Computergestütztes wissenschaftliches Rechnen
SoSe 2004

Alexander K. Hartmann, Universität Göttingen

2. Juli 2004

8.3 Simulation der Boltzmann Verteilung

Statistische Physik definiert durch Boltzmann Verteilung $P(y) \sim e^{-H(y)/k_BT}$, $H(y)$= Energie von Zustand y, T= Temperatur.

Aus der detaillierten Ballance

$$W_{zy}P(z) - W_{yz}P(y) = 0 \quad \forall y, z \tag{1}$$

folgt:

$$W_{zy} = W_{yz}e^{-H(y)/k_BT}/e^{-H(z)/k_BT} = W_{yz}e^{-[H(y)-H(z)]/k_BT}$$

Dabei können viele $W_{zy} = 0$ sein (Übergänge nur zwischen "Nachbarn").

Eine (von vielen) Möglichkeiten, (1) zu erfüllen, ist:

$$W_{zy} = 1/\left[1 + e^{[H(y)-H(z)]/k_BT}\right] \tag{2}$$

denn es gilt

$$\begin{aligned} W_{yz}e^{-[H(y)-H(z)]/k_BT} &= \frac{1}{1 + e^{[H(z)-H(y)]/k_BT}}e^{-[H(y)-H(z)]/k_BT} \\ &= \frac{1}{e^{[H(y)-H(z)]/k_BT} + 1} = W_{zy} \end{aligned} \tag{3}$$

Eine andere bekannte Wahl ist

$$W_{zy} = \begin{cases} 1 & \text{if} H(y) - H(z) < 0 \\ e^{-[H(y)-H(z)]/k_BT} & \text{else} \end{cases}$$

(Metropolis Kriterium).

Realisierung in der Praxis: Man generiert aus $y(t)$ einen *Testzustand* $y' \neq y$, wobei alle Zustände mit $W(y(t)|y') > 0$ gleichwahrscheinlich sind. Dann wird y' als nächster Zustand mit der Wahrscheinlichkeit $W(y(t)|y') > 0$ akzeptiert, also $y(t+1) = y'$ und ansonsten bleibt $y(t+1) = y(t)$.

Hinweis/Ausblick: Falls die Übergangraten W_{yz} klein sind (also W_{yy} groß), ist es effizienter für jeden möglichen Übergang ein Zeit $t'(y, y')$ auszuwürfeln, wo der Übergang stattfindet, und dann als nächstes Ereignis, also das mit der kleinsten Übergangszeit, zu nehmen (analog zur Ereignisgesteuerten Simulation, hier genannt "Waiting Time Method") und direkt die Uhr von t zu $\min t'$ vorzustellen.

8.4 Oberflächendiffusion

Menge aller Zustände y der diskreten Oberfläche ist die Menge aller $y = \{h(\underline{x})\}$, also die Menge aller Höhenprofile (nicht endlich, s.u.).

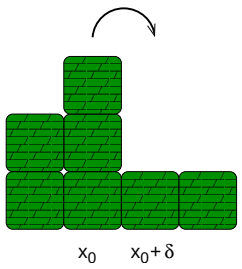
Oberflächen werden durch Diffusion aufgerauht $\rightarrow$ glatte Oberflächen bei $T = 0$.
Ein Modell für Oberflächendiffusion: Hamiltonian

$$H(\{h(\underline{x})\}) := \sum_{\langle \underline{x}, \underline{x}+\underline{\delta} \rangle} \Delta h(\underline{x}, \underline{x}+\underline{\delta})^2 := \sum_{\langle \underline{x}, \underline{x}+\underline{\delta} \rangle} (h(\underline{x}) - h(\underline{x}+\underline{\delta}))^2$$

Summe $\langle \underline{x}, \underline{x}+\underline{\delta} \rangle$ ist über alle Paare von benachbarten Gitterpläze. Für jede Stufe liefert H einen Beitrag. (H hat Energieminimum $H = 0$ bei $h(\underline{x})$=const.)

Hier:
1) Alle Zustände $\{h(\underline{x}) + K\}$, $K = \ldots, -2, -1, 0, 1, 2, \ldots$ sind äquivalent zueinander.
2) Zustände mit $\Delta h \rightarrow \infty$ sind exponentiell unwahrscheinlich
$\rightarrow$ man kann in der Praxis (endliche Simulationszeiten) annehmen, dass man das System mittels einer Markovkette simulieren kann.

Monte Carlo Move:
Zustand $y(t) = \{h(\underline{x})\}$ gegeben.
Als Nachbarzustand wird genommen:
ein Teilchen ist von einer Position $\underline{x}_0$ zu einer Nachbarposition $\underline{x}_0 + \underline{\delta}$ gehüpft (z.B. $\delta = \pm 1$ in einer Dimension)



Also formal $y' = \{h'(\underline{x})\}$ mit

$$h'(\underline{x}) = \begin{cases} h(\underline{x}) - 1 & \text{für } \underline{x} = \underline{x}_0 \\ h(\underline{x}) + 1 & \text{für } \underline{x} = \underline{x}_0 + \underline{\delta} \\ h(\underline{x}) & \text{sonst} \end{cases}$$

Alle Nachbarzustände gleichwahrscheinlich $\rightarrow$ $\underline{x}_0$ und $\underline{\delta}$ werden gleichwahrscheinlich ausgelost.
Energieunterschied:

$$\begin{aligned} \Delta H &= H(\{h'(\underline{x})\}) - H(\{h'(\underline{x})\}) \\ &= (h'(\underline{x}_0) - h'(\underline{x}_0 \pm \underline{\delta}))^2 - (h(\underline{x}_0) - h(\underline{x}_0 \pm \underline{\delta}))^2 \\ &= (h(\underline{x}_0) - 1 - (h(\underline{x}_0 \pm \underline{\delta}) + 1))^2 - (h(\underline{x}_0) - h(\underline{x}_0 \pm \underline{\delta}))^2 = \ldots \end{aligned}$$

Implementierung:

```
/******************** diffusion() ******************/
/** Performs one Monte Carlo sweep for diffusion    **/
/** on 1d surface given by 'h'                      **/
/** Uses periodic boundary conditions (pbc)         **/
/** Activated diffusion is used, see M. Siegert and **/
/** M. Plischke, Phys. Rev. E 50, 917 (1994)        **/
/** PARAMETERS: (*)= return-parameter               **/
/**       glob: global data                         **/
/**     (*)  h: height field                        **/
/** RETURNS:                                        **/
/**    nothing                                      **/
/***************************************************/
void diffusion(system_t *glob, int *h)
{
  int pos;          /* position on x-axis of surface */
  int neighb;       /* position of neighbour         */
  int step;
  int energy_old, energy_new;
  double prob;            /* Monte Carlo probability */

  for(step=0; step<glob->L; step++)     /* one sweep */
  {
    pos = (int) floor(glob->L*drand48());
    if(drand48() < 0.5)           /* choose randomly */
    {
      if(pos==0)              /* left neighbour, pbc */
        neighb = glob->L - 1;
      else
        neighb = pos - 1;
    }
    else
    {
      if(pos==(glob->L-1))   /* right neighbour, pbc */
      neighb = 0;
    else
      neighb = pos + 1;
    }
    energy_old = (h[pos]-h[neighb])*(h[pos]-h[neighb]);
    energy_new = (h[pos]-h[neighb]-2)*(h[pos]-h[neighb]-2);
    prob = 1.0/(1.0+exp((energy_new-energy_old)/glob->T));
    if(drand48() < prob)            /* accept move ? */
    {
      h[pos] -= 1;      /* particle jumps from 'pos' */
      h[neighb] += 1;                /* to 'neighb' */
    }
  }
}
```

Zufallsdeposition zusammen mit Diffusion: Man findet eine geringere Rauhigkeit, $w(t) \sim t^{\beta}$ mit $\beta \approx 0.25$:

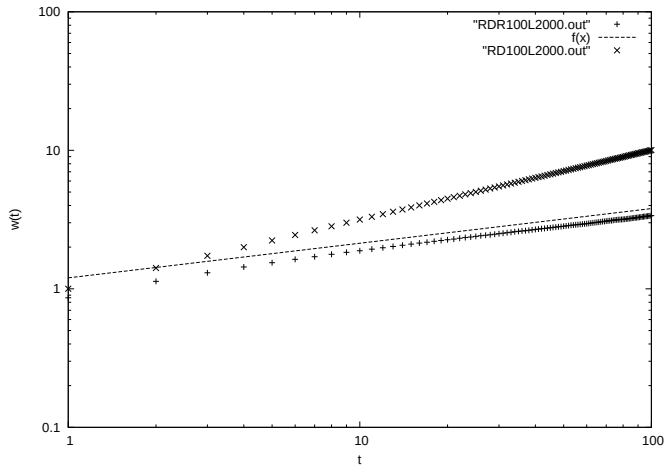


Abbildung 1: Rauhigkeit w als Funktion der Zeit t für Zufallsdeposition + Diffusion ($T = 0.2$, 1Diffusionssweep pro Depositionssweep), gemittelt über 100 Läufe, Systemgröße $L = 2000$. Die Gerade stellt zum Vergleich die Funktion $1.2t^{0.25}$ dar.

9 Datenanalyse

9.1 Daten fitten

Fitten = eine parametrisierte Funktion an Datenpunkte anpassen, indem die Parameter geändert werden.

Hier: Mit `gnuplot`:
Beispiel Fitte Funktion $w(t) = at^\beta$ an Rauhigkeit als Funktion der Zeit für Zufallsdeposition mit Diffusion.

```
gnuplot> f(x)=a*x**b
gnuplot> a=1
gnuplot> b=1
```

Fit erfolgt mit Marquardt-Levenberg Algorithmus [1] Algorithmus. Grundidee: Minimiere gesamten quadratischen Abstand aller Messpunkte (x_i, y_i) von der Kurve

$$\chi^2 = \sum_i (w(x_i) - y_i)^2 \tag{4}$$

Gnuplot Befehl: `fit`. Man muß die Funktion, den Datensatz und die zu ändernden Parameter angeben, hier:

```
gnuplot> fit f(x) "RDR100L2000.out" via a,b
```

Gnuplot gibt Log Informationen (hier nicht dargestellt) und das Ergebnis an:

```
After 8 iterations the fit converged.
final sum of squares of residuals : 0.0677055
rel. change during last iteration : -1.61691e-08

degrees of freedom (ndf) : 98
rms of residuals      (stdfit) = sqrt(WSSR/ndf)      : 0.0262845
```

```
variance of residuals (reduced chisquare) = WSSR/ndf : 0.000690873

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 1.03759          +/- 0.006054     (0.5834%)
b               = 0.256376         +/- 0.001456     (0.568%)


correlation matrix of the fit parameters:

               a      b
a               1.000
b              -0.987  1.000
```

Die Qualität des Fits, wie gut also die Daten zu der gegebenen Funktion passen, kann man aus den 3 Zeilen nach "`degree of freedom`" sehen:

- *Degrees of freedom* = Anzahl der Datenpunkte minus Zahl der Parameters des Fits.
- `WSSR` is χ^2 aus (4).
- Daraus kann man die *Qualität* Q berechnen := Wahrscheinlichkeit, dass der Wert von χ^2 schlechter ist als beim jetzigen Fit, unter der Vorraussetzung, dass die Datenpunkte y_i mit Mittelwert $w(x_i)$ und Varianz 1 verteilt sind. Je größer Q ist, desto besser ist die Qualität des Fits. Um Q auszurechnen, kann man das kleine Programm `Q.c` benutzen.

  ```
  #include <stdio.h>
  #include "nr.h"
  int main(int argc, char **argv)
  {
    float ndf, chi2_per_df;
    sscanf(argv[1], "%f", &ndf);
    sscanf(argv[2], "%f", &chi2_per_df);
    printf("Q=%e\n", gammq(0.5*ndf, 0.5*ndf*chi2_per_df));
    return(0);
  }
  ```

  Es verwendet `gammaq` aus den *Numerical Recipes* [1]. Es wird in der Form `Q <ndf> <WSSR/ndf>`. aufgerufen.

Anmerkung 1:
Die Konvergenz hängt von den Startwerten ab, der Algorithmus kann in lokalen Minima gefangen sein, besonders bei vielen Parametern.

Anmerkung 2:
Genauer ist es, wenn man bei Fits noch die Fehlerbalken ($=\sqrt{\mathrm{Var(y_i)}/(M-1)}$, $\mathrm{Var}(y_i)$=Varianz des Wertes y_i aus M Messungen, siehe Anfängerpraktikum) der y-Werte mit berücksichtigt. Dann muss man die Spalten, wo x-Werte x_i, y-Werte y_i und Fehler σ_i stehen, explizit angeben, z.B.:

```
gnuplot> fit f(x) "RDR100L2000.out" using 1:2:3 via a,b
```

Jeder Datenpunkt geht dann mit einem Gewicht umgekehrt in den Fit ein, und die Abweichung wird relativ zu σ_i gemessen:

$$\chi^2 = \sum_i \left[\frac{w(x_i) - y_i}{\sigma_i} \right]^2 . \tag{5}$$

Ergebnis des Fits:

```
After 4 iterations the fit converged.
final sum of squares of residuals : 10113.5
rel. change during last iteration : -1.05911e-07

degrees of freedom (ndf) : 98
rms of residuals      (stdfit) = sqrt(WSSR/ndf)      : 10.1587
variance of residuals (reduced chisquare) = WSSR/ndf : 103.199

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 0.966968         +/- 0.008133     (0.8411%)
b               = 0.275605         +/- 0.002391     (0.8674%)


correlation matrix of the fit parameters:

               a      b
a               1.000
b              -0.946  1.000
```

Die Qualität des Fits ist nicht gut (WSSR/ndf=100, gut wäre so um 1) →

Anmerkung 3:
Man kann auch den Bereich angeben, wo gefittet werden soll. Für kleine Zeiten passt das Verhalten hier nicht, daher Fit nur für größere:

```
gnuplot> fit [10:100] f(x) "RDR100L2000.out" u 1:2:3 via a,b
```

(ergibt WSSR/ndf : 0.808035 und Kurve passt auch gut, wenn man sie mit `plot` plottet.)

Anmerkung 4:
Auch das Programm `xmgrace` macht nicht-lineare Fits, aber es berechnet keine Fehlerbalken für die Parameter.

Literatur

[1] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C* (Cambridge University Press, Cambridge 1995)