Verfahren spielen Zufallszahlen eine große Rolle. Das ist gerade dort der Fall, wo ein System zu komplex ist, um seine beschreibenden Gleichungen, selbst wenn diese bekannt sind, für einen allgemeinen Fall zu lösen. Oft ist es dann dennoch möglich, ein System, das diesen Gleichungen folgt, zu simulieren und wenigstens approximative Teilaussagen zu erhalten. In vielen Bereichen der Physik geht es darum, hochdimensionale Integrale über einen Phasenraum auszuwerten. Hier erlauben Monte Carlo (MC) Verfahren einen Zugang, und damit wollen wir uns befassen.

3.1 Ideale Zufallszahlen

Im Folgenden werden wir Verfahren studieren, die als Input Zufallszahlen $\eta_i, i = 1, 2, \ldots$ benötigen, die in einem bestimmten Wertebereich liegen und reell sind. Man stelle sich zur "Ziehung" einer Zufallszahl also vor, eine Funktion (Generator) aufzurufen, die als Antwort ohne irgendeine Eingabe einen zufälligen Wert η ausspuckt. Dabei ist die Verteilung $p(\eta) \geq 0$ per Konstruktion bekannt. Sie legt die Wahrscheinlichkeit

$$W(\eta \in [a,b]) = \int_a^b \mathrm{d}u\; p(u) \tag{3.1}$$

fest, daß bei einer beliebigen Ziehung eine Zufallszahl im Intervall $[a, b]$ liegt. Weiter hat p über den gesamten Wertebereich das Integral Eins, bezieht sich also auf eine Ziehung. Das geläufigste Beispiel sind flache Zufallszahlen im Intervall $[0, 1]$ mit $p(\eta) = 1$.

Stochastische Verfahren benötigen viele Zufallszahlen und wir betrachten Folgen $\eta_1, \eta_2, \ldots, \eta_N$, die durch wiederholtes Aufrufen des Generators entstehen. Eine wichtige Eigenschaft, die für die meisten Anwendungen essentiell ist, ist die Unabhängigkeit dieser Zahlen. D. h. die Wahrscheinlichkeit, daß ein solches N-Tupel als Vektor betrachtet in einem N-dimensionalen infinitesimalen Volumen $\mathrm{d}V$ um den Vektor $(u_1, u_2, \ldots, u_N)$ liegt ist gegeben durch $p(u_1)p(u_2)\cdots p(u_N)\mathrm{d}V$. Insbesondere ist sie auch unabhängig davon, an welcher Stelle die N Zahlen einer längeren Gesamtfolge entnommen werden.

Die Einhaltung der vorgegebenen Verteilung p und die statistische Unabhängigkeit einander folgender Zahlen sind die Qualtitätsmerkmale guter

Zufallszahlen Generatoren. In der Realität ist insbesondere die Unabhängigkeit nur näherungsweise erfüllt. Die verbleibenden kleinen Korrelationen aufeinanderfolgender Zahlen führen dann bei Verfahren, die Unabhängigkeit voraussetzten, zu entprechenden systematischen Fehlern. Übliche Generatoren, die ihre Zahlen ja auch schnell liefern sollen, sind deterministisch, weshalb man genauer auch von Pseudozufallszahlen spricht. Sie bestehen aus einem dynamischen System, das nach irgendeiner Initialisierung bei jedem Aufruf nach irgendeinem Algorithmus um einen Schritt evolviert. Solche Systeme ähneln denen, die wir beim Studium von Chaos betrachtet haben. Chaos bedeutete ja gerade ein rasches "Vergessen" der Anfangsbedingung, ist also hier erstrebenswert.

Eine besonders drastische (und primitive) Korrelation stellt die Periodizität dar: dann wiederholt sich die Folge nach M Aufrufen, ist also völlig durch die vorherigen Zahlen festgelegt. Jeder endliche Generator ist periodisch. Es gibt dann nämlich irgendwelche k Bits, die den Zustand des Systems kodieren. Dann muß nach spätestens 2^k Schritten derselbe Zustand wieder vorliegen und das ganze sich wiederholen. Es ist allerdings nicht schwer, die Periodizität astronomisch groß zu machen. Dazu besteht der Generator Zustand aus mehreren Speicher Worten. Bei einfachen Generatoren allerdings arbeitete man mit nur einem 32 Bit Wort, was zu einer Wiederholung nach $\mathrm{O}(10^9)$ Aufrufen führt. Eine schnelle Workstation ist damit in Minuten durch, und moderne Verfahren verbrauchen leicht viel mehr Zufallszahlen. Das größere Problem ist es, subtilere Arten von Korrelationen zu vermeiden.

3.2 Monte Carlo Integration

Bevor wir uns mit der Erzeugung von Zufallszahlen befassen, wollen wir die Anwendung bei MC Integration besprechen. Diese ist interessant bei hochdimensionalen Integralen, wie wir sehen werden.

Wenn wir eine Funktion f über erzeugte Zufallszahlen mitteln,

$$\langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f(\eta_i), \tag{3.2}$$

so ist klar, daß gilt

$$\lim_{N\to\infty} \langle f \rangle = \int \mathrm{d}u \; p(u) f(u) =: \bar{f}. \tag{3.3}$$

Das Integral geht über den Wertebereich der Zufallszahlen. Die entscheidende Frage ist die nach der Konvergenz der Schätzung mit N. Dazu stellen wir uns vor, daß wir die ganze Schätzung M mal durchführen ($M \to \infty$) mit Zufallszahlen $\eta_{i,a}, i = 1, \ldots, N,\ a = 1, \ldots, M$, die alle natürlich unabhängig sind. Dann haben wir die mittlere Abweichung

$$\begin{aligned} \sigma(N,f)^2 &= \frac{1}{M}\sum_{a=1}^{M}\left[\langle f\rangle - \bar{f}\right]^2 \qquad (3.4)\\ &= \frac{1}{N^2}\sum_{i,j=1}^{N}\frac{1}{M}\sum_{a=1}^{M}[f(\eta_{i,a})-\bar{f}][f(\eta_{j,a})-\bar{f}] \end{aligned}$$

Nun gilt

$$\lim_{M\to\infty}\frac{1}{M}\sum_{a=1}^{M}[f(\eta_{i,a})-\bar{f}][f(\eta_{j,a})-\bar{f}] = \delta_{ij}\mathrm{var}(f) \qquad (3.5)$$

mit

$$\mathrm{var}(f) = \int \mathrm{d}u\ p(u)\left[f(u)-\bar{f}\right]^2 \qquad (3.6)$$

und somit

$$\sigma(N,f) = \sqrt{\frac{\mathrm{var}(f)}{N}}. \qquad (3.7)$$

Damit kann man den erwarteten Fehler der Schätzung angeben,

$$\langle f\rangle = \bar{f} \pm \sqrt{\frac{\mathrm{var}(f)}{N}}. \qquad (3.8)$$

Für $i = j$ ist (3.5) einfach eine Anwendung von (3.2). Für $i \neq j$ wird M mal jeweils ein Zweitupel $(\eta_{i,a}, \eta_{j,a})$ von unabhängigen Zufallszahlen gezogen, die mit der Produktverteilung $p(u_1)p(u_2)$ (Dichte) in einer kleinen Fläche um den Vektor (u_1, u_2) liegen. Im Limes $M \to \infty$ ensteht

$$\left(\int \mathrm{d}u\ p(u)f(u) - \bar{f}\right)^2 = 0.$$

Zur Fehlerabschätzung benötigen wir $\mathrm{var}(f)$, und diese Größe ist normalerweise genausowenig bekannt wie $\bar{f}$. Der Ausweg ist, $\mathrm{var}(f)$ gleichzeitig ebenfalls durch MC Integration zu berechnen. Dazu verwendet man

$$\mathrm{var}(f) = \int \mathrm{d}u\ p(u)f^2(u) - \bar{f}^2 \approx \langle f^2\rangle - \langle f\rangle^2. \qquad (3.9)$$

Selbstverständlich ist diese Schätzung wieder mit einem Fehler $\propto 1/\sqrt{N}$ behaftet. Dieser "Fehler des Fehlers" wird normalerweise vernachlässigt; ansonsten müßte man das Ganze noch eine Stufe höher treiben.

Wenn die Schätzwerte $E = \langle f \rangle$ um $\bar{f}$ herum normalverteilt sind, $\propto \exp[-(E-\bar{f})^2/(2\sigma^2)]$, was für nicht zu pathologische f und nicht zu kleine N zu erwarten ist[8], so hat σ mehr quantitative Bedeutung: 68 % aller Schätzungen liegen im Intervall $[\bar{f}-\sigma, \bar{f}+\sigma]$.

Die Konvergenz der MC Integration geht also proportional zu $1/\sqrt{N}$, wo N ja auch die Anzahl der Funktionsberechnungen $f(\eta)$ ist entsprechend der Zahl der Stützstellen. Vergleichen wir mit einem Standard Integrationsverfahren. Für ein Volumen L^D wähle man Stützstellen mit Raster h in jeder Dimension, und der Algorithmus konvergiere proportional zu $(h/L)^n = N^{-n/D}$ mit der Anzahl Stützstellen $N = (L/h)^D$. Für kleine D, insbesondere D=1, kann man den Exponenten n/D leicht größer als $1/2$ machen, so daß Standardverfahren wie Simpson mit $n = 4$ viel effektiver sind. Für jeden gegebenen Algorithmus der Ordnung n ist aber MC in hohen Dimensionen $D > 2n$ asymptotisch im Vorteil. Ebenso simpel wie verblüffend. Es ist damit oft die einzige Möglichkeit. Dennoch muß man stets zur Verdopplung der Genauigkeit den Rechenzeit Etat vervierfachen, was manchmal frustriert.

Es kommt natürlich auch noch auf den Vorfaktor $\mathrm{var}(f)$ an. Dieser verschwindet genau dann, wenn f konstant ist. Die Werte sind dann unabängig von η, und *eine* "Schätzung" genügt. Diese Betrachtung ist nicht so leer wie es scheint. Angenommen, wir wollen

$$I = \int \mathrm{d}u \; h(u) \tag{3.10}$$

schätzen mit p-verteilten Zufallszahlen. Dann müssen wir in der obigen Betrachtung

$$f = \frac{h}{p} \tag{3.11}$$

wählen, und konstantes f bedeutet, daß die Verteilung p bis auf einen Normierungsfaktor dem Integranden h entspricht,

$$p(u) = \frac{1}{I} h(u). \tag{3.12}$$

Um also genau p-verteilte η zu produzieren, wird man I kennen müssen! Praktisch wird man aber ein p wählen, das man gut erzeugen kann, und das

[8] Dies ist der Inhalt des zentralen Grenzwertsatzes.

möglichst viel von der Variation von h erfaßt im Sinne minimaler Varianz von f. Diese Methode heißt Importance Sampling. Je besser dies gelingt, desto schneller konvergiert die MC Schätzung. Man hat prinzipiell jedoch auch immer mit flachen Zufallszahlen einen korrekten Algorithmus, aber die nötigen N könnten unpraktikabel werden. Diese Gefahr besteht besonders bei oszillierenden h, da ja p als Wahrscheinlichkeit nur positiv sein kann.

3.3 Beispiel für Monte Carlo Integration

Als Beispiel wollen wir nun das Integral

$$I = \int_0^1 \mathrm{d}u \, \frac{1}{1+u^2} = \arctan(u)\Big|_0^1 = \pi/4 \tag{3.13}$$

per MC berechnen [1]. Dazu nehmen wir flach verteilte Zufallszahlen, die in MATLAB unter **rand** zur Verfügung stehen. Die Kernzeilen des MATLAB Programms sind

```
absc=[1:N];
x=rand(1,N);    % N Zufallszahlen
f=1./(1+x.*x); % Integrand (Vektor)
f=cumsum(f);    % kumulative Summe
f=f./absc;      % Mittelwerte
```

Mit **cumsum** wird aus einem Vektor ein neuer gebildet, der die Teilsummen enthält (cumulative sum). Damit wurden für $N = 200$ Abb. 7 und 8 erzeugt. Wenn man solche Experimente wiederholt, findet man durchaus auch Fälle, wo mehr oder weniger die ganze Fieberkurve außerhalb des "Konvergenzkorridors" liegt, wenn auch niemals um große Faktoren. Das zeigt, daß die Fehlerabschätzung nur ungefähr gilt, und daß 2σ Abweichungen durchaus praktisch vorkommen. Auch muß man beim Betrachten der Abbildungen bedenken, daß für wachsendes N die früheren MC-Schätzungen weiter im Mittel drin sind. Die verschiedenen Teile der Kurve sind also nicht etwa unabhängig. Liegt sie einmal unter dem exakten Wert, so wird sie eine Weile dort bleiben.

In Abb. 9 sehen wir die Verteilung von 1000 unabhängigen MC Schätzungen mit N=200. Die Varianz ist für diesen Integranden ebenfalls exakt bekannt,

$$\mathrm{var}(f) \;=\; \frac{1}{4} + \frac{\pi}{8} - \frac{\pi^2}{16} \approx 0.0258 \tag{3.14}$$

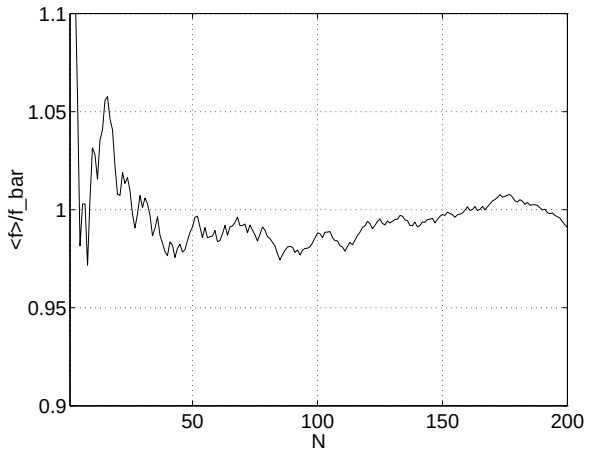

Abbildung 7: $\langle f\rangle/I$ für *eine* MC Schätzung

$$\sqrt{\frac{\mathrm{var}(f)}{200}} \approx 0.011,$$

was gut zu der Breite der Verteilung paßt.

Wir wollen nun für dieses einfache Beispiel anders verteilte Zufallszahlen verwenden. Wenden wir auf einen Typ Zufallszahlen eine monotone Funktion b an,

$$\eta \to \tilde{\eta} = b(\eta), \tag{3.15}$$

so erhalten diese die Verteilung

$$\tilde{p}(\tilde{\eta})\mathrm{d}\tilde{\eta} = p(\eta)\mathrm{d}\eta = p(B(\tilde{\eta}))B'(\tilde{\eta})\mathrm{d}\tilde{\eta}, \tag{3.16}$$

wobei B die Umkehrfunktion von b ist. Der Bereich der $\tilde{\eta}$ ist natürlich der Bildbereich der ursprünglichen Zahlen unter b. Ausgehend von $p = 1$ würde man für optimales Importance Sampling anstreben, daß $\tilde{p}(u)(1+u^2)$ konstant wird,

$$B'(u) \stackrel{!}{=} c\frac{1}{1+u^2}. \tag{3.17}$$

Zusammen mit $B(0) = 0, B(1) = 1$, führt dies auf

$$B = \arctan,\ b = \tan,\ c = 4/\pi, \tag{3.18}$$

und nun ist trivialerweise der konstante Integrand $1/c$ zu integrieren.

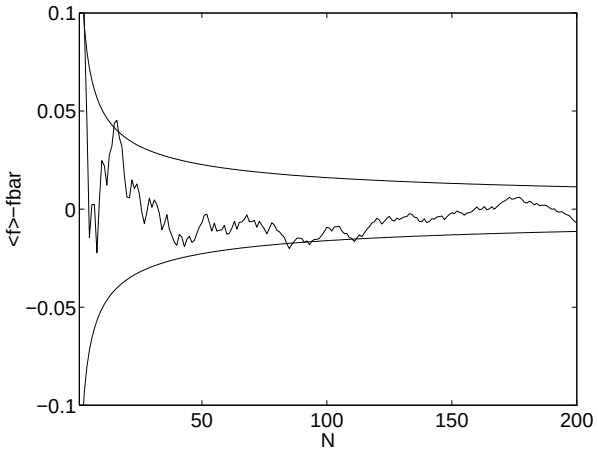

Abbildung 8: $\langle f \rangle - I$ und "Konvergenzkorridor" $\pm\sigma(N, f)$

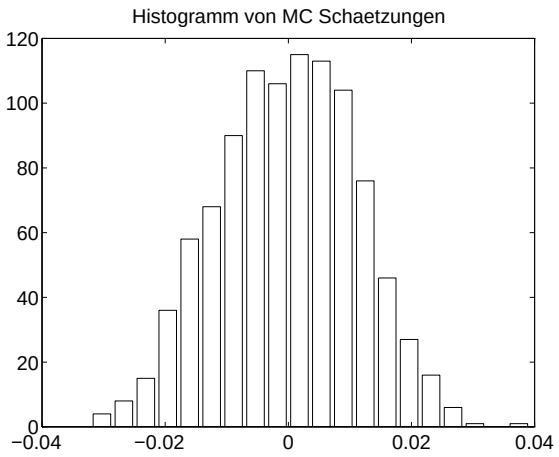

Abbildung 9: Verteilung der Abweichung von 1000 Berechnungen $\langle f \rangle$ mit N=200

Realistischer ist es, daß man keine exakte Konstanz erreicht, aber eine Verteilung findet, die ähnlich ist. Hier wäre z. B. eine lineare Verteilungsfunktion denkbar, die wie der Integrand monoton fällt und $p(0) = 2p(1)$ hat,

$$p(u) = \frac{2}{3}(2-u). \tag{3.19}$$

Sie entsteht durch eine Transformation mit

$$b(\eta) = 2 - \sqrt{4-3\eta} \tag{3.20}$$

aus flachen Zufallszahlen. Der Integrand ist dann gegeben durch

$$f(u) = \frac{1}{p(u)(1+u^2)} \tag{3.21}$$

mit der Varianz[9]

$$\mathrm{var}(f) = \int_0^1 \mathrm{d}u\; p(u)f^2(u) - (\pi/4)^2 = \frac{36\log(2) + 90 + 42\pi - 25\pi^2}{400} \approx 0.0004. \tag{3.22}$$

Der Fehler ist also bei diesem sampling etwa 8 mal kleiner, obwohl auch (nur) mit $N^{-1/2}$ sinkend.

3.4 Zufallszahlen in MATLAB

In MATLAB gibt es selbstverständlich eingebaute Zufallszahlen, die wir zunächst verwenden wollen, bevor wir uns der Erzeugung von Zufallszahlen zuwenden. Wie wir schon sahen, erhält man durch den Aufruf der Funktion **rand** flach verteilte Zufallszahlen zwischen 0 und 1. Nach jedem Start von MATLAB erhält man die gleiche Folge, die einer Standard Initialisierung der Variablen entspricht, die den Zustand des Generators beinhalten. Will man diesen Zustand abspeichern, z. B. um mit Hilfe dieser Information später mit der Zufallsfolge fortzufahren, so geht das wie folgt:

```
>> s=rand('state');
>> whos s
  Name      Size          Bytes  Class
```

[9] Dank sei Maple für dieses Integral

```
  s         35x1             280  double array

Grand total is 35 elements using 280 bytes
```

Der Zustand ist gegeben durch einen 35-elementigen Vektor. Die Initialisierung mit einem solchen Vektor geschieht durch das statement `rand('state',s)`. Will man verschiedene Folgen haben ohne ein `s` zur Verfügung zu haben, das aus dem Generator stammt, so kann man auch mit `rand('state',j)` initialisieren, wobei `j` eine ganze Zahl ist. `j=0` entspricht der Standard Initialisierung bei Programmstart. Ein Start mit irgendwelchen Vektoren ist vorsichtshalber nicht zu empfehlen.

Die MATLAB Dokumentation sagt nicht, mit welchem Algorithmus dieses **rand** arbeitet. Im Internet wurde diese Frage an den Hersteller gerichtet, was zu folgender etwas kuriosen Antwort führte:

The algorithm for the uniform random number generator in MATLAB 5 is based on e-mail correspondence between Cleve Moler and George Marsaglia. The algorithm that is used has not been published.

Nun ja. In MATLAB 4 gab es einen relativ primitiven Generator, der laut Handbuch mit der Rekursion

$$K' = 7^5 K \pmod{2^{31}-1} \tag{3.23}$$

auf ganzzahligen K, K' evolviert. Der Zustandsraum hat also nur $2^{31}-1$ Punkte und damit maximal diese Periode. Genaueres hierzu wird in einem späteren Abschnitt besprochen. Dieser Generator steht auch in MATLAB 5 noch zur Verfügung. Dazu muß man zunächst mit `rand('seed',K)` (seed statt state!) initialisieren, während `rand('seed')` den aktuellen (ganzzahligen) Zustand ausliest. Aufgerufen wird dann identisch wie vorher mit **rand**.

```
>> rand('seed',1)
>> rand

ans =

    0.5129

>> rand('seed')

ans =
```

```
       16807

>> 7^5

ans =

       16807

>>
```

Übrigens kriegt man auch beim Start von `rand('seed',0)` eine brauchbare Folge; das Programm ändert diesen eigentlich unbrauchbaren Wert und stellt eine Standard Initialisierung ein. Ein Qualitätstest für diesen Generator wird später durchgeführt, eine gewisse Skepsis ist angebracht für Anwendungen mit sehr hoher Statistik. Er wird uns als Negativbeispiel dienen.

4 Erzeugung von Zufallszahlen

In diesem Abschnitt wollen wir einige Standard Algorithmen zur Erzeugung von Zufallszahlen studieren. Neben ihrer Definition sollen auch einige statistische Tests beschrieben werden, mit denen man die Qualität der Methoden untersucht. Einige der hierbei benötigten Manipulationen sind in MATLAB gar nicht oder nur umständlich möglich. Daher werden wir an dieser Stelle einfache Programme in der Programmiersprache C betrachten. Der Autor verfügt bisher nur über umfangreiche Erfahrung in Fortran. Daher handelt es sich bei C in besonderem Maße um einen gemeinsamen Lernprozeß. Kommentare zu den vorgestellten Programmen bzw. Fragmenten von C-erfahreneren Hörern sind (wie immer) willkommen.

Der Grund für C liegt in der größeren Bedeutung und dem Nutzen als Berufsausbildung. Hitzige Debatten über Vor- und Nachteile von Fortran und C sind allerdings meist stark ideologisch.

4.1 Generatoren nach der linearen Kongruenz-Methode

Die lineare Kongruenz-Methode (LK) ist *das* Standard Verfahren Zufallszahlen[10] zu erzeugen. Die Folge besteht primär aus ganzen Zahlen $0 \leq I_j < m$, die, ausgehend von irgendeinem Anfangswert ("seed") I_0, rekursiv gebildet werden gemäß

$$I_{j+1} = aI_j + c \pmod{m} \tag{4.1}$$

Die üblichen in $\eta_j \in [0,1)$ ergeben sich durch Multiplizieren mit $1/m$. Zur Wahl der magischen Zahlen a, c, m gibt es einige Theorie, die in [5] in einigem Detail zu finden ist. Klarerweise ist die maximale Periode eines solchen LK Generators durch m gegeben; spätestetens beim m'ten Aufruf muß sich I_0 wiederholen, und alles geht von vorne los. Die Theorie erlaubt rigoros Tripel a, c, m anzugeben, die diese maximale Periode tatsächlich haben. Dazu hinreichend und notwendig ist[5]:

- c und m haben keine gemeinsamen Primfaktoren
- $a - 1$ ist Vielfaches von jeder Primzahl, die m teilt
- falls $a - 1$ Vielfaches von 4 ist, dann auch m

[10] Eigentlich Pseudozufallszahlen, da alles deterministisch ist.

Dann werden also alle m Werte in irgendeiner Reihenfolge durchlaufen, und ein Startwert I_0 ist so gut wie jeder andere. Man erhält lediglich einen anderen Unterabschnitt der Folge. Über die statistischen Eigenschaften der Zufallszahlen jenseits der Periodizität läßt sich wenig rigoros sagen. Hier gibt es ganze Batterien von mehr oder weniger empirischen Tests, von denen später einige betrachtet werden.

Gängige Werte für m sind $2^{31}, 2^{32}$, da man dann die Modulo Operation auf vielen Maschinen einfach durch Ignorieren des Überlaufs bekommt, ohne teure Division. Eine Unterklasse von LK Generatoren hat $c = 0$ und spart so noch die Addition. Dann ist die Periode m typischerweise nicht ganz erreichbar. Der alte MATLAB 4 Generator[11] ist, wie schon erwähnt, von diesem Typ mit $a = 7^5, c = 0, m = 2^{31} - 1 = 2\,147\,483\,647$; seine Periode ist $2^{31} - 2$. Er wird z. B. in [13] diskutiert. Dort finden sich auch weitere empfohlene Kombinationen (a, c, m).

4.2 Fibonacci und Shift Register Generatoren

Die berühmte Fibonacci Folge ist definiert durch die Rekursion

$$F(n+1) = F(n) + F(n-1) \quad F(0) = 0,\ F(1) = 1. \tag{4.2}$$

und ergibt Werte $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$. Nach dieser Bauart kann man auch einen Generator konstruieren,

$$I_{j+1} = I_j + I_{j-1} \pmod m, \tag{4.3}$$

der I_j zwischen 0 und $m - 1$ produziert. Obwohl man bei geignetem Start Perioden länger als m leicht erhält, hat sich die statistische Qualität dieser Zahlen nicht bewährt.

Die längere Periode ist durch das im Vergleich zu LK Generatoren längere Gedächtnis bis zur vorletzten Zahl möglich. Indem man hier noch weitergeht, kann man zu besseren Verfahren kommen, den verzögerten (lagged) Fibonacci Generatoren (LF). In [5] findet man z. B.

$$I_j = I_{j-24} + I_{j-55} \pmod m \tag{4.4}$$

mit geradem m. Dies zu realisieren ist nicht schwierig: man nimmt einen Vektor der Länge 55, den man sich zyklisch geschlossen denkt, mit Indizes

[11] Wird von **rand** verwendet **nach** Initialisierung `rand('seed',i),i=0,1,2,...`

$1, \ldots, 55$. Mit I_{56} darf man nun I_1 überschreiben, da es ja weiter nicht mehr gebraucht wird, I_{57} kann über I_2, usw. Zum Start müssen 55 Werte vorgegeben werden, die nicht alle gerade sein sollen. Die Periode ist beweisbar größer als $2^{55} - 1$, praktisch meist viel größer.

In MATLAB code sähe dieser Generator so aus:

```
% file lag_fib.m

function eta = lag_fib

global r i1 i2;
% Startwerte: i1=1, i2=32, r=r(1:55) 0<r(i)<1

r(i1)=r(i1)+r(i2);
if(r(i1) >= 1.0), r(i1) = r(i1) - 1.0; end

if(i1==55), i1=1; else, i1=i1+1; end;
if(i2==55), i2=1; else, i2=i2+1; end;
eta=r(i1);
```

Man benötigt nur Addition, und wie man sieht, kann er direkt auf dem Niveau der auf $[0, 1)$ normierten Gleitkommazahlen realisiert werden. Der beschriebene zyklische Vektor ist mit **if** Anweisungen realisiert. Die Routine ist für MATLAB so relativ langsam, da sie die schnellen Vektoroperationen nicht ausnutzt. Dies ist am einfachsten zu verbessern, indem man n Kopien des Generators gleichzeitig evolviert. Die dazu nötigen Änderungen sind die Ersetzung der letzten Zeilen durch

```
r(:,i1)=r(:,i1)+r(:,i2);
r(:,i1)=r(:,i1)- ( r(:,i1) >= 1 );
eta=r(:,i1);
```

r muß jetzt als Matrix der Größe $(n, 55)$ initialisiert werden, dann werden über `eta` Spalten der Länge n zurückgegeben. In MATLAB ist `r(:,i1) >= 1` ein Vektor mit Werten 1, wo die Bedingung erfüllt ist, und 0 sonst. Elegant, oder? Man braucht natürlich weiter nur *ein* Paar `i1,i2`, so daß der Überhang (Overhead) zu ihrer Berechnung bedeutungslos wird für größere n.

Laut [5] hat sich dieser Generator in vielen Anwendungen bewährt, aber vor allem das weitgehende Fehlen von Theorie steht seiner Verwendung in kritischen Fällen entgegen.

Bei Shift Register Generatoren in einer populären Variante hat man wieder einen zyklischen Vektor, z. B. der Länge 250 als einer bewährten Wahl magischer Zahlen. Dann lautet die Rekursion:

$$X_j = X_{j-147} \bigoplus X_{j-250}. \tag{4.5}$$

Hier sind X_j einfach als Bit-Ketten der Länge eines Computerwortes anzusehen (z.B. 32), und $\oplus$ verknüpft zwei solche zu einer neuen mit durch die Operation "bitweises exklusives oder" [$0 \oplus 0 = 0 = 1 \oplus 1$, $0 \oplus 1 = 1 = 1 \oplus 0$]. Im Gegensatz zum Rechnen mit ganzen oder Gleitkommazahlen, wo die einzelnen Bits durch Überträge miteinander verknüpft sind, werden hier die Bits der Folgenmitglieder unabhängig verknüpft. Es findet gewissermaßen Parallelverarbeitung über die Wortlänge statt. Man hat also z. B. 32 1-Bit Zufallszahlengeneratoren. Ihre Periode wird durch die Vektorlänge 250 bestimmt und beträgt $2^{250}-1$, wenn man nicht gerade mit lauter Nullen startet. Nun kann man möglichst irreguläre 250 Startworte vorgeben, und dann die neu enstehenden Bitmuster als die Binärdarstellung ganzer Zahlen interpretieren und diese so ausgeben und ggf. umnormieren. Wenn alle Bits genutzt werden, wird auch das Vorzeichen(bit) fluktuieren. Dies läßt sich abschalten, denn offenbar ist es so, daß, wenn man gewisse Bits auf der Startkonfiguration überall Null setzt, diese Null bleiben. Klarerweise sind hier die einzelnen Bits, also insbesondere die Stellen weit links und weit rechts in der Zahl, völlig gleichwertig hinsichtlich Periodizität und Korrelationen. Dies ist anders bei LK, wo die rechten Stellen allein betrachtet zunehmend kürzere Perioden haben können.

Die gerade diskutierten Operationen waren bis MATLAB 4 nicht durchführbar, da es weder explizit ganze Zahlen noch bitweise Operationen gab. Mit MATLAB 5 und 6 hat sich dies geändert, s. z. B. **help bitxor**. Wir wollen hier jedoch bei der Realisierung in C bleiben (Übungen) um uns damit etwas vertraut zu machen.

4.3 Marsaglia-Zaman Generator

Der Marsaglia-Zaman Generator (MZ) funktioniert nach dem "subtract with borrow" Prinzip und erzeugt ganze Zahlen $0 \leq I_j < m$ zusammen mit einem Übertrag (Carry-) Bit $c_j \in \{0, 1\}$. Die Rekursion lautet:

$$\begin{aligned} \Delta &= I_{j-s} - I_{j-r} - c_{j-1} \\ I_j &= \Delta, \qquad c_j = 0 \text{ wenn } \Delta \geq 0 \end{aligned} \tag{4.6}$$

$$I_j \;=\; \Delta + m, \quad c_j = 1 \text{ wenn } \Delta < 0 \tag{4.7}$$

Hier ist Δ nur eine Hilfsgröße, und man sieht leicht ein, daß diese Vorschrift im Bereich $[0, m-1]$ bleibt. Es gibt wieder einige mathematische Theorie zur Wahl von r, s, m, und ein bewährtes und gut getestetes Tripel ist $r = 24, s = 10, m = 2^{24}$. Praktisch arbeitet man nun wieder mit einem zyklischen Vektor der Länge 24. Die generierten Werte bestehen aus 24 Bit Zahlen. Dies ist genau die Länge der Mantisse für einfach genaue Gleitkomma Zahlen im IEEE Standard, so daß MZ auf diese abgebildet werden und mit Gleitkomma Operationen arbeiten kann. Dies ist in [6] ausgenutzt im Hinblick auf Implementierung in Parallelrechnern vom APE Typ, die keine schnelle Integer Arithmetik zulassen. In diesem Kontext ist diese Version von MZ dort ausführlich getestet. Für die Periode ergibt sich ein astronomischer Wert von 5×10^{171}. Dies wird interessanterweise dadurch bewiesen, daß man MZ auf einen LK Generator abbildet mit riesiger effektiver Wortlänge.

4.4 Realisierung von Generatoren in C

Ein meist primitiver Generator kommt mit C Compilern in einer Standard Bibliothek. Hier ist ein erstes Programm, das ihn vorführt. Es existiere ein file, hier `C_rand.c`, des folgenden Inhalts:

```
1    /* Demonstration des C Zufallszahlengenerators */
2
3    #include  <stdio.h>
4    #include <stdlib.h>
5
6    void srand(unsigned seed);
7    int rand(void);
8
9    int main(void)
10   {
11
12   int i,ir[5];
13   float a,r[5];
14
15   printf("\n maximale Zufallszahl = %i (hex: %x) \n\n",RAND_MAX,RAND_MAX);
16
17   a=1.0/(RAND_MAX+1.0);
```

```
18
19   srand(137);
20
21   for(i=0; i<=4; i++)
22     {
23     ir[i]=rand();
24      r[i]=ir[i]*a;
25     printf("\n ir(%i) = %5i, r(%i) = %f ",i,ir[i],i,r[i]);
26     }
27
28   printf("\n\n");
29   }
```

Die Nummern am Anfang der Zeilen sind vorgesetzt (mit MATLAB dbtype) um sie hier zu erläutern. In dem zu compilierenden File dürfen sie *nicht* stehen. Wir wollen dieses Programm kurz besprechen. Dies kann jedoch nicht die begleitende Lektüre und das Nachschlagen eines Buches über C ersetzen. Wir haben dazu [7] und [8] verwendet.

Jedes Statement wird mit ; abgeschlossen, das Eingabeformat ist frei und nur durch Geschmack und Übersichtlichkeit bestimmt.
Z.1: Kommentare können an beliebiger Stelle zwischen `/* ... */` stehen
Z.3,4: Laden von Bibliotheken, die Funktionen (hier printf, rand, srand) zur Verfügung stellen
Z.6,7: Prototypen zum Aufruf von Funktionen; `void` bedeutet kein Argument zur Übergabe bzw. Wertrückgabe. Dadurch weiß der Compiler die Struktur von eventuell erst weiter unten definierten Funktionen. Nicht immer zwingend notwendig, aber guter Stil und klärend. Der Typ unsigned ist eine Integer ohne Vorzeichen (4 Byte auf HP-Workstation[12]) `int` das Gleiche mit Vorzeichen
Z.9: Beginn des Hauptprogramms mit Block von 10 bis 29
Z. 12: Deklaration ganzzahliger Variabler; `ir` ist Feld der Länge 5, anzusprechen in C mit Indizes 0,1,2,3,4
Z.13: Gleitkomma Zahlen (32 Bit lang, ca. 10^{-7} relative Genauigkeit)
Z.15: Druckbefehl, wie schon bei MATLAB besprochen; Formate %i für ganze Zahlen, %x hexaddezimal; `\n` = Zeilenvorschub; `RAND_MAX` wird mit dem Generator `rand` über die Bibliothek `stdlib` als Konstante zur Verfügung

[12] Die Längen von `short, int, long` hängen vom Rechner ab; mit der Funktion `sizeof(int)`, usw. kann man die Länge in Bytes herausfinden

gestellt und gibt die größte Zufallszahl an, die `rand` liefert (i. a. maschinenabhängig)
Z.19: hier wird mit einer heiligen Zahl initialisiert
Z.21: eine `for`-Schleife (von Z.22 bis 26) die bei 0 startet, mit `i++` jeweils um 1 inkrementiert wird (äquivalent $i = i + 1$), und ausgeführt wird solange $i \leq 4$ gilt.

Wir compilieren im pool auf `poolXX` z. B. mit dem GNU-compiler übersetzt mit dem Kommando `gcc C_rand.c`. Dabei entsteht, wenn das Programm fehlerfrei ist, das ausführbare Programm `a.out`. Es wird ausgeführt:

```
> a.out

 maximale Zufallszahl = 2147483647 (hex: 7fffffff)


 ir(0) = 171676246, r(0) = 0.079943
 ir(1) = 1227563367, r(1) = 0.571629
 ir(2) = 950914861, r(2) = 0.442804
 ir(3) = 1789575326, r(3) = 0.833336
 ir(4) = 941409949, r(4) = 0.438378
```

`RAND_MAX` ist also durch die Binärzahl $01111111111111111111111111111111 = 2^{31} - 1$ gegeben.

Der obige Generator ist natürlich für MC-Integration oder gar Simulation ungeeignet. Allenfalls kann man damit einen anderen Generator initialisieren. Er diente somit mehr dazu, Bekanntschaft mit C zu schließen.

Nun wollen wir selbst ein Programm für einen Generator schreiben. Im ersten file wird die Funktion `quickie` definiert:

```
/* file quick.c :
   Quick and Dirty Zufallszahlen Generator */

/* m=2^(32), modulo durch ignorierten Ueberlauf
   IA,IC nach Numerical Recipes in C, S. 284 */

#define IA 1664525
#define IC 1013904223
#define FACT 2.328306437E-10 /* = 1/2^(32) */
```

```
float quickie(unsigned long *idum)
{
  *idum=*idum*IA+IC;
  return FACT*(*idum);
}
```

Neue Elemente sind:

- Mit `#define` wird der sog. Präprozessor angesprochen; Im folgenden Text wird vor dem Compilieren eine Ersetzung durchgeführt. Das Resultat ist also, als ob man überall, wo `IA` usw. vorkommt, die entsprechende Zahl getippt hätte
- Ein neuer Typ ist `unsigned long`, 32 Bit Ganzzahl ohne Vorzeichen. Die Tatsache, daß `*idum` und nicht `idum` im Argument steht, ist von größter Bedeutung: es handelt sich um einen Zeiger. Das ist die Adresse des Speicherplatzes von `idum`, das im aufrufenden Programm definiert sein muß. In der Funktion wird nun mit `*idum` der Inhalt des Speicherplatzes angesprochen. Würde man in C `idum` selbst übergeben und dann entsprechend benutzen (ohne *), so erhielte die Funktion eine *Kopie*, der Wert selbst bliebe ungeändert
- `return` gibt den darauf folgenden Wert zurück; `FACT` ist durch eine Gleitkomma Konstante ersetzt, und bei der Produktbildung wird `idum` in Gleitkomma konvertiert

Nun folgt ein Hauptprogramm:

```
/* file m_quick.c :
   Hauptprogramm zur Demonstration von quickie */

#include  <stdio.h>
#include "quick.c"

float quickie(unsigned long *);

int main(void)
{
unsigned long idum;
```

```
long i;
float x1,x2,rnum;
FILE *dat_ptr;

  idum = 0;
  for(i=0; i<=4; i++)
  {
    rnum=quickie(&idum);
    printf("\n eta = %f, idum = %u (hex = %x) " \
           ,rnum,idum,idum);
  }
  printf("\n\n");

  dat_ptr=fopen("ebenen.dat","w");

  for(i=0; i<=100000000; i++)
  {
    x1=quickie(&idum);
    x2=quickie(&idum);
    if(x1 < 0.00001)
    {
      fprintf(dat_ptr,"%f %f\n",x1,x2);
    }
  }
  fclose(dat_ptr);
}
```

Hierzu folgende Erläuterung;

- Die gerade betrachtete Funktionsdefinition wird mit `#include` geholt und eingefügt
- danach kommt die Prototyp Zeile für den Funktionaufruf (Zeiger Variable im Argument)
- mit `FILE` wird ein Zeiger auf eine Datei definiert
- mit \ kann man Forsetzungszeilen verwenden

- im Argument von `quickie` steht `&idum`; `&` macht das Gegenteil von `*`: es liefert den Adress Zeiger der danach folgenden Variable (statt den Inhalt zum Zeiger)
- bei `fopen` wird ein file eröffnet zum Schreiben (w=write) und mit dem Datei Zeiger `dat_ptr` verknüpft
- `fprintf` ist ähnlich wie `printf`, schreibt aber in die geöffnete Datei
- `fclose` schließt die Datei

Im letzten Teil werden offenbar 10^8 zweikomponentige Vektoren erzeugt und, wenn die erste Koordinate kleiner als 10^{-5} ist, in den File geschrieben. Das wird in einem Test verwendet.

4.5 Tests von Zufallszahlen

In der Literatur gibt es zahlreiche Testverfahren für Zufallszahlen Generatoren. Dies sind mehr oder weniger plausible statistische Anwendungen von Zufallszahlen. Ein Beispiel sind χ^2 Tests. Man simuliert z.B. das simultane Werfen von mehreren Würfeln mit Hilfe aufeinanderfolgender Zahlen der Folge. Dann wird rechnerisch analysiert wie wahrscheinlich ein gefundenes Resultat bei perfekten Würfeln auftreten würde. Wenn man dabei wiederholt sehr unwahrscheinliche Resultate bekommt, dann ist der verwendete Generator suspekt. Man beachte, daß man hier, ganz analog zu Messungen, die mit einer Formel gefittet werden sollen, nur statistische Aussagen machen kann, und kein einfaches richtig oder falsch bekommt.

Der obige Test ist realen Anwendungen ähnlich, z. B. wenn man mit Hilfe von Zufallszahlen eine Wahl trifft. Praktisch kann man jede Anwendung, z. B. MC Integration, auch als Test ansehen. Dazu muß man entweder das exakte Resultat wissen, oder die Konsistenz von Rechnungen oder Simulationen mit mehreren Generatoren vergleichen. Insbesondere die Simulation von komplexen statistischen Systemen, die man lösen kann (Ising oder Gauß'sches Modell, kommt später), mit hoher Statistik, ist ein sehr sensitiver Test.

Eine weitere Klasse von Tests nutzt die Fähigkeit des Auges aus, Strukturen zu erkennen, die visuellen Tests. Am einfachsten ist es, mit Paaren von Zufallszahlen Punkte in der Ebene zu erzeugen und diese zu plotten. Dies sehen wir in Fig.10 für den MATLAB 4 Generator **rand** (Initialisierung mit `rand('seed',0)` !). Im linken Bild sehen wir die Verteilung von

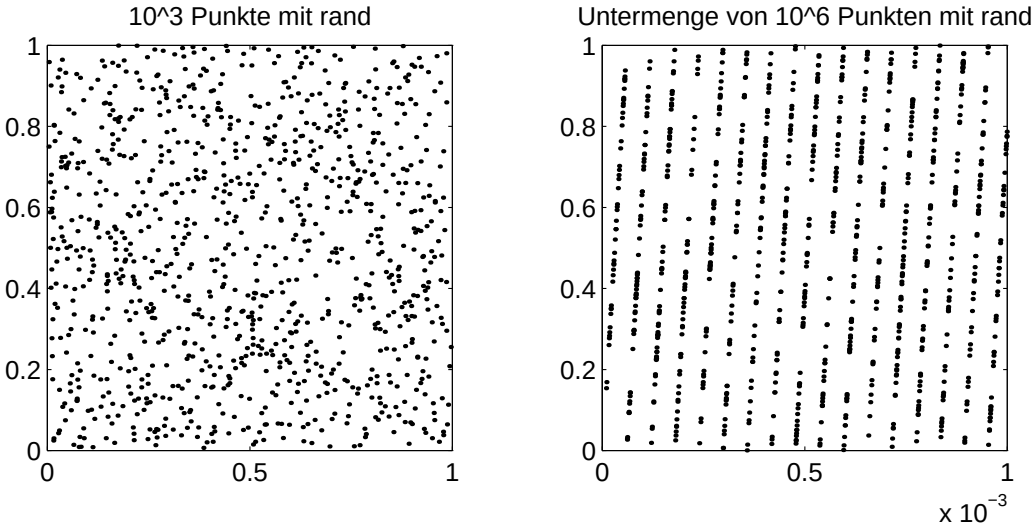

Abbildung 10: Punkte in der Ebene mit MATLAB Funktion **rand**

1000 Vektoren im gesamten Wertebereich, wo visuell wohl nichts besonderes auffällt. Das rechte Bild zeigt den Ausschnitt für x zwischen 0 und 0.001 eines analogen Plots von mehr Punkten. Hier erkennt man das für LK Generatoren typische Phänomen, daß nur gewisse parallele Ebenen besiedelt werden. Ähnliches passiert in höheren Dimensionen, und die Parameterwahl kann nur dafür sorgen, daß die Ebenen möglichst dicht werden. Das analoge Bild für den vorgestellten LF Generator in Abb.11 zeigt bei gleichen Parametern keine solche Anomalie. Für unseren in C geschriebenen LK Generator muß man mehr Zahlen erzeugen und einen engen Ausschnitt wählen, um die Ebenen zu sehen. Die ist in Abb.12 gezeigt, die mit MATLAB aus dem vorher geschriebenen Datenfile erstellt ist (einlesen mit **load**). Man beachte, daß nun der x-Bereich 100 mal kleiner ist als vorher. Durch unsren Wechsel zu C ist das Programm schnell genug um die hier verwendeten 10^8 Punkte in etwa fünf Sekunden zu erzeugen.

Allgemein kann man sagen, daß es zu jedem Generator Tests gibt, die seine Defekte zeigen. Schließlich ist der Folgezustand durch seinen Vorgänger in Wirklichkeit determiniert. Als Zustand ist dabei das anzusehen, was man initialisiern muß; eine Zahl bei LK, und z. B. einen Vektor in 55 Dimensionen bei LF. Nach einem bzw. 55 Schritten liegt ein neuer Zustand vor. Wenn

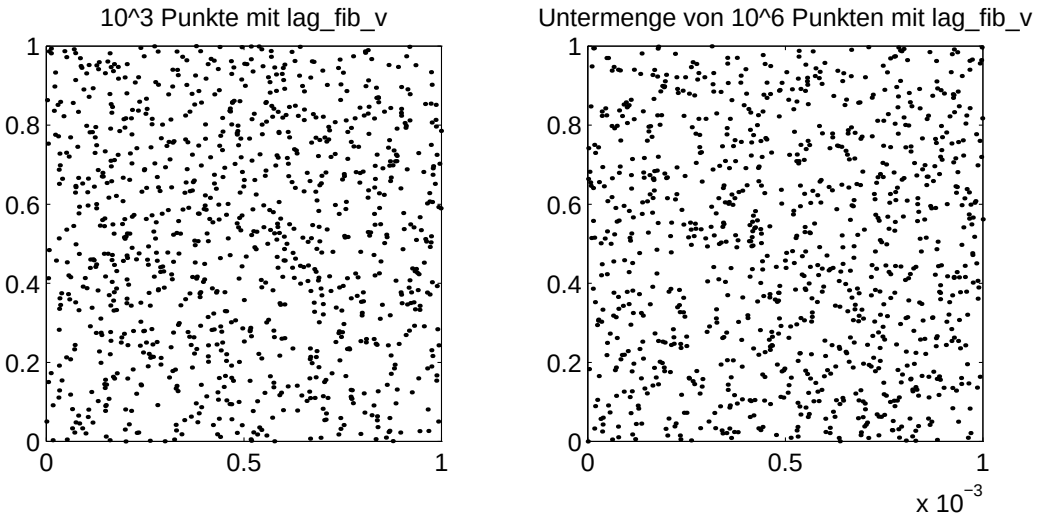

Abbildung 11: Punkte in der Ebene mit LF Generator

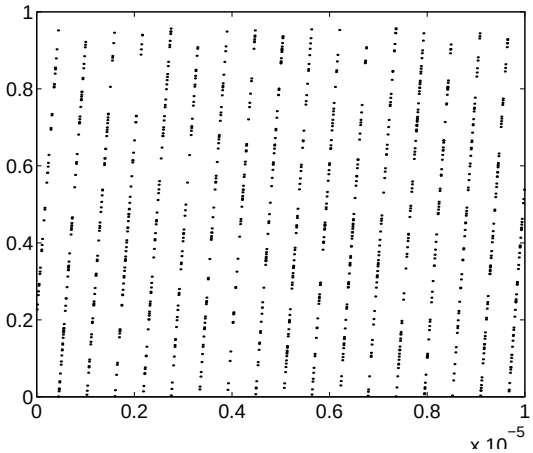

Abbildung 12: LK Generator quickie im "Streifentest"

man das Bildungsgesetz kennt, kann man stets eine Größe definieren, die den Wert 1 hat wenn die beiden Zustände entsprechend zusammenhängen und 0 sonst. Deren Mittelwert, gebildet mit sequentiell erzeugten Zufallszahlen, ist 1, während er integriert über die erstrebte unabhängige Produktverteilung winzig wäre. Die Integrität von Zufallszahlen ist also relativ zu (Klassen von) Anwendungen zu sehen. Die Verwendung von mehreren prinzipiell verschiedenen Typen ist eine gute Vorsichtsmaßnahme gegen Empfindlichkeit auf Korrelationen des einen oder anderen Verfahrens.

In [6] hat Lüscher mit dem MZ Generator zufällige Folgen von Punkten auf einem Torus in 24 Dimensionen erzeugt, also Tupel $(I_0, I_1, \ldots, I_{23})$ deren Koordinaten modulo m identifiziert sind, was wohl mit dem Bildungsgesetz von MZ in Zusammenhang steht. Nach Definition eines Abstandsbegriffes auf dem Torus, kann man nun das Auseinanderstreben von benachbarten Punkten unter Iteration des Generators betrachten. Dabei ergibt sich ein Liapunov Exponent nahe bei 1, der sich sogar analytisch berechnen (und numerisch prüfen) läßt. Er charakterisiert das Abfallen dieses Typs von Korrelationen. Daraus wird für MZ geschlossen, daß man für kritische Anwendungen nach Verwendung von 24 Zahlen die nächsten $24n$ erzeugt aber ignoriert, derart daß $\exp(-n) \ll 1$ gilt. Wie aufwendig diese Maßnahme ist, hängt damit zusammen, welchen Anteil an der Rechenzeit die Erzeugung von Zufallszahlen beim jeweiligen Problem hat.

5 Perkolation

Bei der Perkolation handelt es sich darum, daß Systeme, die aus lokal zufälligen Elementen aufgebaut werden, unter bestimmten Umständen zu "Clustern" zusammenklumpen, die einen langreichweitigen Zusammenhang herstellen können.

Dieser Mechanismus spielt eine entscheidende Rolle bei so verschiedenen Phänomenen wie der elektrischen Leitfähigkeit von Legierungen, der Ausbreitung von Epidemien und Waldbränden oder der Ergiebigkeit von Erdölfeldern.

In diesem Kapitel wollen wir uns der Perkolation von der Seite der Computersimulation nähern, die umfangreichen theoretischen Konzepte würden eine eigene Vorlesung füllen. Eine detailierte Einführung in Theorie und Numerik geben Stauffer und Aharony[9].

5.1 Typen von Perkolationsproblemen

Das einfachste Perkolationsproblem entsteht, indem man die Punkte eines Gitters unabhängig voneinander mit einer gewissen Wahrscheinlichkeit p "besetzt" und dann nach Clustern fragt, die die besetzten Punkte über ihre Nachbarschaftsbeziehungen bilden. Hier spricht man von *Punktperkolation* (site percolation). Die Gitter können verschiedene Dimensionen und Symmetrien (kubisch, dreieckig etc.) haben.

Man kann aber auch die Verbindungslinien (bonds) der Gitterpunkte direkt (zufällig und unabhängig voneinander) aktivieren und dann nach den Clustern der so verbundenen Punkte fragen (*Bondperkolation*).

Schließlich kann man auch auf das Gitter verzichten und z.B. Scheiben, Kugeln o.ä. betrachten, die durch Überlappung zusammenhängen.

In allen Fällen untersucht man, ob bei einem bestimmten Parameterwert von p Perkolation einsetzt. Wenn das geschieht, dann findet man Phänomene wie an kritischen Punkten (Phasenübergängen zweiter Ordnung): langreichweitige Korrelationen, Skalierungsgesetze mit universellen Exponenten usw. Es handelt sich hier aber nicht um Systeme, wie man sie sonst aus der Statistischen Physik kennt, denn es gibt keine Wahrscheinlichkeits–Beschreibung, die durch einen Temperaturparameter gekennzeichnet wäre. Man spricht deshalb eher von "geometrischen" Phasenübergängen.

5.2 Einstieg in die Numerik

Das folgende Hauptprogramm belegt die Punkte eines quadratischen Gitters der Größe $L \times L$ (mit freien Rändern) mit Wahrscheinlichkeit p.

Zunächst wird über die Besetzung entschieden: `feld = -1` markiert leere und `feld = 0` besetzte Punkte. Dann wird ein Analyse–Programm gerufen, das die "0"–Einträge in `feld` mit einer Cluster–Nummerierung überschreibt. Zwei Verfahren dazu werden in den nächsten Abschnitten besprochen.

```
/*    file punkt_perk.c
      Punktperkolation */

#include <stdlib.h>
#define  L 10           /* Gittergroesse: L x L */
#include "baum_analyse.c"
#include "hoshen_kopelman.c"

void srand48(long seed);
double drand48(void);

int main(void){

long feld[L][L];  /* Status der Gitterpunkte:
                             feld = -1 : unbesetzt
                                  =  0 : besetzt
                                  >  0 : Clusternummer */
long x, y;        /* Koordinaten 0..L-1 */
float p = .4;     /* Punktwahrscheinlichkeit */
long  seed = 137; /* Seed fuer drand48 */

srand48(seed);      /* Initialisierung von drand48 */

for (x = 0; x < L; x++)
   {
   for (y = 0; y < L; y++)
      {
      feld[x][y] = -1;
      if (drand48() < p) feld[x][y] = 0;  /* Punkte besetzen */
```

```
      }
   }

for (y = L-1; y >= 0; y--)
   {
   printf("\n");
   for (x = 0; x < L; x++)
      printf("%4i",feld[x][y]);     /* Besetzung ausdrucken */
   }
printf("\n");

 baum_analyse(feld);                  /* Cluster identifizieren */
/* hoshen_kopelman(feld); */                /* Cluster identifizieren */

for (y = L-1; y >= 0; y--)
   {
   printf("\n");
   for (x = 0; x < L; x++)
      printf("%4i",feld[x][y]);     /* Clusterzerlegung ausdrucken */
   }
printf("\n");

}     /* main */
```

Die Ausgangskonfiguration eines 10×10-Gitters, dessen Punkte mit Wahrscheinlichkeit $p = 0.4$ besetzt sind, sieht z.B. so aus:

```
  -1  -1  -1  -1  -1  -1  -1   0  -1  -1
  -1   0   0  -1  -1   0   0  -1  -1  -1
  -1   0   0  -1  -1  -1  -1  -1  -1  -1
  -1   0   0  -1  -1  -1   0   0  -1  -1
   0   0   0   0  -1  -1   0   0   0  -1
  -1  -1  -1   0  -1   0   0  -1  -1  -1
  -1  -1  -1   0  -1  -1   0   0   0  -1
   0  -1  -1  -1  -1  -1   0   0  -1  -1
   0  -1  -1   0   0  -1  -1  -1  -1  -1
  -1  -1   0  -1  -1  -1  -1   0   0   0
```

```
-1  -1  -1  -1  -1  -1  -1   8  -1  -1
-1   6   6  -1  -1   7   7  -1  -1  -1
-1   6   6  -1  -1  -1  -1  -1  -1  -1
-1   6   6  -1  -1  -1   5   5  -1  -1
 6   6   6   6  -1  -1   5   5   5  -1
-1  -1  -1   6  -1   5   5  -1  -1  -1
-1  -1  -1   6  -1  -1   5   5   5  -1
 3  -1  -1  -1  -1  -1   5   5  -1  -1
 3  -1  -1   4   4  -1  -1  -1  -1  -1
-1  -1   1  -1  -1  -1  -1   2   2   2
```

Als erstes sehen wir die Belegung der Gitterpunkte und danach das Resultat der Clusteranalyse. Es wurden insgesamt 8 Cluster gefunden, Nr.5 und Nr. 6 sind mit je 12 Punkten am größten, die anderen umfassen nur wenige Punkte, teilweise sogar nur einen einzigen.

Zum Schluß noch eine C-technische Bemerkung: wir finden den Aufruf `baum_analyse(feld)`, wobei vom Unterprogramm in das im Hauptprogramm definierte Feld `feld` geschrieben wird. Bei skalaren Variablen haben wir betont, daß es dazu nötig ist, eine Adresse zu übergeben und im Untergrogramm mit einem pointer zu arbeiten, da sonst nur eine Kopie übergeben wird. Der Name eine Feldes steht nun automatisch immer für Adresse bzw. Pointer in der Funktion. Felder werden also nicht kopiert.

5.3 Clusterkonstruktion durch Baumsuche

Hier laufen wir in lexikografischer Ordnung (Schreibmaschine, allerdings Zeilen von unten nach oben) über alle Gitterpunkte in der x-y Ebene. Findet man eine Null in `feld`, so ist dieser Punkt belegt und gehört noch keinem bereits mit einem Index gekennzeichneten cluster an. Man wählt einen neuen noch nicht vergebenen Index und heftet ihn diesem Punkt und allen durch belegete Nachbarpaare im Sinne der Punktperkolation mit ihm verbundenen Punkten an. Der Name des Verfahrens stammt daher, dass man jedes Cluster sofort komplett konstruiert, indem man bis in alle Verästelungen ($\rightarrow$ "Baum") seiner Struktur vordringt.

```
/*    file baum_analyse.c
      Punkt-Perkolation:
      Cluster-Analyse mit Baumsuche */
```

```
void baum_analyse(long feld[L][L]){

long xliste[L*L], yliste[L*L];      /* Arbeitslisten fuer Clustersuche */
long xa, ya;                        /* Anfangspunkt eines Clusters */
long cluster;                       /* Cluster-Nummer */
long n;                             /* aktuelle Clustergroesse */
long i;                             /* Arbeitspunkt in den Listen */
long x, y;                          /* aktueller Punkt */

cluster = 0;

for (ya = 0; ya < L; ya++)
   {
   for (xa = 0; xa < L; xa++)
      {
      if (feld[xa][ya] == 0)
         {
         cluster ++;             /* naechstes Cluster anfangen */
         feld[xa][ya] = cluster;
         xliste[0] = xa; yliste[0] = ya;
         n = 0; i = -1;

         while (i < n)
            {
            i++; x = xliste[i]; y = yliste[i];  /* Arbeitspunkt */

            if (x < L-1)               /* rechter Nachbar */
               { if (feld[x+1][y] == 0 )
                  {
                  feld[x+1][y] = cluster;
                  n++; xliste[n] = x+1; yliste[n] = y;
                  }
               }

            if (x > 0)                 /* linker Nachbar */
               { if (feld[x-1][y] == 0 )
                  {
```

```
                  feld[x-1][y] = cluster;
                  n++; xliste[n] = x-1; yliste[n] = y;
                  }
               }

            if (y < L-1)               /* oberer Nachbar */
               { if (feld[x][y+1] == 0 )
                  {
                  feld[x][y+1] = cluster;
                  n++; xliste[n] = x; yliste[n] = y+1;
                  }
               }

            if (y > 0)                 /* unterer Nachbar */
               { if (feld[x][y-1] == 0 )
                  {
                  feld[x][y-1] = cluster;
                  n++; xliste[n] = x; yliste[n] = y-1;
                  }
               }

            }     /* while beendet: Cluster ist komplett */
            /* printf("cluster # %i : Groesse = %i\n",cluster,n+1); */
         }
      }
   }
}     /* baum_analyse */
```

5.4 Clusterzerlegung nach Hoshen–Kopelman

Wenn man von unten links anfängt, reihenweise vorgeht und bei jedem neuen Fragment eines Clusters eine neue Nummer vergibt, dann trifft man in der fünften Reihe unseres Beispiels auf das Problem, die beiden bislang als "5" und "7" nummerierten Fragmente verbinden zu müssen:

```
                  .   .   .
  -1  -1  -1   6  -1   7   ?
```

```
 -1  -1  -1   6  -1  -1   5   5   5  -1
  3  -1  -1  -1  -1  -1   5   5  -1  -1
  3  -1  -1   4   4  -1  -1  -1  -1  -1
 -1  -1   1  -1  -1  -1  -1   2   2   2
```

Man geht nun so vor, daß man dem Verbindungspunkt die kleinere der beiden Zahlen zuweist und in einem zusätzlichen Indexfeld vermerkt, daß "7" ein Teil von "5" ist: `label[7] = 5`. Das ergibt vorläufig

```
                 .   .   .
 -1  -1  -1   6  -1   7   5  -1  -1  -1
 -1  -1  -1   6  -1  -1   5   5   5  -1
  3  -1  -1  -1  -1  -1   5   5  -1  -1
  3  -1  -1   4   4  -1  -1  -1  -1  -1
 -1  -1   1  -1  -1  -1  -1   2   2   2
```

"7" gilt fortan als "schlecht", die anderen Indizes, die (noch) nicht an ein anderes Clusterfragment angeschlossen wurden, erkennt man an `label[i] = i` und nennt sie (vorläufig) "gute" Indizes. Es kann auch vorkommen, daß die Abbildung `label` erst nach mehreren Schritten auf eine gute Zahl führt. Um die Verschachtelung nicht zu tief werden zu lassen, hat es sich bewährt, neue Punkte nicht direkt mit den Zahlen der Nachbarpunkte zu verbinden, sondern, wenn diese schlecht sind, sich erst zum guten Index durchzufragen. (Das betrifft nur den unteren Nachbarn, der linke ist immer gut indiziert, weil er ja gerade eben erst gesetzt wurde.)

In einem (einfachen) weiteren Durchlauf werden schließlich alle besetzten Punkte mit ihrem (endgültig) guten Index belegt.

Das folgende C–Programm implementiert dieses Verfahren. Es sei noch angemerkt, daß die unbesetzten Punkte hier nicht mit -1, sondern vorübergehend mit L^2+1 markiert werden, das ist größer als alle Clusterindizes und erleichtert das Setzen neuer Label auf das Minimum der Nachbarn.

```
/*    file hoshen_kopelman.c
      Punkt-Perkolation:
      Cluster-Analyse nach Hoshen-Kopelman */

void hoshen_kopelman(long feld[L][L]){

long label[L*L+2];                  /* vorlaeufige -> endgueltige Num. */
```

```
long x, y;                          /* aktueller Punkt */
long neu;                           /* neue Clusternummer */
long links, unten;                  /* vorlaeufige Label der Nachbarn */
long LEER;                          /* Label fuer Leerstellen */
long n;

LEER = L*L+1; label[LEER] = LEER;
neu = 1;

for (y = 0; y < L; y++)
   {
   for (x = 0; x < L; x++)
      {
      if (feld[x][y] == -1)         /* leerer Punkt */
         feld[x][y] = LEER;
      else                          /* besetzter Punkt */
         {
         links = LEER;              /* linker Nachbar */
         if (x > 0) links = feld[x-1][y];

         unten = LEER;              /* unterer Nachbar */
         if (y > 0)
            {
            unten = feld[x][y-1];
            while (label[unten] < unten) unten = label[unten];
            }

         if (links == LEER  &&  unten == LEER)
            {
            feld[x][y] = neu;
            label[neu] = neu; neu++;   /* neue Nummer vergeben */
            }
         else if (links < unten)
            {
            feld[x][y] = links;
            if (unten < LEER) label[unten] = links; /* Cluster verbinden */
            }
         else
```

```
            {
            feld[x][y] = unten;
            if (links < LEER) label[links] = unten; /* Cluster verbinden */
            }
         }
      }
   }        /* vorlaeufige Nummerierung fertig */

for (y = 0; y < L; y++)
   {
   for (x = 0; x < L; x++)
      {
      n = feld[x][y];
      while (label[n] < n) n = label[n];  /* gutes Label finden */
      feld[x][y] = n;
      if (n == LEER) feld[x][y] = -1;     /* Leerstellen -> -1 */
      }
   }

}     /* hoshen_kopelman */
```

Das Ergebnis sieht insgesamt so aus:

```
 -1  -1  -1  -1  -1  -1  -1  10  -1  -1
 -1   6   6  -1  -1   9   9  -1  -1  -1
 -1   6   6  -1  -1  -1  -1  -1  -1  -1
 -1   6   6  -1  -1  -1   5   5  -1  -1
  6   6   6   6  -1  -1   5   5   5  -1
 -1  -1  -1   6  -1   5   5  -1  -1  -1
 -1  -1  -1   6  -1  -1   5   5   5  -1
  3  -1  -1  -1  -1  -1   5   5  -1  -1
  3  -1  -1   4   4  -1  -1  -1  -1  -1
 -1  -1   1  -1  -1  -1  -1   2   2   2
```

Man erkennt (natürlich) dieselbe Clusterstruktur wie zuvor, nur ist die Nummerierung jetzt anders — die Indexmanipulationen haben Lücken hinterlassen.

Das Hoshen–Kopelman–Verfahren hat gegenüber der obigen Baumsuche den Vorteil, die Systempunkte nicht in unregelmäßiger Reihenfolge, sondern

in zwei systematischen Durchgängen zu bearbeiten, der Datenzugriff ist deshalb schneller. Das war insbesondere von Bedeutung, als man sehr große Systeme noch nicht im Hauptspeicher unterbringen konnte (sondern z.B. nur auf Magnetbändern!). Es sind aber auch weniger Anfragen an die Nachbarpunkte nötig, weil man jeden Bond nur in einer Richtung ansieht.

Von Nachteil ist, daß kompliziertere geometrische Eigenschaften der Cluster (z.B. die lineare Ausdehnung) nicht unmittelbar ablesbar, sondern nur über eine weitere Durchmusterung zu erhalten sind.

Was den Rechenaufwand in Abhängigkeit von der Systemgröße anbelangt, so ist bei der Baumsuche klar, daß er proportional zum Volumen ist, denn jeder Punkt wird nur einmal aus jeder Richtung berührt (und einmal bei der Suche nach einem neuen Clusteranfang). Beim Hoshen–Kopelman–Verfahren sind dagegen pathologische Fälle möglich, wo die wachsende Tiefe der Index–Rekursionen einen Aufwand verursacht, der schneller wächst als das Volumen. Das scheint aber in realistischen Anwendungen nicht zu passieren.

5.5 Charakteristische Größen der Clusterzerlegung

Am wichtigsten ist natürlich die Frage, ob in einem gegebenen System mit Besetzungswahrscheinlichkeit p Perkolation auftritt, anschaulich: ob ein Cluster das "gesamte" System durchdringt. Wenn dies für $p \geq p_c$ der Fall ist, wird p_c als *Perkolationsschwelle* oder *kritischer Punkt* bezeichnet. Wie kann man das genauer definieren?

Als Maß für die *Größe* eines einzelnen Clusters nimmt man die Anzahl s seiner Punkte ($s \geq 1$). Um weiterhin korrekte Größen zu definieren, verschieben wir den Übergang zum unendlichen System auf später und nehmen die Anzahl der Gitterpunkte zunächst als endlich an. Die mittlere Anzahl *pro Gitterpunkt* der Cluster mit Größe s wird mit n_s bezeichnet. Damit wird bei Punktperkolation

$$\sum_s s n_s = p, \tag{5.1}$$

denn das ist gerade die Wahrscheinlichkeit dafür, daß ein Punkt zu irgendeinem Cluster gehört, also besetzt ist. Wenn wir nun zufällig einen besetzten Punkt auswählen, dann werden wir mit Wahrscheinlichkeit $s n_s / \sum_{s'} s' n_{s'}$ finden, daß er zu einem Cluster der Größe s gehört, und so können wir eine *mittlere Clustergröße*

$$S(p) = \frac{\sum_s s^2 n_s}{\sum_s s n_s} \tag{5.2}$$

definieren.

Nun kann die *Perkolationsschwelle* p_c definiert werden als

$$\lim_{L\to\infty} S(p) = \infty \quad \text{für} \quad p \geq p_c, \tag{5.3}$$

denn der "unendliche" Cluster läßt S divergieren.

Um Größen definieren zu können, die auch für $p \geq p_c$ einen Grenzwert bei $L \to \infty$ haben, geht man folgendermaßen vor: man definiert einen *perkolierenden* oder *überbrückenden* Cluster (spanning cluster) durch die Eigenschaft, daß er entweder
(i) zwei gegenüberliegende Ränder verbindet, oder
(ii) alle Ränder miteinander verbindet.
Die letztere Bedingung ist offenbar stärker. (i) ist leichter zu verifizieren und wird deshalb meist benutzt, hat aber den Nachteil, daß es prinzipiell mehrere solche Cluster nebeneinander geben könnte; die Wahrscheinlichkeit dafür ist aber z.B. in einem großen, quadratischen System winzig klein.

Nun definiert man endgültig

$$S(p) = \frac{\sum_s' s^2 n_s}{\sum_s' s n_s}, \tag{5.4}$$

wobei in $\sum_s'$ der Beitrag des perkolierenden Clusters ggf. wegzulassen ist. Diese Definition ist auch bei $L \to \infty$ sinnvoll und gibt allgemein die mittlere Größe der endlichen Cluster an.

Ferner sei $P_\infty(p)$ die Wahrscheinlichkeit dafür, daß ein besetzter Punkt zum perkolierenden Cluster gehört (wenn es existiert, sonst $P_\infty = 0$). Im unendlichen System erwartet man $P_\infty(p) = 0$ für $p < p_c$ und ein Anwachsen $P_\infty(p) \to 1$ bei $p \to 1$. Hier handelt es sich also um den *Ordnungsparameter* der Perkolation (vergleichbar z.B. mit der Magnetisierung eines Ferromagneten ober– und unterhalb der Curie–Temperatur).

Die Clustergröße war oben durch die Anzahl der Punkte definiert, was keine direkte Aussage über die räumliche Ausdehnung darstellt. Deshalb steht die Einführung einer Längenskala noch aus. Dazu können wir vom "Trägheitsradius" R_s des Clusters s ausgehen: es werde von den Punkten $x_i, i = 1 \ldots s$ gebildet, dann liegt sein "Schwerpunkt" bei

$$\vec{X} = \frac{1}{s} \sum_i \vec{x}_i, \tag{5.5}$$

es wird

$$R_s^2 = \frac{1}{s}\sum_i (\vec{x}_i - \vec{X})^2, \tag{5.6}$$

und durch Mittelung über alle *endlichen* Cluster entsteht die *Korrelationslänge*

$$\xi(p) = \left(\overline{R_s^2}\right)^{1/2}. \tag{5.7}$$

Eine andere mögliche Definition zieht die *Korrelationsfunktion* $G(r)$ heran, das ist die Wahrscheinlichkeit dafür, daß zwei Punkte im Abstand r zum selben Cluster gehören. An der Asymptotik kann man ξ ablesen:

$$G(r) \sim \exp(-r/\xi) \quad r \to \infty. \tag{5.8}$$

Die Übereinstimmung beider Definitionen folgt aus einer längeren Rechnung.

5.6 Exakte Lösung des eindimensionalen Problems

Wie wir sehen werden, gibt es in einer eindimensionalen Kette bei Punkt– oder Bondwahrscheinlichkeit $p < 1$ keine Perkolation. Trotzdem ist es nützlich, die Perkolation in einer Dimension zu diskutieren, denn man findet leicht exakte Ausdrücke für viele interessante Größen.

Wir betrachten beispielsweise Punktperkolation in einer linearen Kette der Länge $L \to \infty$. Cluster kann man hier ohne Mühe als Blöcke besetzter Punkte identifizieren, die durch leere Punkte getrennt sind.

Die Wahrscheinlichkeit dafür, daß ein bestimmter Abschnitt der Kette ein Cluster der Größe s bildet, beträgt $p^s(1-p)^2$. Wir haben einen Faktor p für jeden der besetzten Clusterpunkte und je einen $(1-p)$ für die angrenzenden leeren Punkte. Dieses Cluster kann L verschiedene Positionen einnehmen, wobei wir Randeffekte der (relativen) Größe s/L im Hinblick auf den Limes $L \to \infty$ vernachlässigen. Wenn wir im gleichen Sinne die Überlappung vernachlässigen, dann gibt es im Mittel $Lp^s(1-p)^2$ s–Cluster im System der Größe L und demnach ist

$$n_s(p) = p^s(1-p)^2. \tag{5.9}$$

Die Relation (5.1) kann man leicht nachrechnen:

$$\sum_s s n_s(p) \quad = \quad (1-p)^2 \sum_s s p^s$$

$$\begin{aligned} &= (1-p)^2 p\partial_p \sum_s p^s \\ &= (1-p)^2 p\partial_p \frac{p}{1-p} \\ &= p. \end{aligned}$$

Mit demselben Trick erhält man

$$\begin{aligned} \sum_s s^2 n_s(p) &= (1-p)^2 (p\partial_p)^2 \sum_s p^s \\ &= p\frac{1+p}{1-p} \end{aligned}$$

und damit

$$S(p) = \frac{1+p}{1-p}. \tag{5.10}$$

Man erkennt $S(p) = \infty$ bei $p = 1$, d.h.

$$p_c = 1. \tag{5.11}$$

Die Korrelationslänge bekommt man mit folgender Überlegung: damit zwei Punkte im Abstand r zum selben Cluster gehören, müssen sie mit allen dazwischen liegenden Punkten besetzt sein, was eine Wahrscheinlichkeit p^{r+1} hat. Die Korrelationsfunktion verhält sich also wie

$$G(r) \sim \exp(-r/\xi) \tag{5.12}$$

mit

$$\xi = -\frac{1}{\log p}. \tag{5.13}$$

5.7 Skalengesetze im unendlichen System

Beim Durchgang durch den Perkolationspunkt verhalten sich die Observablen des Systems — zu $L \to = \infty$ extrapoliert — in auffälliger Weise, nämlich nicht–analytisch. Der Ordnungsparameter P_∞, der zuvor identisch null war, beginnt zu wachsen, die mittlere Clustergröße S und die Korrelationslänge ξ durchlaufen eine Singularität. Man versucht häufig, so auch bei der Perkolation (und bei anderen Phasenübergängen höherer Ordnung), dieses Verhalten durch Skalengesetze mit *kritischen Exponenten* zu parametrisieren:

$$P_\infty \sim (p-p_c)^\beta \tag{5.14}$$

$$S(p) \sim |p-p_c|^{-\gamma} \tag{5.15}$$

$$\xi(p) \sim |p-p_c|^{-\nu} \tag{5.16}$$

Die Theorie dieser kritischen Phänomene ist schwierig: Stichwort "Renormierungsgruppe". In der Praxis helfen oft nur numerische Simulationen.

In einer Dimension haben wir explizite Lösungen: an (5.10) liest man $\gamma = 1$ ab. Aus (5.12) folgt wegen $-\log p \simeq (1-p)$ bei $p \to p_c = 1$: $\nu = 1$. β ist hier wegen $p_c = 1$ nicht (direkt) bestimmbar.

Für Punktperkolation in zwei Dimensionen sind die Exponenten auch exakt bekannt:

$$\beta = 5/36 \tag{5.17}$$
$$\gamma = 43/18 \tag{5.18}$$
$$\nu = 4/3 \tag{5.19}$$
$$\tag{5.20}$$

Allgemein gilt in d Dimensionen die Relation

$$2\beta + \gamma = \nu d \tag{5.21}$$

(hyperscaling).

5.8 Skalierung mit der Systemgröße

Die Bestimmung der kritischen Exponenten aus numerischen Simulationen ist schwierig, weil in den endlichen Systemen das obige Verhalten nur gilt, solange $\xi \ll L$, d.h. in gebührender Entfernung vom Perkolationspunkt. Wenn man bei endlichem L durch $p = p_c$ hindurchläuft, erscheinen die Singularitäten "gerundet".

Man kann aber aus der Not eine Tugend machen, direkt *am kritischen Punkt* arbeiten (wo $\xi = \infty$) und die Systemgröße als Längenskala ansehen. Bei Vergrößerung von L ergeben sich dann neue Skalengesetze (*finite size scaling*, FSS), deren Exponenten mit denen des *unendlichen* Systems *in der Umgebung* von p_c zusammenhängen.

Ein (zu) einfaches Argument soll zeigen, wie die neuen Ansätze aussehen: In einer kleinen Umgebung des Perkolationspunkts, wo $\xi > L$, übernimmt L die Rolle der charakteristischen Länge, d.h. man setzt

$$\xi \approx L \sim |p - p_c|^{-\nu} \tag{5.22}$$

und eliminert damit $(p-p_c)$ aus den Skalengesetzen der anderen Observablen:

$$P_\infty(p_c) \sim L^{-\beta/\nu} \tag{5.23}$$
$$S(p_c) \sim L^{\gamma/\nu} \tag{5.24}$$

als asymptotisches Verhalten bei $L \to \infty$ am kritischen Punkt. Aus Messungen von $P_\infty(p_c)$ und $S(p_c)$ an Systemen von wachsendem (aber endlichem!) L kann man also die Verhältnisse von Exponenten β/ν bzw. γ/ν gewinnen.

Dies ist für numerische Simulationen sehr praktikabel. Die Bestimmung von β/ν und γ/ν am Perkolationspunkt des zweidimensionalen quadratischen Gitters ($p_c = 0.5927$) wird in einer Übungsaufgabe behandelt.

6 Monte Carlo Simulation im Ising Modell

In diesem Abschnitt befassen wir uns mit der Monte Carlo Simulation von statistischen Systemen am Beispiel des Ising Modells. Monte Carlo Simulation ist eine zentrale Methode in der statistischen und Festkörper Physik und in einem Zweig der theoretischen Elementarteilchen Physik, der Gitter Feldtheorie. Sie spielt eine zentrale Rolle im Forschungsprogramm des hiesigen Lehrstuhls.

6.1 Das Ising Modell auf einem kubischen Gitter

Zunächst soll das Ising Modell definiert werden [10]. Es ist ein einfaches statistisches System und damit ein natürlicher Startpunkt. Eine mögliche Motivation ergibt sich als stark vereinfachtes Modell eines Ferromagneten. Wir stellen uns ein einfach kubisches Kristallgitter vor, das wir allerdings verallgemeinernd nicht nur in 3 sondern in D Dimensionen betrachten wollen. Gitterplätze (lattice sites) sind durch D ganzzahlige Koordinaten gegeben:

$$\text{Gitterplatz} = x = (x_0, x_1, \ldots, x_{D-1}),\ x_\mu \in Z \tag{6.1}$$

Für thermodynamische Definitionen und natürlich zur Simulation brauchen wir zunächst ein endliches Volumen, obwohl am Ende oft der Grenzübergang zum unendlichen steht. Dazu schränken wir die Koordinaten ein[13],

$$0 \le x_\mu < L,\ V = L^D, \tag{6.2}$$

wobei hier die Anzahl V der so gegebenen Punkte das Volumen ist. Das bezieht sich auf Gittereinheiten, in denen die Elementarzelle das Volumen Eins besitzt.

Ferromagnetische Effekte in Festkörpern stammen von Elektronen Spins. Im Modell werden diese durch Freiheitsgrade $s(x) \in \{-1, +1\}$ dargestellt, die zwei möglichen Spin Ausrichtungen pro Gitterplatz entsprechen. Diese Diskretheit ist offenbar in der zugrundeliegenden Quantentheorie begründet. Eine Konfiguration $s \hat{=} \{s(x)\}$ für alle V Gitterplätze spezifiziert nun im Modell genau einen Zustand des Systems; sein Phasenraum besteht also aus 2^V diskreten solchen möglichen Konfigurationen. In der statistischen Physik gibt man die exakte Beschreibung der Dynamik solcher Systeme auf und macht

[13] Man könnte die Kantenlängen auch richtungsabhängig nehmen

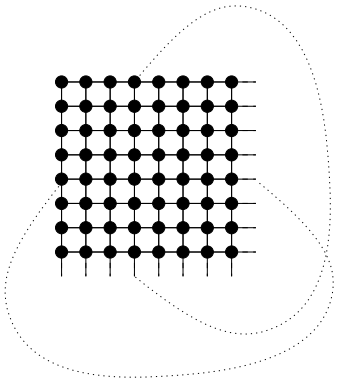

Abbildung 13: Gitter für Ising Modell in $D = 2$.

Wahrscheinlichkeitsaussagen. Das Wunder ist, daß dies für den Bereich, wo thermodynamisches Gleichgewicht gilt, auf einfache Weise möglich ist, wenn man nur die Energie des Systems für jede Konfiguration kennt. Beim Ising Modell lautet diese

$$-H(s) = \epsilon \sum_{\langle xy \rangle} s(x)s(y) + B \sum_x s(x) \tag{6.3}$$

Die erste Summe läuft über alle Kanten (links) im Gitter und $s(x), s(y)$ sind die beiden Spins an den Enden. Der Parameter ϵ, positiv für Ferromagneten, steuert eine energetische Bevorzugung von parallelen gegenüber antiparallelen Spins. Der zweite Term beschreibt den Einfluß eines magnetischen Feldes (reell, skalar, entlang der Quantisierungsachse).

Jeder Punkt im Inneren des Gitters partizipiert offenbar an $2D$ solchen Kanten. Am Rand sind es i. a. weniger, und wir brauchen zusätzliche Definitionen, was zu tun ist. Die Situation ist für zwei Dimensionen in Abb.13 verdeutlicht. Läßt man die am Rand herausragenden Kanten bzw. ihre Energieterme einfach weg, so nennt man dies freie Randbedingungen. Identifiziert man sie, wie mit den punktierten Linien in zwei Fällen angedeutet, alle mit den Punkten am gegenüberliegenden Rand, so hat man periodische Randbedingungen und das Gitter ist ein Torus. Dies ist die Standardwahl. Sie hat eine besonders hohe Symmetrie, da alle Punkte gleichwertig sind. Auch hat man Verschiebungsinvarianz um Vielfache der Kantenlänge.

Wenn wir dies nun noch in den Koordinaten formalisieren, auch im Hinblick auf Programmierung, so sehen wir gleich, daß das Konzept sich problem-

los auf D Dimensionen verallgemeinern läßt. Man führt Einheitsvektoren für die Gitterrichtungen ein,

$$\hat{\mu} = (0, \ldots, 0, \underbrace{1}_{\mu-\text{Komponente}}, 0, \ldots, 0),\ 0 \le \mu \le D-1. \tag{6.4}$$

Wenn man nun die Addition von Torus Koordinaten immer modulo L versteht, so kann man den ersten Energieterm in (6.3) als

$$\sum_{x,\mu} s(x)s(x+\hat{\mu}) \tag{6.5}$$

schreiben. Die Summe geht nun also über $V \times D$ völlig gleichwertige Kanten.

Zum Programmieren mit Feldern auf dem Torus, ist es sicher naheliegend jeweils Felder mit D Indizes einzuführen. Eleganter und klarer (= weniger Fehler) wird es jedoch, insbesondere für mehrere bzw. größere D, wie folgt. Wir numerieren alle Punkte x eindeutig durch ganze Zahlen,

$$n_x = x_0 + x_1 L + x_2 L^2 + \ldots + x_{D-1} L^{D-1},\ 0 \le n_x < V \tag{6.6}$$

und deklarieren als Konfiguration für alle D ein Feld mit nur einem Index. Zur Berechnung der Indizes der $2D$ Nachbarn zu n_x ist einige Rechnung erforderlich. Diese führt man nur einmal durch und speichert die Nachbarindizes in einem ganzzahligen Feld der Größe $V \times 2D$. Auf diese Weise gelangt man zu kompakten Programmen, in denen man D tatsächlich als veränderbaren Parameter hat. Selbstverständlich ist diese Strategie recht großzügig mit Speicherplatz, was auf modernen Rechnern aber akzeptabel ist in den meisten Fällen.

6.2 Observable im statistischen Gleichgewicht

Es ist ein Resultat der statistischen Mechanik bzw. Thermodynamik, daß man unter bestimmten Bedingungen eine korrekte und vorhersagekräftige statistische Beschreibung der Physik erhält, wenn man postuliert, daß ein System mit einer Wahrscheinlichkeit proportional zu $\exp(-\beta H)$ in allen seinen Zuständen sein kann. Hier ist β ein Parameter der das Ensemble vollständig festlegt und der inversen Temperatur entspricht. Speziell für das Ising Modell ist der Erwartungswert einer beliebigen Funktion der Konfigurationen (Observable) $\mathcal{A}(s)$ gegeben durch

$$\langle \mathcal{A} \rangle = \frac{1}{Z} \sum_s \mathcal{A}(s)\, \mathrm{e}^{-\beta H(s)} \tag{6.7}$$

mit der normierenden Zustandssumme

$$Z = \sum_s \mathrm{e}^{-\beta H}. \tag{6.8}$$

Beide Summen gehen über alle 2^V möglichen Konfigurationen der V Spins.

Eine offensichtliche mögliche Observable ist die mittlere (innere) Energie,

$$E = \langle H \rangle = -\frac{\partial \ln Z}{\partial \beta}. \tag{6.9}$$

Durch wiederholtes Differenzieren nach dem Feld bekommt man Potenzen der mittleren Magnetisierung wie z. B.

$$\begin{aligned} M &= \sum_x s(x) \\ \langle M \rangle &= \frac{1}{\beta}\frac{\partial \ln Z}{\partial B} \end{aligned} \tag{6.10}$$

Neben diesen aus der Energie abgeleiteten Obervablen, erhält man detaillierte Informationen aus n-Punkt Korrelationsfunktionen (jedes x^i ist ein Gitterplatz)

$$G^{(n)}(x^1, x^2, \ldots, x^n) = \left\langle s(x^1)s(x^2)\cdots s(x^n) \right\rangle. \tag{6.11}$$

Speziell für die Zweipunktfunktion schreiben wir

$$G(x, y) = \langle s(x)s(y) \rangle. \tag{6.12}$$

Sie beschreibt die Korrelation zwischen zwei möglicherweise weit entfernten Spins. Ist sie z. B. positiv, so gibt es eine Tendenz für diese Spins parallel zu stehen. Wenn dies für beliebig weit entfernte[14] Spins konstant der Fall ist, so liegt spontane Magnetisierung vor, das eigentliche Phänomen Ferromagnetismus. Dies passiert im Ising Modell für $D \geq 2$ und genügend kleine Temperatur (großes β).

[14] Für die Theorie ist dies nach einem Grenzübergang $L \to \infty$ wörtlich gemeint. In der Physik geht es um makroskopische Distanzen (Weiss'sche Bezirke)

6.3 Exakte Lösung in $D = 1$

In nur einer Dimension ist das Problem wie so oft analytisch lösbar. Die Spins bilden einfach eine Kette, die im Falle periodischer Randbedingungen zu einem Ring zusammengebogen ist. Nur der Einfachheit halber spezialisieren wir uns hier auf $B = 0$ und wählen die Einheit der Temperatur so, daß $\epsilon = 1$ gilt. Wir definieren die Transfermatrix durch ihre Matrixelemente

$$T_{\sigma\sigma'} = \exp(\beta\sigma\sigma'). \tag{6.13}$$

Zeilen und Spalten werden hier durch zwei Spins $\sigma, \sigma' = -1, +1$ abgezählt. Dann können wir im periodischen Fall schreiben[15]

$$Z = \sum_{s(0),s(1),\ldots,s(L-1)} T_{s(0)s(1)} T_{s(1)s(2)} \cdots T_{s(L-1)s(0)} = \mathrm{tr}\left[T^L\right]. \tag{6.14}$$

Die Eigenwerte von T sind

$$\lambda_\pm = \exp(\beta) \pm \exp(-\beta) \tag{6.15}$$

mit Eigenvektoren

$$\psi_\pm = \frac{1}{\sqrt{2}}(\pm 1, 1). \tag{6.16}$$

Somit ist

$$Z = \lambda_+^L + \lambda_-^L \approx (2\cosh(\beta))^L, \tag{6.17}$$

wobei die Näherung für hinreichend große Systeme $L \gg \exp(2\beta)$ gilt und im Folgenden angenommen wird. Damit ist die Energie

$$E = -L\tanh(\beta). \tag{6.18}$$

Die Korrelationsfunktion zwischen entfernteren Spins bei $x < y$ ist für $L \gg y - x$ gegeben durch

$$\begin{aligned} \langle s(x)s(y)\rangle &= \frac{\mathrm{tr}\left[ST^{y-x}ST^{L-y+x}\right]}{\mathrm{tr}[T^L]} \\ &= \frac{\left(\lambda_-^{y-x}\lambda_+^{L-y+x}\right)}{\lambda_+^L} = \tanh(\beta)^{y-x}, \end{aligned} \tag{6.19}$$

[15] In einer Dimension gibt es nur eine Komponente $x = x_1$

wobei S analog zu T aber diagonal ist mit Elementen

$$S_{\sigma\sigma'} = \sigma\,\delta_{\sigma\sigma'}, \tag{6.20}$$

und daher

$$S\psi_{\pm} = \psi_{\mp} \tag{6.21}$$

benutzt werden konnte. Die Korrelationen fallen in $D = 1$ also für jede Temperatur exponentiell zu Null ab, es gibt keine spontane (ohne B) Magnetisierung. Dies ist ein allgemeines Resultat in einer Dimension und nicht auf das Ising Modell beschränkt. Die Korrelation von nächsten Nachbarn lautet

$$G(x, x+\hat{\mu}) = \tanh(\beta) \tag{6.22}$$

und ist wegen der Symmetrie des Torus ortsunabhängig. Daher trägt zur Energie jeder link gleich bei und senkt die Energie monton mit β wachsend ab, vgl. (6.18).

Eine exakte Lösung ist auch in $D = 2$ bekannt, wenngleich sie ungleich schwieriger zu bekommen ist. Eine Möglichkeit hierzu ist wieder die Diagonalisierung der Transfermatrix, die allerdings nun 2^L dimensional ist. Dieses Modell mit seiner bekannten Onsager Lösung [10], die einen Phasenübergang zur magnetisierten Phase aufweist, ist sozusagen der "harmonische Oszillator der statistischen Physik". Jedes Konzept wir dort getestet.

6.4 Monte Carlo Simulation am Beispiel Ising Modell

Nun sollen Mittelwerte von Obervablen numerisch berechnet werden. Vollständiges Summieren ist für interessante Fälle indiskutabel. Die meisten Beiträge wären aber bedeutungslos wegen der Unterdrückung durch den Boltzmann Faktor $\exp(-\beta H)$. Hier kommen nun Monte Carlo Verfahren mit importance sampling, die wir bei der MC Integration kennengelernt hatten, zum Tragen.

Der dominante Faktor im Summanden ist offenbar der Boltzmann Faktor, der für alle Konfiguration zwischen $\exp(\pm\beta DV)$ enorm variieren kann. Typische Observable sind beschränkter, z. B. nur ± 1 bei Korrelationen. Daneben kommt es aber auch auf die Anzahl der existierenden Konfiguration mit vergleichbaren Boltzmann Faktoren an (Entropie). Eine gute Idee wäre offenbar, beliebige Konfigurationen zu produzieren mit der Wahrscheinlichkeit

$$P(s) = \frac{1}{Z}\,\mathrm{e}^{-\beta H(s)}. \tag{6.23}$$

Ist $s^1, s^2, \ldots, s^N$ eine (lange) so verteilte Folge, so gilt wie bei der MC Integration

$$\langle \mathcal{A}(s) \rangle = \frac{1}{N} \sum_{i=1}^{N} \mathcal{A}(s^i) \left(1 + \mathrm{O}(N^{-1/2})\right). \tag{6.24}$$

Für Systeme wie das Ising Modell gibt es keine praktikablen direkten Algorithmen die *unabhängige* korrekt verteilte s (independent sampling) produzieren. Statt dessen gibt es Monte Carlo Algorithmen, die die Konfiguration s^i benutzen und modifizieren in s^{i+1}:

$$\ldots s^i \overset{MC}{\to} s^{i+1} \overset{MC}{\to} s^{i+2} \ldots$$

Wenn s^{i+1} nur von s^i abhängt und einige zusätzliche Bedingungen erfüllt sind, spricht man von einer Markov Kette (Markov chain). Diese Verfahren können so gewählt werden, daß bei einer langen Folge asymptotisch die gewünschte Verteilung mit dem Boltzmann Faktor eintritt. Allerdings ist hierbei s^{i+1} eben *nicht unabhängig* von s^i. Gegenüber der früher diskutierten einfachen Monte Carlo Integration hat dies erhebliche Konsequenzen für die erwarteten statistischen Fehler und ihre Schätzung. Außerdem muß man von irgendeiner Konfiguration s^1 starten. Korrekte Monte Carlo Algorithmen führen beweisbar von jeder beliebigen Startkonfiguration asymptotisch zur Boltzmann Verteilung, brauchen dazu aber eine mehr oder weniger große Zahl von Schritten. Der Beginn der Folge wird vom willkürlichen Start abhängen und somit für die Mittelung besser weggelassen. Wieviel, das hängt vom Algorithmus ab und muß untersucht werden. Man nennt in Analogie zur Thermodynamik dieses "Einschwingen" auf typische Konfigurationen Thermalisierung (equilibration).

Der Übergang von einer Konfiguration zur nächsten erfolgt zufällig und ist charakterisiert durch vorgegebene Übergangswahrscheinlichkeiten $W(s^i \to s^{i+1})$, gemäß denen der Prozess unter Benutzung von Zufallszahlen abläuft. Als Wahrscheinlichkeiten erfüllen sie

$$W(s \to s') \geq 0, \; \sum_{s'} W(s \to s') = 1 \tag{6.25}$$

Damit W einen Algorithmus mit den oben postulierten Eigenschaften ergibt, muß gelten

$$\sum_{s} P(s)\, W(s \to s') = P(s') \qquad \text{Stabilität} \tag{6.26}$$

$$W^n(s \to s') > 0 \; \forall s, s' \qquad \text{Ergodizität} \tag{6.27}$$

mit irgendeinem n. Hier ist W^n die Übergangswahrscheinlichkeit bei Hintereinanderausführung entsprechend der Matrix Potenzierung von W. Die zweite Bedingung besagt, daß man in n Schritten prinzipiell von jedem s zu jedem s' gelangen kann. Das ist u. a. notwendig um von jedem s^1 in die relevanten Bereiche des Phasenraumes zu gelangen. Zur Interpretation der ersten Bedingung stellen wir uns ein großes Ensemble von Markov Ketten vor. Wenn nun die i'-ten Konfigurationen dieser vielen Prozesse mit Häufigkeit $P(s)$ verteilt sind, dann gewährleistet die Stabilität von P unter W, daß dies auch für die s^{i+1} gilt. P ist also (Links)eigenvektor von W mit Eigenwert 1. Ein mathematischer Satz für positive normierte Matrizen wie W besagt nun, daß die Beträge aller anderen Eigenwerte echt kleiner sind als 1. Daher führt wiederholte Anwendung von W (MC Iterationen) auf die Verteilung P, andere Komponenten "sterben aus". Die Geschwindigkeit der Thermalisierung hängt vom Abstand zwischen 1 und dem nächstkleineren Eigenwert ab, mit dem die führende unerwünschte Komponente evolviert.

6.5 Lokale Monte Carlo Algorithmen

Wie wir sehen werden, sind viele W konstruierbar, die einem prinzipiell richtigen Algorithmus entsprechen. Es wird sich weiter zeigen, daß Verfahren effektiv sind, wenn sie für einen bestimmten Rechenaufwand von einander möglichst unabhängige s^i produzieren. Zunächst aber müssen die mit W zulässigen Übergänge überhaupt realisierbar sein. Das ist nur der Fall, wenn $W(s \to s') \neq 0$ nur für wenige s' gilt, denn zwischen diesen muß ja gewählt werden. Dies — weniger die Effektivität — ist sehr allgemein erfüllt für lokale MC Verfahren mit Elementarschritten (x fest, beliebig)

$$W_x(s \to s') = \left(\prod_{y \neq x} \delta_{s(y)s'(y)} \right) \; w_x(s(x) \to s'(x)). \tag{6.28}$$

Es wird also nur die Änderung des einen Spins bei x zugelassen, und somit können nur zwei Zustände erreicht werden, $s' = s^{x,+}$ und $s' = s^{x,-}$, wo in der Konfiguration $s^{x,\pm}$ jeweils an der einen modifizierten Stelle $s(x)$ durch ± 1 ersetzt ist und der Rest bleibt. w_x ist nun eine 2×2 Matrix. Aus (6.25) und (6.26) folgt nun leicht, daß

$$\frac{w_x(+ \to -)}{w_x(- \to +)} = \frac{P(s^{x,-})}{P(s^{x,+})} \tag{6.29}$$

gelten muß.

Diese Bedingung wird standardmäßig mit dem Wärmebad oder mit dem Metropolis Algorithmus erfüllt. Beim Wärmebad wird $\sigma = s'(x)$ unabhängig vom alten Spin $s(x)$ gewählt

$$\begin{aligned} w_x(s(x) \to \sigma) &= \frac{P(s^{x,\sigma})}{\sum_{\tau=-,+} P(s^{x,\tau})} \\ &= \frac{\exp[\beta\sigma b(x)]}{\sum_{\tau=-,+} \exp[\beta\tau b(x)]}. \end{aligned} \tag{6.30}$$

Im letzten Schritt ist $b(x)$ das lokale Magnetfeld

$$b(x) = B + \sum_{y=\text{Nachbarn}(x)} s(y). \tag{6.31}$$

Wichtig ist, daß die Wahl eines neuen Spins bei x (update) nur lokale Operationen benötigt, deren Anzahl nicht mit dem Volumen anwächst. Dies liegt an der lokalen Wechselwirkung in $H(s)$. Alle übrigen Energieterme heben sich weg in (6.30).

Bei Metropolis ist der Gesichtspunkt, daß eine Änderung $s \to \tilde{s} = s^{x,-s(x)}$ *vorgeschlagen* wird (Spinflip bei x). Dieser Vorschlag wird akzeptiert mit der Wahrscheinlichkeit

$$\begin{aligned} w_x(s(x) \to -s(x)) &= \min(1, P(\tilde{s})/P(s)) \\ &= \min(1, \exp(-\beta[H(\tilde{s}) - H(s)])) \\ &= \min(1, \exp(-2\beta s(x) b(x))) \end{aligned} \tag{6.32}$$

sonst bleibt es bei s. In beiden Fällen ist leicht nachzuweisen, daß (6.29) gilt.

Ergodizität wird erreicht, wenn man nacheinander lokale Updates bei allen x durchführt,

$$W_{\text{sw}} = W_{x^1} W_{x^2} \cdots W_{x^V}. \tag{6.33}$$

Dabei ist die triviale Tatsache von Bedeutung, daß (6.25) und (6.26) für Produkte gelten, wenn sie für die Faktoren gelten. W_{sw} hängt durchaus von der Reihenfolge der Faktoren ab, da sie für benachbarte x nicht kommutieren. Verschiedene Reihenfolgen liefern i. a. verschiedene korrekte Algorithmen. Diese können sich bezüglich der Unabhängigkeit der s^i durchaus unterscheiden. Ein solcher kompletter Durchgang durch das Gitter heißt Sweep. Er ist

ergodisch und erfordert O(V) Operationen. Man könnte auch ein site x zum update zufällig wählen (random site updating). Dann hätte man

$$W_{\rm rs} = \frac{1}{V}\sum_x W_x, \tag{6.34}$$

und dies alleine wäre ergodisch. Praktisch hat sich dieses Verfahren als nicht vorteilhaft erwiesen. Es liegt wohl daran, daß nach V Schritten (Aufwand etwa wie ein Sweep) einige sites typischerweise kein update bekommen, andere mehrere.

Viele Algorithmen benutzen das detaillierte Gleichgeweicht (detailed balance) als hinreichende (nicht notwendige) Bedingung für Stabilität,

$$P(s)W(s\to s') = P(s')W(s'\to s). \tag{6.35}$$

Stabilität folgt durch Summation über s oder s'. Diese Form hatte auch die Bedingung an das lokale w_x (6.29), so daß detailed balance in diesem Spezialfall auch notwendig war. Dies liegt an den nur zwei Werten der Ising Spins, ist i. a. aber nicht der Fall. Die Hintereinanderausführung von Schritten mit detailed balance liefert i. a. für das Produkt nur noch Stabilität. Alle hier durchgeführten Betrachtungen lassen sich ziemlich direkt verallgemeinern, auf Spins mit mehr und sogar kontinuierlichen Werten, über die dann an jedem site integriert wird.

6.6 Autokorrelation und statistische Fehler

MC Schätzungen a_i für eine Observable A werden aus den mit irgendeinem Algorithmus sukkzessive im Rechner produzierten Konfigurationen s^i abgeleitet,

$$a_i = \mathcal{A}(s^i),\ i=1,\ldots,N, \quad A = \langle \mathcal{A}(s)\rangle. \tag{6.36}$$

Diese aufeinanderfolgenden Schätzwerte sind nicht unabhängig, was die Fehlerabschätzung verkompliziert. Dabei wollen wir hier annehmen, daß die Folge von Schätzwerten erst nach der Thermalisierung beginnt. Bevor Konfiguration s^1 auftritt, wurden also schon so viele Iterationen durchgeführt, daß das System thermalisiert ist und der willkürliche Anfangszustand in beliebig guter Näherung "vergessen" wurde. Dann ist s^1 also bereits eine "typische" Gleichgewichtskonfiguration. Wir werden später einen Hinweis bekommen, wieviele Iterationen eine solche Thermalisierung praktisch benötigt.

Zur Fehlerabschätzung wollen wir wie bei der einfachen MC Integration die mittlere quadratische Abweichung vom exakten Mittelwert bilden,

$$\sigma(N,A)^2 = \left\langle \left(\frac{1}{N}\sum_{i=1}^{N} a_i - A \right)^2 \right\rangle_{\mathrm{MC}} . \tag{6.37}$$

Die Mittelung $\langle\cdots\rangle_{\mathrm{MC}}$ bezieht sich hier auf (im Limes unendlich) viele MC Schätzungen, während wir mit $\langle\cdots\rangle$ weiter das statistische Mittel im Sinn von (6.7) meinen. Für das MC-Mittel kann man sich entweder wieder ein Ensemble von Markov Prozessen im Gleichgewicht vorstellen, oder aber einen sehr langen solchen, aus dem man viele Teilfolgen der Länge N herausgreift, die weit genug voneinander entfernt sind, um als unabhängig gelten zu können.

Durch Ausmultiplizieren ergibt sich

$$\sigma(N,A)^2 = \frac{1}{N^2}\sum_{i,j=1}^{N} \Gamma_A(i-j), \tag{6.38}$$

wobei wir die Autokorrelation

$$\Gamma_A(i-j) = \langle (a_i - A)(a_j - A)\rangle_{\mathrm{MC}} = \Gamma_A(j-i) \tag{6.39}$$

eingeführt haben. Die Notation impliziert nochmals, daß diese Funktion nur vom Abstand zwischen i und j abhängt und nicht mehr vom Abstand zum Beginn der Folge. Die Autokorrelation bei $i=j$ ist die Varianz,

$$\Gamma_A(0) = \langle (a_i - A)^2\rangle_{\mathrm{MC}} = \langle (\mathcal{A}(s) - A)^2\rangle = \mathrm{var}(A). \tag{6.40}$$

Hier wurde benutzt, daß im Gleichgewicht die Konfiguration s^i im MC mit P verteilt ist und so die MC-Mittelung in die statistische des Ising Modells übergeht. Man kann diesen Fall auch statisch nennen, während Mittelungen, die sich auf $i \neq j$ beziehen, dynamisch sind und vom Algorithmus W abhängen.

Wären aufeinanderfolgende Schätzungen unabhängig, so gälte

$$\Gamma_A(i-j) = \Gamma_A(0)\,\delta_{ij} \leftrightarrow \text{unabhängig} \tag{6.41}$$

und damit

$$\sigma(N,A)^2 = \frac{\mathrm{var}(A)}{N} \tag{6.42}$$

in völliger Analogie zur MC Integration.

Üblicherweise würde mit einer solchen Formel der Fehler unterschätzt. Tatsächlich ist Γ_A typischerweise positiv und klingt asymptotisch mit dem Abstand exponentiell ab,

$$\Gamma_A(t) \overset{t\to\infty}{\propto} \exp(-t/\tau), \tag{6.43}$$

aber τ kann viele Sweeps lang sein. Für eine brauchbare MC Rechnung benötigt man auf jeden Fall $N \gg \tau$, damit man viele unabhängige Schätzungen hat[16]. Dann gilt näherungsweise

$$\sum_{i,j=1}^{N} \Gamma_A(i-j)/\Gamma_A(0) \approx N \sum_{t=-\infty}^{\infty} \Gamma_A(t)/\Gamma_A(0) =: 2N\tau_{\mathrm{int},A}. \tag{6.44}$$

Eine effektive oder integrierte Autokorrelationszeit $\tau_{\mathrm{int},A}$ wurde hier definiert[17]. Diese faßt den Effekt von Autokorrelationen auf den Fehler von A zusammen,

$$\sigma(N,A)^2 = \frac{\mathrm{var}(A)}{N/2\tau_{\mathrm{int},A}}. \tag{6.45}$$

Interpretation: Statistischer Fehler wie bei $N/2\tau_{\mathrm{int},A}$ effektiven unabhängigen Schätzungen. Im Prinzip könnten negative Korrelationen den Fehler unter den des unabhängigen Falles drücken ($2\tau_{\mathrm{int},A} < 1$). Praktisch sind unseres Wissens keine solchen Fälle bekannt.

Offenbar beziehen sich die eingeführten MC Zeitskalen auf updates mit irgendeinem Algorithmus, z. B. lokale Wärmebad Sweeps. Würde man hingegen immer schon nach sehr kleinen (und damit "billigen") Schritten – z. B. einem lokalen random site update – "messen", so bekäme man viele Schätzungen, großes N. Gewonnen wäre nichts, da die τ und damit der Fehler dennoch groß wären. Im Gegenteil, da Messungen auch "kosten", würde man wahrscheinlich schließen, daß es günstiger wäre seltener zu messen[18].

Im Prinzip können $\tau_{\mathrm{int},A}$ für verschiedene A ganz verschiedenen sein, und keine dieser Größen ist ein strenger Indikator für die Anzahl von updates die zur Thermalisierung zu machen sind. Letztlich sind diese dynamischen Größen von den Eigenwerten und -vektoren von W bestimmt. In der Praxis sind die beobachteten $\tau_{\mathrm{int},A}$ von etwa der gleichen Größenordnung, und

[16] Für ein statistisches Verfahren braucht man Statistik!

[17] Wenn (6.43) exakt gilt und $\tau \gg 1$, was oft aber nicht immer realistisch ist, dann ist $\tau_{\mathrm{int},A} \approx \tau$

[18] Die Optimierung hängt von dem Kostenverhältnis Messung/update ab

Thermalisierungen von 20 bis 100 $\tau_{\max}$ sind selbstkonsistent gerechtfertigt. Meist ist die Thermalisierung billig, und man bleibt hier auf der sicheren Seite. Mögliche Tests bestehen hier auch aus Starts von verschiedenen Anfangskonfigurationen, z. B. total magnetisiert ($s(x) = +1$) und zufällig. Bei genügender Thermalisierung müssen im Rahmen der (korrelierten) Fehler kompatible Mittelwerte rauskommen.

Um in einer Simulation praktisch den Fehler zu schätzen ist man darauf angewiesen, neben der Varianz weitere Information über Γ_A numerisch zu bekommen. Eine Schätzung ist gegeben durch

$$\Gamma_A(t) \simeq \frac{1}{N-t} \sum_{i=1}^{N-t} \left(a_i - \frac{1}{N} \sum_{j=1}^{N} a_j \right) \left(a_{i+t} - \frac{1}{N} \sum_{k=1}^{N} a_k \right). \tag{6.46}$$

Auch diese Schätzung selbst hat wieder einen Fehler, den man zumindest bei einfachen Experimenten vernachlässigen muß. Um $\tau_{\mathrm{int},A}$ zu schätzen, sollte man $\Gamma(t)$ nur soweit summieren, bis es klein ist. Es ist nicht sinnvoll, Beiträge aufzusummieren, die nur noch aus "Rauschen" bestehen. Wenn es vom Datenanfall her möglich ist, ist zu empfehlen, während der teuren Simulation alle a_i zu speichern und diese später zu analysieren, also z. B. $\Gamma(t)$ zu bilden und zu plotten.

Eine einfache Analysemethode besteht auch in der Blockbildung, dem binning. Hier werden die ursprünglichen Messungen a_i blockweise vorgemittelt zu b_k,

$$b_k = \frac{1}{B} \sum_{i=1}^{B} a_{(k-1)B+i}, \quad k = 1, \ldots, N_B = [N/B], \tag{6.47}$$

wo N_B die Zahl der (vollständigen) solchen Bins ist. Dies ist auch rekursiv möglich, z. B. mit $B = 2, 4, 8, \ldots$. Die aufeinanderfolgenden b_k sind natürlich auch noch korreliert, aber wenn B einmal einige τ lang ist, so ist dies ein vernachlässigbarer Randeffekt. Der Mittelwert ist immer der gleiche[19]. Der Fehler, für nun unabhängige "Blockmessungen" numerisch über die Varianz geschätzt, ist[20]

$$\sigma^2 = \frac{1}{N_B(N_B-1)} \sum_{k=1}^{N_B} \left(b_k - \frac{1}{N_B} \sum_{l=1}^{N_B} b_l \right)^2. \tag{6.48}$$

[19] Exakt gilt das, solange N durch B teilbar ist und keine Messungen wegfallen

[20] Der Nenner $(N_B - 1)$ ergibt sich bei genauer Analyse daraus, daß die Subtraktion von A auch aus den Schätzungen gebildet ist.

Er steigt zunächst mit wachsendem B an, und saturiert dann beim korrekten Wert, wenn die Statistik reicht um dies alles zu sehen.

Binning ist eine Standard Methode, wenn es darum geht, Fehler für nichtlineare Funktionen von Observablen zu bestimmen. Ein Beispiel wäre eine Messung der 2-Punkt Funktion $G(x)$ zwischen Spins mit Abstand x

$$G(x) = \langle \mathcal{O}(s;x) \rangle \tag{6.49}$$

$$\mathcal{O}(s;x) = \frac{1}{V} \sum_z s(z)s(z+x). \tag{6.50}$$

Alle Summen, z. B. $z+x$, und Differenzen sind natürlich unter Beachtung der Torus Periodizität auszuführen (komponentenweise mod L). Man schließt in $\mathcal{O}$ die Summe über z ein, um durch Verwendung der Translationsinvarianz Statistik zu gewinnen. Ein typisches Problem wäre nun eine komplizierte Funktion $F[G(x)]$ zu schätzen, z. B. Parameter aus einem Fit der Korrelation an eine theoretische Form. Ein Schätzwert ergibt sich offenbar durch Einsetzen der Ensemble Mittel über $a_i(x) = \mathcal{O}(s^i;x)$ als Argumente von $F[.]$. Was aber wäre der auf Autokorrelationen korrigierte Fehler der Schätzung? Hier ist nun eine Möglichkeit, für jeden der näherungsweise voneinander unabhängigen zugehörigen Bins $b_k(x)$ die F-Berechnung (z. B. Fit) durchzuführen mit Resultaten $F_k = F[b_k(x)]$ und dann den Fehler zu schätzen wie in (6.48), nur mit F_k an Stelle von b_k selbst.

Praktisch ergibt sich hierbei oft das Problem, daß die Bins zu klein werden und daher ihre Mittel zu stark fluktuieren, um F sinnvoll zu berechnen. Da hilft der Trick des Jackknife Binning: statt der Bins b_k verwenden wir ihr Komplement,

$$c_k = \frac{1}{N-B} \left(\sum_i a_i - B b_k \right). \tag{6.51}$$

Diese Bins sind $N_B - 1$ mal größer, aber natürlich wieder nicht unabhängig, jedoch in einfacher Weise. Eine kurze Rechnung zeigt, daß sich aus den Jackknife Bins der Fehler schätzen läßt zu

$$\sigma^2 = \frac{N_B - 1}{N_B} \sum_{k=1}^{N_B} \left(c_k - \frac{1}{N_B} \sum_{l=1}^{N_B} c_l \right)^2. \tag{6.52}$$

Eine analoge Formel gilt nun für den Fehler der F-Schätzung, wobei man aus den Jackknife Bins gewonnene Werte $F_k = F[c_k(x)]$ benutzt.

7 Neuronale Netze am Beispiel des Hopfield Modells

In diesem Abschnitt sollen einige Anfangsgründe diskutiert werden von Systemen, deren Aufbau und auch Fähigkeiten in primitiver Weise dem menschlichen Gehirn ähneln. Man nennt sie daher neuronale Netze. Der Aspekt, um den es hier geht, ist die assoziative Speicherung, wobei die Adressierung über Teile des Speicherinhalts erfolgt. Ein Beispiel wäre das Erkennen eines bekannten Gesichtes aus einem verschwommenen Teilbild, oder die Erinnerung an ein Lied aus nur einem nicht ganz richtig gesummten Refrain. Das hier studierte Hopfield-Modell liefert eine Karikatur solcher Fähigkeiten. Zur Vorbereitung dieses Kapitels wurde [12] herangezogen.

7.1 Neuronen und Synapsen

Das Hopfield Netz besteht aus einer großen Zahl N von Neuronen S_i, die zweier Zustände fähig sind. In biologischen Netzen spricht man von feuernden und nicht feuernden bzw. aktivierten Neuronen, was wir in unserem Model durch $S_i = +1$ (feuernd) und $S_i = -1$ (nicht feuernd) darstellen. Dies ist selbstverständlich eine Konvention, $0, 1$ wäre genauso gut. Das Neuron S_i kann Neuron S_j mit variabler Stärke beeinflussen über Verbindungen oder Synapsen. Sie werden im Modell durch reelle Zahlen w_{ij} gegeben, die eine $N \times N$ Matrix bilden. Das Netz evolviert nun im einfachsten Fall in diskreten Zeitschritten von t nach $t+1$ nach dem Gesetz (McCulloch-Pitts Gleichung)

$$S_i(t+1) = \mathrm{sgn}(\sum_{\mathrm{j}} \mathrm{w_{ij} S_j(t) - \Theta_i}). \tag{7.1}$$

Hier sind die Synapsen w_{ij} als gegeben betrachtet, und sgn ist die Vorzeichen (signum) Funktion. S_i feuert also, wenn es genügend von anderen Neuronen über Synapsen angeregt wird relativ zu Schwellenwerten Θ_i. Die w_{ij} können positiv (exzitatorisch) oder negativ (inhibitorisch) sein. In diesen Größen residiert der Speicherinhalt, der bestimmt, wie das Netz auf vorgelegte Anfangsmuster $S_i(t_0)$ "reagiert". Eine wichtige Frage ist natürlich, wie die w_{ij} durch "Trainieren" auf das gewüschte Verhalten, z. B. anhand von Beispielen, gesetzt werden. Man spricht hier von Lernalgorithmen. Verallgemeinerungen von (7.1) bestehen aus anderen Aktivierungsfunktionen (stochastisch, "fuzzy") und der Wahl zwischen synchronem und asynchronem Update (vgl.

Jacobi und Gauss-Seidel Relaxation in CP I). Letzteres ist für das Gehirn wohl plausibler und algorithmisch einfacher.

Beim Hopfield Modell wird (7.1) als ein Schritt zur iterativen Minimierung der Energiefunktion

$$H = -\frac{1}{2}\sum_{ij} w_{ij} S_i S_j + \sum_i \Theta_i S_i \tag{7.2}$$

angesehen. Hier sind offensichtlich nur symmetrische w_{ij} sinnvoll, was eine nichttriviale Spezialisierung von (7.1) darstellt. Setzt man nun jeweils ein einzelnes Neuron S_i — bei Festhalten der Übrigen — auf denjenigen seiner beiden Werte, für den H kleiner ist, so ergibt sich (7.1) im *asynchronen* Modus mit *Einschränkung der Summe auf der rechten Seite* auf $j \neq i$. Da H dabei laufend fällt oder gleichbleibt, läuft man in einen Fixpunkt[21] am (i. a. nur lokalen) Minimum hinein. Dieser Konvergenzbeweis gälte nicht im synchronen Modus!

Im Folgenden wollen wir uns auf das Hopfield Modell mit $\Theta_i = 0$ beschränken.

7.2 Speichern in Synapsen

Eine übliche Anwendung des Hopfield Modells besteht in der assoziativen Speicherung von p Mustern $\xi_i^{(\mu)} = \pm 1, \mu = 1 \ldots p$. Hier bildet man die Neuronen auf bestimmte Bildelementen (Pixel), z. B. in einer Ebene, ab. Wenn nun ein Aktivierungszustand $S_i(t)$ "in der Nähe" von $\xi_i^{(\mu)}$ liegt — etwa einer verrauschten Version des Bildes entspricht —, dann soll er per McCulloch-Pitts Gleichung sich dorthin als dem "nächstliegenden" Minimum entwickeln[22] und im Idealfall bei $S_i = \xi_i^{(\mu)}$ als Fixpunkt verharren. Dann besitzt der Konfigurationsraum zu jedem Fixpunkt-Muster einen assoziierten Bereich im Zustandsraum, sein "basin of attraction".

Für nur ein einziges Bild ist eine geeignete Wahl der w_{ij} trivial,

$$w_{ij} = \frac{1}{N}\xi_i \xi_j. \tag{7.3}$$

[21] Der Spezialfall, daß beide Zustände exakt entartet sind, ist für reelle w_{ij} vernachlässigbar.

[22] Man kann den Abstandsbegriff präzisieren durch den Hamming Abstand = Zahl der differierenden Bits oder Pixel.

Dann gilt

$$H = -\frac{1}{2N}\left(\sum_i \xi_i S_i\right)^2 \tag{7.4}$$

mit absoluten Minimum $H = -N/2$ bei $S_i = \xi_i$. Der gleiche Wert wird angenommen bei $S_i = -\xi_i$, daher müssen mehr als die Hälfte der Bits am Anfang stimmen, und der Attraktionsbereich ist der halbe Konfigurationsraum. Geometrisch betrachtet ist w_{ij} als $N \times N$ Matrix ein Projektor auf die Richtung ξ und die damit gebildete quadratische Form H für S mit fester Länge $S^2 = N$ minimal, wenn S in Richtung ξ zeigt (parallel oder antiparallel).

Eine offensichtliche Verallgemeinerung auf mehrere Muster ist die Hebb'sche Regel

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} \xi_i^{(\mu)} \xi_j^{(\mu)}. \tag{7.5}$$

Hier werden entsprechende Projektoren addiert. Für wenige Bilder werden, wenn S_i nahezu parallel zu einem $\xi_i^{(\mu)}$ ist, die anderen Skalarprodukte typischerweise klein sein, und den Konvergenzprozeß zum betreffenden lokalen Minimum kaum stören. Dies gilt nicht mehr, wenn es voll wird im Speicher. Für $p = \mathrm{O}(N)$ kann sich w_{ij} einem Vielfachen der Einheitsmatrix nähern (exakt der Fall für N orthogonale Bilder), was dem Gedächtnisverlust durch zuviel Speichern entspricht, da dann im Extremfall jeder Zustand Fixpunkt ist. Die Kapazität ist also begrenzt, wie wir unten genauer diskutieren.

Die Diagonalelemente $w_{ii} = p/N$ tragen wegen $S_i^2 = 1$ eine für die Minimierung irrelevante Konstante zu H bei. Es ist praktisch, dort (7.5) zu $w_{ii} = 0$ zu modifizieren, so daß in (7.1) die volle Summe (Matrixmultiplikation) stehen darf.

7.3 Speicherkapazität

Zur Speicherkapazität des Hopfield Netzes wollen wir hier einige heuristische Überlegungen anstellen und dann einige Resultate angeben, deren Ableitung hier zu umfänglich wäre.

Wir wollen nun die Stabilität eines Bildes diskutieren. Wir nehmen einen Anfangszustand der Neuronen, der exakt dem ersten Bild entspricht, $S_i(t) =$

$\xi_i^{(1)}$. Dann ist[23]

$$S_i' \equiv S_i(t+1) = \mathrm{sgn}\left(\frac{1}{\mathrm{N}}\sum_{\nu,\mathrm{j}\neq\mathrm{i}} \xi_{\mathrm{i}}^{(\nu)}\xi_{\mathrm{j}}^{(\nu)}\xi_{\mathrm{j}}^{(1)}\right) \simeq \mathrm{sgn}\left(\xi_{\mathrm{i}}^{(1)}(1-\mathrm{C_i})\right), \qquad (7.6)$$

wobei

$$C_i = -\frac{1}{N}\sum_{\nu>1,j\neq i} \xi_i^{(1)}\xi_i^{(\nu)}\xi_j^{(\nu)}\xi_j^{(1)} \qquad (7.7)$$

die "Störung" durch die übrigen $p-1$ gespeicherten Bilder beinhaltet. Offensichtlich werden Bits von $\xi_i^{(1)}$ instabil, wenn $C_i > 1$ wird.

Für zufällige unkorrelierte Muster wollen wir die Wahrscheinlichkeit dafür abschätzen. Wir betrachten C_i als eine Summe von pN zufälligen Größen ϵ_a mit jeweils gleich wahrscheinlichen Werten ± 1 geteilt durch N. Die Summe solcher Zufallsvariablen ist binomial verteilt, und diese Verteilung geht für viele Beiträge in die Normalverteilung über. Letztere ist festgelegt durch Mittelwert und Varianz,

$$\langle C_i\rangle = 0, \qquad (7.8)$$

$$\langle C_i^2\rangle = \frac{1}{N^2}\left\langle \sum_{a,b=1}^{pN}\epsilon_a\epsilon_b\right\rangle = \frac{1}{N^2}\sum_{a,b=1}^{pN}\delta_{ab} = \frac{p}{N} = \sigma^2. \qquad (7.9)$$

Damit ist die Wahrscheinlichkeit, daß $C_i > 1$ gilt, gegeben durch

$$P_{\text{Fehler}} = \frac{1}{\sqrt{2\pi}\sigma}\int_1^\infty \exp(-x^2/2\sigma^2)\mathrm{d}x = \frac{1}{2}\,\mathrm{erfc}(\sqrt{N/2p}). \qquad (7.10)$$

Hier ist erfc die (komplementäre) Fehlerfunktion. Sie ist in MATLAB vorhanden, und Werte sind

```
>> a=[0.105 0.138 0.185 0.37 0.61]'; % Werte p/N
>> P=0.5*erfc(sqrt(0.5./a));         % P_Fehler
>> [a P]

ans =

    0.1050    0.0010
    0.1380    0.0036
```

[23] Wir vernachlässigen Terme der relativen Ordnung $1/N$.

```
0.1850    0.0100
0.3700    0.0501
0.6100    0.1002
```

Die linke Spalte sind p/N Verhältnisse, die rechte der Anteil fehlerhafter Bits.

Diese Betrachtung betraf instabile Bits nach einem Schritt beginnend beim perfekten Muster. Sie sagt nichts darüber aus, wie es weitergeht, ob z. B. eine Lawine von geflippten Bits einsetzt. Wesentlich kompliziertere Rechnungen [12] zeigen, daß die Grenze dafür bei $p/N \simeq 0.138$ liegt. Genauer gesagt, handelt es sich hier für $N \to \infty$ um einen Phasenübergang erster Ordnung. Bis dahin funktioniert das Gedächtnis gut, danach nicht mehr. Gerade bei diesem Grenzwert sind nach einem Schritt im Mittel 0.36 % Bits falsch. Dies steigt dann auf 1.6 %, wo dann ein Fixpunkt nahe beim Muster erreicht wird. Mit kleineren Füllfaktoren (?, load parameter) funktioniert alles fehlerärmer.

7.4 Simulation

In diesem Abschnitt sollen zur Erleichterung der Übungsaufgaben einige charakteristische Statements in MATLAB zur Realisierung des Hopfield Modells angegeben werden. Diese Umsetzung ist natürlich nicht die einzig mögliche. Zunächst kann man wie folgt Bilder erzeugen und die Hebb Matrix konstruieren:

```
n=100;          % # Neuronen
p=20;           % # Bilder

xi = sign(rand(n,p)-0.5); % Bilder

% Synapsen setzen nach Hebb:
w=zeros(n,n);
for i=1:p
  w=w+xi(:,i)*xi(:,i)';
end
for i=1:n, w(i,i)=0; end  % Diagonale zu Null
w=w/n;
```

Die folgenden Zeilen zeigen Update Schritte asynchron und (auskommentiert) synchron, sowie die Berechnung des Hamming Abstandes:

```
    s=xi(:,1);
%   sp=sign(w*s);                                % synchron
    sp=s; for j=1:n, sp(j)=sign(w(j,:)*sp);end % asynchron
    ham=sum(s ~= sp);                            % Hamming Abstand
```

Wer möchte, kann sich auch ein Bild ansehen, obwohl dies bei Zufallsmustern nicht sehr erleuchtend ist:

```
b=10;    % Kantenlaenge Bild
n=b^2;   % # Neuronen
xi = sign(rand(n,1)-0.5); % 1 Bild
pic=zeros(b,b);
pic(:)= (xi > 0);  % xi -> Spalten des Bildes, -/+ in 0/1 verwandelt
spy(pic);          % plotted eine Matrix
```

7.5 Nebenminima

Außer den erwünschten Mustern, die bei der Hebb Regel in die w_{ij} eingehen, kann es andere unerwünschte Fixpunkte geben (spurious states). Ein Beipiel sind Überlagerungszustände (mixed states) aus einer ungeraden Zahl von Mustern. Wir betrachten als Beispiel drei Zustände,

$$\bar{\xi}_i = \mathrm{sgn}(\xi_{\mathrm{i}}^{(1)} + \xi_{\mathrm{i}}^{(2)} + \xi_{\mathrm{i}}^{(3)}). \tag{7.11}$$

Der Zustand $S_i = \bar{\xi}_i$ entwickelt sich zu $S_i' = \mathrm{sgn}(\bar{\mathrm{h}}_{\mathrm{i}})$ mit

$$\bar{h}_i = \frac{1}{N} \sum_{\nu, j \neq i} \xi_i^{(\nu)} \xi_j^{(\nu)} \bar{\xi}_j. \tag{7.12}$$

Für nicht zu viele Muster kann die Summe näherungsweise auf $\nu = 1, 2, 3$ eingeschränkt werden, da die übrigen Zustände dann typischerweise kleine Skalarprodukte mit den betrachteten haben. Nun sind die Produkte (z. B. $\nu = 1$)

$$\xi_j^{(1)} \bar{\xi}_j = \xi_j^{(1)}\ \mathrm{sgn}(\xi_{\mathrm{j}}^{(1)} + \xi_{\mathrm{j}}^{(2)} + \xi_{\mathrm{j}}^{(3)})$$

in 3 von 4 möglichen Konfigurationen von $\xi_j^{(2)}, \xi_j^{(3)}$ (außer $--$) positiv, und bei Summation über j erhalten wir im Mittel

$$\left\langle \sum_j \xi_j^{(1)} \bar{\xi}_j \right\rangle = \frac{N}{2} \tag{7.13}$$

und somit

$$\langle \bar{h}_i \rangle = \frac{1}{2}(\xi_i^{(1)} + \xi_i^{(2)} + \xi_i^{(3)}), \tag{7.14}$$

was die Stabilität von $\bar{\xi}_i$ zeigt. Das gleiche Resultat folgt für andere Wahlen der drei Vorzeichen auf der rechten Seite von (7.11). Darüber hinaus gibt es weitere unerwünschte Minima, die sogenannten Spinglas Zustände, die mit den gespeicherten Mustern nicht direkt korreliert sind. Trotz dieser Zustände bleibt das Hopfield Netz nützlich als Assoziativspeicher (content addressed), da die Attraktionsbereiche der gespeicherten Muster typischerweise größer sind.

7.6 Endliche Temperatur

Oft ist es nützlich, statt der deterministischen McCulloch-Pitts Gleichung ein stochastisches ("fuzzy") Element (Rauschen) einzuführen, so daß $S_i(t+1)$ sich manchmal auch gegen das Vorzeichen der rechten Seite von (7.1) stellt. Genau das ergibt sich, wenn wir mit der Energie H des Hopfield Modells statistische Mechanik treiben.

Wir betrachten die Zustandssumme

$$Z = \sum_{\{S_i\}} \exp(-\beta H) \tag{7.15}$$

mit der inversen Temperatur β. Nun implementieren wir eine Monte Carlo Simulation mit diesem Boltzmann Gewicht durch lokale Wärmebad Schritte für einzelne S_i:

$$P(S_i' = \pm) = \frac{\exp(\pm\beta h_i)}{\exp(\beta h_i) + \exp(-\beta h_i)} \tag{7.16}$$

mit

$$h_i = \sum_{j \neq i} w_{ij} S_j. \tag{7.17}$$

Hier taucht die sogenannte Sigmoid Funktion auf,

$$P(S_i' = +) = 1 - P(S_i' = -) = f_\beta(h_i) = \frac{1}{1 + \exp(-2\beta h_i)}. \tag{7.18}$$

Offenbar entsprechen diese Schritte für $\beta \to \infty$ genau der McCulloch-Pitts Dynamik. Bei endlicher Temperatur können kleinere Nebenminima durch thermische Fluktuationen wieder verlassen werden um den Weg in größere

basins of attraction zu finden. Dort kann man natürlich keine exakte Konvergenz mehr erwarten, da ja weiterhin Fluktuationen möglich bleiben. Die Wahl der Temperatur stellt natürlich eine gewisse Kunst dar. Es gibt dann ein Phasendiagramm des Hopfield Modells in einer Ebene der Parameter Temperatur und Füllfaktor [12]. Der kritische Wert des Letzteren fällt allerdings mit steigender Temperatur.

In der Optimierungstheorie gibt es das eng verwandte Verfahren des "Simulated Annealing". Dort geht es darum, das absolute Minimum einer Funktion vieler Variabler (Kostenfunktion) zu finden. Ein typisches Problem besteht darin, daß man mit deterministischen Verfahren wie "steepest descent" in lokalen Minima hängen bleibt. Ähnlich wie hier geht man zur statistischen Mechanik mit der Kostenfunktion als Energie über, und erhöht nun β während der Iteration nach einem bestimmten Fahrplan. So kann man z. B. versuchen das klassische Optimierungsproblem anzugehen, die Gesamtwegstrecke des "travelling salesman", der eine vorgegebene Zahl von Städten in beliebiger Reihenfolge besuchen muß, zu minimieren.

Literatur

[1] S.E.Koonin und D.C.Meredith, Computational Physics, Addison–Wesley

[2] E. Ott, Chaos in Dynamical Systems, Cambridge University Press 1993

[3] H. G. Schuster, Deterministisches Chaos, VCH Verlagsgesellschaft, Weinheim

[4] H. Goldstein, Klassische Mechanik, AULA-Verlag, Wiesbaden

[5] D. E. Knuth, Seminumerical Algorithms, Vol. 2 of The Art of Computer Programming (Addison Wesley)

[6] Martin Lüscher, A PORTABLE HIGH QUALITY RANDOM NUMBER GENERATOR FOR LATTICE FIELD THEORY SIMULATIONS. Comput.Phys.Commun.79:100-110,1994, hep-lat/9309020

[7] H. Erlenkötter und V. Reher, Programmiersprache C, rororo Grundkurs Computerpraxis (16,80 DM)

[8] Die Programmiersprache C, herausgegeben vom Regionalen Rechenzentrum Niedersachsen (RRZN)

[9] D. Stauffer und A. Aharony, Perkolationstheorie, VCH Verlagsgesellschaft, 1995

[10] W. Nolting, Grundkurs Theoretische Physik Bd. 6: Statistische Physik
J. Schnakenberg, Algorithmen in der Quantentheorie und Statistischen Physik
beide: Verlag Zimmermann-Neufang, Ulmen

[11] U. Wolff, Numerical Simulation in Quantum Field Theory, in Computational Physics, K.H.Hoffmann und M.Schreiber Eds., Springer 1996

[12] J. Hertz, A. Krogh und R.G.Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling und B. P. Flannery, Numerical Recipes, Cambridge University Press
Von diesem nützlichen Buch gibt es verschiedene Ausgaben mit Programmen in Fortran oder C

[14] I. Montvay und G. Münster, Quantum Fields on a Lattice, (Cambridge Univ. Press, 1994)

[15] M. Creutz, Quarks, Gluons, Lattices, (Cambridge Univ. Press, 1983)

[16] H. J. Rothe, Lattice Gauge Theories: Am Introduction (Wolrd Scientific, 1992)

[17] J. Kogut, An introduction to lattice gauge theory and spin systems, Reviews of Modern Physics 51 (1979) S.659-713

[18] K. Wilson, Confinement of Quarks, Phys. rev. D10 (1974) 2445