Computerphysik

Prof. Dr. Heiko Rieger

Universität des Saarlandes

Fakultät 7

Theoretische Physik (FR 7.1)

Inhaltsverzeichnis

1 Gewöhnliche Differentialgleichungen

1.1 Anfangswertprobleme

(z. B. Bewegung von Mond / Erde / Sonne, Rakete, Ozeanwelle)

$\boxed{\frac{d\mathbf{y}}{dt} = \mathbf{g}(\mathbf{y}, t)}$ $\quad \mathbf{y} = (y_1, \ldots, y_N)$

$\boxed{\mathbf{y}(t=0) = \mathbf{y_0}}$ $\quad \mathbf{g}(\mathbf{y}, t) = (g_1(\mathbf{y}, t), \ldots, g_N(\mathbf{y}, t))$ („verallgemeinerte Geschwindigkeit")

Beispiel:
$F = ma$ (in einer Dimension)

$$\left.\begin{array}{l} F = -kx \quad \text{(Feder)} \\ a = x'' \end{array}\right\} x'' = -\frac{k}{m}x$$

$$\left.\begin{array}{l} y_1 := x \\ y_2 := x' \end{array}\right\} \Rightarrow \begin{array}{l} y_1' = x' = y_2 \\ y_2' = x'' = -\frac{k}{m}y_1 \end{array}$$

d.h. $g_1 = y_2$, $g_2 = -\frac{k}{m}y_1$

Allgemein:
Die meisten gewöhnlichen Differentialgleichungen höherer Ordnung können in ein System von gewöhnlichen Differentialgleichungen 1. Ordnung überführt werden.

1.2 Euler- und Picard-Methode

Diskrete Stützstellen: $t_0, t_1, \ldots \quad ; \quad y_0, y_1, \ldots$

$$y(t_i) = y_i$$

$\frac{dy}{dt} \cong \frac{y_{i+1}-y_i}{t_{i+1}-t_i} \cong g(y_i, t_i)$

$\Rightarrow y_{i+1} \cong y_i + (t_{i+1} - t_i) \cdot g_i \cdot (y_i, t_i)$

Für äquidistante Stützstellen $\forall i$: $t_{i+1} - t_i = \tau \ll 1$

$\hookrightarrow \boxed{y_{i+1} = y_i + \tau g_i + \mathcal{O}(\tau^2)}$ Euler-Verfahren $\circledast$

Integration von $t = 0$ bis $t = T$, $\Rightarrow M = \frac{T}{t}$ Stützstelle

$\hookrightarrow$ Am Ende der Iteration $\circledast$ Fehler $\sim M \cdot \mathcal{O}(\tau^2) = \frac{T}{\tau} \cdot \tau^2 = \underline{\underline{Tt}}$

Euler-Verfahrem sehr ungenau

Formale Lösung von $\dot{y} = g(y)$: $\boxed{y_{i+j} = y_i + \int_{t_i}^{t_{i+j}} dt g(y, t)}$

Picard-Verfahren:

$$y_{i+1} = y_i + \overbrace{\frac{\tau}{2}(g_i + g_{i+1})}^{\text{Trapez-Regel}} + \mathcal{O}(\tau^3)$$

$g_{i+1} = g(y_{i+1}, t_{i+1})$ enthält y_{i+1}

$\Rightarrow$ iteratives Verfahren: 1. Schätzwert y_1: $y_{i+1}^{(0)} = y_1$

Konvergenz kann langsam sein.

$$\boxed{y_{i+1}^{(k+1)} = y_i + \frac{\tau}{2}\left(g_i + g_{i+1}^{(k)}\right)}$$

1.3 Predictor-Korrektor-Methoden

Beispiel:
Benutze Euler-Verfahren zur Voraussage von y_{i+1}, dann Picard-Verfahren, um diese zu verbessern.

$$y_{i+1}^{(pred.)} = y_i + \tau g_i(y_i, t_i)$$

$$y_{i+1}^{(corr.)} = y_i + \frac{\tau}{2}\left\{g(y_i, t_i) + g\left(y_{i+1}^{(pred.)}, t_i\right)\right\}$$

$$y_{i+1} = y_{i+1}^{(corr.)}$$

Alternativ: Anzahl der Gitterpunkte in $y_{i+j} = \ldots$ erhöhen:

Beispiel:

$$y_{i+2} = y_i + \int_{t_i}^{t_{i+2}} dt g(y,t)$$

Lineare Interpolation:
$g^{(inter.)}(y,t) = \frac{t-t_i}{\tau} g_{i+1} + \frac{t_{i+1}-t}{\tau} g_i$
$\Rightarrow g_{i+2}^{(inter.)} = g^{(inter.)}(y, t_{i+2}) = 2g_{i+1} - g_i$

$$\Rightarrow \int_{t_i}^{t_{i+2}} dt g(y,t) \approx \frac{2\tau}{2} \cdot (2g_{i+1} - g_i + g_i) = 2\tau$$

$$\hookrightarrow \boxed{y_{i+2} = y_i + 2\tau g_{i+1} + \mathcal{O}(\tau^3)} \text{ Ⓐ}$$

Zum Start der Iteration braucht man neben Anfangswert y_0 noch $y_1 = y(\tau)$ ($\hookrightarrow g_1 = g(y_1, t)$).
Gewöhnlich benutzt man Taylor-Entwicklung $y_1 = y_0 + \tau g_0 + \frac{\tau^2}{2}\left(\frac{\partial g_0}{\partial t} + g_0 \frac{\partial g_0}{\partial y}\right) + \mathcal{O}(\tau^3)$
Bessere numerische Quadratur:

$$\boxed{y_{i+2} = y_i + \frac{\tau}{8}(g_{i+2} + 4g_{i+1} + g_i) + +\mathcal{O}(\tau^5)}$$ Corrector für Ⓐ

1.4 Runge-Kutta-Verfahren

$$y(t+\tau) = y(t) + \tau y'(t) + \frac{\tau^2}{2} y''(t) + \frac{\tau^3}{3!} y^{(3)}(t) + \ldots$$

$$\begin{aligned}
y' &= g(y,t) \\
y'' &= y' g_y + g_t \\
&= g g_y + g_t \\
y^{(3)} &= (y' g_y + g_t) g_y + g(y' g_{yy} + g_{yt}) + y' g_{ty} + g_{tt} \\
&= g g_y^2 + g_t g_y + g^2 g_{yy} + 2 g g_{yt} + g_{tt}
\end{aligned}$$

$$\hookrightarrow y(t+\tau) = y + \tau g + \frac{\tau^2}{2}(g_t + g g_y) + \frac{\tau^3}{6}\left(g_{tt} + 2 g g_{ty} + g^2 g_{yy} + g g_y^2 + g_t g_y\right) + \mathcal{O}(\tau^4) \text{ ①}$$

Formal:
$y(t+\tau) = y(t) + \alpha_1 c_1 + \alpha_2 c_2 + \ldots + \alpha_m c_m$
mit

$$\begin{aligned}
c_1 &= \tau g(y,t) \\
c_2 &= \tau g(y + \nu_{21}, t + \nu_{21}\tau) \\
c_3 &= \tau g(y + \nu_{31} c_1 + \nu_{32} c_2, t + \nu_{31}\tau + \nu_{32}\tau) \qquad \text{②} \\
&\vdots \\
c_m &= \tau g\left(y + \sum_{i=1}^{m-1} \nu_{mi} c_i, t + \tau \sum_{i=1}^{m-1} \nu_{mi}\right)
\end{aligned}$$

Entwicklung von ②, Vergleich mit ①
$\Rightarrow$ Bestimmungsgleichungen für α_i, ν_{ij} (unterbestimmt, Wahlmöglichkeit)

Beispiel:
Runge-Kutta 4. Ordnung

$y(t+\tau) = y(t) + \frac{1}{6}(c_1 + 2c_2 + 2c_3 + c_4)$	
$c_1 = \tau g(y,t)$	$c_3 = \tau g\left(y + \frac{c_2}{\tau}, t + \frac{\tau}{2}\right)$
$c_2 = \tau g\left(y + \frac{c_1}{\tau}, t + \frac{\tau}{2}\right)$	$c_4 = \tau g\left(y + \frac{c_3}{\tau}, t + \tau\right)$

1.5 Bemerkungen zu Randwertproblemen

Beispiel:
$u'' = f(u, u', x)\ x \in [0, 1]$
mit $u(0) = u_0$, $u(1) = u_1$
[z. B. elastische Wellen $u''(x) = -k^2 u(x)$, $u(0) = 0$, $u(1) = 0$]

1.6 „Shooting-Verfahren"

$y_1 := u$, $y_2 := u'$
$\hookrightarrow \frac{dy_1}{dx} = y_2 \qquad y_1(0) = u(0) = u_0 \leftarrow$ fest
$\frac{dy_2}{dx} = f(y_1, y_2, x) \quad y_2(0) = u'(0) = \alpha \leftarrow$ anzupassender Parameter
① ②
Löse ①, z. B. mit Runge-Kutta, mit Randbedingungen ② und variiere α so lange (z. B. mit kleinen Inkrementen), bis $u(1) = 1$ (innerhalb der gewünschten Toleranz).

1.7 Eigenwertprobleme

Speziell eindimensionale stationäre Schrödinger-Gleichung: $-\frac{\hbar^2}{2m}\psi''(x) + V(x)\psi(x) = E\psi(x)$
$\hookrightarrow \psi''(x) = \frac{\hbar^2}{2m}(E - V(x))\psi(x)$
Eigenwert E ist nun Parameter, der im Shooting-Verfahren variiert wird
$\psi(0) = 0$
$\psi'(0) = c$ wenn $V(x) = +\infty$ für $x \leq 0$
anschließend Normierung!

2 Partielle Differentialgleichungen

Beispiel:

linear
- $\Delta\phi(\mathbf{r}) = -\frac{\rho(\mathbf{r})}{\varepsilon_0}$ — Poisson-Gleichung (elliptisch)
- $\frac{\partial n(\mathbf{r})}{\partial t} - \nabla \cdot (D(\mathbf{r}) \nabla n(\mathbf{r}, t)) = S(\mathbf{r}, t)$ — Diffusionsgleichung (parabolisch)
- $\frac{1}{c^2} \frac{\partial^2 u(\mathbf{r},t)}{\partial t^2} - \Delta u(\mathbf{r}, t) = R(\mathbf{r}, t)$ — Wellengleichung (hyperbolisch)
- $-\frac{\hbar}{i} \frac{\partial \psi(\mathbf{r},t)}{\partial t} = \hat{H}\psi(\mathbf{r}, t)$ — Schrödinger-Gl. ($\hat{=}$ Diffusionsgl. in Imaginärzeit)

nichtlinear
- $\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} + \frac{1}{\rho}\nabla p - \eta \Delta \mathbf{v}$ — Navier-Stokes-Gleichung
- $\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$ — Kontinuitätsgleichung
- $f(p, \rho)$ — Zustandsgleichung

(typisch: Hydrodynamik)

Randbedingungen $\rightarrow$ ⊛

2.1 Diskretisierung der Gleichungen

$$\frac{\partial A(\mathbf{r}, t)}{\partial t} \rightarrow \frac{A(\mathbf{r}, t_{k+1}) - A(\mathbf{r}, t_k)}{\tau} \qquad \tau = t_{k+1} - t_k$$

$$\text{oder} \rightarrow \frac{A(\mathbf{r}, t_{k+1}) - A(\mathbf{r}, t_{k-1})}{2\tau}$$

$$\frac{\partial^2 A(\mathbf{r}, t)}{\partial t^2} \rightarrow \frac{A(\mathbf{r}, t_{k+1}) - 2A(\mathbf{r}, t_k) + A(\mathbf{r}, t_{k-1})}{\tau^2}$$

analog

$$\frac{\partial A(\mathbf{r}, t)}{\partial x} \rightarrow \frac{A(x_{k+1}, y, z, t) - A(x_{k-1}, y, z, t)}{2h_x} \qquad h_x = x_{k+1} - x_k$$

bzw.

$$\frac{\partial^2 A(\mathbf{r}, t)}{\partial x^2} \rightarrow \frac{A(x_{k+1}, y, z, t) - 2A(x_k, y, z, t) + A(x_{k-1}, y, z, t)}{h_x^2}$$

Die partiellen Differentialgleichungen können dann an diskreten Punkten gelöst werden.

Bemerkung:

⊛ Randbedingungen i. A. wichtig: D = Lösungsgebiet, ∂D = Rand des Lösungsgebietes

z. B.: $A(\mathbf{r})|_{r \in \partial D} = g(\mathbf{r})$ Dirichlet-Randbedingung

$\underbrace{\frac{\partial}{\partial \mathbf{n}}}_{Normalenableitung,\, \mathbf{n} \perp D} A(\mathbf{r})|_{r \in \partial D} = h(\mathbf{r})$ von Neumann-Randbedingung

+gemischt

Für nicht-homogene Gitterpunkte („Mesh") oder für Inhomogenitäten wie in $\nabla \cdot (D(\mathbf{r}) \nabla A(\mathbf{r}, t))$ mit ortsunabhängiger Diffusionskonstanten benutzt man andere Formen der Diskretisierung.

Beispiel:

$\frac{d}{dx}\left(\varepsilon(x) \frac{d\phi(x)}{dx}\right) = -\rho(x)$ für $x \in [0, L]$ ⊛

eindimensionale Poisson-Gleichung mit ortsabhängiger elektrischer Permittivität

Diskretisierung: $\boxed{\frac{\frac{\varepsilon_{k+1}+\varepsilon_k}{2}\phi_{k+1} - 2\varepsilon_k\phi_k + \frac{\varepsilon_{k-1}+\varepsilon_k}{2}}{h^2} = -\rho_k}$ ⊛

Diskretisierung von linearen partiellen Differentialgleichungen $\rightarrow$ $\boxed{\hat{\mathcal{L}}U(\mathbf{r}, t) = f(\mathbf{r}, t)}$

$\Rightarrow$ Differenzenvergleich $\rightarrow$ $\boxed{\mathbf{Au} = \mathbf{b}}$

↳Lösung mit numerischen Methoden der Linearen Algebra

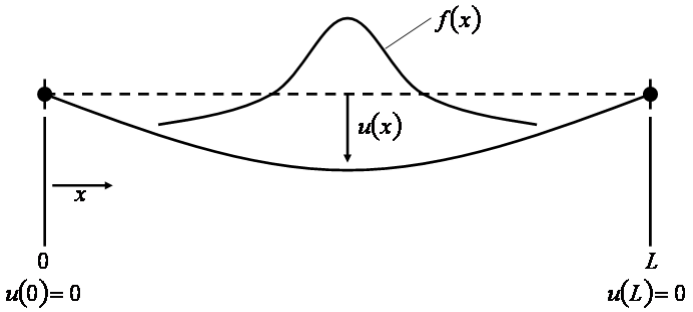
Beispiel:

$$\circledast \Rightarrow A_{ij} = \begin{cases} -4\varepsilon_i & \text{für } i = j \\ \varepsilon_i + \varepsilon_{i+1} & \text{für } i = j-1 \\ \varepsilon_i + \varepsilon_{i-1} & \text{für } i = j+1 \\ 0 & \text{sonst} \end{cases}$$

$$b_{ij} = -2h^2\rho_i$$

Illustration:

Deformation einer Bank bei Belastung



$Y \cdot I \cdot u'' = f(x)$ $\quad$ Y: Young-Modulus

I: geometrische Konstanten $I = d^3w/3$

d: Dicke

w: Breite

$f(x)$: Gewichtsverteilung

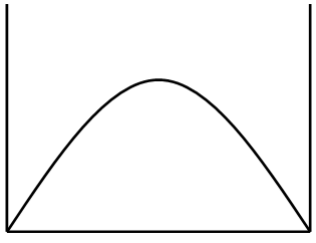
$\hookrightarrow u_{i+1} - 2u_i + u_{i-1} = \frac{h^2 f_i}{YI} =: b_i$

d. h. $\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$

$[u_0 = u_{n+1} = 0]$

z. B. $f(x) = \begin{cases} -f_0 \left(e^{-(x-L/2)^2/x_0^2} \right) - \rho g & \text{für } |x - L/2| \le x_0 \\ -\rho g & \text{sonst} \end{cases}$

$\hookrightarrow$



2.2 Relaxationsmethode

Beispiel:
$-\frac{d}{dx}\left(D(x)\frac{dn(x)}{dx}\right) = S(x)$
Diskretisierung:
$\hookrightarrow \boxed{n_i = \frac{1}{D_{i+1/2}+D_{i-1/2}}\left(D_{i+1/2}n_{i+1} + D_{i-1/2}n_{i-1} + h^2 S_1\right)}$ ⊛
Starte mit $n_i^{(0)}$ $(i = 0, \ldots, L)$, welches die Randbedingungen erfüllt.
Update: $\boxed{n_i^{(k+1)} = (1-p)\, n_i^{(k)} + p n_i}$ ⊛⊛
$p \in [0,2]$, $n_i = r.h.s.$ von ⊛ evaluiert mit $n_i^{(k)}$
Parameter
Iteriere ⊛⊛ bis zur Konvergenz, Randbedingungen berücksichtigen

2.3 Anfangswertprobleme

Partielle Differentialgleichungen höhere ordnung werden wie üblich in Systeme von partiellen Differentialgleichungen 1. Ordnung überführt.
Beispiel:
Wellengleichung: $v(\mathbf{r},t) := \frac{\partial u(\mathbf{r},t)}{\partial t}$
$$\Rightarrow \quad \frac{\partial u(\mathbf{r},t)}{\partial t} = v(\mathbf{r},t)$$
$$\frac{1}{c^2}\frac{\partial v(\mathbf{r},t)}{\partial t} = \Delta u(\mathbf{r},t) + R(\mathbf{r},t)$$
n. b.: Nun ähnliche mathematische Struktur wie Diffusionsgleichung.
$$\frac{\partial n(\mathbf{r},t)}{\partial t} = D\Delta n(\mathbf{r},t) + S(\mathbf{r},t)$$
in einer Dimension, Diskretsierung: $\hookrightarrow$
$$\boxed{n_i(t+\tau) = n_i(t) + \gamma\left[n_{i+1}(t) + n_{i-1}(t) - 2n_i(t)\right] + \tau S_i(t)}$$
$\gamma = \frac{D\tau}{h^2}$
Nur stabil für $\gamma \lesssim \frac{1}{2}$.
Besser: Crank-Nicolson:
$$n_i(t+\tau) = n_i(t) + \frac{1}{2}\left\{\left(H_i n_i(t) + \tau S_i(t)\right) + \left(H_i n_i(t+\tau) + \tau S_i(t)\right)\right\}$$
mit $H_i n_i(t) := \gamma\left(n_{i+1}(t) + n_{i-1}(t) - 2n_i(t)\right)$
$$\Rightarrow \boxed{(2 - H_i)\, n_i(t+\tau) = (2 + H_i)\, n_i(t) + \tau\left(S_i(\tau) + S_i(t+\tau)\right)}$$
↑
Tridiagonalmatrix
Stabil für alle γ, konvergiert für $h \to 0$,
Fehler $\propto h^2$

3 Zufallszahlen

3.1 Echte Zufallszahlen

Zufallszahlen (Abk. Zz.) spielen heutzutage in vielen Bereichen des Alltags eine Rolle, einst z.B. in der Kryptographie großer Banken, Versicherungen, aber auch zur Verschlüsselung beim Militär. Aufgrund der stetigen „Nachfrage" nach neuen Zufallszahlen hat sich eine ganze Industrie entwickelt die sich ausschließlich mit der Erzeugung und Bereitstellung „guter" Zufallszahlen in ausreichender Menge beschäftigt. Es werden kommerzielle CD-Roms angeboten, die nichts weiter als $10^8 - 10^9$ Zufallszahlen enthalten, gewonnen z. B. aus der Beobachtung stochastischer Prozesse wie radioaktivem Zerfall.
Aktuellere Kryptographiemethoden beruhen auf einem anderem Konzept, deren Sicherheit auf der Faktorisierung eines Produktes sehr großer Primzahlen beruht. Diese alternativen Verfahren können umgekehrt zur Generation von „high-qualitiy"-Zufallszahlen genutzt werden, was für unsere Anwendungen aber mit zu hohem Aufwand verbunden ist.

3.2 Pseudo Zufallszahlen

Für unsere Zwecke vollkommen ausreichend sind Pseudo-Zufallszahlen (Abk.: PZ) die durch Computer-Algorithmen „berechnet" werden. Diese sind zumeist ganzzahlig (Integer) und nach oben beschränkt: $rand \in \{0, 1, ..., m-1\}$. Eine Division durch m überführt sie in nicht-ganzahlige Werte im Intervall $[0, 1]$, die Form in der sie am häufigsten gebraucht werden. Die einfachste Methode, um PZ zu generieren, ist die iterative Anwendung einer Funktion f auf eine bereits ermittelte Zufallszahl:

$$i_n = f(i_{n-1}). \tag{3.1}$$

Man erhält somit ausgehend von einem sog. Seed-Wert i_0 eine Folge von PZ, die immer wieder reproduziert werden kann, nur für verschiedene Seed-Werte erhält man unterschiedliche Sequenzen von PZ. Die Funktion f sollte hierbei nur Integeroperationen enhalten $(+, -, \cdot, :)$, man hofft durch eine möglichst nichtlineare Abbildung genügend Chaos zu stiften um gute PZ zu erhalten.
Wichtig für RNG (Random Number Generators) wie (3.1) ist die Tatsache das sie immer auf Zyklen von Zufallszahlen führen, d.h. gilt $i_{n_0} = i_{n_0-k}$ für ein n_0 und ein k, dann ist $i_n = i_{n-k} \forall n$ (Zyklus der Länge k).
Da Rechnerarchitekturbedingt nur m ($m \leq 2^{32}$ oder $m \leq 2^{64}$) verschiedene Zahlen erzeugt werden können, ist die maximale Zykluslänge $k = m$. Um möglichst viele verschiedene PZ zu erzeugen, ist man daher auf der Suche nach „guten" Funktionen f mit großer Zykluslänge.
Eine Erweiterung der Zykluslänge kann mit aufwendigeren RNG erreicht werden, die mehr als eines der vorhergehenden Folgenglieder miteinbezieht:

$$i_n = f(i_{n-1}, i_{n-2}, ..., i_{n-l}) \tag{3.2}$$

Diese RNG haben immerhin eine maximale Zykellänge von m^l.

3.3 Lineare Kongruenz Generatoren

Am weitesten verbreitet sind einfache Generatoren der Form
$$i_n = (a \cdot i_{n-1} + c) \, mod \, m. \tag{3.3}$$

[$p \, mod \, q$ gibt den Rest der ganzahligen Divison von p durch q an, z.B. $5 \, mod \, 2 = 1$]
Der maximale Wert von m ist architekturbedingt gegeben, oft durch $w = 2^{32}$ auf modernen Computern mit 32-Bit Integer Arithmetik. Die Wahl von $m = w$ wäre bequem, man müsste nicht einen evt. auftretenden Overflow abfangen. Leider sind die niedrigsten Bits, welche die führenden Stellen einer Zahl bestimmen, wenig zufällig. Bspw. wird für $m = w$ das kleinste Bit immer 0 oder 1 sein oder zwischen 0 und 1 alternieren, in Abh. von der Wahl von a und c (gerade, ungerade). Daher muss m kleiner als w gewählt werden.
Mit dieser Wahl hat man sich leider ein neues Problem eingehandelt. Im Ablauf des Algorithmus kann es nun passieren, dass $a \cdot i_{n-1} + c$ einen Overflow produziert. Aus diesem Grund muss m wesentlich kleiner als w gewählt werden (in Abh. von a und c), alternativ kann man sich auch mit einem Trick behelfen (s.u.).

Auch hier ist m obere Grenze für die Zykluslänge.
Bsp.: a, c, m gerade $\Rightarrow$nur gerade Zahlen $\Rightarrow k \leq \frac{m}{2}$
Andere a, c, m geben „nicht sehr zufällige" Zahlen.
$\Longrightarrow$ Knuth, ('81): a und m Coprime (keine gemeinsamen Faktoren außer 1), durch Probieren und Tests eine ausreichend gute Sequenz finden.
Bsp.:
$a = 2416, c = 374441, m = 1771875$ (unsigned)
$a = 9301, c = 49297, m = 233280$ (signed)
s.a. Press et al ('88), „Numerical Recipes"
Der folgende Trick von Schruge ('79) erlaubt die Berechnung von $i_n = (a \cdot i_{n-1} + c) \, mod \, m$ ohne Overflow selbst für große m. Sei hierzu m gegeben.
Definition:
$q := \llcorner \frac{m}{a} \lrcorner$, $r := m \, mod \, a$ ($\llcorner \lrcorner \; floor \; function$)
$\Rightarrow m = a \cdot q + r \; mit \; r < q$
damit

$$
\begin{aligned}
a \cdot i_n \, mod \, m &= (a \cdot i_n - \llcorner \frac{i_n}{q} \lrcorner \cdot m) \, mod \, m \\
&= ((a \cdot i_n - \llcorner \frac{i_n}{q} \lrcorner \cdot (a \cdot q + r)) \, mod \, m) \\
&= (a \cdot \underbrace{\underbrace{(i_n - \llcorner \frac{i_n}{q} \lrcorner)}_{i_n mod \, q \in [0, q-1]} / q -}_{<aq<m} \underbrace{\llcorner \frac{i_n}{q} \lrcorner}_{i_n (r/q) < i_n < m} \cdot r) \, mod \, m
\end{aligned}
$$

$\Rightarrow$Die Operation $a \cdot i_n \, mod \, q - \llcorner \frac{i_n}{q} \lrcorner r$ ergibt eine Zahl aus [-m,+m], man benötigt zwar Signed-Integer-Arithmetik (d.h.$m = 2^{31} - 1$ maximal), aber es gibt keinen Overflow.

$$
\Rightarrow (a \cdot i_n + c) \, mod \, m = \begin{cases} a(i_n \, mod \, q) - \llcorner \frac{i_n}{q} \lrcorner r, & falls \, nicht - negativ \\ a(i_n \, mod \, q) - \llcorner \frac{i_n}{q} \lrcorner r + m & sonst \end{cases}
$$

Diese Implementierung wurde bereits gründlich getestet und ist sehr weit verbreitet.
Bsp.: Lewis ('69) RNG:

$m = 2^{31} - 1 = 2147483647$
$a = 16807$
$c = 0$

[Knuth empfiehlt $c = 0$ für alle RNG mit $m = 2^n \pm 1$]
Dies führt auf

$q = 127773$
$r = 2896$

Mit einer Zykluslänge von $\sim 2 \cdot 10^{12}$ (in Ordnung für kleine Monte-Carlo Simulationen) und ca. 10^6 verschiedenen Zahlen in [0,1].
Beachte: $i_0 = 0 \Rightarrow \forall n \; i_n = 0$ vermeide unbedingt $i_0 = 0$!$\Rightarrow$nur $m - 1$verschiedene Integer.

3.4 Shuffling-Shema

Moderne State-of-the-Art MC-Sim. brauchen über 10^{12} Zufallszahlen. Z.B. für ein Ising-Modell mit $L = 1000$, $10^6 Swaps \Rightarrow 10^{15} Zufallszahlen$.
Weiterhin sind die über Lineare-Kongruenz RNG generierten PZ „versteckt korreliert", d.h sie liegen auf k-dimensionalen Hyperebenen im $(i_{n-1}, i_{n-2}, ..., i_{n-l})$-Raum.
Verbesserung: „Mischen" (Shuffling) der PZ eines Linearen-Kongruenz RNG.
Initialisiere hierzu ein N-elementiges Array $i[0], ..., i[N-1]$ mit RNG.

$$
\begin{aligned}
y &= rand() \,(\in \{0,1,...,m-1\}) \\
k &= \lfloor yN/m \rfloor \,(\in \{0,...,N-1\}) \\
x &= j[k] \\
j[k] &= rand() \\
&\Rightarrow \text{Zyklus } m^N
\end{aligned}
$$

3.5 Shift -Register-RNG

Zusammen mit Shuffling gute Alternative (etwas schneller) zu Lin.-Kongr-RNG. Sie arbeiten mit Bit-Shift-Operationen.
R^s Rechts-shift um s Plätze
L^t Links-shift um l Plätze
z.B. s=18, t=13 für 31-Bit-Generatoren oder s=15, t=17, für 32-Bit-Generatoren

$$
\begin{aligned}
i'_{n-1} &= \mathrm{i_{n-1}} \oplus \mathrm{R^s i_{n-1}} \\
i_n &= i'_{n-1} \oplus L^t i'_{n-1}
\end{aligned}
$$

„$\oplus$" bezeichnet hierbei ein „exklusives oder" (XOR). Dieser logische Operator genügt der folgenden Wahrheitstabelle:

	0	1
0	0	1
1	1	0

$A \oplus B$ ist also genau dann „1" (true), wenn entweder A oder B true sind, sonst „0" (false). Die Anwendung auf Integerwerte ist Bitweise zu verstehen, der komplette Ablauf geht aus dem folgenden Beispiel (hoffentlich) verständlich hervor:

1	0	1	1	1	0	0	1	i_{n-1}
					s	$\longrightarrow$	$\longrightarrow$	
0	0	0	1	0	1	1	1	$R^s i_{n-1}$
$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	
1	0	1	0	1	1	1	0	i'_{n-1}
$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	t				
1	1	1	0	0	0	0	0	$L^t i'_{n-1}$
$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	
0	1	0	0	1	1	1	0	i_n

3.6 Lagged-Fibonacci-Generatoren

Sogenannte Lagged-Fibonacci-Generatoren besitzen den Vorteil geringen Ressourcenbedarfs bei gleichzeitig hoher Zyklenlänge. Sie sind von der Form (3.2). Ihre theoretischen Grundlagen sind bisher leider nur unzureichend ergründet, sodass ihre Qualität nur schwer abgeschätzt werden kann. Die nächste PZ geht aus einer einfachen Kombination zweier vorhergehender hervor (vgl. die Entstehung der Fibonacci-Zahlen):

$$i_n = (i_{n-r} \circ i_{n-s}) \, mod \, m.$$

Der Operand „$\circ$" könnte hierbei z.B. $+, -, \cdot$ oder $\oplus$ (XOR) sein, letztere Wahl hätte den Vorteil auf den *mod*-Operator verzichten zu können (Lewis & Payne ['73], „verallgemeinerte feedback Shift Register Generatoren"). Häufige Wahlen für r und s sind bspw.

$r = 418$, $s = 1279$,
$r = 147$, $s = 250$,
$r = 24$, $s = 55$.

Erfahrungsgemäß sind für kleine r und s Addition und Multiplikation besser („zufälliger"). Da zum „Anwerfen" von RNG dieser Bauart $max(r, s)$ Zufallszahlen benötigt werden, muss man zur Initialisierung einen anderen RNG gebrauchen, oft Lineare-Kongruenz-Generatoren. Zur Abspeicherung ist der Circular FIFO Buffer gebräuchlich, ein eindimensionaler Array, dessen Indexbereich einen geschlossenen Kreis bildet.

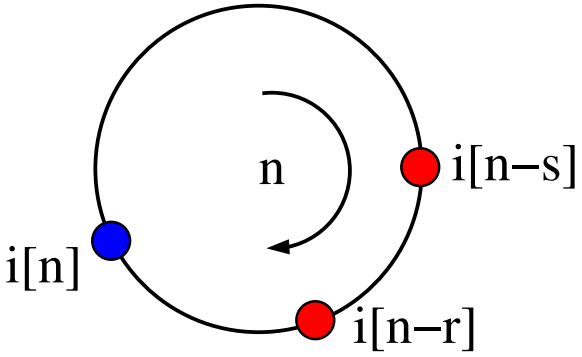


Abbildung 1: Der Circulare FIFO-Buffer

Die Verwendung der Multiplikation besitzt leider den Nachteil, dass irgendwann alle generierten PZ gerade oder ungerade sind. Für die Addition kann man dies noch vermeiden, wenn man darauf achtet, den RNG nicht ausschließlich mit geraden oder ungeraden Zahlen zu initialisieren.

3.7 Nicht-gleichverteilte Zufallszahlen

Fast alle bisher vorgestellten RNG erzeugten gleichmäßig verteilte PZ aus dem Intervall $[0, 1]$, d. h., die Wahrscheinlichkeit $P(I)$, dass eine Zahl im Intervall $I = [a, b] \subset [0, 1]$ „gewürfelt" wird, ist gleich der Intervallbreite $P(I) = b - a$. In konkreten Anwendungen ist es jedoch oft notwendig, Zufallszahlen zu kennen, die nicht gleichverteilt sind, sondern einer gegebenen Verteilung $p(z)$ genügen. Um nicht für jede Verteilungsfunktion einen eigenen RNG entwickeln zu müssen, versucht man, die zu einer gleichmäßigen Verteilung gehörenden Zufallszahlen auf $p(z)$ umzurechnen.
Eine Möglichkeit ist die sog. Invertierungsmethode. Für eine gegebene Wahrscheinlichkeitsdichte definiert man zunächst

$P(z) =$ Wahrscheinlichkeit, dass Zz. kleiner $z = Prob(Z \leq z) = \int_{-\infty}^{z} dz' p(z')$.

Ziel ist es nun eine Funktion $g(x)$ zu finden, so daß nach der Trafo $Z = g(x)$ die Werte von Z gemäß $P(z)$ verteilt sind. Unter der Annahme, dass $g(x)$ invertierbar und streng monoton wachsend ist, folgt:

$P(z) = Prob(Z \leq z) = Prob(g(u) \leq z) = Prob(u \leq g^{-1}(z))$.

Für gleichverteilte Zz. u ist $F(U) = Prob(u \leq U) = u$, sofern $u \in [0, 1]$. Damit folgt

$P(z) = g^{-1}(z) \Longleftrightarrow g(z) = P^{-1}(z)$.

Beispiele:

1. Exponentialverteilung $p(z) = \lambda e^{-\lambda z}$
   $\Rightarrow P(z) = 1 - e^{-\lambda z}$
   $\Rightarrow$erzeuge gleichverteilte Zz. $U \in [0, 1]$, wähle $Z = \frac{-ln(1-U)}{\lambda}$
2. Lorentz-Verteilung $p(z) = \frac{1}{\pi} \frac{\Gamma}{\Gamma^2 + z^2}$
   $\Rightarrow P(z) = \frac{1}{\pi}[arctan(\frac{x'}{\Gamma})]_{-\infty}^{z} = \frac{1}{2} + \frac{1}{\pi} arctan(\frac{z}{\Gamma})$
   $\Rightarrow$erzeuge gleichverteilte Zz. $U \in [0, 1]$, wähle $Z = \Gamma \cdot tan(\pi(U - \frac{1}{2}))$

Diese Methode funktioniert jedoch nur, wenn die Verteilung $P(z)$ invertierbar ist. Die sog. Rejection-Methode kann hier Abhilfe verschaffen. Hierzu muss die Wahrscheinlichkeitsdichte $p(z)$ den folgenden Bedingungen genügen:

$p(z) = 0 \; für \; z \notin [x_0, x_1] \; und \; p(z) \leq p_{max} \forall z$.

Der Ablauf dieser Methode wird durch folgenden Pseudo-Quelltext erläutert, *rand*() erzeugt bei Aufruf eine gleichmäßig verteilte PZ aus $[0, 1]$, x ist die letztlich generierte Zufallszahl.

```
not-found=1
while{not-found}
{
u1 = rand();
x = x0 + (x1-x0)*u1;
u2 = rand();
y=pmax*u2;
if(y<=p(x)) not-found = 0;
}

return(x);
```

Das heißt es wird nur dann eine entsprechend skalierte Zz. aus dem Intervall $[x_0, x_1]$ zurückgeliefert, wenn man die Fläche unter dem Graphen von $p(z)$ „trifft".

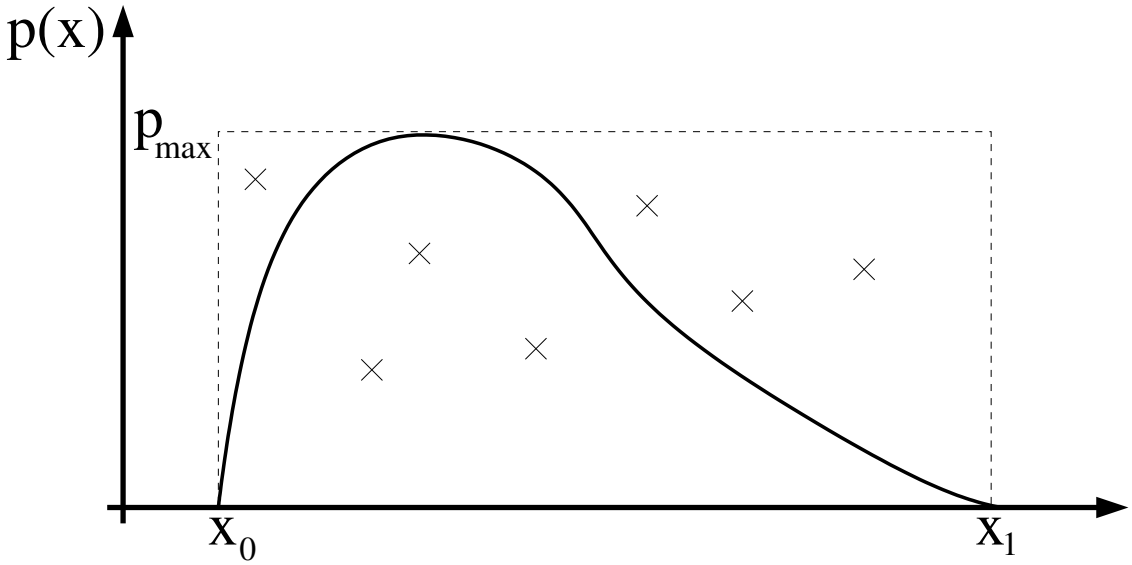


Abbildung 2: Die Rejection Methode. Nur solche Zufallszahlen innerhalb der Fläche werden akzeptiert.

Diese Art der Umrechung funktioniert, da $p_{gen}(x)$ (Verteilung die der rand()-Methode zugehörig ist), $P_{accept}(x)$ (Wahrscheinlichkeit, dass bei x eine PZ akzeptiert wird) und $p(x)$ folgenden Relationen genügen:

$$p_{gen}(x) = \frac{dx}{x_1 - x_0}$$
$$P_{accept}(x) = \frac{p(x)}{p_{max}}.$$

Und damit ($\tilde{p}$ =generierte Verteilung):

$$\begin{aligned}\tilde{p}(x)dx &= p_{gen}(x) \cdot p_{accept}(x)dx \\ &= \frac{p(x)dx}{p_{max} \cdot (x_1 - x_0)}.\end{aligned}$$

Daher ist $\tilde{p}$ proportional zur vorgegebenen Verteilung und stimmt nach Normierung mit dieser überein. Diese Methode ist leider nicht sehr effektiv, da eine große Zahl der von *rand*() erzeugten PZ nötig um eine „richtige" PZ, die $p(z)$ entspricht, zu generieren. Konkret lässt sich die Zahl benötigter Aufrufe abschätzen durch

$$\#total\ calls = \frac{2 \cdot p_{max}(x_1 - x_0)}{\int_{x_0}^{x_1} p(x)dx}.$$

Für ein auf ein zentriertes Intervall der doppelten Standardabweichung σ beschränktes Gaußprofil ($\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$) ergibt sich z. B.:

$$\#total\ calls = \frac{24}{\int_{-\sigma}^{\sigma} e^{-\frac{x^2}{2}} dx} \approx 9.57.$$

Aus diesem Grund werden Rejection-Methoden nur ungern verwandt, häufiger sind hybride Verfahren im Einsatz die die Vorteile beider Abläufe ausnutzen können.

3.8 Die Gaußverteilung

$$P_\sigma(z) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(2-m)^2}{2\sigma^2})$$

Normalverteilung: $m = 0,\ \sigma = 1$, sonst $\sigma \rightarrow \sigma z + m$.
Einfachste Methode zur Realisierung: Zentraler Grenzwertsatz.
Seien hierzu $u_1, u_2..., u_N$ unabhängige Zz. mit Mittelwert m, Varianz v
$\Rightarrow P(z = \sum_{i=1}^{N} u_i) \overset{N\rightarrow\infty}{\longrightarrow} P_\sigma(z)$ mit $\tilde{m} = Nm,\ \tilde{\sigma} = Nv$.
Wähle z. B. für ein gleichverteiltes $u_1 \in [0, 1] \quad (m = 0.5,\ v = 12) \quad N = 12$, dann ist $P(z) := \sum_{i=1}^{6} u_i - 6$ annähernd normalverteilt.
Nachteil: 12 Zz. müssen für eine normalverteilte Zz. „verbraucht" werden, Werte größer als sechs treten nicht auf.
Bessere Methode nach Box-Müller (arbeitet exakt):
Erzeuge zuerst zwei gleichverteilte Zz. u_1, u_2, dann sind

$$z_1 = \sqrt{-2\log(u_1)}\cos(2\pi u_2)$$
$$z_2 = \sqrt{-2\log(u_1)}\sin(2\pi u_2)$$

normalverteilt.

3.9 Diskrete (nicht gleichförmige) Verteilungen

Nachdem bisher ausschließlich kontinuierliche Verteilungen betrachtet wurden, werden in diesem Abschnitt Methoden vorgestellt, um diskrete Verteilungen zu implementieren. Man geht dabei immer von endlich vielen Ergeinissen mit Wahrscheinlichkeiten $p_1, ..., p_N$ aus, die der Normierungskondition $\sum_{i=1}^{N} p_i = 1$ genügen.

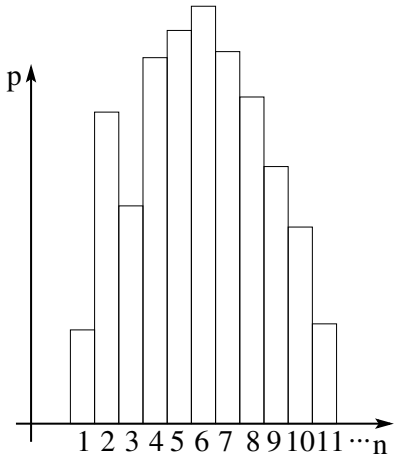


Abbildung 3: Diskrete Wahrscheinlichkeitsverteilung.

Eine direkte Methode dieses Verfahren umzusetzen kann durch leichte Modifikation des Rejection-Vefahrens erhalten werden. Der folgende Code stellt diese Möglichkeit dar,

```
rand()
```

erzeugt dabei eine PZ aus $[0, 1]$:

```
pmax = max{p1,...,pN}

1

n = (int) (N+1)*rand()
x = pmax*rand()

if(x>pn) goto 1

return n
```

Wie im kontinuierlichen Fall geht auch hier die Rejection-Methode mit einem großen Verbrauch an Zufallszahlen einher. Wesentlich effizienter ist das sog. Tower-Sampling. Hier werden die Wahrscheinlichkeiten $p_1, ..., p_N$ sukzessive zu den $q_j := \sum_{i=1}^{j} p_i$ addiert und in einem Array abgespeichert (der „Tower").

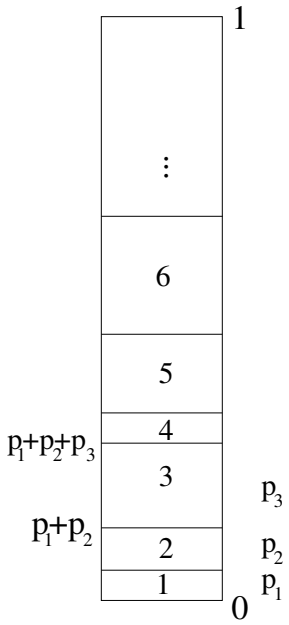


Abbildung 4: Der „Tower".

Anschließend wird eine gleichverteilte Zz. *rand* aus $[0, 1]$ mit den Werten im Tower verglichen, es tritt das Ereignis j ein, für das gilt:

$$q_{j-1} < rand < q_j. \tag{3.4}$$

Somit wird nur eine einzige Zz. „verbraucht". Der folgende Code zeigt eine Möglichkeit der Implementierung:

```
input{p1,...,pN}

q[0] = 0

for(l=1...k) q[l] = q[l-1] + pl
x= rand()

find j with q[j-1]<x<q[j]

output j
```

Zwar spart das Tower-Sampling an Zufallszahlen, jedoch wird dieser Vorteil mit dadurch erkauft, einen Array der Größe N nach einem j zu durchsuchen, welches (3.4) genügt. Ein effizientes Verfahren stellt hierbei die Bisektionsmethode dar, welche den Array nach jedem Suchschritt „halbiert". Es wird durch den Code

```
input x, {q[0],q[1],...,q[N]}

nmin = 0
nmax = N+1
not_found = 1

while(not_found)
{
n=(nmin+nmax)/2

if(q[n]<x) nmin = n
else if(q[n-1]>) nmax = n
else output n, not_found = 0
}
```

erstellt.

3.10 Statistische Tests

1. Gleichförmigkeit der Zz.-Sequenz
   Standard X^2-Test. n Zz. aus $[0,1]$, Multipliziert mit $v \Rightarrow$ Int. aus $[0,v]$
   Anzahl des Auftretens v. jedem v-Bin, vgl. mit theoretischer Erwartung$\rightarrow$ X^2-Test.

2. Serielle Korrelationen: Auftreten v d-Tupeln von n Zz. aus $[0,1]$.
   Z.B. $d=2$: Tabelliere $\forall i \in \{0,...,n-1\} : (X_{2i}, X_{2i+1})$
   $\Rightarrow$Jedes d-Tupel hat Wahrscheinlichkeit $v-d,\ v = \#$ Bins im Intervall $[0,1]$
   $\rightarrow$dann X^2-Test.

3. Gap-Test: auf Gleichförmigkeit
   Sei $x_1 \in [\alpha,\beta]$, beobachte Anzahl darauffolgender Zz. $x_{i+1},...,x_{i+j-1} \notin [\alpha,\beta]$
   Wenn $x_{i+j} \in [\alpha,\beta] \Rightarrow$Gap j (def. maximale Gap-Länge)
   $\rightarrow$dann X^2-Test.

4. Maximum-t-Test: $x_i \in [0,1[, i = 1,...,n$
   Untersequenzen der Länge t, Maximum jeder Untersequenz$\rightarrow$ x^t-Verteilung

5. Kollisionstest: $n = \#Zz. \ll \#Bins = w$, zähle wie oft Zz. in dasselbe Bin fallen!

6. Run-Test: Anzahl von aufsteigenden od. absteigenden Sequenzen der Länge $1 \leq i \leq l$ in $x_1, x_2, ..., x_n$
   z. B. $l = 6$

7. Park-Test:Wähle zufällig Punkte im d-dim Raum und ziehe um jeden einen Kreis mit vorgegebenem Radius$\rightarrow$"parke Autos", so viele wie möglich.

8. Spektraltest: Eigentlich für LCG, testet maximale Distanz zwischen Hyperebenen (je kleiner, desto besser)

Visuelle Tests

1. Plotte Verteillung der Zz., suche geordnete Strukturen, für LCG Hyperebenen

2. Binäre Sequenzen als Comp-Werte in Ebene$\rightarrow 2d$ Ising Modell bei $T = \infty$

3. Plotte $|x_i - x_j| \forall i,j \in \{1,...,n\}$ in Grey-Scale in $2d$

3.11 Ein einfacher stochastischer Prozess - Der Random Walk

Der Random-Walk (dt. „Irrweg") in diskreter Zeit. Ein Betrunkener bewegt sich entlang einer Linie und macht in jeder Sekunde einen Schritt, entweder vorwärts oder rückwärts mit gleicher Wahrscheinlichkeit $\frac{1}{2}$.

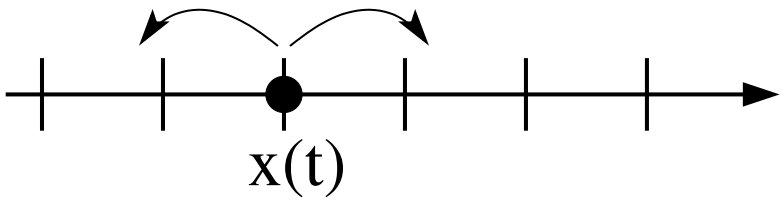


Abbildung 5: Der Random Walk.

Startpunkt: $x(t=0)=0$,$w(x \to x') = \begin{cases} \frac{1}{2} & \text{, wenn } x' = x \pm 1 \\ 0 & \text{, sonst} \end{cases}$
$P_t(x)$ =Wahrscheinlichkeit dafür, dass Walker zur Zeit t an Ort x zu finden ist.
Mastergleichung:

$$P_{t+1}(x) = -\sum_{x'} w(x \to x')P_t(x) + \sum_{x'} w(x' \to x)P_t(x)$$
$$= -P_t(x) + \frac{1}{2}(P_t(x-1) + P_t(x+1))$$

Offenbar ist Position x_t des Walkers nach t Schritten eine stochastische Variable, die Summe von t unabhängigen Schritten $S_{t'} \in \{-1, +1\}$
$x_t = \sum_{t'=1}^{t} S_{t'},\ Prob(S_{t'} = \pm 1) = \frac{1}{2}.$
Dies führt auf die Binomialverteilung
$P_t(x) = \frac{1}{2^t}\binom{t}{\frac{1}{2}(t-x)}.$

Für große Zeiten geht diese Verteilung wegen $\overline{S}_{t'} = 0$ und $\overline{S^2}_{t'} = 1$ m.h. des zentralen Grenzwertsatzes über in eine Gaußverteilung

$$\lim_{t\to\infty} P_t(\frac{x}{\sqrt{t}}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}.$$

Wichtig: Auftreten einer Längeskala $\sqrt{t}$ = typische Entfernung des Betrunkenen von der Kneipe nach t Schritten! (Es gilt: $\Delta x(t) = \sqrt{< x^2(t) >} = \sqrt{2Dt}$).
Dies entspricht einem physikalischem Diffusionsprozess.
In kont. Zeit ist zu beachten, dass Raten statt Wahrscheinlichkeiten auftreten: $\frac{\partial}{\partial t}P(x,t) = \frac{1}{2}(P(x,t-1) - 2P(x,t) + P(x,t+1)$
Zusätzlich mit kont. Raum: $\frac{\partial}{\partial t}P(x,t) = D\frac{\partial^2}{\partial x^2}P(x,t)$ bzw. in $3d$: $\frac{\partial}{\partial t}P(x,t) = D\Delta P(x,t)$
In der konkreten Simulation muss über viele Realisierungen gemittelt werden, um die analytischen Ergebnisse wiederzufinden.

4 Simulation gekoppelter chemischer Reaktionen

(Gillespie Algorithmus)

Das zu untersuchende Problem lässt sich im Allgemeinen wie folgt formulieren: In einem gegebenen Volumen V sind N verschiedene chemisch reaktive Spezies S_i, $i \in \{1, \ldots, N\}$ enthalten mit den jeweiligen Teilchenzahlen $X_i, i \in \{1, \ldots, N\}$. Alle Spezies S_i können chemisch reagieren, wobei die Form der Reaktionen einer der folgenden Typen $R_\mu, \mu \in \{1, \ldots, M\}$ entspricht:

$$\begin{array}{ccccc} * & \rightarrow & \text{Produkte} & \\ S_j & \rightarrow & " & \\ S_j + S_k & \rightarrow & " & j \neq k \\ 2S_j & \rightarrow & " & \\ S_i + S_j + S_k & \rightarrow & " & i \neq j \neq k \neq i \\ S_j + 2S_k & \rightarrow & " & j \neq k \\ 3S_j & \rightarrow & " & \end{array}$$

Die entstehenden Produkte enthalten erneut nur Kombinationen der Spezies S_j. Weiterhin läuft jede Reaktion nur in einer Richtung ab, d.h. reversible Reaktionsabläufe müssen als zwei parallel ablaufende, entgegengesetzt gerichtete Reaktionen aufgefasst werden.
Die Geschwindigkeit, mit welcher jede der Reaktionen abläuft, soll durch einen reaktionsspezifischen, konstanten Parameter $c_\mu, \mu \in \{1, \ldots, M\}$ charakterisiert werden. Die fundamentale Hypothese der dem Gillespie-Algorithmus zugrundeliegenden numerischen Simulation der chemischen Kinetik kann dann wie folgt formuliert werden:
$c_\mu \delta t =$ durschnittliche Wahrscheinlichkeit (in erster Ordnung in δt), dass eine bestimmte Kombination von Reaktanden der Reaktion $R_\mu$im nächsten Zeitintervall δt entsprechend miteinander reagieren.
Die obige Wahrscheinlichkeit ist somit exakt gegeben durch $c_\mu \delta t + o(\delta t)$ mit $\frac{o(\delta t)}{\delta t} \rightarrow 0$ für $\delta t \rightarrow 0$.
Hierbei ist es wichtig, zu erwähnen, dass die Wahrscheinlichkeit für das Auftreten von mehr als einer Reaktion im Zeitintervall δt als von der Ordnung $o(\delta t)$ angenommen wird, somit verschwindet für $\delta t \rightarrow 0$.
Ziel ist es nun, unter Annahme dieser Grundhypothese einen Algorithmus abzuleiten, der es ermöglicht, die zeitliche Entwicklung der $\{X_i\}$ bei Kenntnis ihrer Anfangswerte $\{X_i^0\}$ sowie aller Koeffizienten $\{c_\mu\}$ zu berechnen.
Der übliche Weg, diese Aufgabenstellung analytisch anzugehen läuft über das Aufstellen der sog. Mastergleichung, die die stochastische Funktion
$\mathcal{P}(\{X_i\}; t) :=$Wahrscheinlichkeit, dass zum Zeitpunkt t X_1 Moleküle der Spezies S_1, X_2 Moleküle der Spezies S_2, usw... im Volumen V sind
enthält. Die Mastergleichung beschreibt dann die Zeitentwicklung von $\mathcal{P}$ durch

$$\frac{d}{dt}\mathcal{P}(\{X_i\}; t) = -\sum_{\{X_i'\}} w(\{X_i\} \rightarrow \{X_i'\})\mathcal{P}(\{X_i\}; t) + \sum_{\{X_i'\}} w(\{X_i'\} \rightarrow \{X_i\})\mathcal{P}(\{X_i'\}; t).$$

Aus $\mathcal{P}$ können die weiteren Momente der Teilchenzahlen über

$$X_i^{(k)} = \sum_{\{X_i\}} X_i^k \cdot \mathcal{P}(\{X_i\}; t)$$

gewonnen werden. Von großer Bedeutung sind die Momente für $k = 1$ (Mittelwert) und $k = 2$ (Varianz, misst Fluktuationen um den Mittelwert). Im Regelfall ist die Lösung der Mastergleichung nur in sehr einfachen Fällen möglich und selbst ihre numerische Behandlung ist schwierig bis unmöglich. Aus diesem Grund verzichtet man in konkreten Anwendungen oft ganz auf die Lösung des Problems über die Mastergleichung und wählt einen stochastischen Zugang. Ausgangspunkt ist hierfür die sog. Reaktionswahrscheinlichkeitsdichtefunktion $P(\tau, \mu)$, welche definiert wird über

$P(\tau, \mu)\, d\tau$ =Wahrscheinlichkeit zur Zeit t das die nächste Reaktion in V im differentiellen Zeitintervall $(t + \tau, t + \tau + d\,\tau)$ stattfindet und vom Typ $R_\mu$ist.
Im Folgenden wird ein exakter, analytischer Ausdruck für $P(\tau, \mu)$ hergeleitet. Hierzu definiert man zunächst die M reaktionsspezifischen Variablen $h_\mu$über

h_μ = Anzahl möglicher Reaktanden-Kombinationen in V für die Reaktion $R_\mu$zur Zeit t.
Für die einzelnen oben definierten Reaktionen sind die $h_\mu$nur von den $\{X_i\}$ abhängig und können über einfache kombinatorische Überlegungen abgeleitet werden:

$$\begin{aligned} h_\mu &= 1 \\ h_\mu &= X_j \\ h_\mu &= X_j X_k \\ h_\mu &= \tfrac{1}{2} X_j (X_j - 1) \\ h_\mu &= X_i X_j X_k \\ h_\mu &= \tfrac{1}{2} X_j X_k (X_k - 1) \\ h_\mu &= \tfrac{1}{6} X_j (X_j - 1)(X_j - 2) \end{aligned}$$

Da im Limes $\delta t \to 0$ nur eine einzige Reaktion im Zeitintervall δt stattfinden kann, dürfen die zu jeder möglichen Kombination von Reaktanden gehörenden Wahrscheinlichkeiten zur Gesamtwahrscheinlichkeit für die Reaktion R_μ addiert werden:
$h_\mu c_\mu \delta t$ = Wahrscheinlichkeit (in erster Ordnung in δt), dass im Volumen V im nächsten Zeitintervall δt eine Reaktion vom Typ R_μ stattfinden wird.
Bezeichnet nun $P_0(\tau)$ die Wahrscheinlichkeit, dass zur Zeit t keine Reaktion im Zeitintervall $(t, t + d\tau)$ stattfindet, so kann formal ein Ausdruck für $P(\tau, \mu)$ angegeben werden:
$P(\tau, \mu)d\tau = P_0(\tau) \cdot h_\mu c_\mu d\tau$.

Im nächsten Schritt soll nun $P_0(\tau)$ berechnet werden. Hierzu unterteilt man zunächst das Intervall $(t, t + d\tau)$ in K Unterintervalle der gleichen Länge $\epsilon = \frac{\tau}{K}$. Die Wahrscheinlichkeit, das keine der Reaktionen $\{R_\mu\}$im ersten Unterintervall $(t, t + \epsilon)$auftritt kann dann durch

$$\prod_{\nu=1}^{M} [1 - h_\nu c_\nu \epsilon + o(\epsilon)] = 1 - \sum_{\nu=1}^{M} h_\nu c_\nu \epsilon + o(\epsilon)$$

angegeben werden. Für alle folgenden Intervalle $(t + \epsilon, t + 2\epsilon), (t + 2\epsilon, t + 3\epsilon) \ldots$ folgt analog der gleiche Ausdruck, da es insgesamt K solcher Intervalle gibt ergibt sich $P_0(\tau)$ zu

$$\begin{aligned} P_0(\tau) &= \left[1 - \sum_{\nu=1}^{M} h_\nu c_\nu \epsilon + o(\epsilon) \right]^K \\ &= \left[1 - \sum_{\nu=1}^{M} h_\nu c_\nu \frac{\tau}{K} + o(K^{-1}) \right]^K \\ &= \left[1 - \left(\sum_{\nu=1}^{M} h_\nu c_\nu \tau + \frac{o(K^{-1})}{K^{-1}} \right) \frac{1}{K} \right]^K \end{aligned}$$

und im Limes$K \to \infty$ folgt schließlich

$$P_0(\tau) = exp \left[- \sum_{\nu=1}^{M} h_\nu c_\nu \tau \right]$$

und damit

$$P(\tau, \mu) = h_\mu c_\mu exp \left[- \sum_{\nu=1}^{M} h_\nu c_\nu \tau \right].$$

Mit dieser Formel lässt sich $P(\tau, \mu)$ im kontinuierlichen Wertebereich $0 \leq \tau \leq \infty$ und für alle natürlichen Zahlen μ mit $1 \leq \mu \leq M$ berechnen, für alle anderen Werte von τ oder μ verschwindet $P(\tau, \mu)$.
Die korrekte Normierung von $P(\tau, \mu)$ lässt sich einfach nachrechnen:

$$\int_0^\infty d\tau \sum_{\mu=1}^{M} P(\tau, \mu) = \sum_{\mu=1}^{M} h_\mu c_\mu \int_0^\infty d\tau exp \left[- \sum_{\nu=1}^{M} h_\nu c_\nu \tau \right] = 1.$$

4.1 Gillespie's „Direct Method“

In diesem Abschnitt soll nun basierend auf der oben hergeleiteten Wahrscheinlichkeitsdichte $P(\tau, \mu)$ ein Algorithmus aufgestellt werden, der die numerische Berechnung der Reaktionskinetik erlaubt. Die Dichtefunktion $P(\tau, \mu)$ allein enthält alle nötigen Informationen, um den stochastischen Reaktionsprozess mittels einer Monte-Carlo-Simulation zu behandeln. Ausgangspunkt jedes Iterationschrittes sind die zur aktuellen Zeit t als bekannt vorausgesetzten Werte der Reaktionskonstanten $\{c_\mu\}$ sowie der Teilchenzahlen $\{X_i\}$. Mit ihrer Hilfe kann zunächst $P(\tau, \mu)$ zum Zeitpunkt t berechnet werden und durch Monte-Carlo-Methoden ein Paar von Zufallszahlen (τ, μ) generiert werden, die der Verteilung $P(\tau, \mu)$ genügen. Das Paar (τ, μ) beschreibt nun, welche Reaktion als nächstes wann stattfindet. Abhängig vom Typ der Reaktion werden die Teilchenzahlen $\{X_j\}$ aktualisiert sowie die Zeitskala um τ inkrementiert. Ausgehend von diesen aktualisierten Teilchenzahlen kann nun vollkommen analog der nächste Iterationsschritt zur Systemzeit $t + \tau$ erfolgen. Man erhält zu jedem ermittelten Zeitschritt alle Anzahlen aller Spezies.
Von großer Bedeutung ist die numerische Prozedur, mit welcher bei Kenntnis der Funktion $P(\tau, \mu)$ das Tupel (τ, μ) generiert wird. Eine Möglichkeit stellt die sog. „Direct-Method“ dar, die im Folgenden erläutert wird. Diese Methode macht sich zunutze, dass die von zwei Variablen abhängige Dichtefunktion $P(\tau, \mu)$ mithilfe bedingter Wahrscheinlichkeiten als Produkt zweier Wahrscheinlichkeitsdichten $P_1(\tau)$, $P_2(\mu|\tau)$ geschrieben werden kann, die jeweils nur von einer Variablen abhängen:

$$P(\tau, \mu) = P_1(\tau) \cdot P_2(\mu|\tau).$$

Dabei steht $P_1(\tau)$ für die Wahrscheinlichkeit, dass die nächste Reaktion im Zeitintervall $[t + \tau, t + \tau + d\tau]$ stattfindet (unabhängig davon, um welche Reaktion es sich genau handelt) und $P_2(\mu|\tau)$ für die Wahrscheinlichkeit, dass die nächste Reaktion vom Typ $R_\mu$ist, unter der Bedingung, dass sie sich zum Zeitpunkt $t + \tau$ ereignet. $P_1(\tau)$ kann direkt aus $P(\tau, \mu)$ durch Summation über alle möglichen μ-Werte gewonnen werden:

$$P_1(\tau) = \sum_{\mu=1}^{M} P(\tau, \mu),$$

was durch einsetzen unmittelbar auf einen Ausdruck für $P_2(\mu|\tau)$ führt:

$$P_2(\mu|\tau) = \frac{P(\tau, \mu)}{\sum_\nu P(\tau, \nu)}.$$

Mit der im vorigen Abschnitt hergeleiteten Formel für $P(\tau, \mu)$ erhält man schließlich:

$$P_1(\tau) = a\, exp(-a\tau)$$
$$P_2(\mu|\tau) = \frac{a_\mu}{a}$$

mit den Abkürzungen $a_\mu := h_\mu c_\mu$und $a = \sum_\mu h_\mu c_\mu = \sum_\mu a_\mu$. Mittels dieser beiden Dichtefunktionen kann nun das Tupel (τ, μ) konstruiert werden, welches der Verteilung $P(\tau, \mu)$ gehorcht. Zu Beginn jedes Iterationsschrittes ermittelt man zuerst alle Reaktionsraten a_μ und hierüber die Funktionen $P_1(\tau)$ und $P_2(\mu|\tau)$. Anschließend generiert man stochastisch eine Zeit τ, die der Verteilung $P_1(\tau)$ genügt, danach eine natürliche Zahl μ entsprechend der Verteilung $P_2(\mu|\tau)$.

4.2 Die First-Reaction-Methode

Diese Art der Implementierung wählt einen anderen Ansatz zur Produktion des Paares (τ, μ) von Zufallszahlen. Da $a_\mu dt$ die Wahrscheinlichkeit für das Stattfinden der Reaktion R_μ in dt ist, ist

$$\lim_{K\to\infty} (1 - \frac{\tau}{K})^K a_\mu dt,$$

die Wahrscheinlichkeit dafür, dass zunächst keine R_μ-Reaktion in $[0, \tau]$ erfolgt, aber anschließend im Intervall $[\tau, \tau + dt]$ abläuft. Hieraus erhält man
$P_\mu(\tau)d\tau = exp(-a_\mu\tau)a_\mu d\tau$ = Wahrscheinlichkeit, dass eine R_μ-Reaktion zur Zeit τ stattfindet, vorausgesetzt, dass keine andere Reaktion die Teilchenzahl zuvor geändert hat.

In einem ersten Schritt generiert man zuerst vorläufige Reaktionszeiten $\tau_\nu = -\frac{1}{a_\nu} \ln x_\nu,\ \nu \in \{1, ..., M\}$ mithilfe einer gleichverteilten Zufallszahl x_ν aus dem Intervall $[0, 1]$. Im nächsten Schritt wird aus allen Reaktionszeiten die kleinste ermittelt und μ gleich demjenigen ν gesetzt, für das das zugehörige τ_ν am kleinsten ist. Somit gelangt man zu einem Tupel (τ, μ) von Zz.. Es bleibt zu zeigen, dass diese tatsächlich der eingangs beschriebenen Verteilung genügen. Bezeichne hierzu $\widetilde{P}(\tau, \mu)$ die generierte Wahrscheinlichkeitsverteilung, dann gilt:

$$\widetilde{P}(\tau, \mu)d\tau = Prob(\tau < \tau_\mu < \tau + d\tau) \times Prob(\tau_\nu > \tau,\ \forall \nu \neq \mu)$$

Da

$$Prob(\tau < \tau_\mu < \tau + d\tau) = exp(-a_\mu \tau) a_\mu d\tau$$

und

$$\begin{aligned} Prob(\tau_\nu > \tau,\ \forall \nu \neq \mu) &= Prob(-\frac{1}{a_\nu} \ln(x_\nu) > \tau \forall \nu \neq \mu) \\ &= Prob(x_\nu < e^{-a_\nu \tau} \forall \nu \neq \mu) \\ &= \prod_{\nu \neq \mu} Prob(x_\nu < e^{-a_\nu \tau}) \\ &= \prod_{\nu \neq \mu} e^{-a_\nu \tau} \end{aligned}$$

ergibt sich insgesamt

$$\begin{aligned} \widetilde{P}(\tau, \mu)d\tau &= e^{-a_\mu \tau} a_\mu d\tau \cdot \prod_{\nu \neq \mu} e^{-a_\nu \tau} \\ &= a_\mu d\tau \cdot \exp(-\sum_{\nu 1}^{M} a_\nu \tau) \\ &= P(\tau, \mu)\ q.e.d. \end{aligned}$$

Wir betrachten nun folgendes Beispiel für ein konkretes Reaktionsschema in den Spezies X, Y und Z:

$$\begin{aligned} X &\rightleftharpoons Y \\ 2X &\rightleftharpoons Z \\ W + X &\rightleftharpoons 2X \end{aligned}$$

mit den entsprechenden Raten $c_1, c_2, ...c_6$. Die geraden Indixes beziehen sich auf die Hin-, die ungeraden auf die Rückreaktionen. Deterministisch wäre das folgende System nichtlinearer Gleichungen zu lösen:

$$\begin{aligned} \frac{dW}{dt} &= -c_5 WX + \frac{1}{2} c_6 X^2 \\ \frac{dX}{dt} &= -c_1 X + c_2 Y - c_3 X^2 + 2c_4 Z + c_5 WX - \frac{1}{2} c_6 X^2 \\ \frac{dY}{dt} &= c_1 X - c_2 Y \\ \frac{dZ}{dt} &= \frac{1}{2} c_3 X^2 - c_4 Z \end{aligned}$$

Und die ersten Glieder der Mastergleichung würden

$$\begin{aligned} \frac{d\mathcal{P}(W, X, Y, Z; t)}{dt} &= c_1\{(X+1)\mathcal{P}(W, X+1, Y-1, Z; t) - X\mathcal{P}(W, X, Y, Z; t)\} \\ &+ c_2\{(Y+1)\mathcal{P}(W, X-1, Y+1, Z; t) - Y\mathcal{P}(W, X, Y, Z; t)\} \\ &+ ... \end{aligned}$$

4.3 Ausblick: Next-Subvolume Method

Die bisher vorgestellten Methoden erlauben eine Simulation räumlich homogener Systeme, das heißt die Diffusionskonstanten sind für die betrachteten Ausdehnungen der Systeme derart groß, dass zu jedem Zeitpunkt von einer gleichmäßigen Verteilung aller Reaktanden ausgegangen werden kann. Diese Annahme ist jedoch nicht immer berechtigt, in vielen Fällen interessiert man sich auch für die spatiale Auflösung der beteiligten Spezies. Hierzu wurde von *Johan Elf et al* eine Methode entwickelt, welche Gillespies Algorithmus auf die Behandlung solcher Probleme anwendbar macht. Zunächst wird das zu untersuchende Volumen in zahlreiche, indexierte Subvolumen unterteilt, welche so klein gewählt werden, dass die Spezies in ihnen erneut als homogen verteilt angenommen werden können.

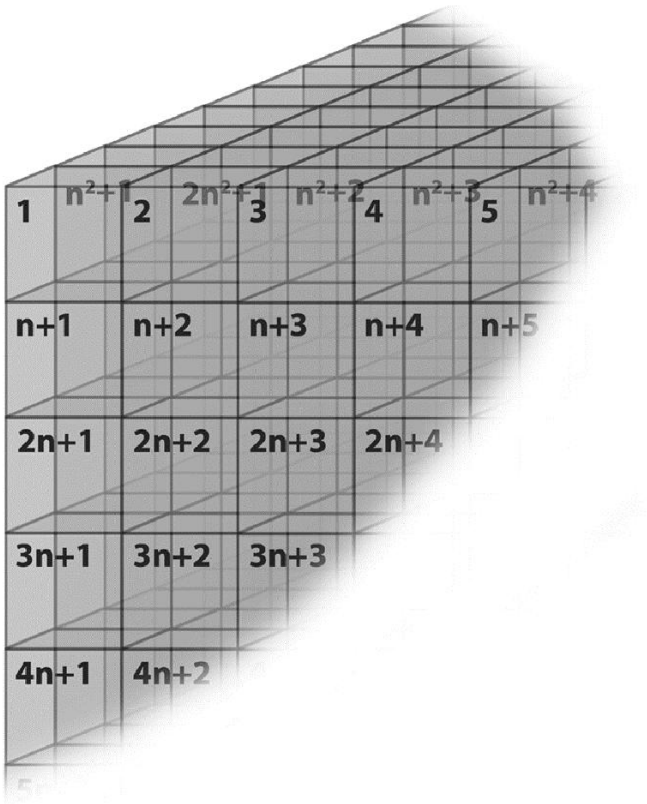


Abbildung 6: Schematische Darstellung der Unterteilung in SV. Abbildung aus „Mesoscopic reaction-diffusion in intracellular signaling, Johan Elf et al 2003".

In jedem SV wird nun seperat Gillespies „Direct-Method" durchgeführt, zwischen benachbarten SV können Diffusionen ablaufen. Die hierzu erforderlichen Parameter werden aus den physikalischen Diffusionskonstanten $D_{A,B,C...}$ für die verschiedenen Teilchensorten unter Beachtung der jeweiligen SV-Geometrie (i.d.R. Kuben mit fester Kantenlänge) umgerechnet. Bei jeder Iteration wird zuerst - ähnlich dem Ablauf in der First-Reaction-Methode - für jedes SV eine Probezeit ausgewürfelt, nach der jeweils ein Event (Reaktion- oder Diffusion) stattfindet. Das nächste Event wird dann jeweils in dem SV gesampelt, dessen Probezeit am kleinsten ist, anschließend werden neue Zeiten für die betroffenen SV ausgewürfelt. Um möglichst effizient das SV zu finden, in welchem die nächste Reaktion abläuft wird ein Bisektionsverfahren angewendet, die SV werden nach ihren Probzeiten in einen binären Baum einsortiert, das SV mit der kleinsten Probezeit befindet sich dabei stets an der Spitze des Baumes.

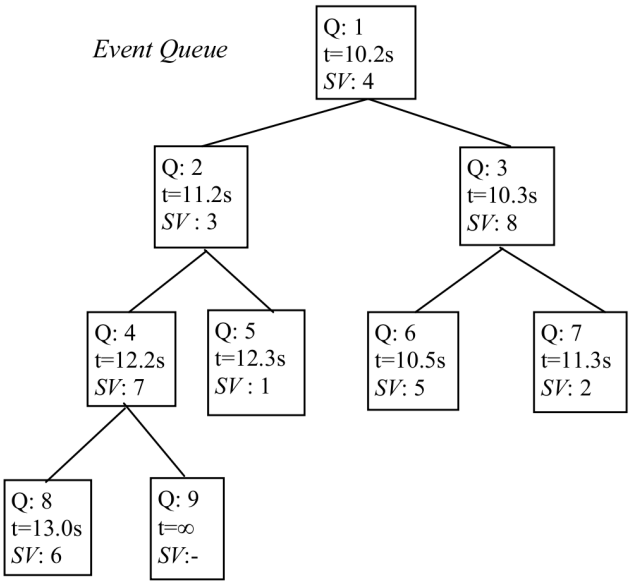


Abbildung 7: Die Anordnung von SV in einem binären Baum. Für spätere Sortierungszwecke werden die Elemente mit einem Index Q durchnummeriert. Abbildung entstammt „Mesoscopic reaction-diffusion in intracellular signaling, Johan Elf et al 2003".

Dies führt zu einer lediglich logarithmischen Abhängigkeit des Rechenaufwandes von der Systemgröße. Die Anwendungen des so zur sog. „Next-Subvolume-Method" erweiterten Algorithmus sind zahlreich, ein populäres Beispiel ist die Modulation der Min-Oszillationen in Bakterien.

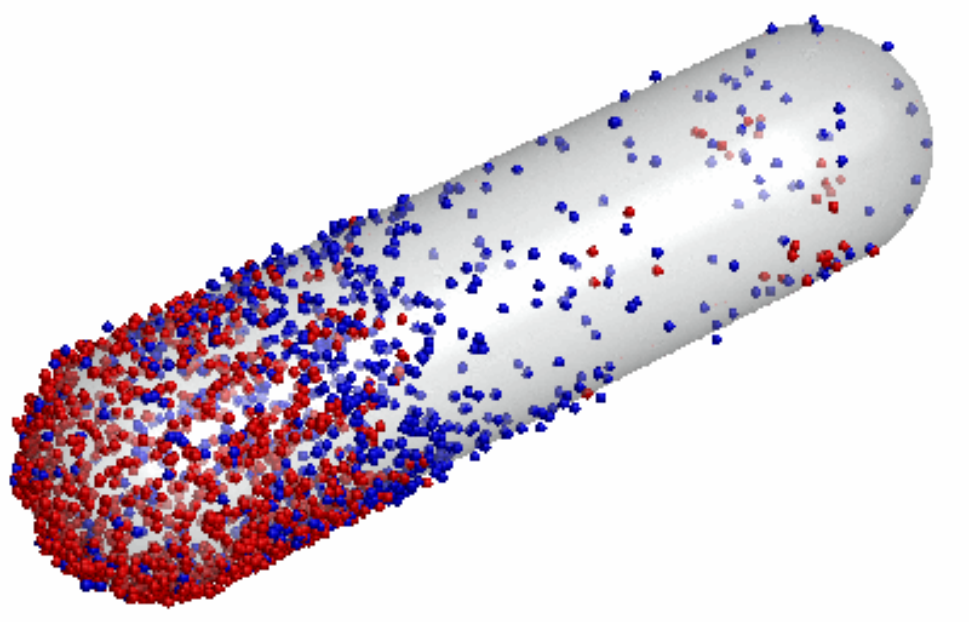


Abbildung 8: Die Simulation der Min-Oszillationen erfolgte mit dem freien Programmpacket MesoRD, welches auf der Next-Subvolume-Methode aufbaut.

5 Monte-Carlo-Simulation

Statistische Physik befasst sich mit Systemen mit vielen $\left(N \sim 10^{23}\right)$ Freiheitsgraden, z. B. $x_1, \ldots, x_N$, $x_i \in \mathbb{R}$, z. B. ein System mit M klassischen Teilchen im dreidimensionalen Raum $(\mathbf{r}, \mathbf{p})$, $\mathbf{r}$: Ort, $\mathbf{p}$: Impuls, $\mathbf{r}_i = (x_i, y_i, z_i)$, $\mathbf{p}_i = (p_{x_i}, p_{y_i}, p_{z_i})$ oder N Ising-Spins $\mathbf{S} = (S_1, , S_N)$,$S_i = \pm 1$ (Modell für mikroskopische magnetische Momente).
Ein System im thermodynamischen Gleichgewicht wird beschrieben durch seine Energie
$H(\mathbf{r}, \mathbf{p})$ die „Hamilton-Funktion"
oder
$H(\mathbf{S})$
und eine Temperatur $T \geq 0$, mit der es im thermischen Kontakt steht.
z. B.

$$H(r,p) = \underbrace{\sum_i \frac{\mathbf{p}_i^2}{2m}}_{\text{kinetische Energie}} + \underbrace{\sum_{i,j} V(\mathbf{r}_i - \mathbf{r}_j)}_{\text{potentielle Energie}}$$

oder

$$H(\mathbf{S}) = -\sum_{(ij)} J_{ij} S_i S_j - h \sum_i S_i$$

Mikroskopische Zustände $(\mathbf{r}, \mathbf{p})$ oder $\mathbf{S}$ (im sog. „kanonischen" Ensemble) mit der sog. „Boltzmann- Wahrscheinlichkeit"

$$P(\mathbf{r}, \mathbf{p}) = \frac{1}{\mathcal{Z}} e^{-\beta H(\mathbf{r}, \mathbf{p})}$$

oder

$$P(\mathbf{S}) = \frac{1}{\mathcal{Z}'} e^{-\beta H(\mathbf{S})}$$

angenommen.
die inverse Temperatur

$$\beta = 1/T$$

die Zustandssumme

$$\mathcal{Z} = \int d\mathbf{p} d\mathbf{q} e^{-\beta H(\mathbf{r}, \mathbf{p})}$$

$$\mathcal{Z}' = \sum_{\mathbf{S}} e^{-\beta H(\mathbf{S})}$$

Makroskopische Observablen physikalischer Messgrößen $\mathcal{O}(\mathbf{r}, \mathbf{p})$ oder $\mathcal{O}(\mathbf{S})$ sind dann durch gewichtete Mittel mit obiger Boltzmann-Wahrscheinlichkeit gegeben:

$$\langle \mathcal{O} \rangle = \int d\mathbf{p} d\mathbf{q} \mathcal{O}(\mathbf{r}, \mathbf{p}) P(\mathbf{r}, \mathbf{p})$$

oder

$$\sum_{\mathbf{S}} \mathcal{O}(\mathbf{S}) P(\mathbf{S})$$

Beispiel:
kinetische Energie

$$E_{kin} = \left\langle \sum_i \frac{\mathbf{p}_i^2}{2m} \right\rangle$$

Magnetisierung

$$M = \left\langle \sum_i S_i \right\rangle$$

Diese gewichteten Mittel (thermodynamische Erwartungswerte) sind also hochdimensionale Integrale oder Summen über alle möglichen Zustände $(\mathbf{r}, \mathbf{p})$ oder $\mathbf{S}$ des Sytems - eine numerische Aufgabe, die selbst bei bescheidenen Teilchenzahlen $N \sim 100$ bereits einen modernen Computer völlig überfordert.
Die Idee der Monte-Carlo-Simulation in der statistischen Physik besteht darin, diese Integrale oder Summen stochastisch auszuwerten
Hierbei werden mehr oder weniger zufällig mikroskopische Zustände $(\mathbf{r}, \mathbf{p})$ oder $\mathbf{S}$ ausgewürfelt, über die dann die physikalische Messgröße gemittelt wird.

→"Simple Sampling"

Man sieht jedoch sofort, dass bei völlig unvoreingenommener Wahl der Zustände in den meisten Fällen Zustände ausgewürfelt werden, die ein exponentiell kleines Gewicht in der Summe haben $\left(e^{-\beta H}!\right)$.
Ein ungeheur hilfreicher Trick besteht darin, diese Zustände bereits gemäß ihrer Boltzmann-Wahrscheinlichkeit $e^{-\beta H}$ auszuwürfeln - das nennt man "Importance Sampling".
Wie erzeugt man nun auf dem Computer eine stochastische Sequenz von Zuständen $(\mathbf{r}, \mathbf{p})$ oder $\mathbf{S}$, die diesem Anspruch, dass sie zumindest asymptotisch, d. h. im statistischen Limes, der Verteilung $P(\mathbf{r}, \mathbf{p})$ bzw. $P(\mathbf{S})$ genügt?

5.1 Ising-Modell

Aufgabe:
Generierung einer Sequenz von Zuständen

$$\mathbf{S}_1 \to \mathbf{S}_2 \to \mathbf{S}_3 \to \dots$$

mit Übergangswahrscheinlichkeiten $\boxed{w(\mathbf{S}_n \to \mathbf{S}_{n+1})}$ derart, dass $\boxed{\lim_{n\to\infty} P_n(\mathbf{S}) = P_{eq}(\mathbf{S}) = e^{-\beta H(\mathbf{S})}/\mathcal{Z}}$
$P_n(\mathbf{S})$ Wahrscheinlichkeit dafür, dass im n-ten Schritt der Sequenz der Zustand $\mathbf{S}$ generiert wird
Wie nun müssen diese $w(\mathbf{S} \to \mathbf{S}')$ gestaltet sein?
Die generierte Sequenz kann als eine Realisierung eines Markov-Prozesses angesehen werden (bedeutet, dass Übergangswahrscheinlichkeiten $w(\mathbf{S}_1, \dots, \mathbf{S}_n \to \mathbf{S}_{n+1})$ nur von $\mathbf{S}_n$ abhängen, also dem letzten Zustand der Sequenz), dessen Wahrscheinlichkeitsverteilung $P_n(\mathbf{S})$ der Mastergleichung

$$\boxed{P_{n+1}(\mathbf{S}) = P_n(\mathbf{S}) + \sum_{\mathbf{S}'} \Big\{ \underbrace{w(\mathbf{S}' \to \mathbf{S})\, P_n(\mathbf{S}')}_{\text{Prozesse, die nach } \mathbf{S} \text{hinführen}} - \underbrace{w(\mathbf{S} \to \mathbf{S}')\, P_n(\mathbf{S})}_{\text{Prozesse, die von } \mathbf{S} \text{wegführen}} \Big\}}$$

genügt (eine Art Bilanzgleichung)
Wir wollen, dass $P_{eq}(\mathbf{S})$ die Summe auf der rechten Seite zum Verschwinden bringt, d. h. $P_n(\mathbf{S}) \equiv P_{eq}(\mathbf{S})$ produziert wieder $\Rightarrow P_{n+1}(\mathbf{S}) \equiv P_{eq}(\mathbf{S})$
Hierzu reicht es, $w(\mathbf{S} \to \mathbf{S}')$ so zu wählen, dass jeder einzelne Term in der geschweiften Klammer verschwindet, wenn $P_n(\mathbf{S}) = P_{eq}(\mathbf{S})$,
d. h. $\boxed{\forall \mathbf{S}', \mathbf{S} :\ w(\mathbf{S}' \to \mathbf{S})\, P_{eq}(\mathbf{S}) = w(\mathbf{S} \to \mathbf{S}')\, P_{eq}(\mathbf{S}')}$ $\circledast$
das nennt man „detaillierte Bilanz"
Das heißt, ein Satz von Übergangswahrscheinlichkeiten $w(\mathbf{S} \to \mathbf{S}')$, für den gilt

$$\boxed{\forall \mathbf{S}', \mathbf{S}\ \frac{w(\mathbf{S} \to \mathbf{S}')}{w(\mathbf{S}' \to \mathbf{S})} = \exp\left(-\beta\left(H(\mathbf{S}') - H(\mathbf{S})\right)\right) = e^{-\beta \Delta H(\mathbf{S}, \mathbf{S}')}}$$

garantiert, dass die stationäre, d. h. zeit- bzw. n-unabhängige, Lösung der Mastergleichung, also die stationäre Veteilung des generierten Markov-Prozesses, die Boltzmann-Verteilung $P_{eq}(\mathbf{S}) = \frac{1}{\mathcal{Z}} e^{-\beta H(\mathbf{S})}$ ist!
Eine mögliche Wahl für $w(\mathbf{S} \to \mathbf{S}')$ ist die Metropolis-Wahl:

$$\boxed{w(\mathbf{S} \to \mathbf{S}') = \begin{cases} 1 & \text{für } \Delta H(\mathbf{S}, \mathbf{S}') \leq 0 \\ e^{-\beta \Delta H(\mathbf{S}, \mathbf{S}')} & \text{für } \Delta H(\mathbf{S}, \mathbf{S}') \leq 0 \end{cases}}$$

ΔH ist die Energiedifferenz zwischen dem aktuellen Zustand $\mathbf{S}$ und einem neuen, vorgeschlagenen Zustand $\mathbf{S}'$. Ist diese negativ, d. h. der neue Zustand energetisch günstiger als der alte, wird er in jedem Fall angenommen, ist sie positiv, so wird er nur mit Wahrscheinlichkeit $0 < e^{-\beta \Delta H} < 1$ angenommen.
Prozedur: generiere neuen Zustand $\mathbf{S}'$
generiere Zufallszahl x
wenn $x < e^{-\beta \Delta H} \Rightarrow$ akzeptiere $\mathbf{S}'$
Um im Falle $\Delta H > 0$ effektive Akzeptanzraten zu haben, sollte die Veränderung $\mathbf{S} \to \mathbf{S}'$ nicht zu groß sein (es sei denn, man generiert große Veränderung „geschickt", sodass sie nicht viel Energie kostet $\to$ Cluster-Algorithmus, später). Im Falle von Ising-Spins $S_i = \pm 1$ benutzt man gewöhnlich Single-Spin-Flip-Dynamik $S_i \to -S_i$, d. h. $\mathbf{S}' = (S_1, \ldots, S_{i-1}, -S_i, S_{i+1}, \ldots, S_N)$.
Angenommen, $\mathbf{S}_1, \ldots, \mathbf{S}_{MC-Steps}$ ist eine Sequenz von Zuständen, die wie oben generiert ist und $P(\mathbf{S}_l) = P_{eq}(\mathbf{S}_l)$. Dann ist der thermodynamische Mittelwert einer Observable $\mathcal{O}(\mathbf{S})$ einfach

$$\langle \mathcal{O} \rangle = \lim_{MC \to \infty} \left(\frac{1}{MC} \sum_{l=1}^{MC} \mathcal{O}(\mathbf{S}_l) \right),$$

denn der Gewichtungsfaktor $\mathcal{O}(\mathbf{S}_l)$ ist schon in der Häufigkeit enthalten, mit der der Zustand $\mathbf{S}_l$ auftritt. Die Monte-Carlo-Update-Prozedur, die aus einer aktuellen Spinkonfiguration $\mathbf{S}$ eine neue $\mathbf{S}'$ vorschlägt und mit $w(\mathbf{S} \to \mathbf{S}')$ akzeptiert, operiert (im einfachsten Fall) mit Einzel-Spinflips $S_i \to -S_i$. Die Energieänderung beim Flip eines einzelnen Spins S_i beträgt

$$\begin{aligned} \Delta H(S_i \to -S_i) &= H(S_1, \ldots, S_{i-1}, -S_i, S_{i+1}, \ldots, S_N) - H(\mathbf{S}) \\ &= 2 \sum_{\substack{j=1 \\ (j \neq i)}}^{N} J_{ij} S_j S_i - 2hS_i \\ &= 2 S_i b_i \end{aligned}$$

mit $\boxed{b_i = h + \sum_{\substack{j=1 \\ (j \neq i)}}^{N} J_{ij} S_j}$
das „lokale Feld"

$$\Rightarrow w(S_i \to -S_i) = \begin{cases} 1 & \text{, falls } S_i b_i < 0 \leftarrow \text{ Spin i steht antiparallel zum lokalen Feld} \\ e^{-2\beta S_i h_i} & \text{, falls } S_i b_i < 0 \leftarrow \text{ parallel} \end{cases}$$

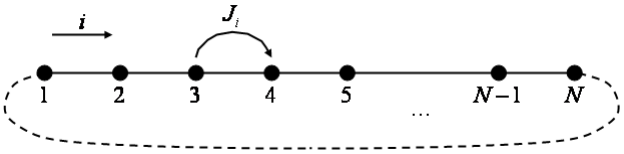
Ein Schema für ein Monte-Carlo-Programm, das zunächst MC-Eq Sweeps durch das ganze Gitter macht, um zu „equilibriere", und dann erst anfängt, zu messen, wäre:

```
choose start-conf. S (z. B. zufällig S_i = ±1 ≙ ∞ Temp)
set O_av = ï¿œ
for step=1,...,MC-Steps
{
    for i = 1,...,N
    if (S_i b_i < 0)
        S_i = -S_i
    else
        if (ran()< e^{-2βS_i h_i})
            S_i = -S_i
        if (step > MC-Eq)
            O_av+ = O(S)
}
O_av = O_av/MC-Steps-MC-Eq
```

Betrachten wir also ein konkretes Beispiel, das Ising-Modell wechselwirkender ± 1-Spins:

$$\boxed{H(\mathbf{S}) = -\frac{1}{2}\sum_{i \neq j} J_{ij} S_i S_j - h \sum_{i=1}^{N} S_i}$$

Die Wechselwirkungen J_{ij} sind positiv in einem Modell für einen Ferromagneten, denn dann ist die parallele Ausrichtung der Spins ($S_i = S_j$) günstiger, und sind negativ in einem Modell für einen Antiferromagneten, denn dann ist $S_i = -S_j$ günstiger.
Das magnetische Feld h kann auch ausgeschaltet sein ($h = 0$).
„Realistische" dreidimensionale Ising-Ferromagnete werden durch ein Modell beschrieben, bei dem die Spins z. B. auf einem kubischen Gitter angeordnet sind und die Wechselwirkungen $J_{ij} = J > 0$ nur für benachbarte Gitterplätze nicht verschwinden.
Analog in $2d$ und in $1d$; letzterer Fall konkret (s. übung)



$\hookrightarrow$ meist nimmt man periodische Randbedingungen (d. h. $S_{N+1} \equiv S_1$!) um das System völlig homogen zu machen (jeder Gitterplatz ist gleichberechtigt)

$$H = -J \sum_i J_i S_i S_{i+1} - h \sum_i S_i$$

Wichtige physikalische Observablen sind

Magnetisierung

$$m = \left\langle \frac{1}{N} \sum_{i=1}^{N} S_i \right\rangle \quad \begin{cases} \neq 0 & h \neq 0 \\ = 0 & h = 0 \end{cases} \text{ in jedem } \textbf{endlichen} \text{ System im } \textbf{Gleichgewicht}$$

auch $|m|$

Suszeptibilität

$$\chi = \left\langle \frac{1}{N} \sum_{i,j} S_i S_j \right\rangle = N \left\langle m^2 \right\rangle$$

(falls $\langle m \rangle \neq 0$, $\chi = N \left(\left\langle m^2 \right\rangle - \langle m \rangle^2 \right)$)

spezifische Energie

$$e = \frac{1}{N} \langle H(\mathbf{S}) \rangle$$

spezifische Wärme

$$c = N \left(\left\langle e^2 \right\rangle - \langle e \rangle^2 \right)$$

Betrachte noch einmal Ising-Modell

$$\boxed{H = -\sum_{(ij)} J_{ij} S_i S_j - b \sum_i S_i}$$

Thermodynamische Größen

Zustandssumme:

$$\mathcal{Z} = \sum_{\mathbf{S}} e^{-\beta H(\mathbf{S})}$$

Freie Energie:

$$\mathcal{F} = -\beta^{-1} \ln(\mathcal{Z})$$

($\beta = 1/T$, Boltzmann-Konstante $k_B = 1$)

thermodynamische Erwartungswerte:

$$\langle (\dots) \rangle_{(T)} = \frac{\sum_{\mathbf{S}} (\dots) e^{-\beta H}}{\mathcal{Z}}$$

Magnetisierung:

$$M = \left\langle \sum_{i=1}^{N} S_i \right\rangle = -\frac{\partial \mathcal{F}}{\partial b}$$

$$\begin{aligned}
\frac{\partial \mathcal{F}}{\partial b} &= -\frac{\partial}{\partial b}\left[-\beta^{-1} \ln\left(\sum_{\mathbf{S}} e^{\overbrace{\beta \sum_{(ij)} J_{ij} S_i S_j + \beta b \sum_i S_i}^{=-\beta H}} \right) \right] \\
&= \beta^{-1} \frac{\sum_{\mathbf{S}} \beta \left(\sum_i S_i\right) e^{-\beta H}}{\sum_{\mathbf{S}} e^{-\beta H}} \\
&= \sum_{\mathbf{S}} \frac{M e^{-\beta H}}{\mathcal{Z}} = \langle M \rangle \ \surd
\end{aligned}$$

Suszeptibilität:

$$\begin{aligned}
X &= \frac{\partial M}{\partial b} = -\frac{\partial^2 \mathcal{F}}{\partial b^2} \\
&= \frac{\partial}{\partial b} \frac{\sum_{\mathbf{S}} \left(\sum_i S_i\right) e^{-\beta H}}{\mathcal{Z}} = \frac{\sum_{\mathbf{S}} \left(\sum_i S_i\right)^2 e^{-\beta H}}{\mathcal{Z}} - \frac{\left(\sum_{\mathbf{S}} \left(\sum_i S_i\right) e^{-\beta H}\right)^2}{\mathcal{Z}} \\
&= \beta \left\{ \left\langle \left(\sum_i S_i \right)^2 \right\rangle - \left\langle \sum_i S_i \right\rangle^2 \right\} \leftarrow \text{Fluktuationen in der Magnetisierung}
\end{aligned}$$

Wichtig:
Numerisch wird χ immer über die Fluktuationen bestimmt, nicht über die Ableitung.

Magnetisierung pro Spin ($\in [-1, 1]$) $m = M/N$
Suszeptibilität pro Spin ($\in [-\beta N, \beta N]$) $\chi = X/N$

Analog:

Energie:

$$E = \langle H \rangle = -\frac{\partial \ln(\mathcal{Z})}{\partial \beta} = \mathcal{F} - T \frac{\partial \mathcal{F}}{\partial T}$$

spezifische Wärme:

$$\begin{aligned}
C &= \frac{\partial E}{\partial T} = -T \frac{\partial^2 \mathcal{F}}{\partial T^2} \\
&= \left(\left\langle H^2 \right\rangle - \left\langle H \right\rangle^2 \right) / T^2 \leftarrow \text{Fluktuationen in der Energie!}
\end{aligned}$$

pro Spin: $e = E/N$
$c = C/N$

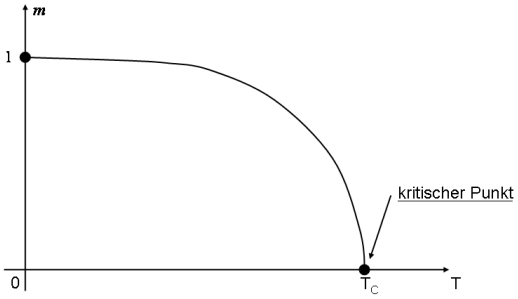
Exakte Lösung für $1d$ im Limes $N \to \infty$ (leicht):

$$f = \lim_{N\to\infty} \frac{\mathcal{F}}{N} = \lim_{N\to\infty} \frac{1}{\beta N} \ln\left(\mathcal{Z}_N\right)$$
$$= \beta^{-1} \ln\left(e^{\beta J}\cosh\left(\beta b\right) + \sqrt{e^{2\beta J}\sinh^2\left(\beta b\right) + e^{-2\beta J}}\right)$$

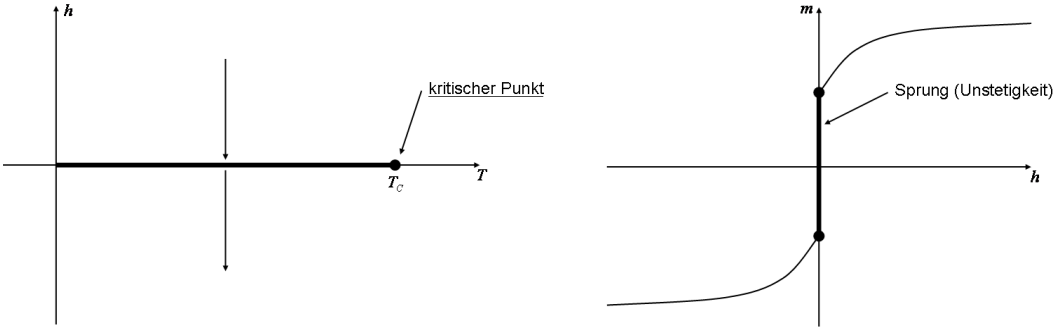
$$\Rightarrow m = \frac{\partial f}{\partial b} = \frac{e^{\beta J}\sinh\left(\beta b\right)}{\sqrt{e^{2\beta J}\sinh^2\left(\beta b\right) + e^{-2\beta J}}}$$

χ, e, c selbst Vergleich mit Ergebnissen für endliches N aus den Übungen
In $d > 1$ ($d = 2, 3, \ldots$) hat das Ising-Modell im Nullfeld ($b = 0$) (für $N \to \infty$) einen Phasenübergang 2ter Ordnung (d. h. mindestens eine 2te Ableitung der freien Energie ist unstetig oder divergent) von einer paramagnetischen Phase ($m = 0$) für $T > T_C$ zu einer ferromagnetischen Phase ($m \neq 0$) für $T < T_C$
$d = 2$ ist exakt lösbar (schwer, Onsager '44), T_C und kritische Exponenten bekannt
$d = 3$: nicht exakt lösbar, Renormierungsgruppe (ϵ-Entwicklung), Reihenentwicklung, numerische Simulation (Transfer-Matrix, Monte-Carlo!)

$h = 0$:



$h \neq 0$: Phasenübergang 1ter Ordnung



$\hookrightarrow \hat{=}$ in $3d$ dem kritischen Punkt im phasendiagramm von Wasser! (Gas-Flüssigkeit!)

Der Phasenübergang bei $T = T_C$ (und $h = 0$) ist ganz analog zu dem Perkolationsübergang: Thermodynamische Observablen haben hier (im Limes $N \to \infty$) Singularitäten.

Magnetisierung	$m \propto \begin{cases} 0 & \text{für } T \geq T_C \\ (T_C - T)^{\beta} & \text{für } T < T_C \end{cases}$
Suszeptibilität	$\chi \propto \lvert T - T_C\rvert^{-\gamma}$
spezifische Wärme	$c \propto \lvert T - T_C\rvert^{-\alpha}$

Die Korrelationslänge ξ ist über das Verhalten der Spin-Spin-Korrelationsfunktion definiert.

$$C(r) := \lim_{N\to\infty} \frac{1}{N} \{\langle S_i S_{i+\mathbf{r}}\rangle - \langle S_i\rangle \langle S_{i+\mathbf{r}}\rangle\}$$
$$\propto r^{-(d-2+\eta)} \exp(-r/\xi) \quad \text{mit } \boxed{\xi \propto |T-T_C|^{-\nu}}$$

Am kritischen Punkt: $\boxed{C(r) \propto r^{-(d-2+\eta)}}$

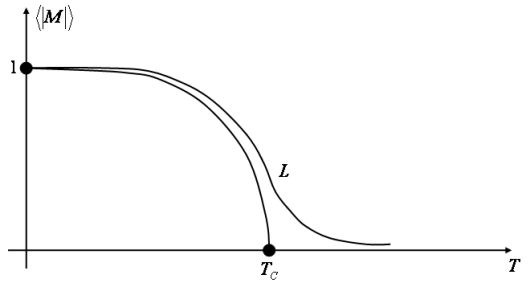
Die Exponenten sind wie üblich durch Skalenrelationen miteinander verknüpft:
Für $T \neq T_C$ ist

$$\sum_{\mathbf{r}} C(r) \propto \int dr r^{d-1} C(r)$$
$$\approx \int_1^\xi dr r^{d-1} r^{-d+2-\eta} = \int_1^\xi dr r^{1-\eta}$$
$$\sim \xi^{2-\eta} \sim |T-T_C|^{-(2-\eta)\nu}$$

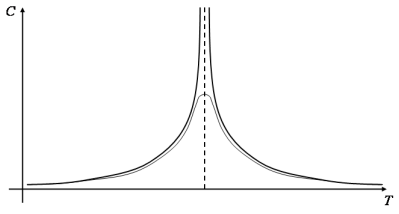
Da aber

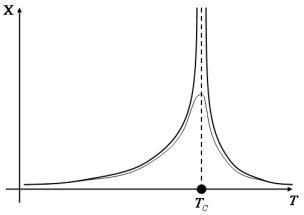
$$\sum_{\mathbf{r}} C(r) = \sum_{i,j} \{\langle S_i S_j\rangle - \langle S_i\rangle \langle S_j\rangle\}$$
$$= \langle M^2\rangle - \langle M\rangle^2 = X \propto |T-T_C|^{-\gamma}$$

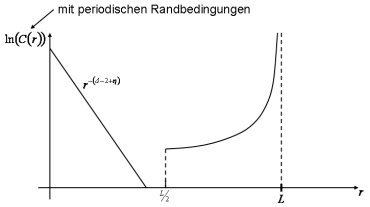
folgt $\boxed{2-\eta = \gamma/\nu}$. Außerdem gilt (s. FSS) die Rushbroke-Identität $\boxed{\alpha + 2\beta + \gamma = 2}$.
Der singuläre Anteil der freien Energie $\mathcal{F}$ ist umgekehrt proportional zum korrelierten Volumen: $\mathcal{F}_{sing} \sim \xi^{-d} \sim |T-T_C|^{\nu d}$. Die spezifische Wärme ist $c = \frac{\partial^2 \mathcal{F}}{\partial T^2} \propto |T-T_C|^{-\alpha}$.
$\Rightarrow F \propto |T-T_C|^{2-\alpha} \Rightarrow \boxed{2-\alpha = \nu d}$ Hyper-Skalenrelation



Im endlichen System sind diese Singularitäten wie in der Perkolation systematisch aufgeweicht; außerdem ist M immer Null (keine Ergodizitätsbrechung in endlichen Systemen!), deshalb betrachte als Observable $|M|$!





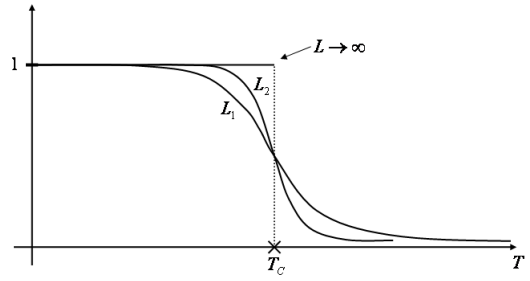


Wie in der Perkolation beschreiben Skalenformen das asymptotische Finite-Size-Scaling ($\delta = T - T_C$, L die lineare Systemgröße $N = L^d$)

$$m(L,T) \sim L^{-\beta/\nu} \tilde{m}\left(L^{1/\nu}\delta\right), \quad \tilde{m}(x) \sim \begin{cases} const & \text{, falls } x \ll 1 \\ x^\beta & \text{, falls } x \gg 1 \end{cases}$$
$$\chi(L,T) \sim L^{\gamma/\nu} \tilde{\chi}\left(L^{1/\nu}\delta\right), \quad \tilde{\chi}(x) \sim \begin{cases} const & \text{, falls } x \ll 1 \\ x^{-\gamma} & \text{, falls } x \gg 1 \end{cases}$$
$$c(L,T) \sim L^{\alpha/\nu} \tilde{c}\left(L^{1/\nu}\delta\right), \quad \tilde{c}(x) \sim \begin{cases} const & \text{, falls } x \ll 1 \\ x^{-\alpha} & \text{, falls } x \gg 1 \end{cases}$$

das Argument der Skalenfunktion ist wieder L/ξ (oder $L^{1/\nu}/\xi^{\gamma/\nu} = L^{1/\nu}\delta$), das Verhältnis der einzigen beiden relevanten Längenskalen L und ξ.
Die geeignetste Größe zur Lokalisierung des kritischen Punktes T_C ist diesmal die Kumulante g (oder Binder-Konstante):

$$\boxed{g := \frac{1}{2}\left(3 - \frac{\langle M^4\rangle}{\langle M^2\rangle^2}\right) \sim \tilde{g}\left(L^{1/\nu}\delta\right)}$$



Sie beschreibt die Abweichung der Wahrscheinlichkeitsverteilung von M ($P(M)$) von einer Gaußverteilung:
$g = 0$ für $P(M)$ Gaußverteilung
$g = 1$ für $M = \pm m$
$\Rightarrow$ Für $N \to \infty$ ist g eine Stufenfunktion.

Die Kumulante ist dimensionslos am kritischen Punkt (d. h., hängt für $\delta = 0$, d. h. $T = T_C$) nicht von L ab. Dies liegt daran, dass die Momente von M nicht simpel skalen:

$$m^{(n)} := \langle |M|^n \rangle \sim L^{-n\beta/\nu}\tilde{m}^{(n)}\left(L^{1/\nu}\delta\right) \circledast$$

g ist so konstruiert, dass nur das Verhältnis von dem kten Moment und dem 2ten Moment zum Quadrat eingeht. $\Rightarrow$ Die L-abhängigen Vorfaktoren von den Skalenfunktionen heben sich weg, bei $\delta = 0$ schneiden sich alle Kurven $g_L(\delta = 0)$ in einem Punkt bei T_C. Dies ist ein gutes Kriterium zur Bestimmung von T_C (wie immer: bis auf convections to Scaling). Mit $\circledast$ folgt auch eine Beziehung zwischen β und γ, denn m ist das erste Moment von $P(M)$, χ ist das zweite multipliziert mit N (beachte: pro Spin!) (für $T > T_C$, bei $T = T_C$ 2k Kumulante)
$\Rightarrow L^{\gamma/\nu} \sim L^d L^{-2\beta/\nu} \Rightarrow \boxed{\gamma/\nu = d - 2\beta/\nu}$
Mit der Hyper-Skalenrelation $2 - \alpha = \nu d$ folgt die Rushbroke-Identität.

$$\alpha + 2\beta + \gamma = 2 - \nu d + 2\beta + \nu d - 2\beta = 2 \surd$$

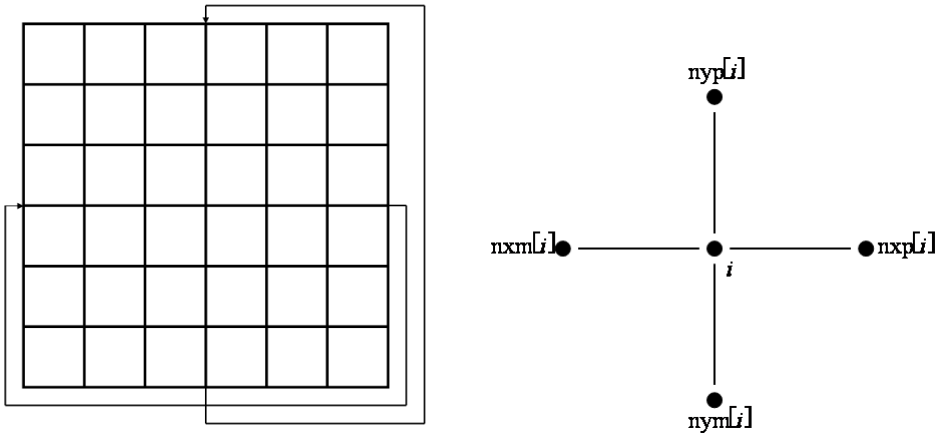
Bekannte Resultate für das d-dimensionale Ising-Modell:

d	1	2	3	$\overset{(MFT)}{\geq 4}$	...	Bethe
T_C	0	2.26917...	4.511...			
ν	1	1	0.629...	-		
α	1	0	0.105...	0		$\hat{=}$MFT
β	0	1/8	0.326...	1/2		
γ	1	7/4	1.239...	1		
η	1	1/4		-		
δ	∞	15		3		

$$\beta\delta = 1$$

$m(h, T_C) \propto h^{1/\delta}$
$\gamma = \beta(\delta - 1)$

Tips zur MC-Simulation des $2d$-Ising-Modells mit Einzelspinflip-Update: $J = 1!$ $\leftarrow$ d. h. T in Einheiten von Jk_B
Benutze periodische Randbedingungen und neighbor-Arrays:



$i = 1, \ldots, N$
4 zusätzliche Arrays mit je $L \cdot L$ Elementen (oder $neighb[N][L]$)
Berechne in der Initialisierung nxm, nxp, nym, nyp
(z. B. $nxp\,[i] = i + 1 \qquad$ falls $i\%L \neq 0$
$\qquad = i + 1 - L \;$ falls $i\%L = 0$ etc.)

Tabelliere die Akzeptanz-Wahrscheinlichkeiten für $\Delta E > 0$: (vermeide Berechnung transzendenter Funktionen in der innersten Schleife)
$\Delta E = 2S_i h_i \qquad \left(h_i = S_{nxm[i]} + S_{nxp[i]} + S_{nym[i]} + S_{nyp[i]}\right)$
$\in \{4, 8\} \qquad$ (in $3d$ $\Delta E \in \{4, 8, 12\}$)
$\hookrightarrow e^{-\beta \Delta E} \in \left\{e^{-4\beta}, e^{-8\beta}\right\}$
Def. $prob\,[j] = max_{rn} \cdot e^{-2j\beta}$ $j = 1, 2, 3, 4$
Benutze RNG $r250$, der Integers aus $[0, \ldots, \max\_\mathrm{m}]$ generiert

Benutze Array für die lokalen Felder h_i
(Wenn kein Spin geflippt wird, muss nichts gerechnet werden!)
Nach Initialisierung des Spins $S_i$berechne $h_i = S_{nxm[i]} + S_{nxp[i]} + S_{nym[i]} + S_{nyp[i]}$
Im Update-Loop: Wenn $S_i \to -S_i$ (Flip) modifiziere $h_{nxm[i]} += 2S_i \leftarrow$ neuer Spinwert
$h_{nxp[i]} += 2S_i$ etc.
Benutze die Hälfte der gesamten MC-Sweeps durchs Gitter zur Equilibrierung, die andere zur Messung der Observablen! (m, $|m|$, m^2, m^4, e, e^2, χ, c, g)

5.2 Andere „Spin-Modelle"

5.2.1 q-Zustands-Potts-Modell

$S_i \,\epsilon\, \{1, 2, ..., q\}$, $J > 0 \;\to$ ferromagnetisch

$$\boxed{H = -J \sum_{(ij)} \delta_{S_i S_j}} \qquad \delta_{SS'} = \begin{cases} 1 & \text{für } S = S' \\ 0 & \text{sonst} \end{cases}$$

$q = 2 \;\hat{=}\;$ Ising-Modell $\left(\text{aber } J \to \frac{J}{2}\right)$

Einzelspinflip klar (wähle zufällig einen Wert aus $\{1, ..., q\} \setminus \{S_i\}$)
2d: Phasenübergang $q \leq 4$ 2. Ordnung $(q > 2$ nicht Ising-Universalitätsklasse)
$q > 4$ 1. Ordnung

Ordnungsparameter: $m_\sigma = \left\langle \frac{1}{N} \sum_{i=1}^{N} (q\delta_{S_{i\sigma}} - 1) \right\rangle \quad \begin{cases} = 0 & \text{für } T > T_c \\ > 0 & \text{für } T < T_c \end{cases}$

Wenn q sehr groß ist, ist das zufällige Auswürfeln von neuen Spin-Werten ineffizient, besser:

$$\text{Heut-Baltz-Algorithmus:} \quad \omega(S_i \to \underbrace{n}_{\epsilon\{1,...,q\}}) = \frac{e^{-\beta E_n}}{\sum_{m=1} e^{-\beta E_m}}$$

equilibriert bei tiefen Temperaturen schneller.

5.2.2 XY-Modell, Heisenberg-Modell

(kontinuierliche Spin-Freiheitsgrade)

$$S_i \begin{cases} = (cos(\varphi_i),\, sin(\varphi_1)) & \text{(XY)} \\ = (cos(\varphi_i) \cdot sin(\theta_i),\, sin(\varphi_i)sin(\theta_i),\, cos(\theta_i) & \text{(Heisenberg)} \end{cases}$$

$$\boxed{H = -J\sum_{(ij)} S_i S_j}$$

$\hat{=} -J\sum_{(ij)} cos(\varphi_i - \varphi_j)$ für XY (Modell für Supraleitung und Suprafluidität)

$$\downarrow$$

Einzelspinupdate: $\varphi_i \to \varphi_i + \underbrace{\Delta\varphi}_{\epsilon[0,\, 2\pi] \text{ zufällig}}$

T_c des Potts-Modells auf dem Quadratgitter (J_1, J_2):
$x_r = q^{-\frac{1}{2}}(e^{\beta J_r} - 1)$, selbst-dual bei $x_1 x_2 = 1$

$$\text{d.h.} \quad \boxed{\frac{1}{q}(e^{\beta J_1} - 1)(e^{\beta J_2} - 1) = 1}$$

für $J_1 = J_2 = J$ (isotrop):

$$e^{\beta_c J} - 1 = \sqrt{q} \;\rightsquigarrow\; \boxed{T_c = \frac{1}{ln(1+\sqrt{q})}}$$

$q = 2:\ T_C = 1.13...$
$q = 8:\ T_C = 0.74...$

kritische Exponenten:

q	α	β	δ
1	$-\frac{2}{3}$	$\frac{5}{36}$	$18\frac{1}{5}$
2	0	$\frac{1}{8}$	15
3	$\frac{1}{3}$	$\frac{1}{9}$	14
4	$\frac{2}{3}$	$\frac{1}{12}$	15

$q > 4$: Phasenübergang 1. Ordnung

5.3 Histogramm-Methode

Mit dieser Methode kann man Daten, die aus einer MC-Simulation bei einer Temperatur T_0 gewonnen werden, auf eine Temperatur $T(\neq T_0)$ extrapolieren.
Erinnerung: $\langle\mathcal{O}\rangle = \lim_{M\to\infty} \frac{1}{M}\sum_{t=1}^{M} \underbrace{\mathcal{O}(S_t)}_{=:\,\mathcal{O}_t}$ (*)

M =Anzahl der Messungen, S =Zustand bei der t-ten Messung

Wir ergänzen das Gewicht $P(S_t) = e^{-\beta H(S_t)}$, mit dem -im Gleichgewicht- der Zustand $P(S_t)$ auftauchen sollte:

$$\langle\mathcal{O}\rangle_T = \lim_{M\to\infty} \frac{\sum_{t=1}^{M} \mathcal{O}(S_t)(P(S_t))^{-1} e^{-\beta H(S_t)}}{\sum_{t=1}^{M} (P(S_t))^{-1} e^{-\beta H(S_t)}} \quad (**) \quad \left(\beta = \frac{1}{T}\right)$$

Für $P(S_t) = e^{-\beta H(S_t)}$steht hier nichts anderes als (*), nur aufgebläht. Aber angenommen, die MC-Simulation wurde bei einer Temperatur $T_0 \left(= \frac{1}{\beta_0}\right)$ durchgeführt, so dass $P(S_t) \propto e^{-\beta_0 H(S_t)}$.

Dann ergibt (**) eine Formel für $\langle\mathcal{O}\rangle_T$ mit Daten, die bei T_0 gewonnen wurden:

$$\boxed{\langle\mathcal{O}\rangle_\mathcal{T} = \lim_{M\to\infty} \frac{\sum_{t=1}^{M} \mathcal{O}(S_t) e^{-(\beta-\beta_0)H(S_t)}}{\sum_{t=1}^{M} e^{-(\beta-\beta_0)H(S_t)}}}$$

Dies ist die fundamentale Gleichung für die Histogramm-Methode. Zur Berechnung des thermodynamischen Erwartungswertes $< E >$ der Energie macht man also zweckmäßig ein Histogramm der auftretenden Energiewerte gemäß ihrer Häufigkeit $N(E)$, dann

$$\boxed{\langle E\rangle_T = \frac{\sum_E E \cdot N(E) e^{-(\beta-\beta_0)E}}{\sum_E N(E) e^{-(\beta-\beta_0)E}}}$$

Für, beispielsweise, Magnetisierung m braucht man ein Histogramm $N(E, M)$

$$\boxed{m = \frac{1}{N}\langle M\rangle_T = \frac{\sum_{M,E} M \cdot N(E,M) e^{-(\beta-\beta_0)E}}{\sum_{M,E} N(E,M) e^{-(\beta-\beta_0)E}}}$$

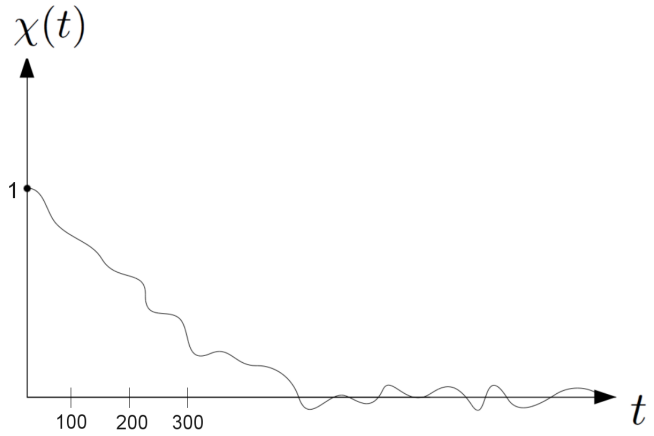
5.4 Autokorrelations-Funktionen

Um ein Maß für die Güte unserer Statistik in einer Monte-Carlo-Simulation zu haben, brauchen wir Informationen darüber, wieviele MC-Schritte benötigt werden, um zwei aufeinanderfolgende Messungen als unkorreliert ansehen zu können. (Im Idealfall wollten wir ja Zustände S auswürfeln, die unabhängige Ereignisse gemäß der Boltzmann-Statistik sind).
Diese Info ist in der „Korrelationszeit" τ der Simulation enthalten, die auf verschiedene Weisen bestimmt werden kann.

Betrachte z. B. die Autokorrelationsfunktion der Magnetisierung:

$$\begin{aligned}\chi(t) &= \int dt'(m(t') - \langle m\rangle)(m(t'+t) - \langle m\rangle) \\ &= \int dt'(m(t')m(t'+t) - \langle m\rangle^2)\end{aligned}$$

Typisch:⟶



$T = 2.4$ in $2d$ Ising-Modell

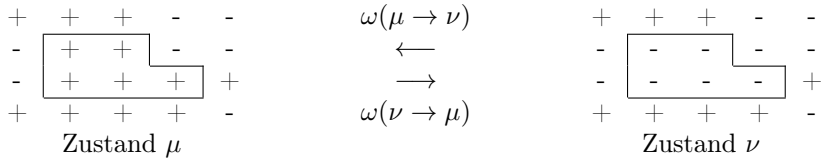
↓

$\chi \approx e^{-\frac{t}{\tau}}$, exponentieller Abfall auf der charakteristischen Zeitskala $\tau \hat{=}$ Estimate für die Korrelationszeit (der Magnetisierung)
Hier: $\tau \approx 100$, d.h.: Nur bei jedem 100. MC-Sweep bekommt man eine nahezu unabhängige Messung.

Schätzwert für τ: Integrierte Korrelationszeit
Wenn $\chi \propto e^{-\frac{t}{\tau}}$, dann ist $\underbrace{\int_0^\infty dt \frac{\chi(t)}{\chi(0)}}_{\text{Dies kann als Schätzwert für } \tau \text{ verwendet werden, auch wenn } \chi(t) \text{ eine komplizierte Zeitabhängigkeit hat und nur asymptotisch wie } e^{-\frac{t}{\tau}} \text{ abfällt.}} = \int_0^\infty dt\, e^{-\frac{t}{\tau}} = \tau$

5.5 Wolf-Cluster-Update für Ising-Modell

Betrachte Cluster-Flip:



$$
\begin{aligned}
m = B(\mu \to \nu) &= \#\{\text{Bonds zwischen } ++, \text{ die bei } \mu \to \nu \text{ gebrochen werden}\} \\
&= 6 \\
n = \tilde{B}(\mu \to \nu) &= \#\{\text{Bonds zwischen } +\text{-}, \text{ die bei } \mu \to \nu \text{ befriedigt werden}\} \\
&= 4 \\
\text{analog: } B(\nu \to \mu) &= \#\{\text{Bonds zwischen } ++, \text{ die bei } \nu \to \mu \text{ gebrochen werden}\} \\
&= \tilde{B}(\mu \to \nu) = 4 \\
\tilde{B}(\nu \to \mu) &= \#\{\text{Bonds zwischen } \text{-}+, \text{ die bei } \nu \to \mu \text{ befriedigt werden}\} \\
&= B(\mu \to \nu) = 6 \\
\Delta E = E_\nu - E_\mu &= 2JB(\mu \to \nu) - 2J\tilde{B}(\mu \to \nu) = 2J(m-n)
\end{aligned}
$$

Für die Übergangswahrscheinlichkeiten muss also gelten:

$$\frac{\omega(\mu \to \nu)}{\omega(\nu \to \mu)} = e^{-2\beta J(m-n)}$$

$\omega(\mu \to \nu)$ setzt sich zusammen aus einer Selektionswahrscheinlichkeit $g(\mu \to \nu)$ für den Übergang $\mu \to \nu$ und einer Akzeptorwahrscheinlichkeit $A(\mu \to \nu)$

$$\frac{\omega(\mu \to \nu)}{\omega(\nu \to \mu)} = \frac{g(\mu \to \nu)\, A(\mu \to \nu)}{g(\nu \to \mu)\, A(\nu \to \mu)}$$

Wir konstruieren nun $g(\mu \to \nu)$ so, dass $A = 1$ wird (optimal!).
Wähle dazu (zufällig) einen „Seed"-Spin S_i für den zu flippenden Cluster.
Addiere nun sukzessive Nachbarspins, die in dieselbe Richtung wie S_i zeigen, bzw. Kopplungen zwischen befriedigten Bonds, zu dem Cluster, und zwar mit Wahrscheinlichkeit P_{add}.
Dann ist

$$\frac{g(\mu \to \nu)}{g(\nu \to \mu)} = \frac{(1-P_{add})^m}{(1-P_{add})^n} \cdot \frac{(P_{add})^{\#,,addierte\,Bonds''}}{(P_{add})^{\#,,addierte\,Bonds''}}$$

denn es werden ja m (++)-Nächste-Nachbarn-Paare bei $\mu \to \nu$ nicht mit hinzugenommen, und n (- -)-Nächste-Nachbarn-Paare bei $\nu \to \mu$.
Da die exakte Gestalt des Clusters in beide Richtungen $\mu \to \nu$ und $\nu \to \mu$ dieselbe ist, ist auch die Anzahl der „addierten Bonds" dieselbe und kürzt sich weg.

Damit:

$$\frac{\omega(\mu \to \nu)}{\omega(\nu \to \mu)} = \frac{A(\mu \to \nu)}{A(\nu \to \mu)} \cdot (1-P_{add})^{m-n} \stackrel{!}{=} e^{-2\beta J(m-n)}$$

$$\Rightarrow \boxed{\frac{A(\mu \to \nu)}{A(\nu \to \mu)} \stackrel{!}{=} [(1-P_{add})e^{2\beta J}]^{n-m}}$$

Mit der Wahl

$$\boxed{P_{add} = 1 - e^{-2\beta J}}$$

erfüllt, damit $A(\mu \to \nu) = A(\nu \to \mu) = 1$ detaillierte Bilanz, ein wie oben (stochastisch) konstruierter Cluster kann in jedem Fall geflippt werden!
Damit ist das Schema des Wolf-Cluster-Updates klar:

```
i = ran()≠N
stack[u] = i
pointer = 1
old_S = S[i]
new_S = -S[i]
while(pointer ≠0)
  j=stack[--pointer]
  for all neighbors k of j do
     if(S[k]==old_S)
        if(ran() < P_add)
        stack[pointer++]=h
        s[h]=new_S
```

5.5.1 Der Swendsen-Wang-Algorithmus

funktioniert ähnlich wie der Wolf-Algorithmus, nur wird hier nicht ein Cluster konstruiert, sondern viele - ganz wie in unserem hypothetischen Algorithmus zur Herleitung der „improved estimations":

- In jedem Schritt werden stochastisch Bonds (oder Links) zwischen parallel stehenden Nachbarspins mit Wahrscheinlichkeit $P_{add} = 1 - e^{-2\beta J}$ gesetzt
- Dann werden die durch die Links verknüpften zusammenhängenden Cluster (mittels Hashem-Kapelman o.ä.) identifiziert
- Diese individuellen Cluster werden dann mit Wahrscheinlichkeit $\frac{1}{2}$ ($A = \frac{1}{2}$) geflippt

$\rightsquigarrow$ist fast genauso effizient wie Wolf (in der Nähe von T_C, verschwendet jedoch viel Rechenzeit bei hohen Temperaturen)

Es existieren auch „improved estimations" für Magnetisierung, Energie etc.

5.5.2 Cluster-Algorithmus für Potts-Modelle

Wolf genau wie für Ising-Modell, nur:

- Wähle für Seed-Spin mit Wert $\mathcal{O}_i \epsilon \{1, ..., q\}$ zufällig neuen Wert aus $\{1, ..., q\} \backslash \{\mathcal{O}_i\}$
- $\boxed{P_{add} = 1 - e^{-\beta J}}$ (statt $1 - e^{-2\beta J}$, da ein gebrochenes Bond J kostet, nicht $2J$)

Swendsen-Wang analog, indiviuelle Cluster werden mit Wahrscheinlichlkeit $\frac{1}{q}$ nach$\sigma_i \epsilon \{1, ..., q\}$ geflippt

5.5.3 Cluster-Algorithmus für XY- und Heisenberg-Modelle

Wiederum Wolf wie für Ising, nur Spins im Cluster in eine vorher zufällig gewählte Richtung $\vec{n}$ ($|\vec{n}| = 1$) zeigen.
Die Wahrscheinlichkeit zur Addition von Bonds/Links zwischen Nachbar-Spins, die auf derselben Seite der zu $\vec{n}$ senkrechten Halbebene liegen (d.h. $\vec{n} \cdot \vec{S_i} > 0$) muss, um detailed balance zu erfüllen

$\boxed{P_{add}(S_i, S_j) = 1 - e^{-2\beta(\vec{n} \cdot \vec{S_i})(\vec{n_i} \cdot \vec{S_i})}}$

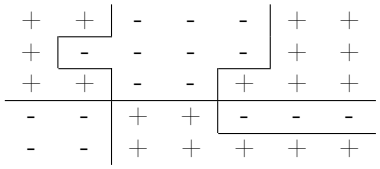
Senkrechte Spins mit $\vec{n_i} \cdot \vec{S_i} < 0$ werden nicht gelinkt!

„Improved Estimator" für die Suszeptibilität
Wir zeigen nun, dass χ im Wesentlichen die mittlere Größe $\langle n \rangle$ der im Wolf-Algorithmus geflippten Cluster ist:

$\chi = \beta \langle n \rangle$

Dazu betrachten wir eine (nur hypothetische) Variante des Wolf-Algorithmus. Statt von einem Seed-Spin einen Cluster durch sukzessive stochastische Addition von Bonds zu konstruieren, verbinden wir erst überall im Gitter parallel stehende Nachbarspins durch Setzen von Bonds mit Wahrscheinlichkeit P_{add}.

z. B.:



Erst jetzt wählen wir einen beliebigen Seed-Spin und flippen dann den dazugehörigen Cluster

Dann können wir die Magnetisierung wie folgt schreiben:
$M = \sum_c S_c n_c$
wobei c ein Label für die verschiedenen Cluster ist, $S_c$der zugehörige Spin-Wert (± 1), und n_c die Größe des Clusters c.

Dann ist

$$\begin{aligned}\langle M^2 \rangle &= \left\langle \sum_c S_c n_c \sum_{c'} S_{c'} n_{c'} \right\rangle \\ &= \left\langle \underbrace{\sum_{c \neq c'} S_c S_{c'} n_c n_{c'}}_{= 0 \text{ da } S_c \text{ und } S_{c'} \epsilon [\pm 1] \text{ unkorelliert}} \right\rangle + \left\langle \sum_c S_c^2 n_c^2 \right\rangle \\ &= \left\langle \sum_c n_c^2 \right\rangle\end{aligned}$$

und $\langle m^2 \rangle = \frac{1}{N^2} \left\langle \sum_c {n^2}_c \right\rangle$
Die Wahrscheinlichkeit, dass ein Cluster im Wolf-Algorithmus geflippt wird, ist $p_c = \frac{n_c}{N}$, denn die Wahrscheinlichkeit einen Seed-Spin aus c zu wählen ist proportional zu n_c.
Damit ist die mittlere Größe der geflippten Cluster:
$\langle n \rangle = \left\langle \sum_c p_c n_c \right\rangle = \frac{1}{N} \left(\sum_c n_c^2 \right) = N \langle m^2 \rangle = T \cdot \chi \quad (T \geq T_C!)$

Dec 03, 01 20:08 **ising.c** Page 1/2

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include "defs.h"

int main(int argc, char** argv)
{
  int L, N, step, step_max, seed, pc, i, j, m, new_s, old_s, pointer,
    itemp, M, E, *spin, *stack, **neighb;
  float  T, prob, mm, ee, mag, mag1, mag2, mag4,
         e, e2, e4, chi, chi2, c, c2, g;

  L        = 0;
  T        = 1.0;
  seed     = 54321;
  step_max = 10000;
  pc       = 0;
  if (!ReadCommandLine(argc,argv,L,T,seed,step_max,pc))
    exit(1);
  N    = L*L;
  seed = -seed;
  ran3 ( &seed );

  spin   = ivector ( 1, N );
  stack  = ivector ( 0, N );
  neighb = imatrix ( 1, N, 1, 4 );

  for ( i=1; i<=N; i++ ) {
    neighb[i][1]=i+1;         // right  neighbor
    neighb[i][2]=i+L;         // top    neighbor
    neighb[i][3]=i-1;         // left   neighbor
    neighb[i][4]=i-L;         // bottom neighbor
  };
  for ( i=1; i<=L; i++ ) {    // periodic boundary conditions!
    neighb [i*L]        [1] = 1+(i-1)*L;
    neighb [i+(N-L)   ] [2] = i;
    neighb [1+(i-1)*L] [3] = i*L;
    neighb [i]          [4] = i+(N-L);
  };

  for ( i=1; i<=N; i++ ) {
    spin[i] = 1;
    if ( ran3(&seed)<0.5 ) spin[i] = -1;
  };


  for ( itemp = 800; itemp >= 100; itemp-- ) {

    T = 0.01 * (float) itemp;
    prob = 1.-exp(-2./T);
    mag  = 0.;
    mag1 = 0.;
    mag2 = 0.;
    mag4 = 0.;
    e    = 0.;
    e2   = 0.;
    e4   = 0.;

    for ( step = 1; step <= 2*step_max; step++ ) {

      i = 1 + (int) (N*ran3(&seed));
      stack[0] =  i;
      pointer  =  1;
      old_s    =  spin[i];
      new_s    = -spin[i];
      spin[i]  = new_s;

      while (pointer) {
```

Dec 03, 01 20:08 **ising.c** Page 2/2

```
        i = stack[--pointer];
        for ( m = 1; m<=4; m++ ) {
          j = neighb[i][m];
          if ( spin[j] == old_s )
            if ( ran3(&seed)<prob ) {
              stack[pointer++] = j;
              spin[j] = new_s;
            };
        };
      };

      if ( step > step_max ) {
        M = 0;
        E = 0;
        for ( i=1; i<=N; i++ ) {
          M += spin[i];
          for ( m=1; m<=4; m++ ) E += spin[i]*spin[neighb[i][m]];
        };
        mm    = (float) M;
        mag  += mm;
        mag1 += fabs(mm);
        mag2 += mm*mm;
        mag4 += mm*mm*mm*mm;
        ee    = (float) E;
        e    -= E;
        e2   += E*E;
        e4   += E*E*E*E;

      };

    };

    mag  /= ((float) N*step_max);
    mag1 /= ((float) N*step_max);
    mag2 /= ((float) N*N*step_max);
    mag4 /= ((float) N*N*N*N*step_max);
    e    /= ((float) N*step_max);
    e2   /= ((float) N*N*step_max);
    e4   /= ((float) N*N*N*N*step_max);


    chi = N*(mag2-mag1*mag1);
    c   = N*(e2-e*e);
    g   = (3.-mag4/(mag2*mag2))/2.;

    printf("%6.3f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n",
           T, mag, mag1, mag2, mag4, chi, e/2., c, g);

  };

}
```

Apr 23, 01 0:39 **ising.c** Page 1/3

```
#include<stdio.h>
#include<math.h>

#define N_BITS 31
#define IRND1  250
#define IRND2  103
#define MAX_RN 0X7FFFFFFF
#define MAX_R2 0X40000000

int ir[256], irndx1[256], irndx2[256];
int rndini();

main ( void )
{
  int i, j, counter, seed,
    L = 128, N = L*L, step_max = 100000,
    x, y, itemp, M, E, prob[5], step, itmp,
    spin[N], field[N], nxm[N], nxp[N], nym[N], nyp[N];
  double  mm, ee, mag, mag1, mag2, mag4, e, e2, e4, chi, chi2, c, c2, g, T;


  seed    = 12381;
  counter = 0;
  rndini  ( seed );

  for ( i=0; i<N; i++ ) {

    j = counter++ & 255;
    ir[j] = ir[irndx1[j]] ^ ir[irndx2[j]];
    spin[i] = 1;
    if (ir[j] < MAX_R2) spin[i] = -1;

    x = i%L;
    y = (int) (i/L);

    nxm[i] = i-1;
    nxp[i] = i+1;
    if ( x==0     ) nxm[i] = (y+1)*L-1;
    if ( x==(L-1) ) nxp[i] = y*L;
    nym[i] = i-L;
    nyp[i] = i+L;
    if ( y==0     ) nym[i] = x+(N-L);
    if ( y==(L-1) ) nyp[i] = x;
    //    printf ("%3d    %3d %3d  %3d %3d\n",i,nxm[i],nxp[i],nym[i],nyp[i]);

  };

  for ( i=0; i<N; i++ )
    field[i] = spin[nxm[i]] + spin[nxp[i]] + spin[nym[i]] + spin[nyp[i]];

  for ( itemp = 250; itemp >= 150; itemp-- ) {

    T = 0.01 * (double) itemp;
    for ( j=0; j<=4; j++ ) prob[j] = MAX_RN * exp( -2.*((double) j) / T );
    mag  = 0.;
    mag1 = 0.;
    mag2 = 0.;
    mag4 = 0.;
    e    = 0.;
    e2   = 0.;
    e4   = 0.;

    for ( step = 1; step <= 2*step_max; step++ ) {

      for ( i=0; i<N; i++ ) {
        itmp = spin[i]*field[i];
        if ( itmp < 0 ) {
          spin[i]  = -spin[i];
          field[nxm[i]] += 2*spin[i];
```

Apr 23, 01 0:39 **ising.c** Page 2/3

```
          field[nxp[i]] += 2*spin[i];
          field[nym[i]] += 2*spin[i];
          field[nyp[i]] += 2*spin[i];
        } else {
          j = counter++ & 255;
          ir[j] = ir[irndx1[j]] ^ ir[irndx2[j]];
          if ( ir[j] < prob[itmp] ) {
            spin[i]  = -spin[i];
            field[nxm[i]] += 2*spin[i];
            field[nxp[i]] += 2*spin[i];
            field[nym[i]] += 2*spin[i];
            field[nyp[i]] += 2*spin[i];
          };
        };
      };
      if ( step > step_max ) {
        M = 0;
        E = 0;
        for ( i=0; i<N; i++ ) {
          M += spin[i];
          E += spin[i]*field[i];
        };
        mm    = (double) M;
        mag  += mm;
        mag1 += abs(mm);
        mag2 += mm*mm;
        mag4 += mm*mm*mm*mm;
        ee    = (double) E;
        e    -= E;
        e2   += E*E;
        e4   += E*E*E*E;

      };

    };

    mag  /= ((float) N*step_max);
    mag1 /= ((float) N*step_max);
    mag2 /= ((float) N*N*step_max);
    mag4 /= ((float) N*N*N*N*step_max);
    e    /= ((float) N*step_max);
    e2   /= ((float) N*N*step_max);
    e4   /= ((float) N*N*N*N*step_max);


    chi = N*(mag2-mag1*mag1);
    c   = N*(e2-e*e);
    g   = (3.-mag4/(mag2*mag2))/2.;

    printf("%6.3f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n",
           T, mag, mag1, mag2, mag4, chi, e/2., c, g);

  };

}


/*************************************************************
*  Initialization of the random number generator.           *
*  (uses a simple linear congruence method to initialize    *
*  each single bit of the array ir - one can do better.     *
*************************************************************/
rndini ( int seed )
{
  int irseed, imult=105, i2h23m1=8388607, warmup=10000;
  int i,j,irt,irhalf;

  irseed=920331+2*seed;
  irhalf=i2h23m1/2;
```

Apr 23, 01 0:39 **ising.c** Page 3/3

```
  irt=irseed;
  for ( i=0; i<=255; i++ ) ir[i]=0;
  for ( i=1; i<=warmup; i++ ) irt = imult*irt % i2h23m1;
  for ( i=0; i<=255; i++) {
    for (j=1;j<=105;j++) irt = imult*irt % i2h23m1;
    for (j=1;j<=N_BITS;j++) {
      irt = imult*irt % i2h23m1;
      ir[i] = (ir[i]<<1);
      if (irt > irhalf) ir[i] |= 1;
    };
  };
  for (i=0; i<=255; i++) {
    irndx1[i] = (6+i)   & 255;
    irndx2[i] = (153+i) & 255;
  };
}
```

6 Grundlagen der Molekular-Dynamik Simulation

Literatur: unleserlich

Wir haben nun 2 verschiedene, häufig verwendete Verfahren zum Studium von komplexen Vielteilchen-Systemen, wie sie in der theoretischen Physik (FKP, stat. Physik, Elementarteilchenphysik, Biophysik) und auch in der theoretischen Chemie häufig verwendet werden kennengelernt:

Monte-Carlo Simulation

Exakte Diagonalisierung

Letztere ist (s. letzte Vorlesung) für quantenmechanische Systeme konzipiert, erstere typischerweise für klassische (Ausnahme: QMC etc.). Bei ersterem wird typischerweise eine stochastische Sequenz von Konfigurationen generiert, deren stationäre Wahrscheinlichkeitsverteilung eine der kanonischen Gesamtheiten ist, mittels derer mit Methoden der statistischen Physik Aussagen über die thermodynamischen Eigenschaften des Systems gewonnen werden.
Typischerweise:

$$\langle O \rangle = \frac{\sum_s O(s) \cdot e^{-\beta U(s)}}{\sum_s e^{-\beta U(s)}}$$

wobei $U(s)$ die (potentielle) Energie des Systems in Konfiguration s ist. In der MC-Simulation durchläuft s eine Markoff-Kette mit

$$\lim_{t\to\infty} P_t(s) = \frac{e^{-\beta U(s)}}{Z},$$

sodass in der Simulation

$$\langle O \rangle = \lim_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} O(s_{t'}),$$

wobei $s_{t'}$ die im MC-Schritt t' erzeugte Konfiguration ist.

Der Ansatz der dritten Methode, die wir nun behandeln ist anders: deterministisch, aber weder mikroskopisch noch klassisch, und eigentlich ganz elementar: man betrachtet die Dynamik von Molekülen (oder größeren Objekten).
Klassische Teilchen genügen Bewegungsgleichungen für Ort und Geschwindigkeit, die sich aus der (klassischen) Hamilton-Funktion (siehe TP 1) für das betrachtete System ergeben.
Typischerweise (Zweikörperwechselwirkung) hat man bei N wechselwirkenden Teilchen:

$$\boxed{H = E_{kin} + E_{pot} = \frac{1}{2m} \sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} u_1(q_i) + \frac{1}{2} \sum_{i,j} u_2(q_i, q_j)}$$

wobei (p_i, q_i) die generalisierten Impulse/Koordinaten der Teilchen sind.
Hier betrachten wir nun den Fall, wo

$\mathbf{p}_i = m_i \cdot \mathbf{v}_i$ ($\mathbf{v}_i$ die Geschwindigkeit des Teilchens i)

$\mathbf{q}_i = \mathbf{r}_i$ ($\mathbf{r}_i$ der Ort des Teilchens i)

$(\dot{q_i} = \frac{\partial H}{\partial p_i}, \dot{p_i} = -\frac{\partial H}{\partial q_i})$

Über die Hamiltonschen Gleichungen erhalten wir sofort die grundlegende Newtonsche Bewegungsgleichung:

(N) $\boxed{m_i \cdot \ddot{\mathbf{r}}_i = \mathbf{F}_i}$ (=Kraft, die auf das Teilchen i wirkt)

$$\mathbf{F}_i = -\nabla_{\mathbf{r}_i} E_{pot} = -\Big\{\nabla u_1 + \sum_j \underbrace{\nabla_{\mathbf{r}} u_2(\mathbf{r}_i, \mathbf{r}_j)}\Big\}$$

$$\downarrow \qquad\qquad \downarrow$$
$$\mathbf{F}_{ext}(\mathbf{r}_i) \qquad \mathbf{F}_{int}(\mathbf{r}_i, \mathbf{r}_j)(= \mathbf{f}_{ij})$$

$\mathbf{F}_{ext}$ ist die von außen auf das Teilchen wirkende Kraft (z. B. Gravitation, elektrisches Feld im Falle von Ladungen, Wand des Behälters...). $\mathbf{F}_{int}$ ist die zwischen den Teilchen wirkende Kraft.

Nach vollständiger Spezifizierung eines Anfangs-Zustandes (d.h. aller Orte und Geschwindigkeiten) ist der Zustand aufgrund von (N) für alle nachfolgenden Zeiten determiniert.

Wie stellt man den Kontakt zur statistischen Physik wieder her?

$$\langle O \rangle_{MC} = \frac{\int d\mathbf{r}_N d\mathbf{v}_N O(\mathbf{r}_N, \mathbf{v}_N)) e^{-\beta H(\mathbf{r},\mathbf{v})}}{\int d\mathbf{r}_N d\mathbf{v}_N e^{-\beta H(\mathbf{r},\mathbf{v})}} \qquad (MC)$$

$$? \updownarrow (*)$$

$$\langle O \rangle_{MC} = \lim_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} O(\mathbf{r}_N(t'), \mathbf{v}_N(t')) \qquad (MC)$$

Nach der Ergondenhypothese ist $(*)$ durch Gleichheit zu ersetzen! D.h. das Ensemble-Mittel (MC) ist identisch mit dem Zeit-Mittel (MD), welches aus der zeitlichen Entwicklung eines einzigen Systems gewonnen wird.

| N.b.: MC: kanonische Gesamtheit bei Temperatur T
MD: mikrokanonische Gesamtheit bei Energie E
d.h. $E(T)$ des Anfangszustandes von MD muss stimmen! (später) |

Die Aufgabe eines MD-Programms ist es also (N) zu integrieren (über eine hinreichend große Zeit) und im Verlauf der Simulation die interessierenden Observablen zu messen.
Die Temperatur T eines Systems aus N klassischen Teilchen in d Dimensionen ist

$$T = \frac{1}{dN} \sum_{i=1}^{N} \mathbf{v}_i^2 \qquad \text{(keine Rotation, } k_B = 1)$$

Bemerkung: Anfangszustand: $\mathbf{v}_i$ zufällig, $\mathbf{v}_i^2 = dT$
(wir erinnern uns: jeder Translations-Freiheitsgrad trägt $\frac{k_B T}{2}$ zur kinetischen Energie bei. Besitzen die Teilchen Rotationsfreiheitsgrade - modifizieren)
Streng genommen wegen Impulserhaltung der Freiheitsgrade weniger
$\rightarrow dN \rightarrow d(N-1) \Rightarrow$Bestimmung von v_{mag} : $T = \frac{1}{d(N-1)} n \cdot v_{mag}^2$
$\Rightarrow v_{mag} = \sqrt{d(1-\frac{1}{N})T}$

Der Druck ergibt sich aus $(k_B = 1)$:

$$pV = NT + \frac{1}{d} \underbrace{\sum_{i=1}^{N} \mathbf{r}_i \mathbf{F}_i}$$

$$\uparrow \qquad\qquad = \sum_{i<j} \mathbf{r}_{ij} \mathbf{f}_{ij}$$

d-dim Volumen

Das Volumen V und die Gesamtenergie (pro Teilchen) ist - modulo Rundungsfehler - konstant, aber Größen wie T und p fluktuieren. In der Simulation muss $\langle T \rangle$ und $\langle p \rangle$ durch Zeitmittelung bestimmt werden.

Außerdem spezifische Wärme: $c_V = \frac{3}{2} k_B \left(1 - \frac{2N \left(\langle E_{kin}^2 \rangle - \langle E_{kin} \rangle^2 \right)}{3(k_B)^2} \right)$

Paar-Korrelationsfunktion: $g(\vec{r}) = \frac{2V}{N^2} \left\langle \sum_{i<j} \delta \left(\vec{r} - \vec{r_{ij}} \right) \right\rangle$ $\rightsquigarrow$ Strukturfaktor

$\rightsquigarrow$Kristall-Ordnung?

$\rightsquigarrow$...

Bemerkung zur expliziten Gestalt der Potentiale u_1 und u_2:
Die Wahl der Wechselwirkungspotentiale u_2 sowie der äußeren Kräfte hängt natürlich vom betrachteten System ab. Der Name Molekular-Dynamik stammt vom ursprünglichen Ziel Eigenschaften atomarer oder molekularer Gase, Flüssigkeiten oder Festkörper zu verstehen - insbesondere den möglichen Phasenübergang von einem in den anderen Aggregat-Zustand. Dies legt sofort bestimmte Energie-, Längen- und Zeitskalen fest, auf denen die relevante Physik durch die Paarpotentiale beschrieben wird. Das wohl bekannteste Beispiel ist das

Lennard-Jones-Potential: $u_2(r_{ij}) = 4e \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right]$ für $r_{ij} = r_i - r_j \leq r_c$

für sphärische Teilchen (Edelgase)
mit:

- e: Wechselwirkungsstärke, Energieskala
- σ: Längenskala
- $\left(\frac{\sigma}{r_{ij}} \right)^{12}$: Abstoßung durch Überlapp der Elektronenwolken
- $\left(\frac{\sigma}{r_{ij}} \right)^{6}$: Van-der-Waals-Term
- $\rightsquigarrow \mathbf{f}_{ij} = \frac{48e}{\sigma^2} \left[\left(\frac{\sigma}{r_{ij}} \right)^{14} - \frac{1}{2} \left(\frac{\sigma}{r_{ij}} \right)^{8} \right] \mathbf{r}_{ij}$ für $r_{ij} < r_c$, 0 sonst

z.B. flüssiges Argon: $\sigma = 3.4\text{Å}$ (Längen-Skala)
$\frac{e}{k_B} = 120k$ (Energie-Skala)
$= 1.3806 \cdot 10^{-16} \frac{erg}{atom}$
$= 1.987 \cdot 10^{-3} \frac{kcal}{mol}$
$= 8.314 \frac{J}{mol}$

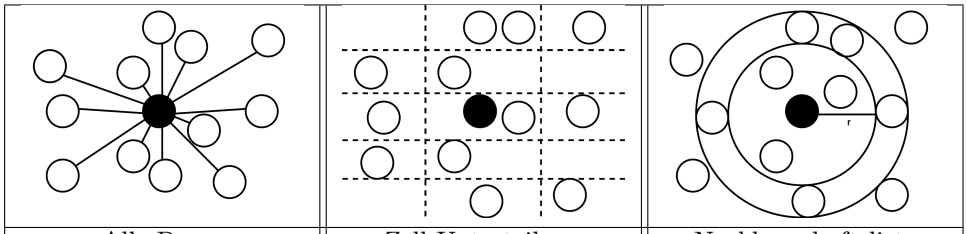
- $m = 39.95 \cdot 1.6747 \cdot 10^{-24} g$ Masse des Argon-Atoms
- $\tau = 2.161 \cdot 10^{-12} s$ Zeitskala
- Zeitschritt $\Delta t = 0.005 \hat{=} 10^{-14} s$
- N-Atome in Kubus der Länge L bei Dichte von $0.942 \frac{g}{cm^3}$
- $\Rightarrow$Länge $L \hat{=} 4.142 N^{\frac{1}{3}}$ Å

Probleme dieser Art haben die Entstehung der MD motiviert - die Entwicklung ging aber rasch in die Breite: von mikroskopischen über mesoskopischen zu makroskopischen Teilchen:

	Durchmesser	Anzahl Moleküle pro Teilchen	
Atome, Moleküle	10^{-10}	1	
Makromoleküle, Polymere	10^{-9}	10^3	
Kolloide, Suspensionen, Rauchpartikel	10^{-8}	10^6	
↓ Puder, Staub	10^{-6}	10^{12}	↑
↓ Sand	10^{-4}	10^{18}	↑
↓ Kies, Schotter	10^{-2}	10^{24}	↑
Planeten			
Galaxien...			

Von oben nach unten ändert sich die relevante Energieskala, langreichweitige Kräfte (sofern keine elektrischen Ladungen beteiligt sind) verlieren an Bedeutung und Adhäsions- und Kontakt-Kräfte werden wichtiger. Außerdem wird die Gravitation relevant (→granulare Medien) und dominiert den astrophysikalischen Bereich (dann auch wieder langreichweitig).
Ladungen, elektrische und Magnetfelder bringen wiederum neue Physik ins Spiel - all diese Szenarien werden mittels MD durch geeignete Wahl der Potentiale in der Hamiltonfunktion bzw. der Kräfte in den Bewegungsgleichungen studiert.

Berechnung der Wechselwirkungen (n.b. $f(r_{ij}) = 0$, für $r_{ij} > r_c$)



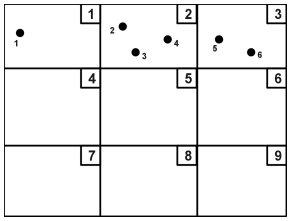
Alle Paare	Zell-Unterteilung	Nachbarschaftsliste

1.) Alle Paare:
Test $r_{ij} < r_c \forall ij$ braucht $O(N^2)$ Operationen
→siehe einfaches Beispielprogramm von letzter Vorlesung
Ziel: Reduktion auf $O(N)$

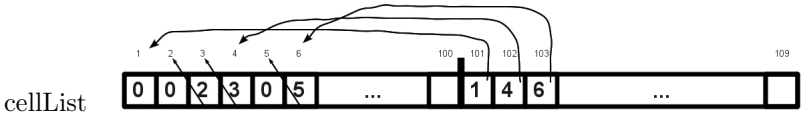
2.) Zell-Unterteilung:
Unterteile Simulationsbereich ($L \times L \times L$) in ein Gitter kleiner Zellen, deren Kantenlängen alle größer als r_c sind. Ordnet man jedem Atom die Zelle zu, in der es sich befindet, so wechselwirkt es nur mit den Atomen, die sich in derselben oder in benachbarten Zellen befinden.
Aus Symmetriegründen nur die Hälfte der Zellen betrachten
⇒es müssen für jedes Atom in 3d nur 14 Zellen (5 in 2d) examiniert werden.
N.b.: $L > 4r_c$, sonst nutzlos
Vor jeder Kraftberechnung: Ordne Atome und Zellen einander zu! ($O(N)$ Operationen)

Daten-Organisation Sogenannte „Linked-List”
Sei nAtom = 100

Beachte: Anzahl von Atomen pro Zelle kann stark variieren, triviale Konzepte verschwenden also viel Speicher.



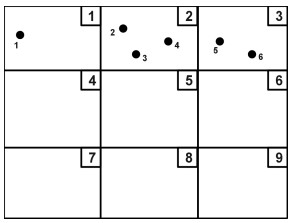
Bilde Liste wo $\forall i = 1, ..., \texttt{nAtom}$
cellList[i] zeigt an, welches Atom
das nächste Atom in der
gleichen Zelle wie i ist
(ï¿œbedeutet i ist letztes Atom)
und $\forall k = 1, ..., \#\text{Zellen}$
cellList[nAtom+k] zeigt 1.Atom in Zelle k an



Algorithmus:

```
int *cellList, cells[NDIM+1];
for(k=1; k<=NDIM; k++) cells[k]=region[k]/rCut;
cellList=AllocVecI (nAtom+cells[1]*cells[2]*cells[3]);
for(k=1; k<=NDIM; k++) inv Wid[4]=cells[4]/regin[4];
for(n=Atom+1;n<=nAtom+cells[1]*cells[2]*cells[3];n++) cellList[4]=?
for(n=1;n<=nAtom;n++){
c=( (int)((r[3][n]+regionH[3])*invWid[3])*cells[2]
+(int)((r[2][n]+regionH[2])*invWid[2])*cells[2])*cells[1]
+   (int)((r[1][n]+regionH[1])*invWid[1])
+ nAtom+1;
cellList[n]=cellList[c];
cellList[c]=n;
};
```

nAtom= 100;



n	L	
1	101	cellList[1] = 0
		cellList[101] = 1←
2	102	cellList[2] = 0
		cellList[102] = 2
3	102	cellList[3] = 2
		cellList[102] = 3
4	102	cellList[4] = 3
		cellList[102] = 4←points to 1.Atom in cell 2
5	103	cellList[5] = 0
		cellList[103] = 5
6	103	cellList[6] = 5
		cellList[103] = 6←
		⋮

Anschließend werden die Kräfte berechnet - hierbei werden Zell-Paare (jede Zelle mit sich und mit der Hälfte der anderen) nach möglichen Wechselwirkungen abgesucht.
Bei der Bestimmung der Zell-Paare sind die periodischen Randbedingungen zu berücksichtigen.

```
m1Z=1,...,cells[3]
 m1Y=1,...,cells[2]
  m1X=1,...,cells[1]
   m1=( (m1Z-1)*cells[2] + (m1Y-1) )*cells[1] + m1X + nAtom + 1
    offset=1,...,14
     m2X=m1X + i ??? fX[offset]            i ??? fX 0110
     shift[1]=0                            i ??? fY 0011 ....
     if(m2X > cells[1]) {
       m2X=1
       shift[1]=region[1];
     } else if(m2X==0) {
       m2X=cells[1]
       shift[1]=-region[1]
     }
     m2Y=...
     .
     .
   m2=( (m2Z-1)*cells[2] + (m2Y-1) )*cells[1] + m2X + nAtom + 1
   j1=cellList[m1]
   while(j1>0) {
```

```
    j2=cellList[m2]
    while(j2>0) {
     if(m1 !m m2 // j2<j1)
             .
             .
     Abstand j1-j2 berechnen
     Kraft j1-j2 berechnen
    }
    j2=cellList[j2]
  }
  j1=cellList[j1]
```

Je nach Computer-Architektur kann man die Schleifen auch umorganisieren (z.B. außen über relative Offsets, innen über Zellen).

3.) Nachbarschaftsliste:
Nur ein Bruchteil der durch die Zell-Methode erfassten Atome liegt innerhalb des WW-Radius ($\frac{4\pi}{81} \approx 0.16$ in 3d, $\frac{\pi}{9} \approx 0.15$ in 2d).
Besser wäre die Konstruktion einer Nachbarschaftsliste mit Hilfe der Zell-Methode, die über mehrere Zeitschritte (10-20) hinweg brauchbar ist. Dazu ersetzt man r_c durch $r_n = r_c + \Delta r$ im Nachbarschaftstest und nutzt die sich langsam verändernde mikroskopische Umgebung aus. Wegen $r_n > r_c$ erscheinen während der Zeitschritte in denen dieselbe Liste benutzt wird keine neuen Nachbar-Paare, sofern Δr groß genug gewählt ist.

$$\Delta r = \begin{cases} \text{umgekehrt proportional zur Erneuerungsrate der Liste} \\ \text{bestimmt die Anzahl der zusätzlichen, nicht-ww Nachbarpaare, die in der Liste aufgeführt sind} \end{cases}$$

$\rightsquigarrow \Delta r$ beeinflusst Rechenzeit und Speicherplatz-Anforderung

Entscheidungskriterium für die Erneuerung der Nachbarschaftsliste basiert auf der Maximal-Geschwindigkeit der Teilchen in jedem Zeitschritt.
Man wartet bis:

$$\sum_{steps} (\max_i |v_i|) > \frac{\Delta r}{2\Delta t} \quad (*)$$

bevor man ein „refreshing" durchführt.
Üblicherweise (bei Flüssigkeitsdichten): $\Delta r \sim 0.3 \sim 0.4$

Algorithmus:
Neue Variablen:

```
real dispHi, rNebrshell, rrMax;
int **nebrTab, nebrNon, nebrTabFac, nebrTabLen, nebrTabMax;
in set Paarung
  for(k=1;k<=NDIM;k++){
     cells[k]=region[k]/(rCut+rnebrShell)
  }
in setup Job
  nebrNon=1
input: INAME  (nbrTabFac)
       RNAME  (rnebrshell)    (=Δr)
in AllocArray
  nebrTab=AllocMatI(2,nebrTabMax)
in Evalprops
  rrMax=0
for(n=1;n<=nAtom;n++){
```

```
      ...
      if(rr>rrMax) rrMax=rr
   }
   dispHi=dispHi+sqrt(rrMax)*deltaT
   if(dispHi>0.5*rNebrShell)
      nebrNon=1   (Kriterium (*) erfüllt)
   in SingleStep
      if(nebrNon){
         nebrNon=0
         dispHi=0
         BuildNebrList 0
      }
```

Nachbarschaftslisten-Erneuerung benutzt Zell-Methode:

```
   BuildNebrList 0 {
      .
      .
      rrNebr=Sqr(rCut + rNebrShell)
      .
      .
      nebrTabLen=0
      for(m1Z=1;m1Z<=cells[3];m1Z++){
       .
       .
       if(rr<rrNebr){
         nebrTabLen++;
         if(nebrTabLen>nebrTabMax)
           ErrExit(‘‘neighbor list overflow’’)
         nebrTab[1][nebrTabLen]=j1
         nebrTab[2][nebrTabLen]=j2
       }
   ....
```

Nun kann die Nachbarschaftsliste zur Kraftberechnung gebraucht werden:

```
   ComputeForces()
      .
      .
     for(n=1;n<=nebrTabLen;n++){
       j1=nebrTab[1][n]
       j2=nebrTab[2][n]
       for(k=1;k<=NDIM;k++){
        dr[k]=r[k][j1]-r[k][j2]
          .
          .
        rr<rrCut?
       }
     }
```

N.B.:
ApplyBoundaryCond nur beim Refreshing der Nachbarschaftsliste erforderlich.

Weniger gebräuchlich:
Replication-Methode (kopiere Teile des Systems, wegen BC)
Multiple-Timestep-Methode (Kräfte weiter entfernter Paare werden weniger häufig berechnet)
Kraft-Tabellierung (z.B. bei exp-Funktion)

Integrations-Methoden:
Es gibt viele numerische Integrations-Methoden - viele können von Anfang an wegen einer zu häufigen Kraft-Berechnung verworfen werden. Jede Methode, die die Kraft mehr als einmal pro Zeitschritt berechnet ist verschwenderisch - wenn nicht Δt entsprechend vergrößert werden kann. Bei LJ-Potentialen (starke Singularität) gibt es sowieso eine Obergrenze für Δt, so dass die wohlbekannten Runge-Kutta-artigen Methoden nicht in der Lage sind, den Zeitschritt über diese Grenze hinaus anzuheben.
Das selbe gilt für Methoden mit adaptiver Schrittweite, bei denen Δt dynamisch geändert wird um eine bestimmte Genauigkeit zu bewahren. Aufgrund der lokalen Rearrangements der Atome erfährt jedes eine rapide wechselnde Umgebung, die einen solchen Ansatz fehlschlagen lassen.

Nur 2 Klassen von Methoden sind heutzutage weit verbreitet:
- Leap-Frog-Technik
- Predictor-Corrector-Verfahren

Merke:
Eine hohe Genauigkeit in der Trajektorien-Berechnung zu erzielen, ist weder ein realistisches, noch ein praktisches Ziel! (Chaos für $t > 2,3$ Kollisionszeiten)

Wichtig:
Kriterium für Auswahl einr Methode ist Energie-Erhaltung und die Reproduzierbarkeit bestimmter räumlicher und zeitlicher Korrelationen.

1.) Leap-Frog und Verlet-Methoden (algebraisch äquivalent)
Taylor-Funktion:

$x(t+h) = x(t) + h\dot{x}(t) + (\frac{h^2}{2})\ddot{x}(t) + O(h^3)$
$h = \Delta t$, $\ddot{x}(t) = f(t)$
$x(t-h) = x(t) - h\dot{x}(t) + (\frac{h^2}{2})\ddot{x}(t) + O(h^3)$
$\overset{\text{Addition}}{\Rightarrow}$Verlet-Formel $\boxed{x(t+h) = 2x(t) - x(t-h) + h^2\ddot{x}(t) + O(h^4)}$
$\dot{x}(t)$ kann mittels $(x(t+h) - x(t-h))/2h$ berechnet werden $(O(h^2))$

Schreibe Taylor um: $x(t+h) = x(t) + h(\dot{x}(t) + (\frac{h}{2})\ddot{x}(t)) + O(h^3)$

$$\rightsquigarrow \boxed{x(t+h) = x(t) + h\dot{x}(t+\frac{h}{2}) + O(h^3)}$$

$\dot{x}(t+\frac{h}{2}) = \dot{x}(t) + \frac{h}{2}\ddot{x}(t) + O(h^2)$ (Leap-Frog)
$\dot{x}(t-\frac{h}{2}) = \dot{x}(t) - \frac{h}{2}\ddot{x}(t) + O(h^2)$

$$\rightsquigarrow \boxed{\dot{x}(t+\frac{h}{2}) = \dot{x}(t-\frac{h}{2}) + h\ddot{x}(t) + O(h^3)}$$

$\dot{x}(t)$ mittels $\dot{x}(t \mp \frac{h}{2}) \pm \frac{h}{2}\ddot{x}(t)$

Predictor-Corrector-Methoden

Predictor-Corrector-Methoden benutzen mehrere Daten, die zu einem oder mehreren früheren Zeitschritten berechnet worden sind.
Adams-Verfahren: Benutze Beschleunigungsdaten von früheren Zeitschritten
Nordsieck-Verfahren:Benutze höhere Ableitung der BEschleunigung zum momentanen Zeitschritt
Verfahren von höherer Ordnung als Leap-Frog, also auch mehr Speicher und Rechenaufwand

Ziel: Löse $\ddot{x} = f(x, \dot{x}, t)$
$P()$ = Predictor-Schritt der Berechnung, $C()$ = Corrector-Schritt der Berechnung
$P(x)$ für $t + h$ ist eine Extrapolation von Werten, die zu einem früheren Zeitpunkt berechnet wurden.

$$P(x): \boxed{x(t+h) = x(t) + h\dot{x}(t) + h^2 \sum_{i=1}^{k-1} \alpha_i f(t + (1-i)h)}$$

für gegebenes k sogenannte Adams-Bashforth-Methode
Exakte Resultate für $x(t) = x^q$, sofern $q \leq k$, Fehler $\mathcal{O}\left(h^{k+1}\right)$ und die Koeffizienten $\{\alpha_i\}$ die Gleichung

$$\sum_{i=1}^{k-1} (1-i)^q \alpha_i = \frac{1}{(q+1)(q+2)} \quad q = 0, \ldots, k-2$$

erfüllen.
Analog für $\dot{x}$.

$$P(\dot{x}): \boxed{h\dot{x}(t+h) - x(t) + h^2 \sum_{i=1}^{k-1} \alpha_i' f(t + (1-i)h)}$$

mit Koeffizienten, die

$$\sum_{i=1}^{k-1} (1-i)^q \alpha_i' = \frac{1}{q+2}$$

erfüllen.
Nun wird $f(t+h)$ mithilfe der „vorausgesagten Werte" von x und $\dot{x}$ berechnet - die Korrekturen werden mithilfe der Adams-Moulton-Methode gemacht.

$$C(x): \quad x(t+h) = x(t) + h\dot{x}(t) + h^2 \sum_{i=1}^{k-1} \beta_i f(t + (2-i)h)$$

$$C(\dot{x}): \quad h\dot{x}(t+h) = x(t+h) - x(t) + h^2 \sum_{i=1}^{k-1} \beta_i' f(t + (2-i)h)$$

mit Koeffizienten, die aus

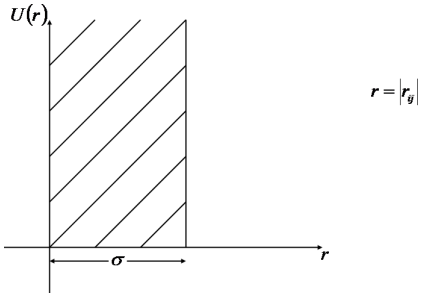
$$\sum_{i=1}^{k-1} (2-i)^q \beta_i = \frac{1}{(q+1)(q+2)} \quad , \quad \sum_{i=1}^{k-1} (2-i)^q \beta_i' = \frac{1}{q+2}$$

erhalten werden
Die vorausgesagten Werte x, $\dot{x}$ erscheinen nirgendwo in den Corrector-Formeln außer bei der Berechnung der Kräfte f.

$k = 4$ ($x^1/_{24}{}^3$)	1	2	3
$P(x)$	19	−10	3
$P(\dot{x})$	27	−22	7
$C(x)$	3	10	−1
$C(\dot{x})$	7	6	−1

Einfache Anwendung eines Event-Driven-Algorithmus: Hart-Kugel-Flüssigkeit

Für theoretische Studien ist das sogenannte Hart-Kugel-Modell von Interesse, bei dem die Teilchen nur im Moment des Zusammenstoßes wechselwirken und dann z. B. einen elastischen Stoß ausführen.
Das Paarpotential wäre in diesem Fall



$r = |r_{ij}|$

Im Prinzip kann man natürlich die Bewegungsgleichung integrieren mit besonderer Beachtung der Singularität bei $r = \sigma$, aber viel effektiver ist hier (wie auch bei der Simulation von granularen Medien) ein sogenannter Event-Driven-Algorithmus:

Man braucht nur zu wissen, wann im System der nächste Zusammenstoß zweier Teilchen zu erwarten ist. Bis dahin bewegen sich die Teilchen geradlinig weiter. Die Zeitdifferenz bis zum erwähnten Moment sei τ: Man verschiebt alle Teilchen ($i = 1, \ldots, N$) des Systems um $\tau \mathbf{v}_i$ zu neuen Orten und ändert dann nur die Geschwindigkeit der beteiligten Teilchen. Das wiederholt sich, bis das Ende der Simulation (t_{max}) erreicht ist.

2 Teilchen stoßen, wenn $|\mathbf{r}_{ij}| = |\mathbf{r}_i - \mathbf{r}_j| < \sigma$

und sie aufeinander zufliegen $\mathbf{v}_{ij} \cdot \mathbf{r}_{ij} < 0$ ($\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$)

Die Zeit t_{ij} bis zum möglichen Stoß ist Lösung von

$$\mathbf{r}_{ij} + t_{ij}\mathbf{v}_{ij})^2 < \sigma^2$$

$$\Rightarrow \boxed{t_{ij} = \left(-\mathbf{v}_{ij} \cdot \mathbf{r}_{ij} - \sqrt{\left(\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}\right)^2 - \left(\mathbf{r}_{ij}^2 - \sigma^2\right)\mathbf{v}_{ij}^2}\right) / \mathbf{v}_{ij}^2}$$

Geschwindigkeitsänderung beim Stoß der Teilchen (elastisch)

Impuls-Erhaltung $\quad m_i \Delta \mathbf{v}_i = m_j \Delta \mathbf{v}_j$

$+$ Energie-Erhaltung $\Rightarrow$ $m_i \Delta \mathbf{v}_i = -2 m_{ij} \left(\mathbf{e} \cdot \mathbf{v}_i\right) \mathbf{e}$, $\mathbf{e} = \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}$, $m_{ij} = \frac{m_i m_j}{m_i + m_j}$ reduzierte Masse

Also schematisch (Messung immer bei $t = t_{ev}$)

Init { Initialisiere $\mathbf{r}_i$, $\mathbf{v}_i$
Berechne t_{ij} und mögliche Stoßpartner

$\rightarrow \circledast$ Siehe kleinste Stoßzeit $\tau = t_{kl}$ k, l =aktuelle Stoßpartner

Ende { if $(t + \tau \geq t_{max})$ stop }

Messung {
if $(t + \tau \geq t_{ev})$
$\mathbf{r}_i = \mathbf{r}_i + \mathbf{v}_i \left(t_{ev} - t\right)$
$t_{ij} = t_{ij} - t_{ev} + t$
$\tau = \tau - t_{ev} + t$
Wähle neues t_{ev}

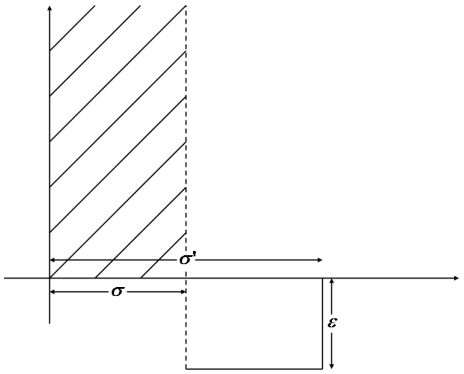
Freier Flug {
if $\mathbf{r}_i = \mathbf{r}_i + \mathbf{v}_i \tau$
$t = t + \tau$
$t_{ij} = t_{ij} + \tau$ für alle i, j

Stoß {
if $v_k = v_k + \Delta \mathbf{v}_k$ s. o.
$v_l = v_l + \Delta \mathbf{v}_l$ s. o.

Berechne t_{kl} neu

Modifizierung

a)

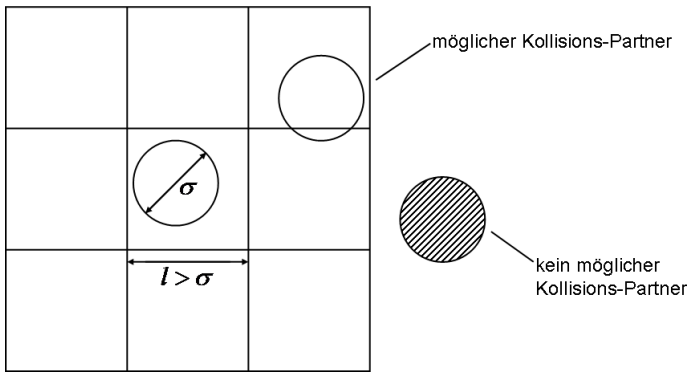


„reaktiver Hartkugel-Stoß"
(Bei σ' wird E_{kin} um ε größer.)
$m_i \Delta \mathbf{v}_i = m_{ij} \left(-\mathbf{e} \cdot \mathbf{v}_{ij} - \sqrt{(\mathbf{e} \cdot \mathbf{v}_{ij})^2 + 2\varepsilon / m_{ij}} \right) \mathbf{e}$

b) Harte Kugeln im konstanten Kraftfeld (z. B. Schwerkraft)
t_{ij} ergibt sich aus $\left(r_{ij} + v_{ij} t_{ij} + \frac{b_{ij}}{2} t_{ij}^2 \right)^2 = b^2$
$b_{ij} = b_i - b_j$ Differenz der Beschleunigung durch Kraftfeld
Stoßgesetze ändern sich nicht

c) Harte Wände (statt p. b. c.)

Problem: Zur Bestimmung der nächsten Kollision eines Atoms sind $\mathcal{O}(N)$ Operationen erforderlich. $\rightsquigarrow$ schlechte Performance
Ausweg: Zell-Unterteilung wie in Soft-Sphere-Modell.



2 Arten von Ereignissen: 1) Kollision
2) überqueren der Zell-Grenze

Process Collision

Nach einer Kollision werden die beiden Geschwindigkeiten $\boxed{\Delta v_i = -\left(\mathbf{r}_{ij} \cdot \mathbf{v}_{ij}\right) \mathbf{r}_{ij} / \mathbf{r}_{ij}^2}$ der Stoßpartner geändert, alle Atome in den Nachbarzellen der Zelle, in der der Stoß stattgefunden hat, müssen auf Kollision mit den soeben kollidierten Atomen untersucht und die lokale Zeit der Atome in der Kollisions-Zelle sowie von deren Nachbarn aktualisiert werden.
(lokale Zeit atomTime[i] gibt an, wann das letzte Ereignis von Atom i stattgefunden hat)

Process Cell Crossing

Nach einem Überqueren der Zell-Grenze wird das betreffende Atom von der alten in die neue Zelle verschoben und dort auf neue mögliche Kollisionen mit Atomen der Nachbarzellen untersucht.

Implizit haben wir angenommen, dass das System weiß, wann das nächste Ereignis fällig ist, und ob es eine Kollision oder eine Zellüberschreitung ist, und welche Atome involviert sind. Dies erfordert die Existenz eines Ereignis-Kalenders mit den folgenden Eigenschaften:

1) er produziert das nächste Ereignis
2) ist einfach modifizierbar
   enthält viel zukünftige Ereignisse,
   bei Kollisionen werden neue hinzugefügt und einige gelöscht

Die Implementierung des Kalenders besprechen wir später - zunächst brauchen wir die Info, dass die Routine

nextEvent ()

das nächste Ereignis liefert, indiziert durch zwei Zahlen:

evIdA und **evIdB**
$$\begin{cases} \in \{1,\ldots,N\} \text{ wenn Kollision} \\ > ATOM\_LIMIT + 100 \text{ wenn Cell-Crossing} \\ \ulcorner < ATOM\_LIMIT + 100 \\ \quad > N \text{ für andere Ereignisse} \lrcorner \end{cases}$$
$\cap$
$\{1,\ldots,N\}$ ein Atom

Die Routine

PredictEvent ()

berechnet alle neu entstehenden Ereignisse, die nach einer Kollision oder einem Cell-Crossing entstehen, mittels

ScheduleEvent ()

werden diese dann in den Kalender eingefügt und alte Ereignisse mittels

DeleteEvent ()

gelöscht (d. h. bei Kollision von i und j, d. h. Ereignisse, die Atome i oder j enthalten)

Struktur des Programms:

```
main: SetParams()
      SetupJob()
      while (moreCycles): SingleEvent()



SetupJob(): AllocArrays()
            InitCoords()
            InitVels()
            timeNow=0
            StartRun()
```

```
SingleEvent(): NextEvent()↙evIdA & evIdB globale Variablen!
               if (evIdB<=ATOM_LIMIT)
               {
                   ProcessCollision ()
               }
               else if (evIdB<ATOM_LIMIT+100)
               {
                   ProcessCellCrossing()
               }
               else
               {
                   UpdateSystem
                   ...
               }
```

`Start Run()`: Initialisiere linked Zell-Liste `cellList[1,...,n Atom + #cells]` (wie in MD mit lokale `und in Cell[k][n].` Zellunterteilung)
Zeit des `}`→ `atomtime[n]=timeNow`
Atoms n `InitEventList()` ←Initialisiere Ereigniskalender!
`PredictEvent(n,-1)` ←Berechne für jedes Atom
↑ erstes Ereignis
$n = 1, \ldots, N$

```
UpdateSystem(): for (n=1, n<=N, n+1) UpdateAtom(n)
UpdateAtom(id): r[k][id]=r[k][id]*(timeNow-atomtime[id])
                atomtime[id]=timeNow
```

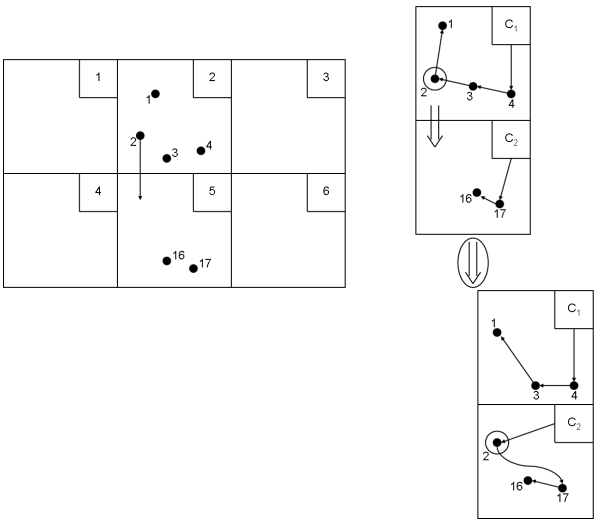
↙sind globale Variablen, aus `NextEvent()` bekannt!

```
ProcessCollision(): UpdateAtom(evIdA)
                    UpdateAtom(evIdB)
                    cellRange[1,3 und 5] = −1
                              2,4 und 6  = +1   } für Predict-Event
                                                =r_i − r_j mod. per. R.B.
                                                        ↓
                    Neue Geschwindigkeiten: Δv_i = −(r_ij · v_ij) r_ij / r²_ij
                                                 i,j∈{evIdA,evIdB}
                    PredictEvent(evIdA,0)
                    PredictEvent(evIdB,evIdA)
```

```
ProcessCellCrossing(): Update(evIdA)
                       n=Nummer der aktuellen Zelle von Atom evIdA
                       while(cellList[n]! = evIdA)n = cellList[n]   } Setze Pointer von
                       cellList[n] = cellList[evIdA]                } evIdA um auf
                                                                      nächstes Element
                       cellRange[1,3 und 5] = −1
                                 2,4 und 6  = +1   } für Predict-Event
                       k=evIdB-ATOM_LIMIT-100 (Index der durchquerten Zellen)
                       rv[k][evIdA]>0: cellRange[2k-1]=-1
                                       inCell[k][evIda]++
                                      -per. R.B.-
```

```
                    <0: cellRange[2k]=+1
                        inCell[k][evIdA]--
                       -per. R.B.-
PredictEvent(evIdA,evIdB)
n=Nummer der neuen Zelle
cellList[evIdA]=cellList[n]
                ↙setze Pointer von n auf evIdA
cellList[n]=evIdA
```

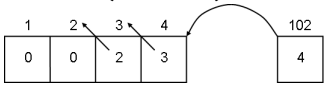
`ProcessCellCrossing()`: Verschiebt Atom `evIdA` in Nachbarzelle (nach Maßgabe der Geschwindigkeiten)



`n=nAtom+{(inCell[2][evIdA]-1)*cells[1]+inCell[1][evIdA]}`

Bsp. hier: `evIdA=2`

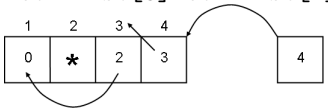
`n=102 (Cell-No)`



`while cellList[n]≠2` `n=cellList[n]`

`evIdA`

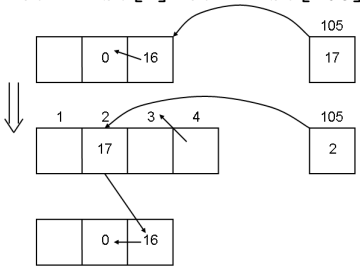
`cellList[3]=cellList[2]=1`



`...`

`n=105` (neue Zelle für Atom 2)

`cellList[2]=cellList[105]`



`PredictEvent(int na, int nb):`

Berechne Index $k \in \{1,2,3\} \hat{=} x,y,z$ der Zellwand, die von Atom `na` als erstes erreicht wird, sowie die Zeit `tm[k]`, die es dazu braucht

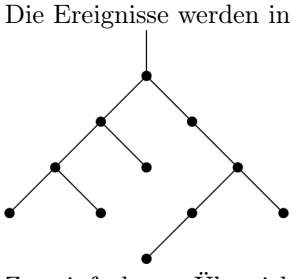
`evCode=100+k`

`ScheduleEvent(na, ATOM_LIMIT + evCode, )`$timeNow + tm[k]$

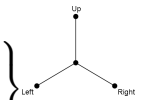
Check für mögliche Kollisionen in ausgewählten (nicht alle → `cellRange`[...]!) Nachbarzellen ⓝ (beachte p. R.B.)
Loop über diese Nachbarzellen:

```
n=Nummer der Zelle
for (n=cellList[n]; n>0; n=cellList[n])
{
    if (n≠na, n≠nb, (nb≥0 od n<na))
    {
        tInt=timeNow-atomTime[n]
        dr = r_na − (r_n + v_n * tInt)
        dv = v_na − v_n
        b = dr · dv
        if (b<0)
        {
            d = b² − dv² · (dr² − 1)
            if (d>0)
            {
              t = (−b − √d) /dv²
              ScheduleEvent(na,n,timeNow+t)
            }
        }
    }
}
```

Der Ereignuskalender mit sämtlichen notwendigen Infos wird in einem Array `tree`$[1,\ldots,9]$`[`$1,\ldots,$`poolsize]` abgespeichert.
Die Ereignisse werden in einem binären Baum arrangiert:



Zur einfacheren Übersicht definieren wir



`treeLeft` = `tree`[1]
`treeRight` = `tree`[2]
`treeUp` = `tree`[3]

`treeCircAL` = `tree`[4]
`treeCircBL` = `tree`[5]
`treeCircAR` = `tree`[6]
`treeCircBR` = `tree`[7]

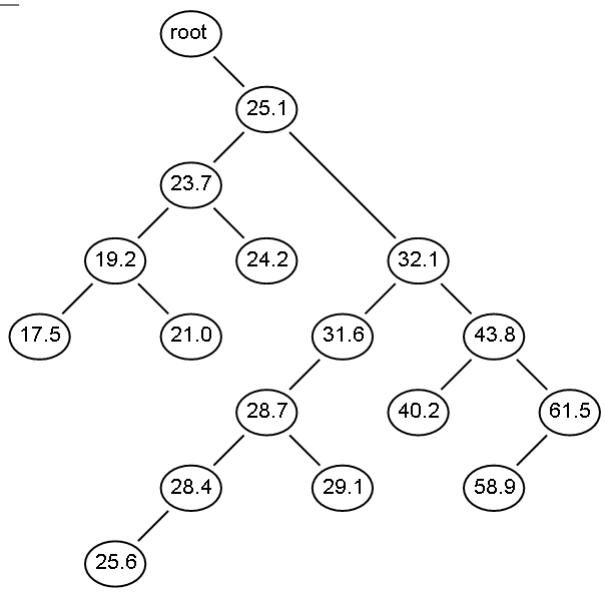
} zirkulär gelinkte Kisten, die das Auffinden aller Ereignisse mit einem bestimmten Atom erleichtern

`treeIdA` = `tree`[8] } Teilchen-Nummer der (des)
`treeIdB` = `tree`[9] } beteiligten Atome (Atoms)

D. h. Jeder Satz `tree`[1, . . . , 9][`evId`] von 9 (integer) Einträgen definiert einen Knoten im binären Baum.
Außerdem noch eine real-Variable für Ereignis: die Zeit

```
treeTime[evId]
```

Der Ereignis-Baum fluktuiert in seiner Größe, da im Verlaufe der Simulation Knoten hinzugefügt und gelöscht werden.
Ein Pool von leeren Knoten steht als Reservoir zur Verfügung.
Der erste Knoten enthält kein Ereignis, sondern dient als feste Verankerung „`root`".
Einer seiner Pointer dient dem Zugriff auf den Pool, dessen Knoten mit einer gelinkten Liste verknüpft sind (→ `treeCivicAR`).
Die `nAtom` Knoten unmittelbar unter dem `root`-Knoten entsprechen `CellCrossings`, denn es ist immer mindestens ein Ereignis dieser Art pro Atom vorhanden.
Diese Knoten dienen auch als Verankerung der zirkulären Listen, die mit diesem Atom verknüpft sind.
Die übrigen Knoten werden dynamisch Kollisionen und anderen Ereignissen zugeordnet.
Wir stellen nun die Routinen vor, die Ereignisse in den Baum einfügen, das nächste Ereignis bestimmen oder Ereignisse löschen.
Der binäre Baum ist so aufgebaut, dass für jeden Knoten mit Ereignis-Zeit t der linke Nachfolge-Knoten eine frühere Ereignis-Zeit $t_1 < t$ hat und der rechte Nachfolge-Knoten eine spätere Ereignis-Zeit $t_2 > t$.
z. B.



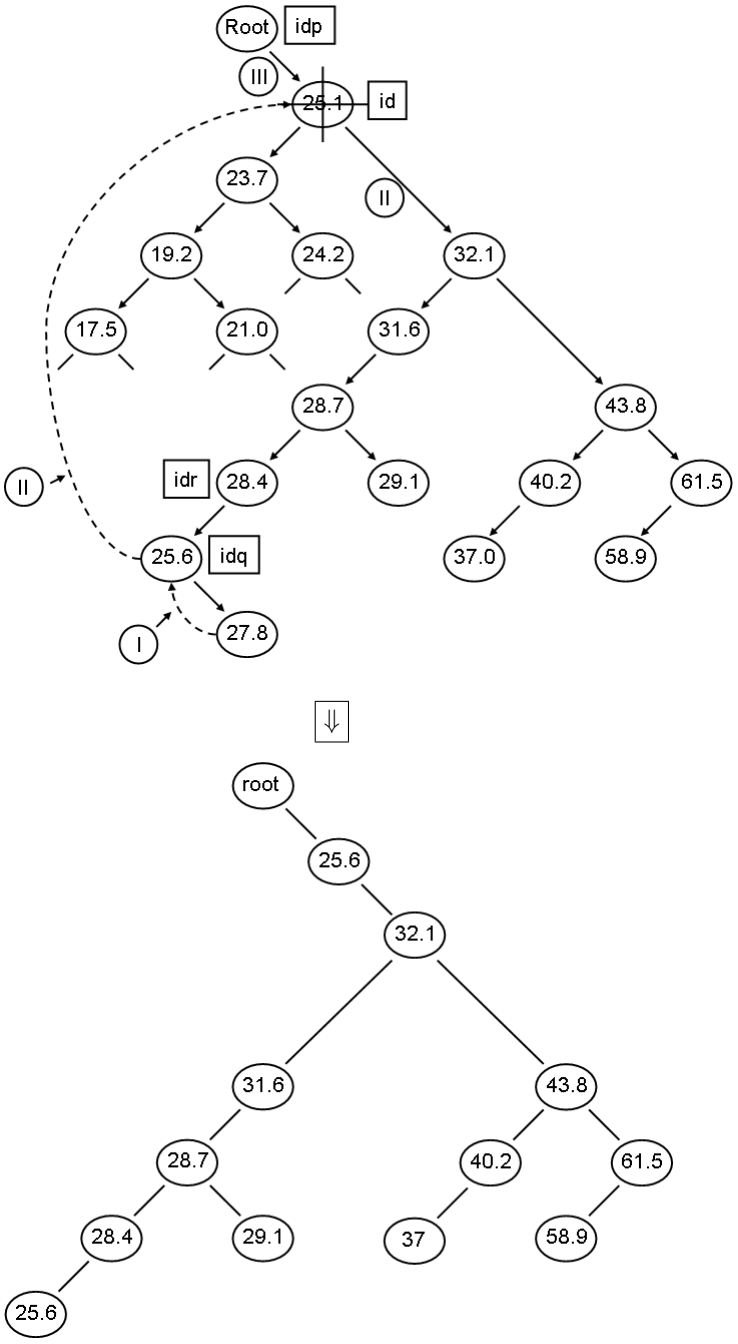
Die Ereignis-Liste wird mittels

`InitEventList() :`

initialisiert.

```
treeLeft[1]=0
treeRight[1]=0
treeIdA[1]=nAtom+2
(id=treeIdA[1];id≤podSize-1):treeCircAR[id]=id+1
(id=2,id≤nAtom+1):
     treeCircAL[id]=id
     treeCircAR[id]=id
     treeCircBL[id]=id
     treeCircBR[id]=id
```

Ad: `Delete − Event`
Bsp. für Entfernen eines Events `id` aus dem binären Ereignis-Baum




```
I)   Left[idv]=Right[idq]
     Up[Right[idq]]=idv
II)  Right[idq]=Right[id]
     Right[id]=idq
     Up[Left[id]]=idq
     Left[idq]=Left[id]
III) Up[idq]=idp
     ( Left  )
     ( Right ) [idp]=idq
```

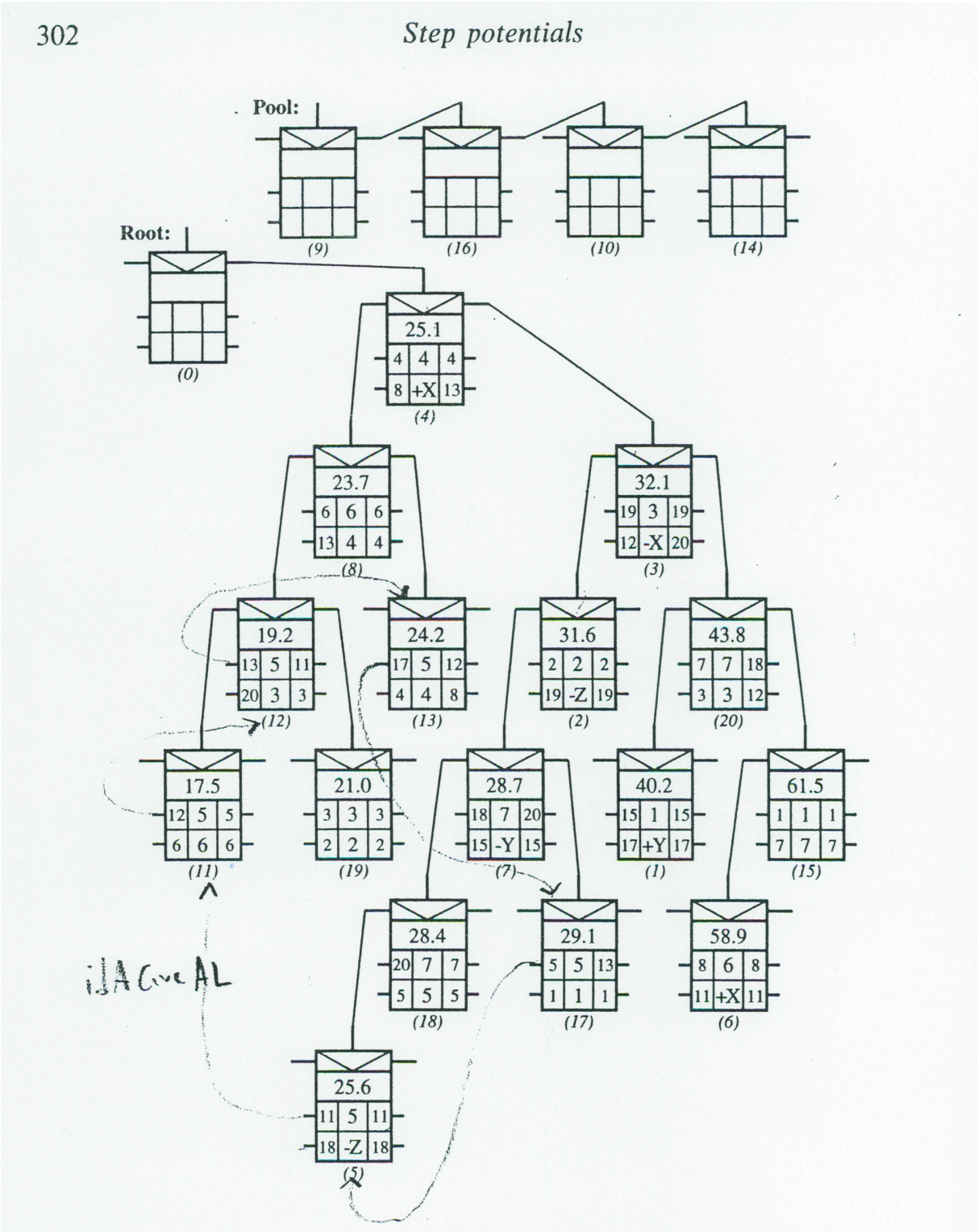


Fig. 12.1 A hand-crafted miniature event tree (the real one is much larger) – the tree links are shown, but for clarity the circular lists are omitted and the four pointer values are shown instead (on either side of the node). Each node includes the event time, the atom(s) involved, and the cell face crossed (if relevant); the value beneath the node is its 'address' in the tree.

Jan 24, 02 10:50 **hard–disk–event.c** Page 1/2

```
#define treeLeft    tree[1]
#define treeRight   tree[2]
#define treeUp      tree[3]
#define treeCircAL  tree[4]
#define treeCircBL  tree[5]
#define treeCircAR  tree[6]
#define treeCircBR  tree[7]
#define treeIdA     tree[8]
#define treeIdB     tree[9]

#define ATOM_LIMIT 1000000
#define NDIM       2

  void ScheduleEvent (int idA, int idB, real tEvent) {
    int id, idNew, more;
    id = 1;
    if (idB <= ATOM_LIMIT ||
       idB > ATOM_LIMIT + 2 * NDIM && idB <= ATOM_LIMIT + 100) {
      if (treeIdA[1] == 0) ErrExit ("empty event pool");
      idNew = treeIdA[1];
      treeIdA[1] = treeCircAR[treeIdA[1]];
    } else idNew = idA + 1;
    if (treeRight[id] == 0) treeRight[id] = idNew;
    else {
      more = 1;     id = treeRight[id];
      while (more) {
        if (tEvent <= treeTime[id]) {
          if (treeLeft[id] > 0) id = treeLeft[id];
          else {
            more = 0;     treeLeft[id] = idNew;
          }
        } else {
          if (treeRight[id] > 0) id = treeRight[id];
          else {
            more = 0;     treeRight[id] = idNew;
    } } } }
    if (idB <= ATOM_LIMIT) {
      treeCircAR[idNew] = treeCircAR[idA + 1];
      treeCircAL[idNew] = idA + 1;
      treeCircAL[treeCircAR[idA + 1]] = idNew;
      treeCircAR[idA + 1] = idNew;
      treeCircBR[idNew] = treeCircBR[idB + 1];
      treeCircBL[idNew] = idB + 1;
      treeCircBL[treeCircBR[idB + 1]] = idNew;
      treeCircBR[idB + 1] = idNew;
    }
    treeTime[idNew] = tEvent;
    treeIdA[idNew] = idA;
    treeIdB[idNew] = idB;
    treeLeft[idNew] = treeRight[idNew] = 0;
    treeUp[idNew] = id;
  }

  void NextEvent () {
    int id, idAx, idBx, idd, idNow, idtx;
    idNow = treeRight[1];
    while (treeLeft[idNow] > 0) idNow = treeLeft[idNow];
    timeNow = treeTime[idNow];
    evIdA = treeIdA[idNow];     evIdB = treeIdB[idNow];
    if (evIdB <= ATOM_LIMIT + 2 * NDIM) {
      if (evIdA < evIdB) {
        idAx = evIdA + 1;     idBx = evIdB + 1;
      } else {
        idAx = evIdB + 1;     idBx = evIdA + 1;
      }
      idtx = idBx - idAx;
      if (evIdB > ATOM_LIMIT) idBx = idAx;
      for (id = idAx; id <= idBx; id += idtx) {
        DeleteEvent (id);
```

Jan 24, 02 10:50 **hard-disk-event.c** Page 2/2

```
      for (idd = treeCircAL[id]; idd != id; idd = treeCircAL[idd]) {
        treeCircBR[treeCircBL[idd]] = treeCircBR[idd];
        treeCircBL[treeCircBR[idd]] = treeCircBL[idd];
        DeleteEvent (idd);
      }
      treeCircAR[treeCircAL[id]] = treeIdA[1];
      treeIdA[1] = treeCircAR[id];
      treeCircAL[id] = treeCircAR[id] = id;
      for (idd = treeCircBL[id]; idd != id; idd = treeCircBL[idd]) {
        treeCircAR[treeCircAL[idd]] = treeCircAR[idd];
        treeCircAL[treeCircAR[idd]] = treeCircAL[idd];
        DeleteEvent (idd);
        treeCircAR[idd] = treeIdA[1];    treeIdA[1] = idd;
      }
      treeCircBL[id] = treeCircBR[id] = id;
    }
  } else {
    DeleteEvent (idNow);
    if (evIdB <= ATOM_LIMIT + 100) {
      treeCircAR[idNow] = treeIdA[1];
      treeIdA[1] = idNow;
  } }
}

void DeleteEvent (int id) {
  int idp, idq, idr;
  idr = treeRight[id];
  if (idr == 0) idq = treeLeft[id];
  else {
    if (treeLeft[id] == 0) idq = idr;
    else {
      if (treeLeft[idr] == 0) idq = idr;
      else {
        idq = treeLeft[idr];
        while (treeLeft[idq] > 0) {
          idr = idq;    idq = treeLeft[idr];
        }
        treeLeft[idr] = treeRight[idq];
        treeUp[treeRight[idq]] = idr;
        treeRight[idq] = treeRight[id];
        treeUp[treeRight[id]] = idq;
      }
      treeUp[treeLeft[id]] = idq;
      treeLeft[idq] = treeLeft[id];
  } }
  idp = treeUp[id];    treeUp[idq] = idp;
  if (treeRight[idp] != id) treeLeft[idp] = idq;
  else treeRight[idp] = idq;
}

void InitEventList () {
  int id;
  treeLeft[1] = treeRight[1] = 0;
  treeIdA[1] = nAtom + 2;
  for (id = treeIdA[1]; id <= poolSize - 1; id ++) treeCircAR[id] = id + 1;
  treeCircAR[poolSize] = 0;
  for (id = 2; id <= nAtom + 1; id ++) {
    treeCircAL[id] = treeCircBL[id] = id;
    treeCircAR[id] = treeCircBR[id] = id;
  }
}
```

`ScheduleEvent(idA, idB, tEvent)`

Kollision:
Nehme leere Adresse aus dem Pool: `idNEw = treeIdA[1]`
Speichere nächste leere Adresse des Pools: `treeIdA[1] = AR[tree[IdA[1]]]`
nicht Kollision: `idNew = idA + 1`

↙ d. h. `treeLeft[id]` oder `treeRight[id] = 0`
Finde unbesetzten Anknüpfungspunkt in passendem Zweig
des binären Baumes: ↑
dann `tree` $\begin{pmatrix} \texttt{Left} \\ \texttt{Right} \end{pmatrix}$ `id = idNew` beachte Zeitordnung!

Falls Kollision:
Verknüpfe `idNew` mit den zirkulären Listen
`AL` & `AR` (die alle Ereignisse mit Beteiligung von Atom `idA` enthalten)
`BL` & `BR` (die alle Ereignisse mit Beteiligung von Atom `idA` enthalten)

```
treeTime[idNew]=tEvent
treeIdA[idNew]=idA
treeIdB[idNew]=idB
treeLeft[idNew]=0
treeRight[idNew]=0
treeUp[idNew]=id
```

Schedule-Event
z. B. Kollision von Atom `i` und Atom `j` zur Zeit `t` (+`tNow`)

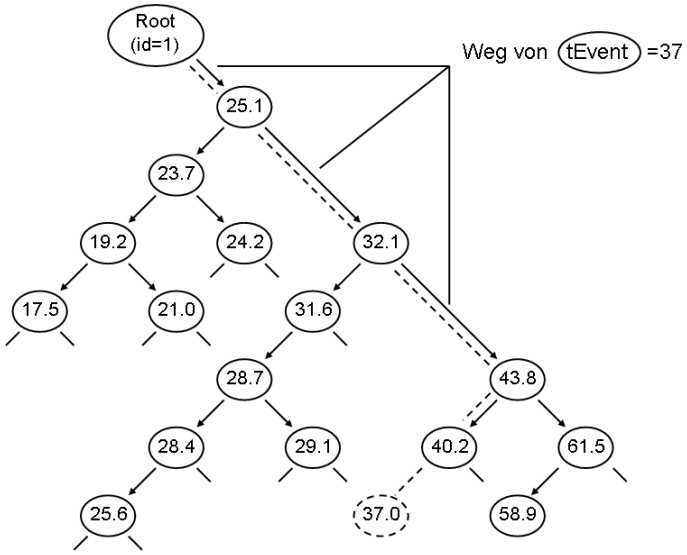
```
                    |         |       |
                  idA       idB   tEvent
```

```
id=1
----------
idNew=treeIdA[1]    ← erster Knoten im Pool
treeIdA[1]=treeCircAR[treeIdA[1]]    ← neuer erster Knoten im Pool
----------
if (treeRight[id]==0)
     tree right[id]=idNew
else
{
    more=1
    id=treeRight[id]
    while (more)
    {
        if (tEvent≤treeTime[id])
        {
            if (treeLeft[id]>0)
              id=treeLeft[id]
            else
            {
              more=0
              treeLeft[id]=idNew
            }
        }
        else
        {
            if (treeRight[id]>0)
              id=treeRight[id]
```

```
            else
            {
              more=0
              treeRight[id]=idNew
            }
          }
        }
    }
```

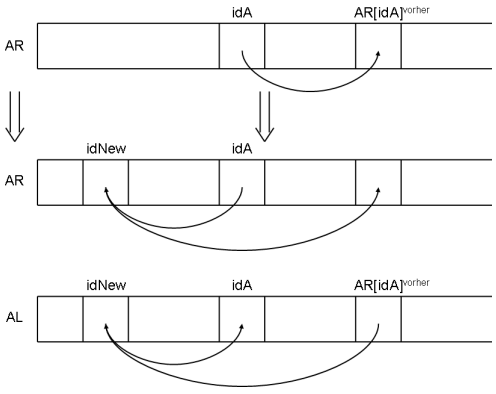
Ad: `ScheduleEvent()`

Bsp. für Einfügen eines Event sbei `tEvent` = 37 in den binären Ereignisbaum



Bem. `NextEvent` läuft einfach an der linken Flanke des Baumes bis zum Ende, dies ist die kleinste Zeit `treeTime`
Dieses Ereignis wird aus dem Baum entfernt und mit ihm alle, die mittels der zirkulären Listen mit ihm verknüpft sind (denn sie enthalten ein am Ereignis beteiligtes Atom!)

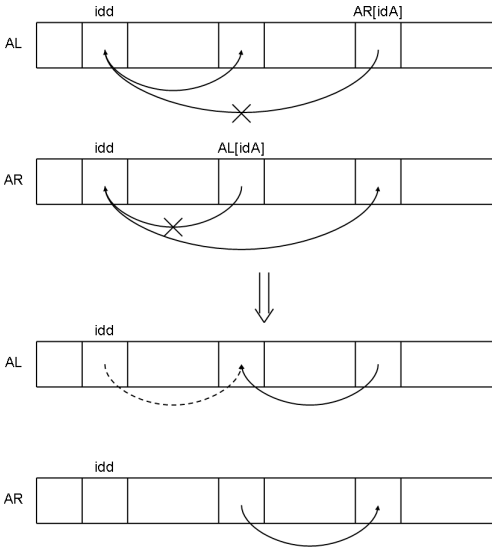
Schema für:
Einfügen eines Events `idNew` in $\begin{pmatrix} \texttt{doubly linked} \\ \texttt{circular List} \end{pmatrix}$



`AR[idNew]=AR[idA+1]`

```
AL[idNew]=idA+1
AL[AR[idA+1]]=idNew
AR[idA+1]=idNew
```

Löschen eines Events `idd` aus $\begin{pmatrix} \texttt{doubly linked} \\ \texttt{circular List} \end{pmatrix}$




```
AL[AR[idd]]=AL[idd]
AR[AL[idd]]=AR[idd]
```

Jan 08, 02 18:48 **cc** Page 1/8

```
// hd Model class

class HD_model {

    int scale = 2 ;  // pixel size
    int dimX,dimY ;  // display width size
    int LatticeWidth, LatticeHeight, LatticeSites ;
    double Temperature;

    double r[][], rv[][], atomTime[], treeTime[], region[],
        regionH[], collCount, crossCount, density,
        intervalSum, nextSumTime, pi, sKinEnergy, temperature,
        timeNow, vMag, vSum;
    int inCell[][], tree[][], cellList[], cellRange[],
        cells[], initUcell[], eventCount,
        eventCountLimit, eventMult, evIdA, evIdB, moreCycles,
        nAtom, poolSize, randSeed, runId;

    public HD_model(int lx,int ly,int dx,int dy,double temp,double rho) {

        region  = new double[3];
        regionH = new double[3];
        initUcell = new int[3];

        cells     = new int[3];
        cellRange = new int[5];

        runId        =  1;
        initUcell[1] =  lx;
        initUcell[2] =  ly;
        temperature  =  temp;
        density      =  rho;
        intervalSum  =  5.;
        randSeed     =  17;
        eventMult    =  4;
        eventCountLimit = 9999999;

        // System.out.print(lx);
        // System.out.print("   " + temp);
        // System.out.print("   " + density);
        // System.out.println(" ");

        dimX = dx;
        dimY = dy;

        SetParams ();
        SetupJob ();
        // moreCycles = 1;     eventCount = 0;
        // while ( moreCycles != 0 ) {
        //    SingleEvent ();
        //    eventCount = eventCount + 1;
        //    if (eventCount >= eventCountLimit) moreCycles = 0;
        // }
    }


    void SetupJob () {
        AllocArrays ();
        InitCoords ();
        InitVels ();
        timeNow = nextSumTime = 0.;
        collCount = crossCount = 0.;
        StartRun ();
        ScheduleEvent (0, 1000000  + 7, nextSumTime);
    }


    void SingleEvent () {
        double vvSum;
```

Jan 08, 02 18:48 **cc** Page 2/8

```
        int k, n;
        NextEvent ();
        // System.out.println("Event:  " + evIdB);
        if (evIdB <= 1000000 ) {
            ProcessCollision ();
            collCount = collCount + 1.;
            // new
            // System.out.println("Collision  " + collCount);
            UpdateSystem ();
        } else if (evIdB > 1000000  + 100) {
            ProcessCellCrossing ();
            crossCount = crossCount + 1.;
            // new
            // System.out.println("Crossing  " + crossCount);
            UpdateSystem ();
        } else if (evIdB == 1000000  + 8) {
            UpdateSystem ();
        } else {
            UpdateSystem ();
            nextSumTime = nextSumTime + intervalSum;
            ScheduleEvent (0, 1000000  + 7, nextSumTime);
            vSum = 0.;    vvSum = 0.;
            for (n = 1; n <= nAtom; n ++) {
                for (k = 1; k <= 2 ; k ++) {
                    vSum = vSum + rv[k][n];
                    vvSum = vvSum + rv[k][n]*rv[k][n];
                } }
            vSum = Math.abs (vSum) / nAtom;
            sKinEnergy = vvSum * 0.5 / nAtom;
            //PrintSummary (stdout);
        }
    }


    void SetParams () {
        int k;
        pi = 4. * Math.atan (1.);
        for (k = 1; k <= 2 ; k ++)
            region[k] = initUcell[k] / Math.sqrt (density);
        nAtom = initUcell[1] * initUcell[2];
        for (k = 1; k <= 2 ; k ++) regionH[k] = 0.5 * region[k];
        vMag = Math.sqrt (2  * (1. - 1. / nAtom) * temperature);
        poolSize = eventMult * nAtom;
        for (k = 1; k <= 2 ; k ++) cells[k] = (int) region[k];
    }


    void AllocArrays () {
        r  = new double [3][nAtom+1];
        rv = new double [3][nAtom+1];
        atomTime = new double [nAtom+1];
        cellList = new int [ nAtom + 1 + cells[1]*cells[2] ];

        inCell   = new int  [3][nAtom+1];
        tree     = new int [10][poolSize+1];
        treeTime = new double  [poolSize+1];
    }


    void ProcessCollision () {
        double dr[], fac, s1, s2;
        int k;
        dr = new double [3];

        UpdateAtom (evIdA);
        UpdateAtom (evIdB);
        for (k = 1; k <= 2 ; k ++) {
            cellRange[2*k-1] = -1;    cellRange[2*k] = 1;
        }
```

Jan 08, 02 18:48 **cc** Page 3/8

```
        for (k = 1; k <= 2 ; k ++) {
            dr[k] = r[k][evIdA] - r[k][evIdB];
            if (Math.abs (dr[k]) > regionH[k])
                dr[k] = dr[k] -
                ( (dr[k]>=0) ? (region[k]) : (-region[k]) ) ;
        }
        s1 = s2 = 0.;
        for (k = 1; k <= 2 ; k ++) {
            s1 = s1 + dr[k] * (rv[k][evIdA] - rv[k][evIdB]);
            s2 = s2 + (( dr[k] ) * ( dr[k] )) ;
        }
        fac = - s1 / s2;
        for (k = 1; k <= 2 ; k ++) {
            rv[k][evIdA] = rv[k][evIdA] + dr[k] * fac;
            rv[k][evIdB] = rv[k][evIdB] - dr[k] * fac;
        }
        PredictEvent (evIdA, 0);
        PredictEvent (evIdB, evIdA);
    }


    void ProcessCellCrossing () {
        int k, n;
        UpdateAtom (evIdA);
        n = (inCell[2][evIdA] - 1) * cells[1] + inCell[1][evIdA] + nAtom;
        while (cellList[n] != evIdA) n = cellList[n];
        cellList[n] = cellList[evIdA];
        for (k = 1; k <= 2 ; k ++) {
            cellRange[2 * k - 1] = -1;    cellRange[2 * k] = 1;
        }
        k = evIdB - 1000000  - 100;
        if (rv[k][evIdA] > 0.) {
            cellRange[2 * k - 1] = 1;
            inCell[k][evIdA] = inCell[k][evIdA] + 1;
            if (inCell[k][evIdA] == cells[k] + 1) {
                inCell[k][evIdA] = 1;
                r[k][evIdA] = - regionH[k];
            }
        } else {
            cellRange[2 * k] = -1;
            inCell[k][evIdA] = inCell[k][evIdA] - 1;
            if (inCell[k][evIdA] == 0) {
                inCell[k][evIdA] = cells[k];
                r[k][evIdA] = regionH[k];
            }
        }
        PredictEvent (evIdA, evIdB);
        n = (inCell[2][evIdA] - 1) * cells[1] + inCell[1][evIdA] + nAtom;
        cellList[evIdA] = cellList[n];    cellList[n] = evIdA;
    }



    void PredictEvent (int na, int nb) {
    double dr[], dv[], shift[], tm[], b, d, t, tInt, vv;
    int signDir[], evCode, iX, iY, jX, jY, k, n;

    dr    = new double [3];
    dv    = new double [3];
    shift = new double [3];
    tm    = new double [3];
    signDir = new int [3];


    for (k = 1; k <= 2 ; k ++) {
      if (rv[k][na] != 0.) {
        if (rv[k][na] > 0.) signDir[k] = 0;
        else signDir[k] = 1;
        tm[k] = ((inCell[k][na] - signDir[k]) * region[k] /
```

Jan 08, 02 18:48 **cc** Page 4/8

```
        cells[k] - r[k][na] - regionH[k]) / rv[k][na];
    } else tm[k] = 1e12;
  }
  if (tm[1] <= tm[2]) k = 1;
  else k = 2;
  evCode = 100 + k;
  ScheduleEvent (na, 1000000  + evCode, timeNow + tm[k]);
  for (iY = cellRange[3]; iY <= cellRange[4]; iY ++) {
    jY = inCell[2][na] + iY;     shift[2] = 0.;
    if (jY == 0) {
      jY = cells[2];     shift[2] = - region[2];
    } else if (jY > cells[2]) {
      jY = 1;     shift[2] = region[2];
    }
    for (iX = cellRange[1]; iX <= cellRange[2]; iX ++) {
      jX = inCell[1][na] + iX;     shift[1] = 0.;
      if (jX == 0) {
        jX = cells[1];     shift[1] = - region[1];
      } else if (jX > cells[1]) {
        jX = 1;     shift[1] = region[1];
      }
      n = (jY - 1) * cells[1] + jX + nAtom;
      for (n = cellList[n]; n > 0; n = cellList[n]) {
          if (n != na && n != nb && (nb >= 0 || n < na)) {
              tInt = timeNow - atomTime[n];
              for (k = 1; k <= 2 ; k ++) {
                  dr[k] = r[k][na] - (r[k][n] + rv[k][n] * tInt) - shift[k];
                  dv[k] = rv[k][na] - rv[k][n];
              }
              b = dr[1] * dv[1] + dr[2] * dv[2];
              if (b < 0.) {
                  vv = (dv[1]*dv[1] + dv[2]*dv[2]) ;
                  d = (b*b - vv * (dr[1]*dr[1] + dr[2]*dr[2] - 1.));
                  if (d >= 0.) {
                      t = - (Math.sqrt (d) + b) / vv;
                      ScheduleEvent (na, n, timeNow + t);
                  } } } } } }
}


void StartRun () {
    int j, k, n;
    for (j = 1; j <= cells[1] * cells[2] + nAtom; j ++) cellList[j] = 0;
    for (n = 1; n <= nAtom; n ++) {
        atomTime[n] = timeNow;
        for (k = 1; k <= 2 ; k ++)
            inCell[k][n] =
              (int) ((r[k][n]+regionH[k]) * cells[k]/region[k] + 1);
        j = (inCell[2][n] - 1) * cells[1] + inCell[1][n] + nAtom;
        cellList[n] = cellList[j];     cellList[j] = n;
    }
    InitEventList ();
    for (k = 1; k <= 2 ; k ++) {
        cellRange[2 * k - 1] = -1;     cellRange[2 * k] = 1;
    }
    for (n = 1; n <= nAtom; n ++) PredictEvent (n, -1);
}


void UpdateAtom (int id) {
    double tInt;
    int k;
    tInt = timeNow - atomTime[id];
    for (k = 1; k <= 2 ; k ++) {
        r[k][id] = r[k][id] + rv[k][id] * tInt;
    }
    atomTime[id] = timeNow;
}
```

Jan 08, 02 18:48 **cc** Page 5/8

```
void UpdateSystem () {
    int n;
    for (n = 1; n <= nAtom; n ++) UpdateAtom (n);
}


void ScheduleEvent (int idA, int idB, double tEvent) {
    int id, idNew, more;
    id = 1;
    if (idB <= 1000000  ||
        idB > 1000000  + 2 * 2  && idB <= 1000000  + 100) {
        if (tree[8] [1] == 0) System.out.print ("empty event pool");
        idNew = tree[8] [1];
        tree[8] [1] = tree[6] [tree[8] [1]];
    } else idNew = idA + 1;
    if (tree[2] [id] == 0) tree[2] [id] = idNew;
    else {
        more = 1;    id = tree[2] [id];
        while (more != 0) {
            if (tEvent <= treeTime[id]) {
                if (tree[1] [id] > 0) id = tree[1] [id];
                else {
                    more = 0;    tree[1] [id] = idNew;
                }
            } else {
                if (tree[2] [id] > 0) id = tree[2] [id];
                else {
                    more = 0;    tree[2] [id] = idNew;
                } } } }
    if (idB <= 1000000 ) {
        tree[6] [idNew] = tree[6] [idA + 1];
        tree[4] [idNew] = idA + 1;
        tree[4] [tree[6] [idA + 1]] = idNew;
        tree[6] [idA + 1] = idNew;
        tree[7] [idNew] = tree[7] [idB + 1];
        tree[5] [idNew] = idB + 1;
        tree[5] [tree[7] [idB + 1]] = idNew;
        tree[7] [idB + 1] = idNew;
    }
    treeTime[idNew] = tEvent;
    tree[8] [idNew] = idA;    tree[9] [idNew] = idB;
    tree[1] [idNew] = tree[2] [idNew] = 0;
    tree[3] [idNew] = id;
}


void NextEvent () {
    int id, idAx, idBx, idd, idNow, idtx;
    idNow = tree[2] [1];
    while (tree[1] [idNow] > 0) idNow = tree[1] [idNow];
    timeNow = treeTime[idNow];
    evIdA = tree[8] [idNow];    evIdB = tree[9] [idNow];
    if (evIdB <= 1000000  + 2 * 2 ) {
        if (evIdA < evIdB) {
            idAx = evIdA + 1;    idBx = evIdB + 1;
        } else {
            idAx = evIdB + 1;    idBx = evIdA + 1;
        }
        idtx = idBx - idAx;
        if (evIdB > 1000000 ) idBx = idAx;
        for (id = idAx; id <= idBx; id += idtx) {
            DeleteEvent (id);
            for (idd = tree[4] [id]; idd != id; idd = tree[4] [idd]) {
                tree[7] [tree[5] [idd]] = tree[7] [idd];
                tree[5] [tree[7] [idd]] = tree[5] [idd];
                DeleteEvent (idd);
            }
            tree[6] [tree[4] [id]] = tree[8] [1];
            tree[8] [1] = tree[6] [id];
```

```
            tree[4] [id] = tree[6] [id] = id;
            for (idd = tree[5] [id]; idd != id; idd = tree[5] [idd]) {
                tree[6] [tree[4] [idd]] = tree[6] [idd];
                tree[4] [tree[6] [idd]] = tree[4] [idd];
                DeleteEvent (idd);
                tree[6] [idd] = tree[8] [1];    tree[8] [1] = idd;
            }
            tree[5] [id] = tree[7] [id] = id;
        }
    } else {
        DeleteEvent (idNow);
        if (evIdB <= 1000000  + 100) {
            tree[6] [idNow] = tree[8] [1];
            tree[8] [1] = idNow;
        } }
}


void DeleteEvent (int id) {
    int idp, idq, idr;
    idr = tree[2] [id];
    if (idr == 0) idq = tree[1] [id];
    else {
        if (tree[1] [id] == 0) idq = idr;
        else {
            if (tree[1] [idr] == 0) idq = idr;
            else {
                idq = tree[1] [idr];
                while (tree[1] [idq] > 0) {
                    idr = idq;    idq = tree[1] [idr];
                }
                tree[1] [idr] = tree[2] [idq];
                tree[3] [tree[2] [idq]] = idr;
                tree[2] [idq] = tree[2] [id];
                tree[3] [tree[2] [id]] = idq;
            }
            tree[3] [tree[1] [id]] = idq;
            tree[1] [idq] = tree[1] [id];
        } }
    idp = tree[3] [id];    tree[3] [idq] = idp;
    if (tree[2] [idp] != id) tree[1] [idp] = idq;
    else tree[2] [idp] = idq;
}


void InitEventList () {
    int id;
    tree[1] [1] = tree[2] [1] = 0;
    tree[8] [1] = nAtom + 2;
    for (id = tree[8] [1]; id <= poolSize - 1; id ++)
        tree[6] [id] = id + 1;
    tree[6] [poolSize] = 0;
    for (id = 2; id <= nAtom + 1; id ++) {
        tree[4] [id] = tree[5] [id] = id;
        tree[6] [id] = tree[7] [id] = id;
    }
}



void InitCoords () {
    double c[], gap[], offset[];
    int k, n, nX, nY;
    c      = new double[3];
    gap    = new double[3];
    offset = new double[3];
    for (k = 1; k <= 2 ; k ++) {
        gap[k] = region[k] / initUcell[k];
        offset[k] = 0.;
```

Jan 08, 02 18:48 **cc** Page 7/8

```
        }
        n = 0;
        for (nY = 1; nY <= initUcell[2]; nY ++) {
            c[2] = (nY - 0.5) * gap[2] - regionH[2] + offset[2];
            for (nX = 1; nX <= initUcell[1]; nX ++) {
                c[1] = (nX - 0.5) * gap[1] - regionH[1] + offset[1];
                n = n + 1;
                for (k = 1; k <= 2 ; k ++) r[k][n] = c[k];
            } }
    }


    void InitVels () {
        double vSum[], ang;
        int k, n;
        vSum = new double[3];
        for (k = 1; k <= 2 ; k ++) vSum[k] = 0.;
        for (n = 1; n <= nAtom; n ++) {
            ang = 2. * pi * RandR();
            rv[1][n] = vMag * Math.cos (ang);
            rv[2][n] = vMag * Math.sin (ang);
            for (k = 1; k <= 2 ; k ++) vSum[k] = vSum[k] + rv[k][n];
        }

        for (k = 1; k <= 2 ; k ++) vSum[k] = vSum[k] / nAtom;
        for (n = 1; n <= nAtom; n ++) {
            for (k = 1; k <= 2 ; k ++) rv[k][n] = rv[k][n] - vSum[k];
        }
    }


    double RandR() {
        randSeed = (randSeed * 314159269  + 453806245 ) & 2147483647 ;
        return (randSeed * 0.4656612873e-9 );
    }


    void paint(Graphics g) {
        int x,y,size,size2;
        double v, v_max, v_sum;
        float  c_red, c_green, c_blue;
        Color color ;

        size = (int) (512./region[1]);
        size2 = size/2;
        v_max = 0.;
        v_sum = 0.;
        for(int i=1; i<=nAtom; i++) {
            v = Math.sqrt(rv[1][i]*rv[1][i]+rv[2][i]*rv[2][i]);
            if ( v>v_max) v_max = v;
            v_sum += (rv[1][i]*rv[1][i]+rv[2][i]*rv[2][i]);
        };
        v_sum = Math.sqrt(v_sum);

        for(int i=1; i<=nAtom; i++) {
            x = (int) (512*(1.+r[1][i]/regionH[1])/2.);
            y = (int) (512*(1.+r[2][i]/regionH[2])/2.);
            // if ( x>512) System.out.println ("x = " + x);
            // if ( y>512) System.out.println ("x = " + y);
            v = Math.sqrt(rv[1][i]*rv[1][i]+rv[2][i]*rv[2][i]);;
            c_red   = (float) (v/v_max);
            c_blue  = (float) (1.-c_red);
            c_green = (float) 0.;
            color = new Color(c_red,c_green,c_blue);
            g.setColor(color) ;
            g.fillOval(x-size2,y-size2,size,size) ;
        }
```

Jan 08, 02 18:48 **cc** Page 8/8

```
    }

    void clear(Graphics g) {
        g.setColor(Color.white) ;
        g.fillRect(0,0,dimX,dimY) ;
    }

}
```

7 Quanten Monte Carlo Simulationen

Für einen Einstieg in die Quanten-Monte-Carlo (QMC) Simulationen ist es sinnvoll sich zuerst noch einmal den Ablauf der MC Simulation für klassische Systeme in Erinnerung zu rufen. Hier ging es vorrangig um die Bestimmung thermodynamischer Erwartungswerte für Systeme, deren Hamilton-Operatoren $\mathcal{H}$ als bekannt vorausgesetzt wurden:

$$\langle O \rangle = \frac{\int Dx\, O(x)\ exp(-\beta\mathcal{H})}{\int Dx\, exp(-\beta\mathcal{H})}.$$

$\int Dx \ldots$ bezeichnet hierbei ein Integral über den ganzen Phasenraum. Die Berechnung eines solchen Erwartungswertes wurde auf dem Computer implementiert, indem man einen stochastischen Markov-Prozeß realisierte, der den Phasenraum gemäß der Wahrscheinlichkeitsverteilung $\frac{exp(-\beta\mathcal{H})}{Z}$ durchläuft und dabei die einzelnen Beiträge zu $< O >$ aufsummiert. Die Übergangswahrscheinlichkeiten des Markov-Prozesses mussten dabei einer bestimmten Bilanzgleichung gehorchen, bisher die sogenannte detaillierte Bilanz. Zum Beispiel ergeben sich für den einfachen Fall eines einzigen Ising-Spins im Magnetfeld

$$\mathcal{H}(S) = -hS,\ S = \pm 1$$

die übergangswahrscheinlichkeiten zu

$$w(S \to -S) = exp(\beta[\mathcal{H}(S) - \mathcal{H}(-S)]),$$
$$w(-S \to S) = exp(\beta[\mathcal{H}(-S) - \mathcal{H}(S)])$$

und damit für die Wahrscheinlichkeit eines Zustandes

$$\mathcal{P}(S) = \frac{exp(\beta hS)}{exp(\beta hS) + exp(\beta h - S)}.$$

Wir bemerken, dass ein Exponential von $\mathcal{H}$ für die Simulation des Markov-Prozesses entscheidend ist. Diese exponentielle Wahrscheinlichkeitsverteilung tritt de facto in vielen Problemen der statistischen Physik auf, lediglich die Gestalt von$\mathcal{H}$ wechselt von Fall zu Fall. Gerade an dieser Stelle kann nun die Analogie von der klassischen zur quantenmechanischen MC-Simulation gezogen werden, wenn man z.B. den statistischen Dichteoperator mit dem Zeitentwicklungsoperator der Quantenmechanik vergleicht:

$$\varrho(x) \propto e^{-\beta\mathcal{H}(x)},$$
$$\mathcal{U}(x) \propto e^{-\frac{it}{\hbar}\mathcal{H}(x)}.$$

Gelingt es den Zeitentwicklungsoperator eines Systems explizit zu bestimmten, so kann die Schrödinger-Gleichung (bei bekannten Anfangsbedingungen) vollständig gelöst werden. Aus obiger Gegenüberstellung ergibt sich unmittelbar eine formale Analogie zwischen der thermodynamischen inversen Temperatur β und der quantenmechanischen imaginären Zeit $\frac{it}{\hbar}$.
Für den Fall, dass $\mathcal{H}$ exakt diagonalisierbar ist, kann die Zeitentwicklung eines Zustandes $|\psi\rangle$ direkt angegeben werden:

$$e^{\lambda\mathcal{H}}|\psi\rangle = \sum_{i=1}^{M} e^{\lambda E_i}|\phi_i\rangle\langle\phi_i|\psi\rangle.$$

Hierbei bezeichnet $\{|\phi\rangle\}$ die Eigenbasis von $\mathcal{H}$.
Die Herausforderung numerischer Simulation liegt nun darin für den überwiegenden Teil der Fälle in denen $\mathcal{H}$ nicht explizit diagonalisiert werden kann, eine Möglichkeit zu liefern $e^{\lambda\mathcal{H}}$ dennoch zu berechnen. In vielen System ist zwar $\mathcal{H}$ selbst nicht analytisch diagonalisierbar, lässt sich aber als Summe zweier trivial diagonalisierbarer Teile darstellen (z.B. kinetische Energie $\frac{p^2}{2m}$ und potenzielle Energie $V(x)$))Die sog. Trotter-Suzuki-Methode macht sich auf dieser Annahme aufbauend folgende Identität zu Nutze:

$$e^{\lambda(A+B)} = \lim_{m\to\infty}\left(e^{\lambda\frac{A}{m}}e^{\lambda\frac{B}{m}}\right)^m,$$

deren Richtigkeit unmittelbar mit Hilfe einer Taylorentwicklung gezeigt werden kann:

$$e^{\lambda\frac{(A+B)}{m}} = 1 + \frac{\lambda}{m}(A+B) + \frac{1}{2}\frac{\lambda^2}{m^2}\left(A^2+AB+BA+B^2\right) + \mathcal{O}\left(\frac{\lambda^3}{m^3}\right),$$
$$e^{\lambda\frac{A}{m}}e^{\lambda\frac{B}{m}} = 1 + \frac{\lambda}{m}(A+B) + \frac{1}{2}\frac{\lambda^2}{m^2}\left(A^2+2AB+B^2\right) + \mathcal{O}\left(\frac{\lambda^3}{m^3}\right).$$

Beide Ausdrücke stimmen überein bis auf Terme der Ordnung $\mathcal{O}\left(\frac{\lambda^2}{m^2}|||[A,B]|||\right)$ so, dass für hinreichend große m gilt

$$e^{\lambda(A+B)} \approx \left(e^{\lambda\frac{A}{m}}e^{\lambda\frac{B}{m}}\right)^m$$

(genauer gilt: $||e^{\lambda(A+B)} - \left(e^{\lambda\frac{A}{m}}e^{\lambda\frac{B}{m}}\right)^m|| \leq \frac{\lambda^2}{2m}||\,[A,B]\,||e^{|\lambda|(||A||+||B||)}$, für kommutierende A und B ist somit obige Formel sogar exakt.)
Allgemeiner gilt für $\mathcal{H} = \sum_{i=1}^{p} A_i$:

$$||\exp\left[\lambda\frac{(A+B)}{m}\right] - \left(e^{\lambda\frac{A_1}{m}}\cdot\ldots\cdot e^{\lambda\frac{A_p}{m}}\right)^m|| \leq \frac{\lambda^2}{2m}\sum_{i\leq j<j\leq p}|||[A_iA_j]|||\exp\left[|\lambda|\sum_{i=1}^{p}||A_i||\right].$$

Auf diese Art und Weise kann jede additive Dekomposition des Hamilton-Operators als Kandidat für die Anwendung der Trotter-Suzuki-Methode verwendet werden.
Für die weiteren Betrachtungen definieren wir die m-te Approximation der Zustandssumme über

$$Z_m := Tr\left(e^{-\beta\frac{A}{m}}e^{-\beta\frac{B}{m}}\right)^m,$$

sodass gilt

$$Z = \lim_{m\to\infty} Z_m.$$

Mit $\Delta\tau := \frac{\beta}{m}$ kann die m-te Approximation durch die folgenden Schritte formal berechnet werden :

$$\begin{aligned} Z_m &= Tr\left(e^{-\Delta\tau A}e^{-\Delta\tau B}\cdot\ldots\cdot e^{-\Delta\tau A}e^{-\Delta\tau B}\right) \\ &= \sum_{\psi_1,\ldots,\psi_{2m}} \langle\psi_1|e^{-\Delta\tau A}|\psi_2\rangle\langle\psi_2|e^{-\Delta\tau B}|\psi_3\rangle\cdot\ldots\cdot\langle\psi_{2m-1}|e^{-\Delta\tau A}|\psi_{2m}\rangle\langle\psi_{2m}|e^{-\Delta\tau B}|\psi_1\rangle. \end{aligned}$$

Dabei ist $\{|\psi_i\rangle\}$ eine ONB des d-dimensionalen Hilbertraumes. Es ergibt sich also eine Summation über 2m verschiedene d-dimensionale Zustände. Wie in konkreten Beispielen deutlich wird, kann dies formal als eine Ausdehnung des Systems in eine zusätzliche Dimension aufgefasst werden, die sog. Trotter-Dimension in imaginärer Zeit.
Mit Hilfe dieser Zustandssumme können nun die Erwartungswerte von Observablen bestimmt werden. Zunächst gilt

$$\begin{aligned} \langle O\rangle &= Tr\left(O\mathcal{P}\right) \\ &= Z^{-1}\sum_{\psi}\langle\psi|Oe^{-\beta\mathcal{H}}|\psi\rangle. \end{aligned}$$

Ist O diagonal in $|\psi\rangle$-Darstellung ($\langle\psi|O|\psi'\rangle = O\left(\psi\right)\delta_{\psi\psi'}$), dann gilt:

$$\langle O\rangle = \sum_{\psi_1,\ldots\psi_{2m}} O\left(\psi_1\right)\mathcal{P}\left(\psi_1,\ldots,\psi_{2m}\right).$$

Mit der Gewichtungsfunktion

$$\mathcal{P}(\psi_1,\ldots,\psi_{2m}) = \frac{\langle\psi_1|e^{-\Delta\tau A}|\psi_2\rangle\langle\psi_2|e^{-\Delta\tau B}|\psi_3\rangle\cdot\ldots\cdot\langle\psi_{2m-1}|e^{-\Delta\tau A}|\psi_{2m}\rangle\langle\psi_{2m}|e^{-\Delta\tau B}|\psi_1\rangle}{\sum_{\psi_1,\ldots,\psi_{2m}}\langle\psi_1|e^{-\Delta\tau A}|\psi_2\rangle\langle\psi_2|e^{-\Delta\tau B}|\psi_3\rangle\cdot\ldots\cdot\langle\psi_{2m-1}|e^{-\Delta\tau A}|\psi_{2m}\rangle\langle\psi_{2m}|e^{-\Delta\tau B}|\psi_1\rangle}.$$

Diese Summe kann natürlich in konkreten Simulationen nicht über alle möglichen Konfigurationen ausgeführt werden. Stattdessen wählt man die Zustände über die summiert wird nach einer Wahrscheinlichkeitsverteilung gemäß der Gewichtungsfunktion $\mathcal{P}$ aus. Dies entspricht dem Verfahren bei klassischen MC-Simulationen, funktioniert jedoch nur, sofern $\mathcal{P}$ ausschließlich nichtnegative Werte annimmt.
Im Folgenden soll nun am Beispiel eines Ein-Spin-Systems die vorgehensweise verdeutlicht werden, insbesondere wird der Begriff der Trotter-Dimension formal klarer.
Für einen einfachen $\frac{1}{2}$-Spin im longitudinalen h-Feld und transversalen Γ-Feld ergibt sich der Hamilton-Operator in zwei Dimensionen zu

$$\mathcal{H} = \underbrace{-h\sigma^z}_{=:A}\underbrace{-\Gamma\sigma^x}_{=:B}$$

(wobei $\sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ und $\sigma^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ wie üblich die Pauli-Matrizen bezeichnen)
Insbesondere gilt

$$[A, B] \neq 0,$$

d.h. die Trotter-Suzuki-Näherung ist Nichttrivial. Zur Behandlung des zweidimensionalen Problems verwendet man z.B. die Eigenbasis des Operators A $\{|S\rangle\}, S = \pm 1$, damit gilt

$$\langle S|e^{-\Delta\tau A}|S'\rangle = \delta_{SS'}e^{\Delta\tau hS'},$$
$$\langle S|e^{-\Delta\tau B}|S'\rangle = \begin{cases} \cosh(\Delta\tau\Gamma) & S = S' \\ \sinh(\Delta\tau\Gamma) & S \neq S' \end{cases}.$$

Letzeres lässt sich nach einiger Rechnung geschickt mit Hilfe einer e-Funktion aufschreiben

$$\langle S|e^{-\Delta\tau B}|S'\rangle = \sqrt{\sinh(2\Delta\tau\Gamma)}e^{kSS'}, e^{-2k} = \tanh(\Delta\tau\Gamma).$$

Einsetzen dieser Ausdrücke in unsere Formel für Z_m führt auf

$$\begin{aligned} Z_m &= \sinh(2\Delta\tau)^{\frac{m}{2}} \cdot \sum_{S_1,\ldots,S_{2m}} e^{\Delta\tau hS_2}\delta_{S_1S_2}e^{kS_2S_3} \cdot e^{\Delta\tau hS_4}\delta_{S_3S_4}e^{kS_4S_5}\ldots e^{\Delta\tau hS_{2m}}\delta_{S_{2m-1}S_{2m}}e^{kS_{2m}S_1} \\ &= \sinh(2\Delta\tau)^{\frac{m}{2}} \cdot \sum_{S_1^i,\ldots S_m^i} \exp\left(\Delta\tau h\sum_{n=1}^{m} S_n + k\sum_{n=1}^{m} S_nS_{n+1}\right). \end{aligned}$$

Durch genaues Hinschauen erkennt man, dass der Ausdruck im Exponenten die gleiche Strukur aufweist wie der Hamilton-Operator einer eindimensionalen, gekoppelten Spin-Kette in äußerem Feld b bei Temperatur T_{kl}:

$$\mathcal{H} = -J\sum_{n=1}^{m} S_nS_{n+1} - b\sum_{n=1}^{m} S_n$$

mit

$$\frac{J}{T_{kl}} = k,$$
$$\frac{b}{T_{kl}} = \Delta\tau h.$$

Insgesamt kann in diesem Fall der Ausdruck für die m-te Approximation der Zustandssumme also durch die Zustandssumme eines eindimensionalen Systems mit m Spins ausgedrückt werden. Diese Analogie setzt sich auch in höheren Dimensionen fort, zu den physikalisch vorhandenen Dimensionen kommt durch Diskretisierung eine zusätzliche, imaginäre Trotter-Dimension hinzu.

8 Pfad-Integral-MC

Als nächstes wollen wir uns mit Systemen beschäftigen deren Hamilton-Operator in der Form

$$H = \underbrace{\frac{1}{2}\sum_{i=1}^{N}\frac{p_i^2}{m}}_{T=H_1} + \underbrace{V\left(x_1,\ldots,x_N\right)}_{U=H_2}.$$

aufgeschrieben werden kann. Auch hier vertauschen die beiden Operatoren H_1und H_2nicht wegen

$$[p_i, x_j] = \frac{\hbar}{i}\delta_{ij}.$$

Sei ab sofort $\hbar = 1 = m$. Zur Berechnung der Zustandssumme ziehen wir erneut die Trotter-Suzuki-Formel heran, dieses Mal verwenden wir als Basis abwechselnd die Eigenzustände von $\mathbf{p} = (p_1,\ldots,p_N)$ und $\mathbf{x} = (x_1,\ldots,x_N)$

$$Z_m = \sum_{\substack{\mathbf{p}_1,\ldots,\mathbf{p}_m \\ \mathbf{x}_1,\ldots,\mathbf{x}_m}} \langle\mathbf{p}_1|e^{-\Delta\tau H_1}|\mathbf{x}_1\rangle\langle\mathbf{x}_1|e^{-\Delta\tau H_2}|\mathbf{p}_2\rangle\ldots\langle\mathbf{p}_m|e^{-\Delta\tau H_1}|\mathbf{x}_m\rangle\langle\mathbf{x}_m|e^{-\Delta\tau H_2}|\mathbf{p}_1\rangle.$$

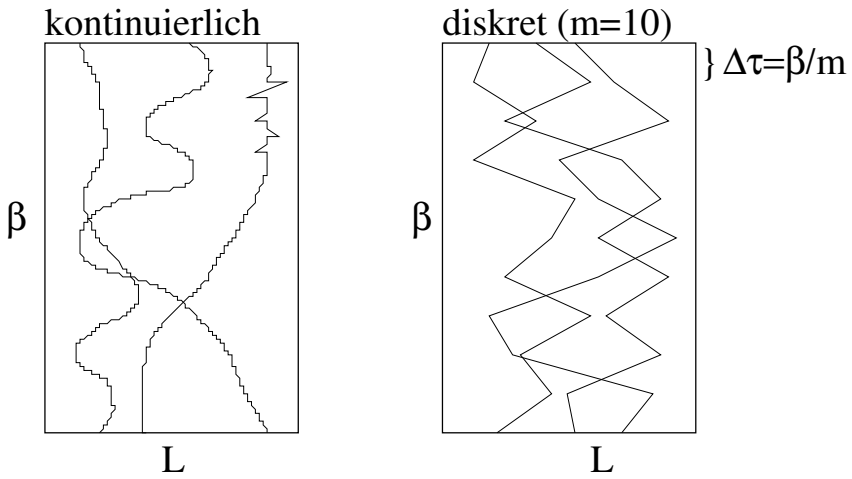
Weiterhin folgt für die Eigenzustände:

$$\begin{aligned}\langle\mathbf{p}_\tau|e^{-\Delta\tau H_1}|\mathbf{x}_\tau\rangle &= e^{-\Delta\tau\frac{p_\tau^2}{2}}\underbrace{\langle\mathbf{p}_\tau|\mathbf{x}_\tau\rangle}_{e^{i(\mathbf{p}_\tau\cdot\mathbf{x}_\tau)}},\\ \langle\mathbf{x}_\tau|e^{-\Delta\tau V}|\mathbf{p}_{\tau+1}\rangle &= e^{-\Delta V(\mathbf{x}_\tau)}\underbrace{\langle\mathbf{x}_\tau|\mathbf{p}_{\tau+1}\rangle}_{e^{-i(\mathbf{p}_\tau\cdot\mathbf{x}_{\tau+1})}}.\end{aligned}$$

Für kontinuierliche Eigenwerte wird die Summe zu einem Integral über den Phasenraum, einsetzen obiger Formeln führt dann auf:

$$\begin{aligned}Z_m &= \int\prod_{\tau=1}^{m}d\mathbf{x}_\tau\int\prod_{\tau=1}^{m}d\mathbf{p}_\tau\exp\left(-\sum_\tau\Delta\tau\mathbf{p}_\tau^2 + i\mathbf{p}_\tau\cdot(\mathbf{x}_{\tau+1}-\mathbf{x}_\tau) - \Delta\tau V\left(\mathbf{x}_\tau\right)\right)\\ &= \int\prod_{\tau=1}^{m}d\mathbf{x}_\tau\exp\left(-\sum_{\tau=1}^{m}\left\{\frac{1}{2}\left(\frac{\mathbf{x}_{\tau+1}-\mathbf{x}_\tau}{\Delta\tau}\right)^2 + V\left(\mathbf{x}_\tau\right)\right\}\Delta\tau\right) \qquad (*)\\ &\underset{\Delta\tau\to0}{=} \int\mathcal{D}x(\tau)\exp\left(-\int_0^\beta d\tau\frac{1}{2}\left(\frac{\partial\mathbf{x}(\tau)}{\partial\tau}\right)^2 + V\left(\mathbf{x}\left(\tau\right)\right)\right).\end{aligned}$$

Hierbei bezeichnet $\int\mathcal{D}x\ldots$ ein Pfadintegral über alle möglichen Konfigurationen in imaginärer Zeit. Rein formal kann daher ein Zusammenhang zum Feynmanschen Pfad-Integral in realer Zeit hergestellt werden $\left(\tau\leftrightarrow\frac{\hbar}{i}t\right)$. In einer praktisch ausführbaren QMC-Simulation wird natürlich die diskrete Form $(*)$ verwendet . Die Folgende Illustration zeigt die Pfadintegrale entlang der Trotter-Dimension für den Fall von 3 Bosonen mit Coulomb-WW in einem Kasten der Länge L bei Temperatur β^{-1}



In einer konkreten Simulation werden die verschiedenen Beiträge entsprechend ihrem Gewicht zur Zustandssumme aufaddiert. Hierbei geht man von einer Startkonfiguration der Weltlinien $\{x_i(\tau)\}_{i=1\ldots N}$ aus und führt nacheinander verschiedene MC-Moves aus, die Weltlinien werden in diskreten Schritten verschoben. Werden diese Moves akzeptiert mit der Wahrscheinlichkeit $e^{-\Delta E}$, wobei ΔE die Energiedifferenz der beiden Zustände bezeichnet, so ergibt sich über hinreichend viele MC-Samples der Wert der Zustandssumme.

9 Das Bose-Hubbard-Modell

In diesem Kapitel soll ein spezielles System von Bosonen/Fermionen untersucht werden, um ein Beispiel für einen QMC-Algorithmus für diesen Fall zu liefern. Konkret betrachten wir den Hamiltonian

$$H = -t \sum_{i=1}^{N} \left(c_i^\dagger c_{i+1} + c_{i+1}^\dagger c_i \right) + V \sum_i n_i \left(n_i - 1 \right).$$

$c_i / c_i^\dagger$ bezeichnen hierbei den Erzeuger/Vernichter-Operator am Orte $i \in \{1...N\}$, n_i die zugehörigen Besetzungszahloperatoren. Der erste Term beschreibt somit das „Hüpfen" einzelner Fermionen von Gitterplatz zu Gitterplatz, wohingegen der zweite Term je nach Vorzeichen von V Mehrfachbesetzungen eines Gitterplatzes „belohnt" oder „bestraft". Das Verhältnis der Parameter t („Hüpf-Matrix-Element") und V ist letzlich ausschlaggebend für das beobachtete Verhalten. Je nach betrachteter Teilchenart sind die Werte für die möglichen Besetzungszahlen eingeschränkt, für Bosonen gilt wie üblich $n_i \in \{0, 1, 2...\}$, für spinlose Fermionen $n_i \in \{0, 1\}$ (in diesem Fall besitzt der Wechselwirkungsterm keine Bedeutung). Der einfachheit Halber betrachten wir im folgenden spinlose Fermionen und setzen direkt V=0. An der Struktur des Hamilton-Operators kann man unmittelbar erkennen, dass dieser in der Form

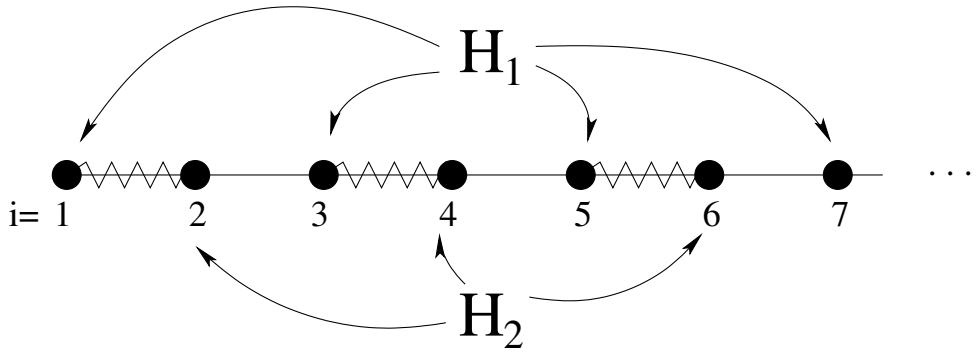
$$H = \sum_{i=1}^{N} H_{i,i+1}$$

aufgeschrieben werden kann. Wir definieren daher geschickt

$$H_1 = \sum_{i\, ungerade}^{N} H_{i,i+1},$$

$$H_2 = \sum_{i\, gerade}^{N} H_{i,i+1}$$

und stellen fest, dass zwar H_1 und H_2untereinander nicht kommutieren, aber jeweils aus einer Summe miteinander kommutierender Terme bestehen.



Wegen $[c_i, c_j]_{i \neq j} = 0$ vertauschen alle geraden Terme sowie alle ungeraden Terme untereinander. Diese Tatsache wiederum ermöglicht uns die zugehörigen Entwicklungsoperatoren exkat zu faktorisieren:

$$U_1 = e^{-\Delta\tau H_1} = \prod_{i\, ungerade} e^{-\Delta\tau H_{i,i+1}},$$

$$U_2 = e^{-\Delta\tau H_2} = \prod_{i\, gerade} e^{-\Delta\tau H_{i,i+1}}.$$

Erneut verwenden wir die Trotter-Suzuki-Formel, wobei wir dieses mal als VONB in die Besetzungszahldarstellung $|\mathbf{n}\rangle = |n_1 n_2 ... n_N\rangle, n_i \in \{0, 1\}$ übergehen. Als Ausdruck für Z_m erhält man damit:

$$Z_m = \sum_{\{\mathbf{n}_i\}_{i=1}^{2m}} \langle \mathbf{n}_1 | e^{-\Delta\tau H_1} | \mathbf{n}_2 \rangle \langle \mathbf{n}_2 | e^{-\Delta\tau H_2} | \mathbf{n}_3 \rangle \cdot ... \cdot \langle \mathbf{n}_{2m-1} | e^{-\Delta\tau H_1} | \mathbf{n}_{2m} \rangle \langle \mathbf{n}_{2m} | e^{-\Delta\tau H_2} | \mathbf{n}_1 \rangle. \qquad (\#)$$

Für die weiteren Berechnungen schreiben wir die Vektoren unserer Basis geeignet um

$$|\mathbf{n}\rangle = |n_1 n_2\rangle \bigotimes |n_3 n_4\rangle \bigotimes ... \bigotimes |n_{N-1} n_N\rangle,$$
$$|\mathbf{n}\rangle = |n_2 n_3\rangle \bigotimes |n_4 n_5\rangle \bigotimes ... \bigotimes |n_N n_1\rangle$$

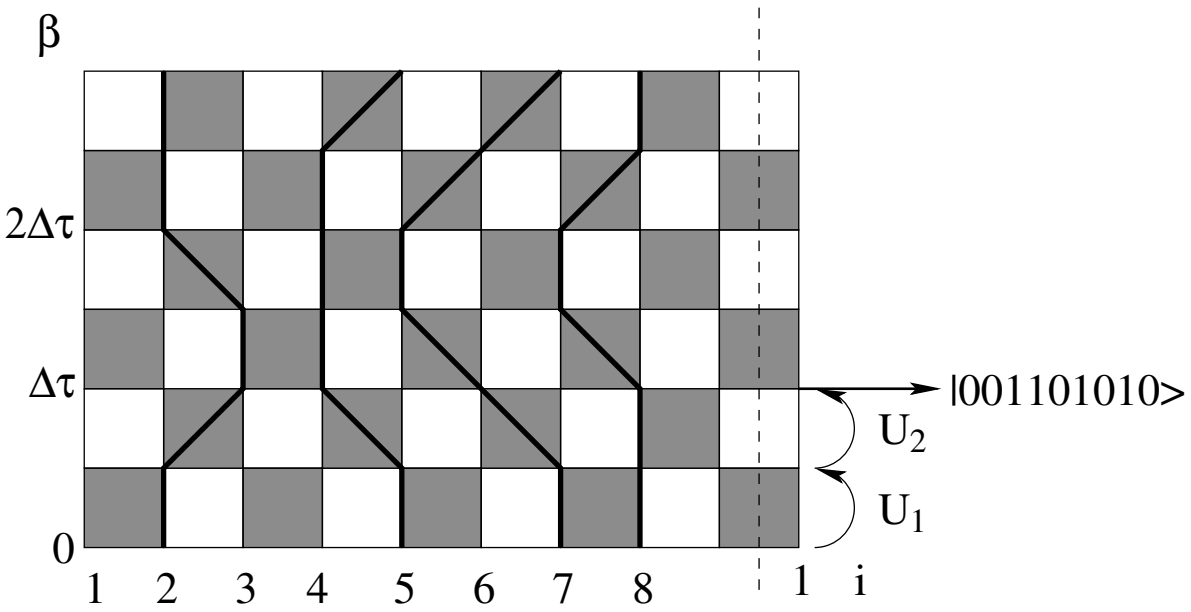
und können damit die Wirkung von U_1 auf unseren Zustand schreiben als

$$e^{-\Delta\tau H_1}|\mathbf{n}\rangle = e^{-\Delta\tau H_{1,2}}|n_1 n_2\rangle \bigotimes e^{-\Delta\tau H_{3,4}}|n_3 n_4\rangle \bigotimes ... \bigotimes e^{-\Delta\tau H_{N-1,N}}|n_{N-1} n_N\rangle.$$

Analog für U_2. Somit wurde die Berechnung der in $Z_m$auftretenden Matrixelemente auf ein effektives Problem mit nur zwei Gitterplätzen zurückgeführt. Im einzelnen gilt für die Matrixelemente

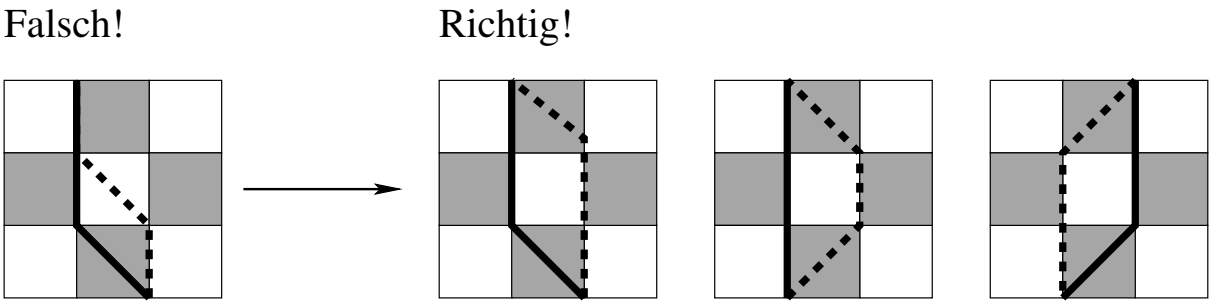
$$e^{-\Delta\tau H_{i,i+1}}|00\rangle = |00\rangle,$$
$$e^{-\Delta\tau H_{i,i+1}}|11\rangle = |11\rangle,$$
$$e^{-\Delta\tau H_{i,i+1}}|10\rangle = \{\cosh(\Delta\tau t)\,|10\rangle + \sinh(\Delta\tau t)\,|01\rangle\},$$
$$e^{-\Delta\tau H_{i,i+1}}|01\rangle = \{\cosh(\Delta\tau t)\,|01\rangle + \sinh(\Delta\tau t)\,|10\rangle\}.$$

Somit bewegt sich ein einzelnes Fermion (in imaginärer Zeit) mit dem Matrixelement $\cosh(\Delta\tau t)$ vorwärts und hüpft nach rechts oder links mit dem Matrix-Element $\sinh(\Delta\tau t)$.
In den Trotter-Zeitschritten werden die Operatoren U_1und U_2 abwechselnd auf die Zustände angewandt. Da U_1 nur die ungeraden, U_2 nur die geraden Terme enthält, ist bei deren Anwendung nicht jeder Zustand erreichbar, es gibt „verbotene" und „erlaubte" Bereiche nach jedem Schritt. Die folgende Abbildung veranschaulicht eine Entwicklung des Systems in Imaginärzeit für vier Fermionen.

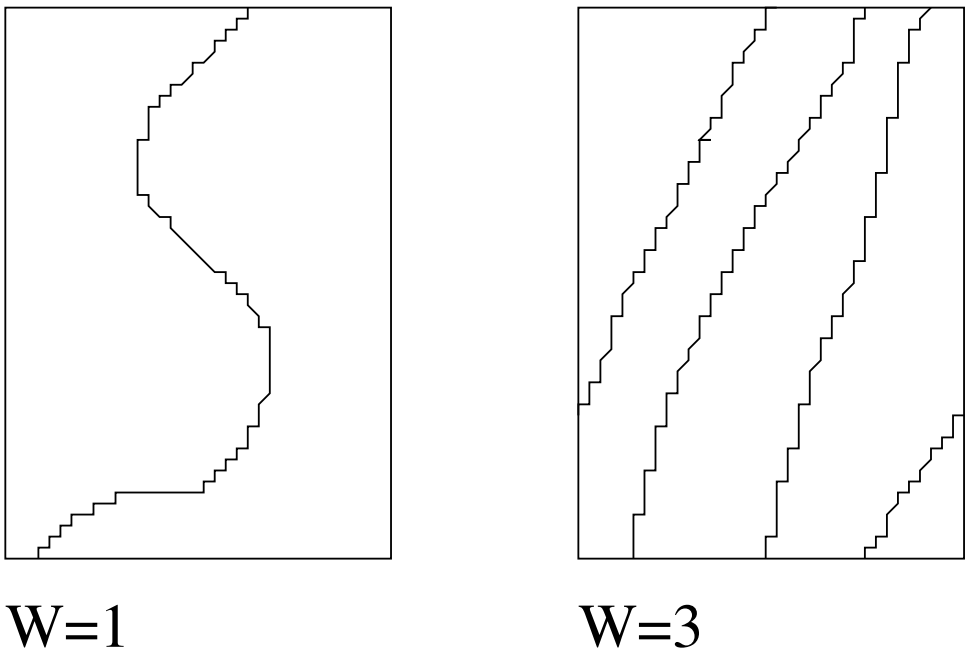


Man spricht in diesem Zusammenhang oft vom Schachbrett (engl.: „Checkerboard") Schema, die unterschiedlich gefärbten Plaketten sollen die erlaubten/verbotenen Bereiche farblich kennzeichnen (schattiert = erlaubt, nicht schattiert = verboten). Die einzelnen Beiträge zur Zustandsumme können nun durch Summation über viele mögliche Weltlinien-Konfigurationen gefunden werden. Im Ablauf der Simulation ist darauf zu achten, dass nur solche Weltlinien beachtet werden, die sich auf dem erlaubten Bereich des Checkerboards befinden sowie die gleiche Anfangs- und Endkonfiguration aufweisen.
Diese zusätzliche Einschränkung ist der Periodizität des Systems in der imaginären Zeitdimension geschuldet, in Formel (#) tritt die Konfiguration $|\mathbf{n}_1>$ sowohl beim ersten, als auch beim letzten Zeitschritt auf. Wir bemerken zusätzlich, dass die Fermionen-Zahl nicht nur insgesamt über einen kompletten Weltlinien-Zyklus sondern auch für jeden imaginären Zeitschritt erhalten bleibt.
Die Summe über die möglichen Weltlinien-Konfigurationen wird erneut durch das aus den MC-Simulationen bekannte „Importance-Sampling" implementiert, d.h. ausgehend von einer gegebenen Startkonfiguratioen,

werden verschiedene Moves durchgeführt und mit einer Wahrscheinlichkeit gemäß detaillierter Bilanz akzeptiert. Dies ist in diesem Fall ohne weiteres möglich, da die zugehörigen Matrixelemente, die zum Sampeln der Übergangswahrscheinlichkeit benötigt werden, alle positiv sind. Die möglichen lokalen MC-Moves dürfen dabei die erlaubten Plaketten des Checkerboards nicht verlassen, wie in folgender Abbildung deutlich wird.



Diese lokalen Moves sind bereits zum Sampeln einiger physikalischer Observablen ausreichend, in manchen Fällen ist jedoch die sogenannte Windungszahl W von Bedeutung. Man beachte, dass aufgrund der räumlichen und zeitlichen Periodizität das System dieses die Topologie eines Torus aufweist. Geht man nun von einem besetzten Gitterplatz aus und folgt der Weltlinie bis der Ausgangsplatz wieder erreicht ist, so ist die Anzahl der Durchgänge durch $\tau = \beta$ gerade $W + 1$.



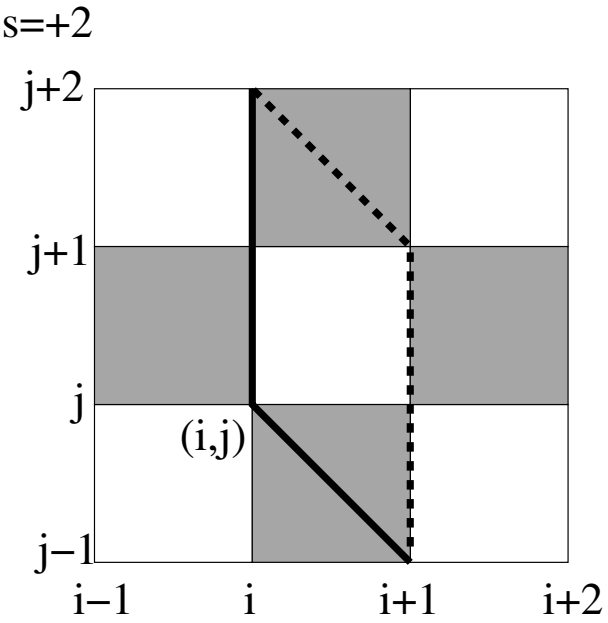
Allein aus der Anschauung wird bereits deutlich, dass Weltlinien mit Windungszahlen deutlich größer als 1 kaum mit lokalen Moves kaum zu verwirklichen sind, da deren Wahrscheinlichkeit zu gering ist. Interessiert man sich für Observablen, für die W von Bedeutung ist, ist es daher angebracht globale Moves einzuführen, die gleichzeitig mehrere Plaketten verschieben können.
Im Folgenden soll auf die konkreten Details des QMC-Algorithmus eingegangen werden. Unsere Simulation erfolgt auf einem zweidimensionalen Quadratgitter (i, j) mit $i = 1, ..., m$ (Raum) und $j = 1, ..., 2m$ (imaginäre Zeit). Die Besetzungszahl am jeweiligen Gitterplatz werde mit $n(i, j) \in \{0, 1\}$ bezeichnet. Bei jedem MC-Update wird zuerst überprüft, ob ein MC-Move einer Weltlinie über ein nichtschattiertes Quadrat möglich ist. Sei hierzu (i, j) die untere linke Ecke eines solchen Quadrates, wir berechnen die Größe

$$s = n(i, j) + n(i, j + 1) - n(i + 1, j) - n(i + 1, j + 1),$$

aus welcher wir ablesen können, welche Moves möglich sind:

$s = +2$	Move von links nach rechts möglich
$s = -2$	Move von rechts nach links möglich
$s \neq \pm 2$	kein Move möglich

Nun muss die Akzeptanzwahrscheinlichkeit berechnet werden, welche vom Verhältnis R des Produkts der Matrix-Elemente nach und vor dem Move abhängt. R hängt von $n(i+1, j-1)$ und $n(i+1, j+2)$ ab, denn diese bestimmen ob die Weltlinie beim Move vertikal oder Diagonal über die Plakette gezogen wird. Weiterhin hängt R von $n(i-1, j)$ und $n(i, j+2)$ ab, diese legen fest, ob sich rechts oder links eine andere, wechselwirkende Weltlinie befindet.



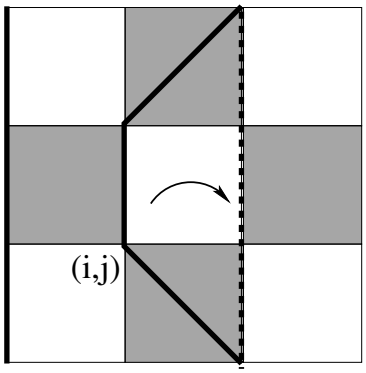
Insgesamt ergibt sich die Akzeptanz des Moves dann zu $P = \frac{R}{1+R}$.
Mit den oben angegebenen Matrixelementen lässt sich R berechnen zu

$$R = \tanh(\Delta\tau t)^{su} \cosh(\Delta\tau t)^{sv}$$

mit

$$u = 1 - n(i+1, j-1) - n(i+1, j+2),$$
$$v = n(i-1, j) - n(i+2, j).$$

Zum Beispiel gilt für folgendene Konfiguration



$s = +2$
$u = -1 \qquad \Rightarrow R = \sinh(\Delta\tau t)^{-2} \cosh(\Delta\tau t)^{4}$
$v = +1$

Exaktes Nachrechnen liefert

$$\begin{aligned} R &= \frac{\langle \star 01 \star | e^{-\Delta\tau H_2} | 1010 \rangle \langle 1010 | e^{-\Delta\tau H_1} | 1010 \rangle \langle 1010 | e^{-\Delta\tau H_2} | \star 01 \star \rangle}{\langle \star 01 \star | e^{-\Delta\tau H_2} | 1100 \rangle \langle 1100 | e^{-\Delta\tau H_1} | 1100 \rangle \langle 1100 | e^{-\Delta\tau H_2} | \star 01 \star \rangle} \\ &= \frac{\cosh(\Delta\tau t) e^{\frac{\Delta\tau}{4}} \cosh(\Delta\tau t)^2 e^{\frac{\Delta\tau}{2}} \cosh(\Delta\tau t) e^{\frac{\Delta\tau}{4}}}{\sinh(\Delta\tau t) e^{\frac{\Delta\tau}{4}} e^{-\frac{\Delta\tau}{2}} \sinh(\Delta\tau t) e^{\frac{\Delta\tau}{4}}} \\ &= s.o. \qquad \checkmark \end{aligned}$$

Von eigentlichem Interesse für den Simulator sind die Werte verschiedener physikalischer Observablen während der Evolution des Systems. Observablen welche in Besetzungszahl-Darstellung diagonal sind, können wie gehabt bestimmt werden. Beispiele hierfür sind der Ordnungsparameter

$$Q := \sum_{i=1} (-1)^i n_i$$

oder die Dichte-Dichte-Korrelationsfunktion

$$\begin{aligned} G(i-j,\tau) &= \langle n_i(\tau)\, n_j(0) \rangle, \\ n_i(\tau) &= e^{\tau H} n_i e^{-\tau H}. \end{aligned}$$

Für Q gilt zum Beispiel

$$\langle Q \rangle = \lim_{M\to\infty} \frac{1}{2LM} \sum_{k=1}^{M} \sum_{j=1}^{2m} \sum_{i=1}^{L} (-1)^i n_k(i,j).$$

Auch von Bedeutung sind Observablen wie

$$\begin{aligned} E &= -\frac{\partial}{\partial\beta} \ln(Z) \qquad \text{(Energie)}, \\ C &= \frac{\partial E}{\partial T} \qquad \text{(Wï¿œrmekapazitï¿œt)}, \\ \mathcal{C}(\tau) &= \langle j(\tau)\, j(0) \rangle = \sum_i c_i^\dagger c_{i+1} - c_{i+1}^\dagger c_i \quad \text{(Strom/Strom-K.fkt.)}. \end{aligned}$$

Abschließend soll noch bemerkt werden, dass die hier geschilderte Implementierung des Bose-Hubbard-Modells nur in einer Dimension ohne weiteres funktioniert. Für höhere Dimensionen (ab $d \geq 2$) ensteht das Problem, dass Produkte der Matrix-Elemente für bestimmte Fermionen-Konfigurationen negativ werden können. Für diese Konfigurationen ist die Interpretation der Gewichte in der Zustandssumme als Wahrscheinlichkeiten hinfällig, andere Algorithmen (z.B. Mean-Field-Methoden) müssen herangezogen werden.

10 Verfahren zur „exakten Diagonalisierung" (Lanczos etc.) in der QM

Anharmonischer Oszillator - Matrix-Darstellung

$\hat{H} = \hat{H}_0 + \xi \hat{x}^4$

$\hat{H}_0 = \frac{1}{2}\left(\hat{p}^2 + \hat{x}^2\right)\ ,\ \ \hat{p} = -\imath\hbar\frac{\partial}{\partial x}$

Eigenzustände $|n\rangle$ $(n \geq 0)$ mit $\hat{H}_0|n\rangle = \varepsilon_n|n\rangle$,
Energie-Eigenwerte $\varepsilon_n = n + \frac{1}{2}$

$$\langle x|n\rangle = \phi_n(x) = \frac{1}{\sqrt{2^n n!\sqrt{\pi}}} e^{-x^2/2} H_n(x)$$
$$\Big| \text{ Hermite-Polynome}$$

$\hat{x} = \frac{1}{\sqrt{2}}\left(\hat{a}^\dagger + \hat{a}\right)\ ,\ \left[\hat{a}, \hat{a}^\dagger\right] = \hat{\mathbb{1}}$

$$\langle n+1|\hat{a}^\dagger|n\rangle = \langle n|\hat{a}|n+1\rangle = \sqrt{n+1}$$
$$i.\,e.\ \hat{a}|n\rangle = \sqrt{n}|n-1\rangle$$
$$\hat{a}^\dagger|n\rangle = \sqrt{n+1}|n+1\rangle$$

Berechne hiermit $\langle n|\hat{H}|m\rangle = \varepsilon_n\delta_{nm} + \xi\langle n|\hat{x}^4|m\rangle$
$\langle n|\hat{x}^4|m\rangle = \frac{1}{4}\langle n|(\hat{a}^\dagger + \hat{a})^4|m\rangle \neq 0$ nur für $|n-m| \leq 4$

z. B. $\langle n+4|(\hat{a}^\dagger + \hat{a})^4|n\rangle \overset{|}{=} \langle n+4|\hat{a}^{\dagger 4}|n\rangle$

$= \langle n|\left(\hat{a}^\dagger + \hat{a}\right)^4|n+4\rangle \overset{|}{=} \sqrt{(n+1)(n+2)(n+3)(n+4)}$

$\vdots$

Einführung der Basis $|n\rangle$ ergibt also ein Eigenwertproblem aus der linearen Algebra. Für numerische Rechnungen muss man die unendlich-dimensionale Matrix abschneiden (d. h. nur Zustände $n < N$ betrachten $\rightarrow$ ergibt $N \times N$-Matrix).
Für $N \leq 10^3 - 10^4$ kann man die üblichen LA-Routinen (Icepack, Lapack, numerical recipes) verwenden, für größere $N \rightarrow$ Lanczos-Alg.

Exakte Diagonalisierung, Lanczos-Verfahren

Quantenmechanische Systeme werden durch einen (Modell-)Hamiltonian $\hat{H}$ sowie die Schrödinger-Gleichung

$$\frac{\partial}{\partial t}\Psi = -\frac{\imath}{\hbar}\hat{H}\Psi$$

beschrieben. Die stationären Lösungen sind die Eigenvektoren des Hamilton-Operators

$\hat{H}\phi_n = E_n\phi_n\ ,\ \ E_n \in \mathbb{R}.$

(Zeitunabhängige) statistische Physik betreibt man mit diesen Eigenvektoren + Eigenwerten und der kanonischen statistischen Gesamtheit

$$\hat{\rho} = \frac{\sum_n e^{-\beta E_n}|\phi_n\rangle\langle\phi_n|}{\sum_n e^{-\beta E_n}}.$$

Bei niedrigen Temperaturen ($\beta \gg 1$) erhalten nur die Eigenvektoren zu den niedrigsten Energieeigenwerten E_n ein signifikantes statistisches Gewicht (s. Boltzmann-Faktor $e^{-\beta E_n}$), für $T \rightarrow 0$ ($\beta \rightarrow \infty$) werden

die physikalischen Eigenschaften des Systems vollständig durch den Grundzustand von $\hat{H}$ erfasst, d. h. von ϕ_0 mit $E_0 = \min_n \{E_n\}$.

Beispiel: Spin-$\frac{1}{2}$ im Magnetfeld

$$\hat{H} = \underset{\nearrow}{-h\hat{\sigma}^z} + \underset{\nwarrow}{\Gamma\hat{\sigma}^x}$$

„longitudinales" Feld — „transversales" Feld

Wähle Darstellung (Basis des Hilbertraums), in der $\hat{\sigma}^z$ diagonal ist.

$$\mathcal{H} = \mathbb{R}^2\,, \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |1\rangle\,, \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |0\rangle$$

$$\hat{\sigma}^z \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}\,, \quad \hat{\sigma}^z \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\hat{\sigma}^x \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}\,, \quad \hat{\sigma}^x \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

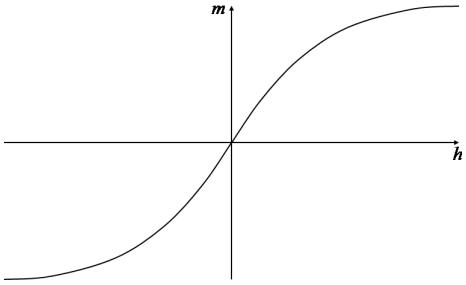
d. h. $\left\{ \begin{matrix} |1\rangle \\ |0\rangle \end{matrix} \right\}$ ist Zustand mit Spin $\left\{ \begin{matrix} \text{parallel} \\ \text{antiparallel} \end{matrix} \right\}$ zum longitudinalen Magnetfeld.

$$\Rightarrow \hat{H} = \begin{pmatrix} -h & \Gamma \\ \Gamma & +h \end{pmatrix}$$

Für $\Gamma = 0$ ist $\hat{H}$ diagonal ($\rightarrow$ Ising-Spin), $|1\rangle$ ist EV zum EW $-h$, $|0\rangle$ EV zum EW $+h$.

$$\hat{\rho} = \frac{e^{\beta h}|1\rangle\langle 1| + e^{-\beta h}|0\rangle\langle 0|}{2\cosh(\beta h)}$$

$$\begin{aligned} m = \langle\hat{\sigma}^z\rangle = Tr\left(\hat{\rho}\hat{\sigma}^z\right) &= \langle 1\,|\hat{\rho}\hat{\sigma}^z|\,1\rangle + \langle 0\,|\hat{\rho}\hat{\sigma}^z|\,0\rangle \\ &= \tanh(\beta h) \\ &\underset{\beta\to\infty}{\to} \operatorname{sign}(h) \end{aligned}$$



Für $T \to 0$ ist $\hat{\rho} = \Theta(h)\,|1\rangle\langle 1| + \Theta(-h)\,|0\rangle\langle 0|$, d. h. bei $\Gamma = 0$ und Temperatur 0 zeigt der Spin vollständig in Richtung des longitudinales Feldes.
Für $\Gamma \neq 0$ wird der Spin auch bei $T = 0$ aus der z-Richtung gelenkt. Nun muss man $\hat{H}$ diagonalisieren, um die Grundzustandseigenschaften (z. B. m, Magnetisierung) zu bestimmen.
In diesem einfachen Beispiel ist das trivial - das Problem wird aber sehr schnell schwieriger, wenn man z. B. mehrere Spins ferromagnetisch zusammenkoppelt $\left(-J\hat{\sigma}_i^z\hat{\sigma}_j^z\right)$ und auf jeden einzelnen ein transversales Feld $(\Gamma\hat{\sigma}_i^x)$ wirken lässt.
Z. B. in 1-dim.

$$\hat{H} = -J\sum_i \hat{\sigma}_i^z\hat{\sigma}_{i+1}^z - \Gamma\sum_i \hat{\sigma}_i^x$$

Ohne das transversale Magnetfeld ist das genau die 1-dim. ferromagnetische Ising-Spinkette, welche bei $T = 0$ einfach im Zustand $|1\rangle_i \, \forall i$ oder $|0\rangle_i \, \forall i$ ist.
Mit $\Gamma \neq 0$ wird die Bestimmung des Grundzustandes von $\hat{H}$ hochgradig nicht-trivial. Bei N Spins $\left(\frac{1}{2}\right)$ ist der Hilbertraum 2^N-dimensional, die Matrix, die den Operator $\hat{H}$ darstellt, ist $2^N \times 2^N$-dimensional. Für $N = 20$ ist $2^N \approx 10^6$ und eine numerische Berechnung der Eigenvektoren mit herkömmlichen Methoden (Householder & QR-Zerlegung, LINPACK, IMS2, numerical recipes) ist schwierig.
In diesem speziellen Fall der 1-dim. ferromagnetischen Ising-Spinkette kann das Grundzustands-Problem auf das 2-dim. klassische Ising-Modell bei endlicher Temperatur abgebildet werden, welches 1944 mit beträchtlichem analytischen Aufwand von Onsager gelöst wurde. Da aber Grundzustands-Probleme dieser Art häufig in der modernen Festkörperphysik (Heisenberg-Modell, Hubbard-Modell, etc.) auftreten und diese dann i. A. nicht mehr analytisch exakt lösbar sind, spielen numerische exakte Diagonalisierungs-Methoden eine sehr wichtige Rolle.
Konzeptionell sind sie sehr einfach (im Wesentlichen Matrix-Iteration), numerisch sehr aufwendig (wegen der Größe des Hilbertraumes, im Wesentlichen Speicherplatz-kritisch), aber straight-forward zu programmieren. Im Folgenden werden wir Methoden zur iterativen numerischen Bestimmung des Eigenvektors zum größten Eigenwert (äquivalent zum Grundzustand-Problem [kleinster EW] via $\hat{H} \to -\hat{H}$) besprechen.
Gegeben reell symmetrische $M \times M$-Matrix $\mathbf{H}$. Bestimme größten EW und dazugehörigen EV.

Brute-Force Potenzmethode

Bem. Im Folgenden bestimmen wir den größten EW (+ zug. EV).
In QM-Problemen interessiert der kleinste EW.
$\to$ Übergang von $\hat{H}$ zu $-\hat{H} + const$!
Da $\hat{H}$ reell-symmetrisch, existiert ONB $\Psi_1, \ldots, \Psi_M$, sodass

$$\hat{H}\Psi_n = E_n\Psi_n \quad \underbrace{E_1 > E_2} > \ldots > E_M$$

o. B. d. A. $E_M > 0$, ansonsten $\mathbf{H} \to \mathbf{H} + E'\mathbf{I}$ mit $E' = |E_M| + \varepsilon$

Im Folgenden nehmen wir der Einfachheit halber an, dass der Grundzustand von $-\hat{H}$ nicht entartet ist!

Betrachte einen zufällig gewählten Startvektor ϕ_0.

$$\phi_0 = \sum_{n=1}^{M} a_n \Psi_n$$

Nun iteriere durch Anwendung von $\hat{H}$ auf ϕ_0:

$$\phi_1 = \hat{H}\phi_0 = \sum_{n=1}^{M} a_n E_n \Psi_n$$

$$\phi_2 = \hat{H}\phi_1 = \sum_{n=1}^{M} a_n E_n^2 \Psi_n$$

$$\vdots$$

$$\phi_k = \hat{H}^k\phi_0 \underset{|}{=} E_1^k \left\{ a_1\Psi_1 + \sum_{n=2}^{M} a_n \underbrace{\left(\overbrace{\frac{E_n}{E_1}}^{<1} \right)^k}_{\substack{\to 0 \\ k\to\infty}} \Psi_n \right\}$$

$$\overset{|}{\to} a_1 E_1^k \Psi_1$$

D. h. durch sukzessive Anwendung von $\hat{H}$ auf den Startvektor wird die ursprünglich in ϕ_0 enthaltene (!) Komponente in Richtung von Ψ_1 gegenüber den übrigen Komponenten verstärkt. Normiert man nach jeder Iteration den durch Anwendung von $\hat{H}$ entstehenden Vektor, so verschwinden im Limes $k \to \infty$ die anderen Komponenten und übrig bleibt nur noch Ψ_1!
Also:

```
sum=0
do n=1,M

    phi(n)=rand()
    sum=sum+phi(n)*phi(n)

end
xnorm=1/sqrt(sum)
do n=1,M

    phi(n)=phi(n)*xnorm

end do
```

Dieser Teil berechnet den Vektor $\hat{H}\phi$ und normiert den resultierenden Vektor - üblicherweise ist M so groß, dass gerade einmal ϕ ganz in den Speicher passt, $\hat{H}$ wird i. A. nicht hineinpassen. Also wird $\hat{H}$ für das vorliegende (physikalische Problem) nicht abgespeichert, sondern immer wieder neu bestimmt! Der Übersichtlichkeit halber platziert man diesen Teil in einem Unterprogramm (z. B. callapply_ham(psi,psi_new)).

```
do iteration=1,itermax

    sum=0
    do n=1,M
        tmp=0
        do m=1,M   = H(n,m)
            tmp=tmp+H(m,n)*phi(m)
        end do
        phi_new(n)=tmp
        sum=sum+tmp*tmp
        end do
        xnorm=1/sqrt(sum)
        do n=1,M
            phi(n)=phi_new(n)*xnorm
        end do

--
C** Das ist ein guter Platz für den Check einer Abbruchbedingung.
end do
```

Wie findet man die nächstliegenden Eigenwerte $E_2, \ldots$?
Dazu braucht man sich nur vor Augen zu führen, dass Ψ_2 (der EV zum EW E_2) im orthogonalen Komplement von span $\{\Psi_1\}$ liegt (da $\hat{H}$ reell-symmetrisch), welches identisch mit span $\{\Psi_2, \ldots, \Psi_M\}$ ist. Also ist Ψ_2 der Grundzustand von $\hat{H}$ eingeschränkt auf span $\{\Psi_2, \ldots, \Psi_M\}$.
Projiziert man also nach jedem Iterationsschritt die Komponente in Richtung Ψ_1 aus dem Vektor $\hat{H}\phi$ heraus, so konvergiert das Verfahren (theoretisch) gegen Ψ_2.
Also

1.) Bestimme Ψ_1 (normiert).

2.) Initialisiere ϕ_0 (zufällig),
orthogonalisiere bzgl. Ψ_1 $\quad \phi_0 \to \phi_0 - \langle \phi_0 \cdot \Psi_1 \rangle \Psi_1$
normiere ϕ_0

3.) Iteriere: $\phi_{k+1} = \hat{H}\phi_k$
orthogonalisiere bzgl. Ψ_1 (Die Anwendung von $\hat{H}$ führt nicht aus span $\{\Psi_2, \ldots, \Psi_M\}$ heraus, aber durch Rundungsfehler empfiehlt es sich, hin und wieder die Orthogonalität herzustellen.)
normiere ϕ_{k+1}

Die Konvergenz dieses Verfahrens ist ziemlich langsam für Ψ_1 und funktioniert immer schlechter für die angeregten Zustände Ψ_2, ...

Versuchen Sie es einmal selber mit der $M \times M$-Matrix

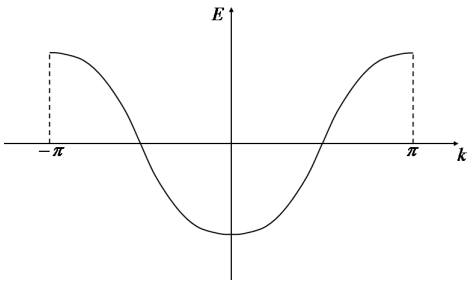
$$H = \begin{pmatrix} 0 & -1 & & & & & & -1 \\ -1 & 0 & -1 & & & & & \\ & -1 & 0 & -1 & & & & \\ & & -1 & 0 & -1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & -1 & 0 & -1 \\ -1 & & & & & & -1 & 0 \end{pmatrix}$$

Beschreibt z. B. freie Fermionen in 1-d

$$\hat{H} = -\sum_i \left(\hat{c}_i^\dagger \hat{c}_{i+1} + \hat{c}_i \hat{c}_{i+1}^\dagger \right)$$

mit p. b. c.

EW: $E_n = -2\cos(k_n)$, $k_n = \frac{2\pi n}{M}$, $n = -\frac{M}{2} + 1, \ldots, 0, \ldots, \frac{M}{2}$



$E = 2$: $\phi(i) = (-1)^i$

E_n: $\phi_n(i) = \frac{1}{\sqrt{M}} \begin{cases} \sin(k_n i) \\ \cos(k_n i) \end{cases}$

$E = -2$: $\phi(i) = 1$

Deshalb benutzt man raffiniertere Methoden.
Bemerkung: Alle Methoden dieser Art laufen unter dem Namen exakte Diagonalisierung.
Eigentlich trifft dieser Name nicht ganz zu:

1.) Die Matrix wird nicht vollständig diagonalisiert, sondern es wird nur der Grundzustand (evtl. noch einige wenige angeregte Zustände) berechnet.

2.) Exakt stimmt nur modulo numerische Rundungsfehler plus Abbruchbedingung. Der Name dient eher zur Abgrenzung von stochastischen Methoden wie Quanten-Monte-Carlo.

Lanczos-Verfahren

Dieses Verfahren basiert auf dem Konzept des invarianten Unterraums.
Betrachte den von $m \leq M$ linear unabhängigen Vektoren $q_1, \ldots, q_m$ aufgespannten Unterraum $Q = \text{span}\{q_1, \ldots, q_m\}$.
Angenommen, Q ist invariant unter der Wirkung von H, d. h.

$$\forall q \in Q : Hq \in Q$$

und $q_1, \ldots, q_m$ seien ONB von Q, $\mathbf{Q} = (q_1, \ldots, q_m) \in SO(m)$ $(*)$. Wegen $(*)$ gibt es eine Matrix T mit

$$Hq_i = \sum_{j=1}^{m} q_j t_{ji} \quad \forall i,$$

d. h. $H\mathbf{Q} = \mathbf{Q}T$
also $T = \mathbf{Q}^T H \mathbf{Q}$ $\quad (**)$
T stellt H eingeschränkt auf den Unterraum $\text{span}\{q_1, \ldots, q_m\}$ in der ONB q dar. T ist wieder reell-symmetrisch ($t_{ij} = t_{ji}$), daher existieren $\xi_1, \ldots, \xi_m \in \mathbb{R}$ und $y_1, \ldots, y_m \in \mathbb{R}^m$, sodass

$$Ty_j = \xi_j y_j$$

Die EW ξ_j sind wegen $(**)$ die Eigenwerte von $H|_Q$, denn $H(\mathbf{Q}y) = (H\mathbf{Q})y = (\mathbf{Q}T)y = \mathbf{Q}(Ty) = \mathbf{Q}(\xi y) = \xi(\mathbf{Q}y)$ und $\mathbf{Q}y_j$ sind die entsprechenden EV von H.

Résumé Die EW und EV der großen Matrix H können mittels der EW und EV der kleinen Matrix T gefunden werden, sofern Q ein invarianter Unterraum von H ist.

Idee von Lanczos: Konstruiere einen solchen Unterraum näherungsweise!
Benutze hierzu die Vektoren $\{\phi_0, H\phi_0, H^2\phi_0, ..., H^{m-1}\phi_0\}$, die wir bei der Potenz-Methode schon kennen gelernt haben, das heißt betrachte den Unterraum $Q = \text{span}\{\phi_0, H\phi_0, H^2\phi_0, ..., H^{m-1}\phi_0\}$.
Dann gilt $HQ = \text{span}\{H\phi_0, H^2\phi_0, ..., H^m\phi_0\}$, das heißt bis auf den letzten Basisvektor $H^m\phi_0$ sind alle schon in Q enthalten, das heißt abgesehen von der Komponente in Richtung $H^m\phi_0$ ist Q schon „fast" ein invarianter Unterraum. Wir haben gesehen, dass $H^m\phi_0$ für $m \to \infty$ gegen den Grundzustand von $-H$ konvergiert, das heißt $H^m\phi_0$ und $H^{m-1}\phi_0$ liegen für hinreichend großes m in fast derselben Richtung. Dies ist die Motivation, Q als nahezu invarianten Unterraum von H zu betrachten (in dem auch der Grundzustand enthalten sein sollte). E und Ψ (GS-Energie und GS) ergeben sich dann aus der exakten Diagonalisierung der „kleinen" Matrix T. Es stellt sich heraus, dass die Konvergenz des Verfahrens sehr schnell ist, typischerweise $m \sim 30 - 100$ sogar für $M \sim 10^7$ $(\approx 2^{24})$. Zudem ist T symmetrisch tridiagonal, was die Diagonalisierung von T vereinfacht. Verglichen mit dem Aufwand einer Matrix-Multiplikation $H\phi$ ist die Zeit für die Diagonalisierung sowieso zu vernachlässigen.

Zur Berechnung von $T = \begin{pmatrix} a_1 & b_1 & & & 0 \\ b_1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_{m-1} \\ 0 & & & b_{m-1} & a_m \end{pmatrix}$:

Die Bestimmungsgleichung ist (**), $H\mathbf{Q} = \mathbf{Q}T$
d. h. $\boxed{Hq_j = b_{j-1}q_{j-1} + a_j q_j + b_j q_{j+1}}$ für $j = 1, ..., m-1$

$$\Rightarrow q_{j+1} = \Big(\underbrace{Hq_j - a_j q_j - b_{j-1}q_{j-1}}_{=:r_j}\Big) / \underbrace{b_j}_{=:||r_j||} \qquad (***)$$

q_0 =zufälliger, normierter Startvektor, $b_0 = 1$
$q_{-1} = 0$

Aus den Bedingungen $q_{j+1} \perp q_j, q_{j-1}$ und $||q_{j+1}|| = 1$ ergeben sich die Rekursionsformeln für a_j und b_j

$a_j = q_j H q_j$ (1)

$b_j = ||r_j||$ $(= ||Hq_j - a_j q_j - b_{j-1} q_{j-1}||)$ (2)

Beweis durch Induktion:

Angenommen, $q_0, ..., q_j$ ist ONB. Dann ist $||q_{j+1}|| = 1$ (aus (***) trivial) und

$$q_j q_{j+1} = \Big(\underbrace{q_j H q_j - a_j \underbrace{||q_j||^2}_{=1} - b_{j-1} \underbrace{q_j q_{j-1}}_{=0}}_{=0} \Big) / b_j \overset{\text{aus(2)}}{=} 0$$

$$q_{j-1} q_{j+1} = \underbrace{q_{j-1} H q_j}_{} - a_j \overbrace{q_{j-1} q_j}^{=0} - b_{j-1} \overbrace{||q_{j-1}||^2}^{=1} = 0$$

$$= q_j H q_{j-1} \text{ da } H \text{ reell-symmetrisch}$$
$$= q_j \left(b_{j-2} q_{j-2} + a_{j-1} q_{j-1} + b_{j-1} q_j \right)$$
$$= b_{j-1} \underbrace{||q_j||^2}_{=1}$$

1) Wähle q_0, $||q_0|| = 1$

2) a) $\tilde{q}_1 = Hq_0 - q_0 \underbrace{(q_0 H q_0)}_{=a_0} \rightsquigarrow \tilde{q}_1 \perp q_0$

b) $q_1 = \tilde{q}_1 / ||\tilde{q}_1||$

3) a) $\tilde{q}_2 = Hq_1 - q_1 \underbrace{(q_1 H q_1)}_{=a_1} - q_0 \underbrace{(q_0 H q_1)}_{=b_0} \rightsquigarrow \tilde{q}_2 \perp q_0, q_1$

b) $q_2 = \tilde{q}_2 / \underbrace{||\tilde{q}_2||}_{b_1}$

$\vdots$

j+2) a) $\tilde{q}_{j+1} = Hq_j - q_j \underbrace{(q_j H q_j)}_{=a_j} - q_{j-1} \underbrace{(q_{j-1} H q_j)}_{=b_{j-1}} \rightsquigarrow \tilde{q}_{j+1} \perp q_j, q_{j-1}$

b) $q_{j+1} = \tilde{q}_{j+1} / \underbrace{||\tilde{q}_{j+1}||}_{b_j}$

Beachte: $q_{j-i} H q_j = q_j H q_{j-1} = q_j \tilde{q}_j = b_{j-1}$

mit $Hq_{j-1} = \tilde{q}_j + a_{j-1} q_{j-1} + b_{j-2} q_{j-2}$

Zudem: $q_{j+1} \perp q_i$ für $i < j - 1$, denn

$$q_i q_{j+1} = q_i H q_j - a_i \overbrace{q_i q_j}^{=0} - b_{j-1} \overbrace{q_i q_{j-1}}^{=0}$$
$$= q_j H q_i \, (\in \text{span}\{q_0, ..., q_{j-1}\} \perp q_j) = 0$$

Nun offensichtlich $T \left(= \mathbf{Q}^T H \mathbf{Q} \right)$ Tridiagonalmatrix.

Bei der praktischen Implementierung berechnet man zuerst

$w_j = Hq_j - b_{j-1} q_{j-1}$

und dann $a_j = q_j w_j = q_j H q_j - b_{j-1} \underbrace{q_j q_{j-1}}_{=0}$ wie gewünscht, das spart eine Matrix-Multiplikation.

Anschließend berechnet man

$r_j = w_j - a_j q_j$

und schließlich

$b_j = ||r_j||$

$q_{j+1} = r_j / b_j$

Ein (theoretisches) Abbruch-Kriterium ist $b_j = 0$ für $j = m$, denn dann liegt r_j innerhalb des Raumes $Q = \left\{ q_0, Hq_0, ..., H^{m-1} q_0 \right\}$. In der Praxis kann man früher abbrechen.

Schematisch:

$q_0 = 0$, q_1 =Startvektor
$b_0 = 1$, $j = 1$
while $b_j \neq 0$ {

$q_{j+1} = Hq_j - b_{j-1}q_{j-1}$ (*)
$a_j = q_j \cdot q_{j+1}$
$q_{j+1} = q_{j+1} - \alpha_j q_j$
$b_j = ||q_{j+1}||$
$q_{j+1} = q_{j+1}/||q_{j+1}||$ (**)
$m = j$, $j = j + 1$

}

Ist man zunächst nur an der Berechnung der Tridiagonalmatrix T interessiert, so erkennt man leicht, dass man mit zwei Vektoren der Länge M (sehr groß!) im Hauptspeicher auskommt, nämlich q_0 und q_1 : ab (*) wird q_{j-1} nicht mehr gebraucht, also kann q_{j+1} auf q_{j-1} abgelegt werden.
Nach (**) vertauscht man dann q_0 und q_1 (s. o. Programm-Listing). Beachte, dass mit bekanntem Startvektor (z. B. zufällig mit festem seed) die Folge der Vektoren $q_1, ..., q_m$ jederzeit rekonstruiert werden kann, was bei der Berechnung des Grundzustandes auch notwendig ist.
Um die Iteration (***) durchführen zu können, muss man nur Speicher für q_j und q_{j-1} reservieren (double precision number = 8 Byte (64 Bits), 2^{24} dp-Zahlen ($L = 24$ im Spin-Modell) brauchen aber 2 MB, 2 Vektoren also 4 MB), denn $q_{j+1}(i)$ kann auf $q_{j-1}(i)$ abgelegt werden.

Also z. B.:
Initialisieren

```
iseed=1234, xtmp=rand(iseed)
sum=0
do i=1,M
    q(i)=rand()
    qn(i)=0
    sum=sum+q(i)*q(i)
end do
do i=1,M
    q(i)=q(i)/sqrt(sum)
end do
b(0)=1
```

Berechnung der $a(j)$, $b(j)$

```
do j=1,m
    aj=0
    bj1=b(j-1)
    do i=1,M
        tmp=... ←hier wird (Hq)(i) berechnet, hängt von H ab (s. u.)
        qn(i)=tmp-bj1*qn(i)
        aj=aj+tmp*q(i)
    end do
    a(j)=aj
    sum=0
    do i=1,M
        qn(i)=qn(i)-aj*q(i)
        sum=sum+qn(i)*qn(i)
```

```
    end do
    b(j)=sqrt(sum)
    do i=1,M
        tmp=qn(i)/b(j)
        qn(i)=q(i)
       end q(i)=tmp
```

Anmerkung:

```
tmp=0
do h=1,M
    tmp=tmp+H(i,k)q(k)
end do
```

Aber $H(i,k)$ i. A. zu berechnen.

Mit $a(j)$ und $b(j)$ geht man nun in eine „handelsübliche" Diagonalisierungsroutine (s. Numerik 1) für symmetrische Tridiagonalmatrizen, z. B. cal(tqli(a,b,m,z)) (Numerical Recipes), dies liefert Eigenwerte von T (in a) sowie die Eigenvektoren in $z(i,j)$. $b(m) = E_0$, $z(j,m) = y(j)$
Will man neben der Grundzustandsenergie $-E_0$ auch den Grundzustandsvektor von H haben, so muss man, da man die q_j nicht abgespeichert hat, die q-Vektoren noch einmal konstruieren (diesmal mit schon bekannten $a(j)$ und $b(j)$), da $\Psi_0 = \sum_{j=1}^{m} y(j)\, q_j$.
Man wiederholt die Prozedur solange mit anwachsendem m, bis die Abweichungen in E_0 zwischen den einzelnen Iterationsschritten hinreichend klein ($\lesssim 10^{-18}$) geworden sind.

Klingt alles gut - wo ist das Problem?
Aufgrund von Rundungsfehlern geht für wachsendes m die Orthogonalität der q_j untereinander verloren!

Abhilfe:
Reorthogonalisieren, wirklich iterieren (mit festem $m \approx 20$ und neuem Startvektor = alter Grundzustand) - oder ignorieren.

Z. B. bei Verwendung von „tqli" aus den „Numerical Recipes":

```
do i=1,M
    d(i)=a(i)
    e(i)=b(i-1)
    do j=1,M
        z(i,j)=0
       end do
       z(i,j)=1.0      z auf Einheitsmatrix initialisieren!
end do
tqli(d,e,m,z)
do i=1,M
    y(i)=z(i,m)
end do
```

Nun wiederhole Lanczos-Prozedur, um den (approximierten) Grundzustand $\Psi = \sum_j y(j)\, q_j$ von H zu berechnen.

Initialisierung:

```
sum=0, iseed= 12345, xtmp=rand(iseed)
do i=1,M
    q(i)=rand()
    psi(i)=0
    qn(i)=0
    sum=sum+q(i)*q(i)
end do
sum=1/sqrt(sum)
do i=1,M
    q(i)=q(i)*sum
end do
```

Konstruktion von Ψ:

```
do j=1,m
    do i=1,M
        Psi(i)=Psi(i)+y(j)*q(i)
    end do
    do i=1,M
        tmp=...
        qn(i)=(tmp-a(j)q(i)-b(j-1)qn(i))/b(j)
    end do
    do i=1,M
        tmp=qn(i)
        qn(i)=q(i)
        q(i)=tmp
    end do
end do
```

Rekursive Variationsverfahren:
Erinnerung: Variationsprinzip: $\forall|\Psi\rangle \neq 0$
$E_0 \leq \dfrac{\langle\Psi|H|\Psi\rangle}{\langle\Psi|\Psi\rangle} =: R_\Psi \rightarrow$ Rayleigh-Quotienten

Beliebiges $|\Psi\rangle$ liefert obere Schranke für E_0.
rekursive Variationsverfahren: systematische Verbesserung der Schranke
$|\Psi\rangle = \sum_{j=1}^{m} \alpha_j |\phi_j\rangle$
Minimierung von $R_\Psi \Leftrightarrow$ Berechnung der kleinsten Eigenwerte der Matrix $H^{(m)}$
(bzgl. α_j)
$$\left[H = \left(\begin{array}{c|c} H^{(m)} & X^T \\ \hline \underbrace{X}_{m} & Y \end{array} \right) \Big\}m \right]$$
Separationstheorem: Zusätzlicher Vektor $|\phi_{m+1}\rangle$ und Variationsparamter α_{m+1}
$\rightarrow \boxed{E_0 \leq E_0^{(m+1)} \leq E_0^{(m)}}$
$\uparrow$
verbesserte Schranke
$\rightsquigarrow$Basis des Lanczos-Verfahrens

Beachte $\frac{\partial R_\Psi}{\partial \alpha_j} \propto \langle\phi_j| \left(H|\Psi\rangle\langle\Psi|\Psi\rangle - |\Psi\rangle\langle\Psi|H|\Psi\rangle\right)$
$\Rightarrow$ Der Vektor, der in der $m+1$-sten Lanczos-Iteration konstruiert wird, ist parallel zu den Vektoren, die die Richtung zur Normierung des Rayleigh-Quotienten spezifizieren.

Die Tatsache, dass jeder Schritt des Lanczos-Prozesses als ein Versuch angesehen werden kann, den Rayleigh-Quotienten zu minimieren, suggeriert, dass man in der Praxis die Berechnung des Grundzustandes so rearrangieren kann, dass keine explizite Diagonalisierung einer Tridiagonal-Matrix notwendig ist:
Angenommen, wir hätten bereits m Lanczos-Iterationen durchgeführt, d. h. wir kennen
$|\Psi\rangle \left(= \sum_{j=1}^{m} \alpha_j |\phi_j\rangle\right) \left(||\Psi|| = 1\right)$.
Der nächste Schritt besteht aus der Berechnung des neuen Vektors $|\Psi'\rangle = \frac{H|\Psi\rangle - |\Psi\rangle\langle\Psi|H|\Psi\rangle}{\sqrt{\langle\Psi|H^2|\Psi\rangle - \langle\Psi|H|\Psi\rangle^2}}$.
Minimierung des Rayleigh-Quotienten bzgl. des Variationszustandes $X|\Psi\rangle + Y|\Psi'\rangle$ bedeutet:
Lösung des Eigenwertproblems für die 2 x 2-Matrix $A = \begin{pmatrix} \langle\Psi|H|\Psi\rangle & \langle\Psi|H|\Psi'\rangle \\ \langle\Psi'|H|\Psi\rangle & \langle\Psi'|H|\Psi'\rangle \end{pmatrix}$
Eigenvektor (X_0, Y_0) zum kleinsten Eigenwert von A wird dann benutzt, um den neuen Variationszustand $|\Psi\rangle = X_0|\Psi\rangle + Y_0|\Psi'\rangle$
zu konstruieren. Dieser dient dann als Input für die nächste Iteration. $\leftrightarrow$ Steepest-Descent-Verfahren

Lanczos mit $m = 2$ oder Steepest-Descent

1.) Initialisierung : Wähle ein q_1 als Startvektor (z. B. zufällig generiert)

2.) Definiere

$$a_1 = q_1 H q_1$$
$$b_1 = ||Hq_1 - a_1 q_1||$$
$$q_2 = \frac{Hq_1 - a_1 q_1}{b_1}$$
$$a_2 = q_2 H q_2$$

3.) Berechne die Eigenvektoren v_1, v_2 und Eigenwerte λ_1, λ_2 von $\begin{pmatrix} a_1 & b_1 \\ b_1 & a_2 \end{pmatrix}$:

$$\lambda_{1,2} = \frac{a_1 + a_2}{2} \pm \gamma,\ \gamma = \sqrt{\left(\frac{a_1 + a_2}{2}\right)^2 - (a_1 a_2 - b_1^2)}$$
$$v_{1,2} = \begin{pmatrix} \frac{a_2 - a_1}{2} \mp \gamma \\ -b_1 \end{pmatrix} = \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix}$$

4.) Wähle den größten Eigenwert $\left(\frac{a_1+a_2}{2} + \gamma\right)$ und normiere den zugehörigen Eigenvektor:
$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \frac{1}{\sqrt{\left(\frac{a_2 - a_1}{2} - \gamma\right)^2 + b_1^2}} \begin{pmatrix} \frac{a_2 - a_1}{2} - \gamma \\ -b_1 \end{pmatrix}$

5.) Konstruiere $q_1^{new} = x_1 q_1 + x_2 q_2$

Beispiele für Systeme, welche mittels des Lanczos-Verfahrens exakt diagonalisiert werden können sind

- das transversale Ising-Modell:

$$\hat{H} = -J \sum_i \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - \Gamma \sum_i \hat{\sigma}_i^x$$

($\hat{\underline{\sigma}}$ Pauli-Spin-Matrizen)

- das Heisenberg-Modell:

$$\hat{H} = J \sum_i \underbrace{\left(\hat{\sigma}_i^x \hat{\sigma}_{i+1}^x + \hat{\sigma}_i^y \hat{\sigma}_{i+1}^y\right)}_{\frac{1}{2}\left(\hat{\sigma}_i^+ \hat{\sigma}_{i+1}^- + \hat{\sigma}_i^- \hat{\sigma}_{i+1}^+\right)} + \Delta \sum_i \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z$$

(Spin-$\frac{1}{2}$-XXZ)

- das Hubbard-Modell:

$$\hat{H} = -t \sum_{i,\sigma} \left(\hat{c}^\dagger_{i,\sigma} \hat{c}_{i+1,\sigma} + \hat{c}_{i\sigma} \hat{c}^\dagger_{i+1,\sigma} \right) + U \sum_i \hat{n}_{i\uparrow} \hat{n}_{i\downarrow}$$

  ($c^\dagger_{i\sigma}, c_{i\sigma}$ Fermionen-Erzeuger/Vernichter)

- das Bose-Hubbard-Modell:

$$\hat{H} = -t \sum_i \left(\hat{a}^\dagger_i \hat{a}_{i+1} + \hat{a}_i \hat{a}^\dagger_{i+1} \right) + U \sum_i \hat{n}_i^2$$

  ($\hat{a}^\dagger_i, \hat{a}_i$ Bose-Erzeuger/Vernichter, für Hardcore-Bosonen $n_i = 0, 1$)

Instruktives Beispiel: transversales Ising-Modell in einer Dimension

$$\hat{H} = \underbrace{-J \sum_i \hat{\sigma}^z_i \hat{\sigma}^z_{i+1}}_{\hat{H}^z} \underbrace{-\Gamma \sum_i \hat{\sigma}^x_i}_{\hat{H}^x}$$

Für einen einzelnen Gitterplatz i gilt in der s^z-Basis:

$\hat{\sigma}^z_i |\uparrow\rangle_i = +1 |\uparrow\rangle_i$
$\hat{\sigma}^z_i |\downarrow\rangle_i = -1 |\downarrow\rangle_i$
$\hat{\sigma}^x_i |\uparrow\rangle_i = |\downarrow\rangle_i$
$\hat{\sigma}^x_i |\downarrow\rangle_i = |\uparrow\rangle_i$

Für L Gitterplätze und periodische Randbedingungen $\left(\hat{\sigma}^z_{L+1} = \hat{\sigma}^z_1\right)$ ist der Hilbertraum 2^L-dimensional, als Basis wählt man die Ising-Basis $\{|s_1 s_2 \dots s_L\rangle\}$ mit $s_i = \uparrow, \downarrow$ (oder $s_i = 0, 1$). In dieser Basis nimmt $\hat{H}$ für $L = 2$ die Gestalt

$$H = \begin{pmatrix} 2J & \Gamma & \Gamma & 0 \\ \Gamma & 0 & 0 & \Gamma \\ \Gamma & 0 & 0 & \Gamma \\ 0 & \Gamma & \Gamma & 2J \end{pmatrix}$$

an.

Bei der numerischen Implementierung ist es sehr praktikabel, den einzelnen Basisvektoren eineindeutige Nummern zuzuordnen, für das transversale Ising-Modell kann dies z. B. auf die Folgende Art und Weise geschehen

$$n\left(|s_1 \dots s_L\rangle\right) = \sum_{i=1}^{L} S_i 2^{i-1}$$

mit

$$S_i = \begin{cases} 1 & \text{für } s_i = \uparrow \\ 0 & \text{für } s_i = \downarrow \end{cases}$$

Zum Beispiel für

$n(|\uparrow\uparrow\downarrow\uparrow\rangle) = 1 + 2 + 0 + 8 = 11$

Es sei kurz an die Identitäten

$\hat{\sigma}^z_i |s_1 \dots s_L\rangle = (2S_i - 1) |s_1 \dots s_L\rangle$
$\hat{\sigma}^x_i |s_1 \dots s_L\rangle = |s_1 \dots \tilde{s}_i \dots s_L\rangle$

mit

$$\tilde{s}_i = \begin{cases} \uparrow & \text{für } s_i = \downarrow \\ \downarrow & \text{für } s_i = \uparrow \end{cases}$$

erinnert. Stellt man nun den Zustand $|\psi\rangle$ in der so numerierten Basis $\{|n\rangle\}$ dar

$$|\psi\rangle = \sum_{n=0}^{2^L-1} \psi(n)\,|n\rangle,$$

ergibt sich die Wirkung des Hamilton-Operators zu

$$\hat{H}^z|\psi\rangle = \sum_{n=0}^{2^L-1} \psi(n)\,|n\rangle \cdot \underbrace{E_{kl}(n)}_{=-J\sum_{i=1}^{L}(2S_i-1)(2S_{i+1}-1)}$$

und

$$\hat{H}^x|\psi\rangle = \sum_{n=0}^{2^L-1} \Gamma \sum_{i=1}^{L} \psi\left(n\left(|s_1 \ldots \tilde{s}_i \ldots s_L\rangle\right)\right)|s_1 \ldots \tilde{s}_i \ldots s_L\rangle$$

Wir können auch jedem Basiszustand $|n\rangle$ seine Magnetisierung $mag(n)$ und Energie $energy(n)$ zuordnen, dies kann z. B. mit folgendem Meta-Code geschehen ($M = 2^L - 1$) .

Für die Magnetisierung:

```
do n=0 to M
mag(n) = 0
n_tmp = n
do i=1 to L
   mag(n) = mag(n) + and(n_tmp,1)
   n_tmp = n_tmp/2
end do
mag(n) = 2*mag(n)-L
end do
```

Die Energie:

```
do n=0 to M
   nen = n/2
   nen = nen + and(n,1)*(2**(L-1))
end do
```

Mithilfe der Energie kann nun die Wirkung des Hamilton-Operators tmp auf einen Zustand q berechnet werden

```
itwo(0) = 1
do ii=1 to L
   itwo(ii)=2*itwo(ii-1)
end do

do n=0 to M
   tmp(n) = energy(n)*q(n)
   do ii = 0 to L
      index = xor(n,itwo(ii))
      tmp(n) = tmp(n) + q(index) * Gamma
   end do
end do
```

Ein weiteres interessantes Beispiel, vor allem bezüglich der Ausnutzung von Symmetrien bei der numerischen Implementierung, liefert das Heisenberg-Modell:

$$H = J \sum_i \underbrace{\left(\hat{\sigma}_i^x \hat{\sigma}_{i+1}^x + \hat{\sigma}_i^y \hat{\sigma}_{i+1}^y\right)}_{\frac{1}{2}\left(\hat{\sigma}_i^+ \hat{\sigma}_{i+1}^- + \hat{\sigma}_i^- \hat{\sigma}_{i+1}^+\right)} + \Delta \sum_i \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z$$

Man kann nachrechnen, dass $\hat{H}$ mit dem Operator $\hat{S}^z := \sum_i \hat{\sigma}_i^z$ der Gesamtmagnetisierung kommutiert, damit ist $M := \sum_i s_i$ eine Erhaltungsgröße. Des Weiteren folgt aus analytischen Kalkulationen, dass sich der Grundzustand stets im Unterraum zu $M = 0$ befindet. Diesen Sachverhalt kann man sich zunutze machen, um die Suche nach dem Grundzustand von Beginn an auf diesen Unterraum einzuschränken und so erheblichen Rechenaufwand zu sparen. Wir wählen erneut die Ising-Basis und betrachten ausschließlich die Vektoren mit $M = 0$, problematisch ist zunächst, dass wir für die numerische Implementierung eine neue Numerierung unserer Zustände finden müssen. Eine Möglichkeit besteht hier zum Beispiel in

$$n\left(|s_1 \dots s_L\rangle\right) = 1 + \sum_{i=1}^{L/2} \begin{pmatrix} P_i - 1 \\ i \end{pmatrix},$$

wobei P_i die Position des i-ten „↑"-gesetzten Spins bezeichnet (da $M = 0$ gelten soll, müssen genau $\frac{L}{2}$ Spins nach oben bzw. unten zeigen).

Zum Beispiel $L = 4$:

Zustand	Nummer
0011	1
1010	2
0110	3
1001	4
1010	5
1100	6

Durch die Ausnutzung dieser Symmetrien nimmt H eine blockdiagonale Gestalt an, das Lanczos-Verfahren wird dann auf den entsprechenden Untervektorraum angewandt. Darüber hinaus können je nach Hamiltonian andere Symmetrien beachtet werden, z. B. Translationssymmetrie, Spiegelungssymmetrie, Spin-Umkehr-Symmetrie ...

Ist man mittels des Lanczos-Verfahrens des Grundzustandes $|\psi_0\rangle = \sum_{i=1}^{M} a_i |q_i\rangle$ habhaft geworden, kann man direkt den Erwartungswert physikalischer Observablen angeben, sofern diese in der gewählten Basis diagonal sind. Sei hierzu O eine physikalische Observable, dann gilt

$$\begin{aligned} \langle O \rangle &= \langle \psi_0 | O | \psi_0 \rangle \\ &= \sum_{i,j}^{M} a_i a_j \langle q_i | O | q_j \rangle \end{aligned}$$

Damit ist die Berechung einer Observablen eine Operation $\mathcal{O}(M^2)$.

$$H = -J \sum_i \left(\sigma_i^z \sigma_{i+2}^z - R \sigma_i^z \sigma_{i+2}^z \right) - \Gamma \sum_i \sigma_i^x$$

```
#include <stdio.h>
#include <math.h>

#define ns_max   16
#define m_max    20
#define deviation 1.e-10

int    *mag;
double *energy, *q, *qn, *psi;

void main ( void )
{

  float ran1(int *idum);
  void  init_mag( int n, int ns );

  double d[m_max], rj, rkappa,
         rj_rec0, rkappa_a, rkappa_e, rkappa_int, xtmp, e0, e1,
         xnorm, xmag, xe, prob;
  int    miter[31], irkappa, irkappa_max,
         i, j, n, ns, irseed, m0, m;

  ns = 8;
  rj_rec0    = 0.05;
  rkappa_a   = 0.2;
  rkappa_e   = 0.8;
  rkappa_int = 0.01;
  irseed     = 12345;

  n   = pow(2,ns) - 1;
  psi = (double *) malloc ( (unsigned) (n+1) * sizeof(double) );
  q   = (double *) malloc ( (unsigned) (n+1) * sizeof(double) );
  qn  = (double *) malloc ( (unsigned) (n+1) * sizeof(double) );

  for ( i=1; i<=30; i++ ) miter[i]=20;
  for ( i=1; i<=10; i++ ) miter[i]=i;

  xtmp =  (double) ran1( &irseed );
  for ( i=0; i<=n; i++ )  psi[i] = (double) ran1( &irseed );

  init_mag    ( n, ns );

  irkappa_max = (rkappa_e-rkappa_a) / rkappa_int;

  for ( irkappa=0; irkappa<=irkappa_max; irkappa++ ) {

    rj     = 1./rj_rec0;
    rkappa = rkappa_a + rkappa_int*(float) irkappa;

    init_energy ( n, ns, rj, rkappa );

    e0 =  99999.;
    e1 = -99999.;
    m0 = 0;
    m  = miter[ns];
    while ( fabs( e1-e0 ) > deviation ) {
      m0++;
      lanczos ( ns, m, psi, d );
      e0 = e1;
      e1 = d[m]/ns;
    };

    xnorm = 0.;
    xmag  = 0.;
    xe    = 0.;
    for ( i=0; i<=n; i++ ) {
```

```
      prob = psi[i]*psi[i];
      xnorm += prob;
      xmag  += prob * fabs( (float) mag[i] ) / (float) ns;
      xe += prob*energy[i];
    };
    xe = -xe * (float) rj_rec0 + (float) rkappa * (float) ns;
    printf("%8.4f %10.6f %14.6g %14.6g %9.5f %5d\n",
           rkappa, xmag, xe, d[m]/ns, xnorm, m0 );

  };

  exit(0);


}
#include <math.h>

#define m_max 50

int    *mag, itwo[50];
double *energy, *q, *qn;

void lanczos ( int ns, int m, double *psi, double *d )
{

/******* Lanzcos routine for arbitrary Ising models in transv. field

 ------- On Input:

         ns    - number of spins
         m     - number of iterations
         psi   - the initial vector

 ------- On Output:

         psi - groundstate
         d   - eigenvalues, d(m) = ground state energy

 ------- Dependencies:

         tqli           - diagonalization procedure for tridiagonal matrices
         apply_hamilton - Applies Hamiltonian to vector

 *********************************************************************/

   void tqli ( double d[], double e[], int n, double z[][m_max+1] );
   void apply_hamilton ( double *q, double *qn, int ns,
                         double *alpha, double *beta);

   double a[m_max+1], b[m_max+1], e[m_max+1],
          z[m_max+1][m_max+1], y[m_max+1],
          sum, aj, bj1, term, ev;
   int    i, j, n, nsm1, ierr;

   nsm1 = ns - 1;
   n    = pow(2,ns)- 1;
   for ( i=0; i<=ns; i++ ) itwo[i] = pow(2,i);

 /******************  Constructing the initial vector  ****************/

   sum = 0.;
   for ( i=0; i<=n; i++ ) sum += psi[i]*psi[i];
   sum = 1./sqrt(sum);
   for ( i=0; i<=n; i++ ) {
     qn[i] = 0.;
     q [i] = psi[i]*sum;
```

```
                                r=sqrt((g*g)+1.0);
                                g=d[m]-d[l]+e[l]/(g+SIGN(r,g));
                                s=c=1.0;
                                p=0.0;
                                for (i=m-1;i>=l;i--) {
                                        f=s*e[i];
                                        b=c*e[i];
                                        if (fabs(f) >= fabs(g)) {
                                                c=g/f;
                                                r=sqrt((c*c)+1.0);
                                                e[i+1]=f*r;
                                                c *= (s=1.0/r);
                                        } else {
                                                s=f/g;
                                                r=sqrt((s*s)+1.0);
                                                e[i+1]=g*r;
                                                s *= (c=1.0/r);
                                        }
                                        g=d[i+1]-p;
                                        r=(d[i]-g)*s+2.0*c*b;
                                        p=s*r;
                                        d[i+1]=g+p;
                                        g=c*r-b;
                        /* Next loop can be omitted if eigenvectors not wanted */
                                        for (k=1;k<=n;k++) {
                                                f=z[k][i+1];
                                                z[k][i+1]=s*z[k][i]+c*f;
                                                z[k][i]=c*z[k][i]-s*f;
                                        }
                                }
                                d[l]=d[l]-p;
                                e[l]=g;
                                e[m]=0.0;
                        }
                } while (m != l);
        };

/*  order eigenvalues and eigenvectors */

        for ( ii=2; ii<=n; ii++ ) {
          i = ii - 1;
          k = i;
          p = d[i];

          for ( j=ii; j<=n; j++ ) {

            if (d[j] <= p) {
              k = j;
              p = d[j];
            };
            if (k != i) {
              d[k] = d[i];
              d[i] = p;
              for ( j=1; j<=n; j++ ) {
                p = z[j][i];
                z[j][i] = z[j][k];
                z[j][k] = p;
              };
            };

          };

        };

}
```

```
  };
  b[0] = 1.;

/*********************   Start the recursion  ***********************
 Recursion relation : A*q[j] = b[j-1]*q[j-1] + a[j]*q[j] + b[j]*q(j+1)
 At this stage q() is q[j] and qn() is q[j-1]. Later the new vector
 q(j+1) will also be stored at qn().
*********************************************************************/

  for ( j=1; j<=m; j++ ) {

    aj  = 0.;
    bj1 = b[j-1];

    apply_hamilton ( q, qn, ns, &aj, &bj1 );

    a[j] = aj;
    sum  = 0.;
    for ( i=0; i<=n; i++ ) {
      qn[i] = qn[i] - aj*q[i];
      sum  += qn[i]*qn[i];
    };
    sum = 1./sqrt(sum);
    b[j] = 1./sum;

    for ( i=0; i<=n; i++ ) {
      term = sum*qn[i];

/* 'term" is now the i-th component of q(j+1). q() is still q[j].
   In the next two steps q[j] is stored in qn() and term, in q(). */

      qn[i] = q[i];
      q[i]  = term;

    };

  };

/***********  Construction of tridiagonal matrix finished ************/

  for ( i=1; i<=m; i++ ) {
    d[i] = a[i];
    e[i] = b[i-1];
    for ( j=1; j<=m; j++ ) z[i][j] = 0.;
    z[i][i] = 1.0;
  };
  tqli ( d, e, m, z );
  for ( i=1; i<=m; i++ ) y[i] = z[i][m];

/*******  Constructing the previously chosen initial vector  ********/

  sum = 0.;
  for ( i=0; i<=n; i++ ) {
    q[i] = psi[i];
    psi[i] = 0.;
    qn[i]  = 0.;
    sum += q[i]*q[i];
  };
  sum = 1./sqrt(sum);
  for ( i=0; i<=n; i++ ) q[i] *= sum;

/***** Repeating the Lanczos procedure for finding eigenvector *****/

  for ( j=1; j<=m; j++ ) {

    bj1 = b[j-1];
```

```
    aj  = a[j];

    apply_hamilton ( q, qn, ns, &aj, &bj1 );

    sum = 0.;
    for ( i=0; i<=n; i++ ) {
      qn[i] = qn[i] / b[j];
      sum  += qn[i]*qn[i];
    };
    sum = 1./sqrt(sum);

    for ( i=0; i<=n; i++ ) {
      term    = sum*qn[i];
      qn[i]   = q[i];
      q[i]    = term;
      psi[i] += y[j]*qn[i];
    };

  };

  ev = d[m];

/*********************************************************************
   psi() is now the (normalised) ground-state eigenvector
   ev is the ground-state energy
*********************************************************************/

}

#include <math.h>

double *energy;
int    itwo[50];

void apply_hamilton ( double *q, double *qn, int ns,
                      double *alpha, double *beta )
{

  int n, i, ii, index;
  double asum, term;

  n= pow(2,ns);
  asum = 0.;
  for ( i=0 ; i<n; i++ ) {
    term = energy[i]*q[i];
    for ( ii=0 ; ii<ns; ii++ ) {
      index = i^itwo[ii];
      term += q[ index ];
    };
    qn[i] = term - *beta*qn[i] - *alpha*q[i];
    asum = asum + term*q[i];
  };
  *alpha = asum;

}
int *mag;
void init_mag ( int n, int ns )
{
  int i, j, itmp;

  mag     = (int *) malloc ( (unsigned) (n+1) * sizeof(int) );
  for ( i=0; i<=n; i++ ) {
    mag[i] = 0;
    itmp = i;
    for ( j=1; j<=ns; j++ ) {
      mag[i] += (itmp &1);
```

```
      itmp >>= 1;
    };
    mag[i] = 2*mag[i] - ns;
  };

}
#include <math.h>

int    *mag;
double *energy;

void init_energy  ( int n, int ns, double rj, double rkappa )
{

  int i, ii, iii, ien1, ien2;

  energy = (double *) malloc ( (unsigned) (n+1) * sizeof(double) );

  for ( i=0; i<=n; i++ ) {

/*-------------
  periodic boundary conditions: circular shift!
  -------------
    ien1 = xor(i,ishftc(i,1,ns))
    ien2 = xor(i,ishftc(i,2,ns))
  -------------*/

    ien1 =  (i>>1);
    ien1 += (i&1)*pow(2,ns-1);
    ien1 ^= i;

    ien2 =  (i>>2);
    ien2 += ((i&1)*pow(2,ns-2) + (i&2)*pow(2,ns-2));
    ien2 ^= i;

    energy[i] = -rj * (double) (mag[ien1] - rkappa*mag[ien2] );

  };


}

void nrerror(error_text)
char error_text[];
{
        void exit();

        printf("Numerical Recipes run-time error...\n");
        printf("%s\n",error_text);
        printf("...now exiting to system...\n");
        exit(1);
}
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349

float ran1 ( int *idum )
{
```

```
        static long ix1,ix2,ix3;
        static float r[98];
        float temp;
        static int iff=0;
        int j;
        void nrerror();

        if (*idum < 0 || iff == 0) {
                iff=1;
                ix1=(IC1-(*idum)) % M1;
                ix1=(IA1*ix1+IC1) % M1;
                ix2=ix1 % M2;
                ix1=(IA1*ix1+IC1) % M1;
                ix3=ix1 % M3;
                for (j=1;j<=97;j++) {
                        ix1=(IA1*ix1+IC1) % M1;
                        ix2=(IA2*ix2+IC2) % M2;
                        r[j]=(ix1+ix2*RM2)*RM1;
                }
                *idum=1;
        }
        ix1=(IA1*ix1+IC1) % M1;
        ix2=(IA2*ix2+IC2) % M2;
        ix3=(IA3*ix3+IC3) % M3;
        j=1 + ((97*ix3)/M3);
        if (j > 97 || j < 1) nrerror("RAN1: This cannot happen.");
        temp=r[j];
        r[j]=(ix1+ix2*RM2)*RM1;
        return temp;
}

#undef M1
#undef IA1
#undef IC1
#undef RM1
#undef M2
#undef IA2
#undef IC2
#undef RM2
#undef M3
#undef IA3
#undef IC3
#include <math.h>
#define m_max 50

#define SIGN(a,b) ((b)<0 ? -fabs(a) : fabs(a))

void tqli( double d[], double e[], int n, double z[m_max+1][m_max+1] )
{
        int m,l,iter,i,j,k,ii;
        double s,r,p,g,f,dd,c,b;
        void nrerror();

        for (i=2;i<=n;i++) e[i-1]=e[i];
        e[n]=0.0;
        for (l=1;l<=n;l++) {
                iter=0;
                do {
                        for (m=l;m<=n-1;m++) {
                                dd=fabs(d[m])+fabs(d[m+1]);
                                if (fabs(e[m])+dd == dd) break;
                        }
                        if (m != l) {
                                if (iter++ == 30)
                                   nrerror("Too many iterations in TQLI");
                                g=(d[l+1]-d[l])/(2.0*e[l]);
```

ED – Technik

EXACT DIAGONALIZATION METHODS FOR QUANTUM SYSTEMS

H.Q. Lin and J.E. Gubernatis

Department Editors: Harvey Gould
hgould@vax.clark.edu

Jan Tobochnik
jant@kzoo.edu

Exact diagonalization methods are important tools for studying the physical properties of quantum many-body systems. These methods typically are used to determine a few of the lowest eigenvalues and eigenvectors of models of many-body sytems on a finite lattice. From these eigenvalues and eigenfunctions, various ground state expectation values and correlation functions are easily computed. Although the methods are limited to small lattice sizes, they have become increasingly popular in the past several years. In addition to providing useful benchmarks for approximate theoretical calculations and quantum Monte Carlo simulations they help to provide insight into the often subtle properties of unsolvable many-body problems in the thermodynamic limit.

In this column, we introduce the Lanczos method[1–3] and a related method, the recursion method.[4,5] To make our discussion concrete, we will use the one-dimensional (1D) Hubbard–Hamiltonian as a working example and provide sufficient detail so that the reader can develop a workable computer code. The ideas and concepts are simple. As we will discuss, the main programming effort is efficiently performing a matrix–vector multiply without storing the matrix. Extensions of the basic ideas and concepts to other many-electron models and to quantum spin models is straightforward.

The 1D Hubbard–Hamiltonian is

$$H = -t\sum_{i,\sigma}(c^{\dagger}_{i,\sigma}c_{i+1,\sigma} + c^{\dagger}_{i+1,\sigma}c_{i,\sigma}) + \frac{1}{2}U\sum_{i,\sigma} n_{i,\sigma}n_{i,-\sigma}, \quad (1)$$

where $c^{\dagger}_{i,\sigma}$ and $c_{i,\sigma}$ are the creation and destruction operators for a fermion at site i with spin σ (up or down) and $n_{i,\sigma} = c^{\dagger}_{i,\sigma}c_{i,\sigma}$ is the fermion number operator at site i for spin σ. The first term in Eq. (1) is the kinetic energy of the electons and describes their hopping, without spin flip, from site to site. The second term is the potential energy (Coulomb interaction) that exists only if two electrons occupy the same site.

Hai Qing Lin is a Visiting Assistant Professor at the Department of Physics, University of Illinois, Urbana, IL 61801. James E. Gubernatis is a Staff Member in the Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545.

A natural orthonormal basis for the model is the occupation number basis that includes states describing all possible distributions of N electrons over the M lattice sites. There are 4 possible electron occupancies at each site (unoccupied, singly occupied for spin up or down, and doubly occupied with one spin up and one spin down). We can represent the up and down spin configurations separately by assigning each lattice site a 1 if the site is occupied or a zero if it is not. An example of a state is this basis that has 5 electrons on an 8 site lattice with 3 up and 2 down spins is

$$|00101010\rangle|00100100\rangle. \quad (2)$$

The up spin electrons are at sites 3, 5, and 7, and the down spin electrons are at sites 3 and 6. The remaining sites are unoccupied.

For M lattice sites, the number of states in the basis is 4^M, which for $M = 16$ equals 4 294 967 296. Thus in this basis, the Hamiltonian matrix has $(4\,294\,967\,296)^2$ elements, i.e., over 10^{19}. Although this matrix is very sparse, the number of nonzero elements is still a very large number, and their storage in a computer's memory is well beyond what is possible. For this reason, the direct numerical solution of eigenvalue problems for quantum many-body systems by conventional dense matrix routines, such as those found in the LAPACK software package,[6] is possible only for lattice sizes up to about 8 sites. Clearly, other methods are needed.

The first step is making the problem more tractable is the use of symmetries to block–diagonalize the Hamiltonian. By similarity transformations, this step produces sequences of smaller matrices along the diagonal. If we want the lowest eigenvalue of the Hamiltonian matrix, i.e., the ground state energy, we simply find the smallest eigenvalue from each of these smaller matrices.

For many-electron models, one of the simplest symmetries is associated with the conservation of the total number of electrons. Using this symmetry, we need only consider the subspace that corresponds to a specified number of electrons for a given number of lattice sites. For a lattice with M sites and N electrons, the number of such states is $(2M)!/N!(2M-N)!$. For $M = 16$ and $N = 16$, the largest block has 601 080 390 states, and hence the number of elements in this block of the Hamiltonian matrix is $(601\,080\,390)^2$, i.e., over 10^{17}.

From the Hubbard–Hamiltonian, Eq. (1), we see that besides the total number of electrons, the number of electrons with a particular spin N_σ also is conserved, because neither term in Eq. (1) changes an up spin to a down spin or vice versa. In condensed matter physics, we typically are interested in systems where N_σ is fixed, so this symmetry is useful and important. Using this symmetry, we produce Hamiltonian blocks of size $\Pi_\sigma M!/N_\sigma!(M-N_\sigma)!$. For $M=16$ with 8 up and 8 down electrons, the largest block size has $(12\,870)^2 = 165\,636\,900$ states. Thus, the Hamiltonian matrix has approximately 2×10^{16} elements, a number that is still too large for most computers.

Another common symmetry is translational invariance. In one dimension with periodic boundary conditions, two states connected by translational symmetry to the one given in Eq. (2) are

$$|00010101\rangle|00010010\rangle \text{ and } |10001010\rangle|00001001\rangle. \quad (3)$$

When translational symmetry is present, the states can be grouped by wave number k, since k is a good quantum number. For the 1D Hubbard model, translational symmetry reduces the block sizes by a factor of M, i.e., by the number of lattice sites, which happens to equal the number of k values.

As an example, if translational symmetry, conservation of N_σ, and the symmetry group of a square are used on a 4×4 lattice with 16 electrons (8 up and 8 down), the dimension of the biggest block of the Hamiltonian matrix is found to be 1 310 242. This number is somewhat larger than what we might expect from simple considerations. *A priori*, we expect translational symmetry to reduce the size of the matrix by a factor of 16, and the point group of the square to reduce the size by a factor of 8, that is, we expect that these two symmetries together to reduce the matrix by a factor of $16\times8=128$. However, $165\,636\,900/128\approx1\,294\,038$ is smaller than what we actually obtain because not all states are affected by both symmetries. Although the actual size of 1 310 242 is much smaller than 4^{16}, $(1\,310\,242)^2$ matrix elements still are too many to store. We note, however, that storing a few vectors of length 1 310 242 is well within the capability of today's largest computers.

Other useful symmetries include particle-hole symmetry and charge conjugation. The use of symmetries, however, can be overdone as a point is easily reached where little is gained in terms of computer memory reduction and much is lost by now having a computer code "hard-wired" for a particular problem. Often, the conservation of N_σ is all that is used.

One might ask, "Why not store the matrix on a disk and read parts of it back as needed?" The difficulty is that even on computers with fast disks, the I/O speed still is too slow compared to the computational speed. To avoid this bottleneck, very fast algorithms for computing matrix elements "on-the-fly" have been developed. These algorithms are highly suitable for vector and parallel computers[7] and make it unnecessary to store the elements.

To see how these algorithms work, we first address the question of how do we represent all the possible configurations on a computer. We observe that each of the possible 4^M states maps uniquely onto an integer I defined by

$$I=\sum_{i=1}^{M}\left[n_\uparrow(i)2^{i-1}+n_\downarrow(i)2^{M+i-1}\right], \quad (4)$$

where $n_\uparrow(i)$ and $n_\downarrow(i)$ are the occupancies of site i for the up and the down spins. The bits of this integer represent a specific state

$$|n_\uparrow(1),n_\uparrow(2),\ldots,n_\uparrow(M)\rangle|n_\downarrow(1),n_\downarrow(2),\ldots,n_\downarrow(M)\rangle. \quad (5)$$

Because we are interested in only those states that correspond to N electrons, we need to set up a one-to-one correspondence between I and a label J that runs from 1 to the number of states with N electrons. With such a label, a table T, defined by $T(J)=I$, compactly expresses the correspondence.

The two-table method of Lin[8] is an efficient and convenient way of establishing this corespondence. In this method, the states of the system are split into two pieces, which, for example, may represent the left- and right-halves of a 1D chain or the "red" and "black" sites of a two-dimensional (2D) checkerboard square lattice. For the Hubbard model, choosing the two pieces to correspond to the up and down spins is especially convenient. For a given value of I, we can write

$$I=I_\uparrow+2^M I_\downarrow, \quad (6)$$

so that the bits of the integers $I_\uparrow$ and $I_\downarrow$ represent the occupancy of the sites for up and down spins. We next define two arrays (tables) $J_\uparrow$ and $J_\downarrow$, so that

$$J=J_\uparrow(I_\uparrow)+J_\downarrow(I_\downarrow). \quad (7)$$

Now, if we know J and properly define $J_\uparrow$, then $J_\downarrow$ is fixed, and we have established a one-to-one correspondence between I and J. Before we define the array $J_\uparrow$, we remark that although each state I is occupied by N electrons, the occupancy of the states $I_\uparrow$ and $I_\downarrow$ varies from 0 to N as I assumes all possible values. For a given $I_\downarrow$, a number of different $I_\uparrow$'s will exist. We label each different states represented by these values of $I_\uparrow$ serially by a number k ($k=1,2,\ldots$), and define the array $J_\uparrow$ by $J_\uparrow(I_\uparrow)=k$. This value of $J_\uparrow$, along with the value of J, fixes $J_\downarrow$. The scheme is illustrated in Table I. Thus, if we know I, we determine $I_\uparrow$ and $I_\downarrow$ by looking at the bits of I. Then, from Eq. (7) we find J. With J, we find I from $T(J)$.

As we discuss below, the Lanczos and recursion methods require the efficient computation of a matrix–vector multiplication. The matrix is the Hamiltonian matrix H and the vectors are linear combinations of the complete set of states $|I\rangle$ described by the bits of the integer I. If $|\psi\rangle=\Sigma_I a(I)|I\rangle$ and $H|\psi\rangle=\Sigma_{I'} b(I')|I'\rangle$ (or equivalently $|\psi\rangle=\Sigma_J a(J)|J\rangle$ and $H|\psi\rangle$

Table I. Electron configurations, their representations $I_\uparrow$ and $I_\downarrow$, and position vectors $J_\uparrow(I_\uparrow)$ and $J_\downarrow(I_\downarrow)$ for a system of 3 electrons and 3 sites.

Up configuration	$I_\uparrow$	$J_\uparrow(I_\uparrow)$	Down configuration	$I_\downarrow$	$J_\downarrow(I_\downarrow)$	$J=J_\uparrow+J_\downarrow$
111	7	1	000	0	0	1
011	3	1	001	1	1	2
101	5	2	001	1	1	3
110	6	3	001	1	1	4
011	3	1	010	2	4	5
101	5	2	010	2	4	6
110	6	3	010	2	4	7
011	3	1	100	4	7	8
101	5	2	100	4	7	9
110	6	3	100	4	7	10
001	1	1	011	3	10	11
010	2	2	011	3	10	12
100	4	3	011	3	10	13
001	1	1	101	5	13	14
010	2	2	101	5	13	15
100	4	3	101	5	13	16
001	1	1	110	6	16	17
010	2	2	110	6	16	18
100	4	3	110	6	16	19
000	0	1	111	7	19	20

$= \Sigma_{J'} b(J')|J'\rangle$), then by the orthonormality of the states, we have

$$b(J') = \sum_J \langle J'|H|J\rangle a(J). \tag{8}$$

Thus, the matrix–vector multiplication problem reduces to the problem of computing the nonzero matrix elements of $\langle J'|H|J\rangle$, or equivalently, the nonzero elements of $\langle I'|H|I\rangle$.

The matrix elements for the Hubbard model separate into matrix elements $\langle I'|K|I\rangle$ for the kinetic energy and matrix elements $\langle I'|V|I\rangle$ for the potential energy. For the kinetic energy, we write

$$\begin{aligned}\langle I'|K|I\rangle &= -t\Delta(I'_\downarrow,I_\downarrow)\sum_{i=1}^{M}\langle I'_\uparrow|(c^\dagger_{i,\uparrow}c_{i+1,\uparrow}+c^\dagger_{i+1,\uparrow}c_{i,\uparrow})|I_\uparrow\rangle \\ &\quad -t\Delta(I'_\uparrow,I_\uparrow)\sum_{i=1}^{M}\langle I'_\downarrow|(c^\dagger_{i,\downarrow}c_{i+1,\downarrow}+c^\dagger_{i+1,\downarrow}c_{i,\downarrow})|I_\downarrow\rangle,\end{aligned} \tag{9}$$

where

$$\Delta(I',I)=\begin{cases}1, & \text{if } I'=I\\ 0, & \text{otherwise}\end{cases}. \tag{10}$$

The effect of the hopping terms $c^\dagger_{i,\uparrow}c_{i+1,\uparrow}+c^\dagger_{i+1,\uparrow}c_{i,\uparrow}$ and $c^\dagger_{i,\downarrow}c_{i+1,\downarrow}+c^\dagger_{i+1,\downarrow}c_{i,\downarrow}$ is to move an electron from site i to $i+1$ or from $i+1$ to i. This action changes the states $|I_\uparrow\rangle$ and $|I_\downarrow\rangle$ to $|I_{\uparrow i}\rangle$ and $|I_{\downarrow i}\rangle$. Thus, if $I_\uparrow$ has its i and $i+1$ bits equal to 10, then $I_{\uparrow i}$ will have these same bits changed to 01. If the bits are 01, they change to 10. (For the bit combinations 00 and 11, hopping is not allowed.) We now rewrite Eq. (10) as

$$\begin{aligned}\langle I'|K|I\rangle = -t\sum_i [&\Delta(I'_\uparrow,I_{\uparrow i})\Delta(I'_\downarrow,I_\downarrow)\\ &+\Delta(I'_\downarrow,I_{\downarrow i})\Delta(I'_\uparrow,I_\uparrow)].\end{aligned} \tag{11}$$

From this expression, we see that for each value of i, there are two values of I' that give nonzero matrix elements. The first value is $I'=I_{\uparrow i}+2^M I_\downarrow$ for which the first term in Eq. (12) is nonzero, and the second value is $I'=I_\uparrow+2^M I_{\downarrow i}$ for which the second term is nonzero.

To find $I_{\uparrow i}$ and $I_{\downarrow i}$, we start by defining a mask equal to 2^i+2^{i+1} so that its only nonzero bits correspond to lattice sites i and $i+1$. To find $I_{\uparrow i}$, we first perform a bitwise and (AND) operation with $I_\uparrow$ and the mask and call the result K. This integer records in its i and $i+1$ bits the occupancies of the up spins. Next, we perform a bitwise exclusive or (XOR) operation with K and the mask and call the result L. The integer L is zero or equal to the mask if hopping between i and $i+1$ is not allowed. If hopping is allowed, its i and $i+1$ bits will be 10 or 01 depending on whether the i and $i+1$ bits of $I_\uparrow$ were 01 or 10. If hopping is allowed, then $I_{\uparrow i}$ is simply $I_\uparrow - K + L$. The subtraction by K removes from $I_\uparrow$ the original occupancy of the i and $i+1$ bits, while the addition of L inserts the new bit configuration. The analogous sequence is performed on $I_\downarrow$ to find $I_{\downarrow i}$, and both sequences are repeated for all values of i.

The evaluation of $\langle I'|V|I\rangle$ is even simpler. We write

$$\langle I'|V|I\rangle = U\Delta(I',I)\sum_i \langle I|n_{i\uparrow}n_{i\downarrow}|I\rangle. \tag{12}$$

From the form of Eq. (12), we see that the state is unchanged, and the computational task is to count the number of doubly occupied sites in $|I\rangle$. To do so, we define a mask equal to $2^i + 2^{M+i}$. If the result of an AND operation on I with this mask equals the mask, then site i is doubly occupied and contributes 1 to the sum. This procedure is repeated for all values of i.

If conservation of N_σ is used, the two-level scheme is still used. All that changes is storing in the look-up table only those values of I that have the correct N_σ. The configurations between the horizontal lines in Table I correspond to the different fixed N_σ cases. To use any one of these blocks, all that is needed is a relabeling of the $J_\downarrow$'s and J's. For example, if we have $N_\uparrow = 2$ and $N_\downarrow = 1$, we resequence J from 2–10 to 1–9, and then change the $J_\downarrow$ so that Eq. (7) is satisfied.

If translational symmetry if used, we can use the sublattice coding scheme of Lin.[8] This method is a bit more difficult to explain and program. The idea is take the two table storage scheme and use translational symmetry to reduce it to a smaller two table scheme. The problem is more complicated than the other cases because the Bloch state represents a linear combination of many-body basis states that is, in general, a complex number because of the e^{ikx} weights. This complication makes it difficult to explain the method succinctly. In many applications, however, the wave number of the ground state is often known beforehand, and we need only consider this one value. To give a flavor of using translational symmetry, we will assume we are interested only in the $k = 0$ state for which the phase factor becomes unity and the Bloch state is a simple linear combination of those states connected by translational symmetry.

We use Table I for illustrative purposes and focus on the configurations in the up configuration table. For a given value of $N_\uparrow$, we assign indices serially to all possible distributions of $N_\uparrow$ electrons among M sites. For $N_\uparrow = 2$ and $M = 3$, these distributions are 011, 101, and 110. For given value of $N_\downarrow$, we determine from all the possible configurations of $N_\downarrow$ among M sites the sets of configurations that are independent under translational symmetry. Then, we choose from each set a representative state and serially label these states. The final step is to associate each of these states with each of the up spin states. For $N_\downarrow = 1$, the only distribution independent under translational symmetry is 001. The result is illustrated in Table II. An important point is that translational symmetry is applied simultaneously to both up and down spins. For $N_\uparrow = 2$ and $N_\downarrow = 1$, translational symmetry generates from each entry in Table II three entries ($J = 2,7,$ and 9) in Table I. This degeneracy leads to an additional requirement that we record and use the weight W of each configuration.

The methods we described are efficient and minimize storage requirements. The specific details of their implementation depend greatly on the architecture of the computer, the bitwise operations in the compiler's library, and the degree of portability desired.[7] On a Cray-2 computer, one Lanczos iteration for a 4×4 Hubbard model with 8 up and 8 down electrons requires only 90 s. Fo a 18 site Hubbard model, the largest size studied by this method, the computation time increases by an order of magnitude.[9]

We now describe the Lanczos method. We will assume in the following that the block of the Hamiltonian matrix H whose eignevalues we want to calculate is called A and is of order n. For the labeling schemes presented above, n equals the maximum value of J. What is needed is a method for determining at least some of the eigenvalues and eigenstates of A without storing A and storing only a few vectors with n components (n vectors). The limiting factor is the amount of memory needed to store the vectors. We will describe a version of the Lanczos algorithm that needs to store only two n vectors, if just a few lowest lying eigenstates are being determined, and three such vectors, if the corresponding eigenvectors also are desired. Storing the matrix elements of A is unnecessary.

The concept of an invariant subspace is important for understanding the Lanczos method.[2] By a subspace, we mean the set of all n vectors that can be written as linear combinations of a set $S = \{s_1, s_2, \ldots, s_m\}$ of n vectors. If a matrix $A = A^T$, which is true in our case, the subspace is said to be invariant under A if for any vector x in the sub-

Table II. Electron configurations, their representations $I_\uparrow$ and $I_\downarrow$ and position vectors $J_\uparrow(I_\uparrow)$ and $J_\downarrow(I_\downarrow)$ for a system of 3 electrons and 3 sites after translational symmetry for k=0 is applied to the configurations in Table I.

Up configuration	$I_\uparrow$	$J_\uparrow(I_\uparrow)$	Down configuration	$I_\downarrow$	$J_\downarrow(I_\downarrow)$	$J = J_\uparrow + J_\downarrow$	W
111	1	1	000	1	0	1	1
011	1	1	001	1	0	1	3
101	2	2	001	1	0	2	3
110	3	3	001	1	0	3	3
001	1	1	011	1	0	1	3
010	2	2	011	1	0	2	3
100	3	3	011	1	0	3	3
000	1	1	111	1	0	1	1

space, the vector Ax also is in the subspace. What does this invariance mean? Any eigenvector y of A, for example, determines an invariant subspace of dimension one because $Ay = \lambda y$, where λ is the eigenvalue, and λy is an obvious linear combination of y. A set of m eigenvectors determines an invariant subspace of dimension m. Thus, any invariant subspace of A is spanned by a set of the eigenvectors of A.

If $Q = \{q_1, q_2, \ldots, q_m\}$ is a basis of an invariant subspace of A, arranged in the form of an $n \times m$ matrix Q whose columns are q_i, then the matrix product AQ is a $n \times m$ matrix whose columns are linear combinations of the columns of Q. Because of the invariance of the subspace, each vector Aq_i is in the subspace. These linear combinations can be expressed as the $m \times n$ matrix QT, where T is an $m \times m$ matrix. Thus, we have

$$AQ = QT. \tag{13}$$

If Q is an orthonormal basis in the subspace, i.e., $Q^T Q = I$, then we can write

$$Q^T A Q = T. \tag{14}$$

What does the relation Eq. (14) do for us? If λ and y are an eigenpair of T,

$$Ty = \lambda y, \tag{15}$$

then multiplying Eq. (15) by Q produces $(QT)y = \lambda(Qy)$. If we use Eq. (13), we find

$$A(Qy) = \lambda(Qy). \tag{16}$$

Equation (16) says that λ and Qy are an eigenvalue and eigenvector of A. Thus, the eigenvalues and eigenvectors of a large matrix A can be found from those of a smaller matrix T, if the space spanned by Q is invariant under A. The Lanczos method generates such an invariant subspace approximately. As a subspace, it uses the Krylov subspace defined by $K = \{b, Ab, A^2 b, \ldots, A^{m-1} b\}$, where b is an arbitrary nonzero vector. The reasoning is roughly as follows: Under the action of A, the vectors $Ab, A^2 b, \ldots, A^m b$ are all in the Krylov space, except for the last vector. It can be shown that $A^{m-1} b$ converges to an eigenvector of A if m is sufficiently large, so that $A^m b$ is approximately proportional to $A^{m-1} b$, and hence is almost in the Krylov space. Thus, the Krylov space for $m < n$ is almost an invariant subspace of A, and to a very good approximation, we can find eigenvalues and eigenvectors of the large matrix A from those of a smaller matrix. Several remarkable properties[2] follow from the selection of K:

- The matrix $T = Q^T A Q$ is a symmetric tridiagonal matrix.
- A three term recursion relation exists for the calculation of the columns of Q.
- The matrix A is needed only to compute matrix–vector multiplications.
- Convergence is fast. (Typically, we need only $m = 30 - 100$ even for a matrix as large as 16 million by 16 million. In some cases, e.g., if all off-diagonal matrix elements have the same sign, convergence is guaranteed.)

Let us see how these properties lead to a practical algorithm. We first want to compute the elements of the matrix,

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{m-1} \\ 0 & \cdots & & \beta_{m-1} & \alpha_m \end{bmatrix}. \tag{17}$$

If we equate columns of $AQ = QT$, we find the three-term recursion relation

$$Aq_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}, \tag{18}$$

where $j = 1$ to $m - 1$, and q_j is the jth column of Q. The procedure begins by setting $\beta_0 = 1$ and $q_0 = 0$. Then we let $q_1 = b$, where b is an arbitrary vector consistent with the symmetry of A. The orthonormality of q_j implies that $\alpha_j = \langle q_j | A q_j \rangle$ and $\beta_{j-1} \langle q_j | A q_{j-1} \rangle = \langle q_{j-1} | A q_j \rangle$. With these values of $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ and $\beta = \{\beta_1, \beta_2, \ldots, \beta_{m-1}\}$, the tridiagonal matrix T is formed, and its eigenvalues λ_i and eigenvectors y_i are found by a standard method.[6] Here, we can use these methods because T is a much smaller matrix than A. The process is repeated until the lowest eigenvalues converge to some desired accuracy. We remark that usually the eigenvalues are first determined; then computer storage is needed for only q_j and q_{j-1}.

If y_k is an eigenvector of T, then to compute the corresponding eigenvector ψ_k of A we see from Eq. (16) that we need to compute $\psi_k = Q y_k$. Because we did not store q_i while generating T, we need to regenerate the q_i by starting with the same b and repeating the Lanczos procedure. Now, in addition to storing q_{j-1} and q_j, we have to store a third n vector, ψ_k.

If this procedure sounds too good to be true, one might ask, "What is the catch?" The difficulty with the Lanczos method is that finite precision arithmetic causes the q_i to lose their orthogonality. One fix is to repeatedly reorthogonalize, another is to partially reorthogonalize, and a third is to ignore the problem.[3,5] We generally choose the latter to avoid the overhead associated with reading from and writing to the disk the long vectors that are needed to reorthogonalize. The cost is that only the extremal (lowest or highest few) eigenvalues can be determined accurately, but these are generally all that were wanted in the first place.

The following pseudocode[3] performs an effective Lanczos method. It assumes the existence of a function `mult(A,q)` that multiplies the matrix A times the vector q and returns the resulting vector. In this pseudocode, α and β are vectors of the order of the maximum number of Lanczos steps planned, and b and q are n vectors. The

components of the vectors are indicated by a subscript i or j; t is a scalar. The symbol $\|q\|$ is a vector norm, usually taken to be the L_2-norm $\sqrt{q^T q}$. The vector b is an arbitrary nonzero vector consistent with the symmetry of A,

```
q(1:n) = 0; β0 = 1; j = 0
while βj ≠ 0
    if j ≠ 0
        for i = 1:n
            t = bi; bi = qi/βj; qi = −βj t
        end
    end
    q = q + mult(A,b)
    j = j + 1; αj = b^T q; q = q − αj b; βj = ||q||
end
```
(19)

How is this code used? After the mth pass through the **while** loop, the accrued components of α and β define a tridiagonal matrix of order m. A conventional eigenvalue routine is used to find the eigenvalues of this matrix. If $\lambda_0(m)$ is the lowest eigenvalue at step m, the procedure is repeated until $|\lambda_0(m) - \lambda_0(m-1)|/|\lambda_0(m)| < \epsilon$, where ϵ is a small number, say 10^{-10}.

Orthogonality breaks down in β_j becomes small. If $\beta_j = 0$, then we have reduced the tridiagonal matrix T to at least two tridiagonal parts, i.e., we have produced vectors q_i that define an invariant subspace of A. Since the method is supposed to produce at least approximately such a subspace, convergence of the method and the loss of orthonormality seem to go hand-in-hand. Fortunately, experience has shown that the convergence to the extermal eigenvalues (largest and smallest) is very rapid and the method can be used without reorthogonalization to determine them. Ignoring the loss of orthogonality is not as cavalier as it sounds.

Mathematicians have studied several versions of the Lanczos method and have developed its relation to variational principles and the method of moments. We have presented a version suggested by Paige,[10] who studied four versions and discovered that two of these versions, including the one described above, are numerically more stable and converge faster than the other two. There are other versions of the Lanczos method that require storing three vectors even for the determination of the eigenvalues. We are uncertain of their advantages.

Usually, the eigenpair corresponding to the lowest energy is all that is determined. Many properties of the ground state can easily be determined knowing this eigenpair. For example, if $|\psi_0\rangle$ is the ground state wave function, then the spin and charge density wave correlations between sites of separation j are given by

$$\chi_\pm(j) = \frac{1}{N}\sum_i \langle\psi_0|(n_{i,\uparrow} \pm n_{i,\downarrow}) \times (n_{i+j,\uparrow} \pm n_{i+j,\downarrow})|\psi_0\rangle, \tag{20}$$

where the plus sign refers to the charge density wave. Similarly, superconducting pairing correlations between sites i and j are calculated by[11]

$$\chi_s(i-j) = \langle\psi_0|\Delta_i \Delta_j^\dagger|\psi_0\rangle, \tag{21}$$

where

$$\Delta_i = \frac{1}{N}\sum_j c_{j,\uparrow} c_{j+i,\downarrow}. \tag{22}$$

Perhaps one of the more interesting things that can be done with the ground-state energy λ_0 and wave function $|\psi_0\rangle$ is to compute dynamical properties of the system by the recursion method.[4,5,12] The type of quantity calculated is a spectral function defined as

$$I(\omega) = \sum_m |\langle\psi_m|B|\psi_0\rangle|^2 \delta(\omega + \lambda_0 - \lambda_m), \tag{23}$$

where B represents some perturbation of the ground state. For example, if B is the current operator or the spin-lowering operator, then $I(\omega)$ is the optical conductivity or the dynamical susceptibility. The action of B on $|\psi_0\rangle$ is to create a new state from the ground state. The expression $|\langle\psi_m|B|\psi_0\rangle|^2$ is the fraction of this new state that is in an excited state $|\psi_m\rangle$ of the Hamiltonian. The δ function expresses conservation of energy. Using the standard relation

$$\delta(x - x') = -\frac{i}{\pi}\,\mathrm{Im}\lim_{\epsilon\to 0}\frac{1}{x - x' + i\epsilon}, \tag{24}$$

and the resolution of unity

$$1 = \sum_m |\psi_m\rangle\langle\psi_m|, \tag{25}$$

we can rewrite Eq. (23) as

$$I(\omega) = -\frac{1}{\pi}\,\mathrm{Im}\lim_{\epsilon\to 0}\langle\psi_0|B^\dagger \frac{1}{zI - H} B|\psi_0\rangle, \tag{26}$$

where $z = (\omega + \lambda_0 + i\epsilon)$. In this form, what must be computed is a specific element of the inverse of the matrix $zI - H$. What the recursion method does is to use the Lanczos method with $B|\psi_0\rangle$ as the initial state to tridiagonalize H. In this simple form, the matrix inverse of $zI - H$ for the specific element is readily obtained. Of course, the procedure is not used on the entire Hamiltonian matrix H. Usually, it is used only on the block that contains the ground state.

The spectral function $I(\omega)$ also can be expressed as a continued fraction[4,5,12]

$$I(\omega) = -\frac{1}{\pi}\,\mathrm{Im}\lim_{\epsilon\to 0}\cfrac{1}{z - \alpha_1 - \cfrac{\beta_1^2}{z - \alpha_2 - \ddots}}. \tag{27}$$

One way to evaluate the continued fraction is by "brute force." Another way is to observe the connection between a continued fraction and a "stair-case" Padé approximant[13] and to use a simple set of recursion equations to

evaluate the approximant. A third way is to use the eigenvalues and eigenvectors of T and evaluate Eq. (23) in this diagonal basis

$$I(\omega) = \lim_{\epsilon \to 0} \sum_m |\langle y_m | B | y_0 \rangle|^2 \frac{\epsilon}{\epsilon^2 + (\omega + \lambda_0 - \lambda_m)^2} . \tag{28}$$

One advantage of the recursion method is that we only collect states that are connected to the ground state by the operators whose spectrum we wish to calculate. This fact saves much computer time. The major disadvantage is that the finite size of the system causes the spectral functions to be a sum of δ functions. Some smoothing of the spectrum is achieved by having ϵ remain a small finite value in the range 10^{-3}–10^{-6}. Other methods also have been proposed.[14]

To conclude, we emphasize that the exact diagonalization method is an important, frequently used tool in theoretical studies of many-body problems. The method has been used, for example, to verify the existence of magnetic long-range order in the 2D Heisenberg model and to support the use of the 2D antiferromagnetic Heisenberg model to describe some properties of the newly discovered high-T_c superconductors.[15] In contrast to quantum Monte Carlo methods, which frequently suffer from the "sign" problem,[16] the exact diagonalization method gives exact answers for many-body models on finite lattices. Hence, the method is a means of studying the applicability of theoretical models to experiments, as well as the validity of approximations used in analytical methods. Often, the results of the exact diagonalization method point to parameter regimes where interesting physics may occur.

The main limitation of this method is its restriction to small lattices, and thus properties in the thermodynamic limit are difficult to obtain. Sometimes, however, this limitation is only an apparent one. The results for the small lattice might meaningfully represent those of the large systems because many-body interactions are short ranged and can lead to phenomena with short coherence lengths. If this length is smaller than the lattice size accessible by the exact diagonalization method, physically meaningful results are obtained. This situation frequently arises for 1D systems. In other cases, an extrapolation of results for finite sizes to infinite system sizes is possible. Such extrapolation processes were used, for example, in the exact diagonalization calculations of the staggered magnetization of two 2D Heisenberg models.[15] The exact diagonalization method is not a replacement for analytical analysis; rather, it is a complementary part of it.

Suggestions for further study

1. Investigate the implementation of the Lanczos method to the spinless fermion[17] and Heisenberg models.[8] What symmetries are useful? Are there exploitable differences between the two models that makes one easier to implement than the other? Do the details of the two-table and sublattice coding schemes change? If so, how?

2. Quantum chemists often use the Davidson method[18] to find the lowest eigenvalues of a large matrix. Investigate its algorithmic structure. What are its advantages and disadvantages in comparison to the version of the Lanczos method described in this column?

3. Investigate more fully the implementation of the recursion method and derive Eq. (27). In deriving Eq. (27), consider the following approach: Replace the α_i in the tridiagonal matrix T in Eq. (17) with $z - \alpha_i$ and call the resulting matrix S_m. Then express S_m as the block matrix

$$S_m = \begin{bmatrix} a_1 & b_1^T \\ b_1 & S_{m-1} \end{bmatrix}, \tag{29}$$

where $a_1 = z - \alpha_1$, b_1 is a column vector of length $m-1$ with elements $\{\beta_1, 0, \ldots, 0\}$, and S_{m-1} is the $(m-1)\times(m-1)$ matrix formed from S_m by deleting its first row and column. Block invert S_m and show that the 11 element of the inverse is

$$(S_m^{-1})_{11} = \frac{1}{a_1 - \langle b_1 | \dfrac{1}{S_{m-1}} | b_1 \rangle}, \tag{30}$$

which equals $[a_1 - \beta_1^2 (S_{m-1}^{-1})_{11}]^{-1}$. To find the 11 element of S_{m-1}^{-1}, block S_{m-1}, and then block invert it. Repeating this procedure until only the inversion of S_1 is

Measurement Errors
Theory and Practice
Semyon Rabinovich

This volume offers practical recommendations and procedures for problems related to the estimation of measurement errors. The author covers a wide range of subjects, including classical concepts of metrology, modern problems of instrument calibration, estimation of single and multiple measurement errors, and modern probability-based methods of error estimation. A valuable resource for graduate students, applied physicists, and engineers.

284 pages, cloth, ISBN 0-88318-866-X
$100.00 (Member price $80.00)
Please indicate your AIP Member Society when ordering.

To order, call 1-800-488-BOOK
In Vermont: 1-802-878-0315. Fax: 1-802-878-1102
Or mail check, MO, or PO (plus $2.75 for shipping) to:

AMERICAN INSTITUTE OF PHYSICS
American Institute of Physics
c/o AIDC, 64 Depot Road
Colchester, VT 05446

left produces the continued fraction. Are the Padé and diagonal-space methods more stable or efficient than the brute force method for evaluating the continued fraction?

4. The recursion method also can be used to perform perturbation theory. For $H = H_0 + V$, where $H_0|\phi_0\rangle = E_0|\phi_0\rangle$, the idea is to compute the Green's function $(zI - H)^{-1}$, where $z = E + i\epsilon$ and obtain the continued fraction. Because the poles of the Green's function are the eigenvalues of the Hamiltonian, the poles of the continued fraction are approximations to these eigenvalues. To develop these approximations, one takes unperturbed state $|\phi_0\rangle$, instead of $B|\psi_0\rangle$, as the initial state in the Lanczos step to tridiagonalize H. Work through this procedure analytically to second order and compare the result with second-order Brillouin–Wigner perturbation theory.

Acknowledgments

This work was supported by the U.S. Department of Energy and the Department of Physics at the University of Illinois. We also thank the editors, Jan Tobochnik and Harvey Gould, for helpful comments and suggestions about the manuscript.

References

1. C. Lanczos, J. Res. Natl. Bur. Stand. **45**, 225 (1950).
2. S. Pissanetsky, *Sparse Matrix Technology* (Academic Press, New York, 1984), Chap. 6.
3. G. H. Goulb and C. F. van Horn, *Matrix Computations* (Johns Hopkins Press, Baltimore, 1989), Chap. 9.
4. R. Haydock, in *Solid State Physics* **35**, edited by H. Ehrenreich, F. Seitz, and D. Turnbull (Acadmeic, New York, 1980), p. 215.
5. R. E. Wyatt in Adv. Chem. Phys. **LXXII**, 231 (1988).
6. LAPACK Users' Guide, edited by E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarrra, J. Du Cruz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostouchov, and D. Sorenson, SIAM, Philadelphia (1992).
7. P. W. Leung and P. E. Oppenheimer, Comp. Phys. **6**, 603 (1993).
8. H. Q. Lin, Phys. Rev. B **42**, 6561 (1990).
9. H. Q. Lin, Phys. Rev. B **44**, 7153 (1991).
10. C. C. Paige, J. Inst. Math. Appl. **10**, 373 (1972).
11. For example, see H. Q. Lin, J. E. Hirsch, and D. J. Scalapino, Phys. Rev. B **37**, 7359 (1988).
12. E. R. Gagliano and C. A. Balserio, Phys. Rev. Lett. **59**, 2999 (1987).
13. G. A. Baker, Jr., *Essentials of Padé Approximants* (Academic, New York, 1975), Chap. 4.
14. E. Y. Loh, Jr. and D. K. Campbell, Synth. Metals **27**, A499 (1988).
15. For a recent general survey, see E. Manousakis, Rev. Mod. Phys. **63**, 1 (1991).
16. For example, see E. Y. Loh, Jr., J. E. Gubernatis, R. T. Scalettar, S. R. White, D. J. Scalapino, and R. L. Sugar, Phys. Rev. B **41**, 9301 (1990), and references within.
17. W. R. Somsky and J. E. Gubernatis, Comput. Phys. **6**, 178 (1972).
18. E. R. Davidson, J. Comp. Phys. **17**, 87 (1975).

From the editors. We appreciate your feedback and encourage your contributions to this column. A copy of the guidelines are available from the editors. Please send comments and suggestions for future columns to hgould@vax.clarku.edu or jant@kzoo.edu.

GET THE
PHYS
ADVANTAGE

AN ONLINE BIBLIOGRAPHIC DATABASE FOR PHYSICISTS AND ASTRONOMERS

PHYS (Physics Briefs) is an online database—available exclusively on STN International—that enables you to search through physics literature worldwide to obtain English-language citations. Abstracts, bibliographic data, extensive index entries—PHYS gives you access to more than 1.4 million citations in all.

You will find citations from 1979 onward—information drawn from more than 2,800 scientific and technical journals, books, reports, conference proceedings, patents, and nonconventional literature (e.g., dissertations and corporate publications). The PHYS database is updated twice monthly with 120,000 new records added each year. Abstracts are often available simultaneously with publication of the articles.

Major advantages of PHYS include:

Immediacy—30% of all journal citations are available within one month of publication

Comprehensiveness—significant coverage of monographs and reports; strong emphasis on Eastern European literature; indexing of all astronomical objects

Searchability—editors and authors are searchable, making citations easier to locate—especially helpful for conference proceedings literature

Flexible Pricing—choose either a discounted rate or an annual flat fee.

Act Now! For more information on the new academic pricing programs for PHYS contact the American Institute of Physics, Electronic Publishing Division.

AMERICAN INSTITUTE OF PHYSICS

Electronic Publishing
500 Sunnyside Boulevard
Woodbury, NY 11797-2999
(516) 576-2262/2264
E-mail: ellen@pinet.aip.org

PHYS is a cooperative effort of Fachinformationszentrum Energie, Physik, Mathematik and the American Institute of Physics.