Das Ising-Modell des Ferromagneten

Ender M. Akçöltekin*

18.03.2005

Zusammenfassung

In dieser Arbeit wird eine Anwendung der Monte Carlo Methode aus dem Bereich der statistischen Physik vorgestellt.Dazu wird konkret ein leicht verständliches Standardmodell diskutiert, welches anschließend mit Hilfe der vorgestellten Konzepte als Java-Applet implementiert wird.

*ender.akcoeltekin@uni-essen.de

Inhaltsverzeichnis

1 Etwas zur Physik **3**
1.1 Ferromagnetismus 3
1.2 Das Ising-Modell 4

2 Implementierung des Ising-Modells **5**
2.1 Vorüberlegung 5

3 Monte Carlo-Simulation **6**
3.1 Metropolis-Algorithmus 6
3.2 Programmierung der Monte Carlo-Simulation 6

4 Ergebnis **10**

5 Anhang **11**

1 Etwas zur Physik

1.1 Ferromagnetismus

Um den Ferromagnetismus genau zu erklären, bedarf es eigentlich der Quantenphysik. Aber für eine kurze anschauliche Beschreibung genügt es, wenn man sich auf das halbklassische Bild der Atome, in dem sich die Elektronen auf bestimmten Bahnen um die Kerne bewegen, beschränken. Elektronen besitzen also einen Bahndrehimpuls, zu dem auch ein *magnetisches Bahnmoment* gehört. Außerdem drehen sich die Elektronen auf ihren Bahnen auch noch um ihre eigene Achse. Diese Drehung bezeichnet man als *Spin* und dieser Spin erzeugt infolge eines Kreisstroms auch einen Moment, das man als *magnetisches Spinmoment* bezeichnet. Wenn nun die Elektronenspins eines Atoms mit denen von benachbarten Atomen wechselwirken, so bewirkt das eine Ausrichtung der *magnetischen Dipolmomente* der verschiedenen Atome. Ein magnetisches Dipolmoment sei in userem Bild einfach nur ein Vektor, der eine resultierende Magnetisierungsrichtug besitzt. Ist diese in eine bestimmte Richtung dauerhaft ausgerichtet, liegt eiene ferromagnetische Substanz vor. Reines Eisen, Cobalt oder Nickel zeigen Ferromagnetismus. Im Alltag begegnet man ihnen z.B an Pinwänden. Sind jedoch die Ausrichtung der magnetischen Dipolmomente zufällig orientiert, also keine bestimmte Richtung dauerhaft aufweist, hat man es mit *Paramagnetismus* zu tun.Dazu sei nur erwähnt, daß paramagnetische Substanzen nur schwach und temporär magnetisch sind.

Die besondere Eigenschaft eines ferromagnetischen Materials ist nun, daß es bei Erhitzung bis über eine kritische Temperatur T_c , der sog. Curie-Temperatur, plötzlich seine permanente Magnetisierung verliert und nur noch paramagnetisch wirkt. Je nach Material kann dieser *Phasenübergang*, für unterschiedliche Temperaturen eintreten. Für Eisen ist es z.B bei 750 ℃ oder für Gondolinium schon bei 15 ℃. Der Phasenübergang ist vergleichbar mit Wasser,das bei seinem Siedepunkt verdampft. Die Ursache für den Phasenübergang beruht auf einem quantenmechanischen Effekt[1], der in dieser Arbeit nicht näher beschrieben werden kann.

[1]Dieser Effekt wird als Austauschkopplung bezeichnet.Interessierte Leser können mehr dazu in [1] nachlesen

1.2 Das Ising-Modell

Das Ising-Modell[2] ist ein einfaches theoretisches Modell für die Beschreibung eines ferromagnetischen Materials. Einfachheithalber betrachtet man dazu ein Metall[3], das aus einem quadratischen Gitter besteht und auf dessen Gitterplätzen L magnetische Spinmomente S_i sitzen. Jedes dieser Spinmomente kann zwei Werte $S_i \in \{-1, +1\}$ annehmen.Man sagt dazu auch *Spin up* $\uparrow$ für $+1$ und *Spin down* $\downarrow$ für -1. Wenn das Gitter komplett mit Spins besetzt ist und eine gegebene Magnetisierungsrichtung besitzt, nennt man dies *Konfiguration* $\vec{S} = S_1, S_2, \ldots, S_N$. Für die Energie H einer Spinkonfiguration gilt

$$H = -J \sum_{(i,j)_{nn}} S_i S j - h \sum_i S_i \tag{1}$$

Dabei bezeichnet der Summationsindex $(i, j)_{nn}$, die Summation über alle Paare von Gitterplätzen, die direkt benachbart (nearest neighbour)sind. J ist ein Maß für die Wechselwirkung zwischen benachbarten Spins und h sei ein externes Magnetfeld. Der linke Term der Gleichung(1) besagt nun, daß es energetisch günstiger ist, wenn zwei benachbarte Spins die gleiche Orientierung ($\uparrow\uparrow$ oder $\downarrow\downarrow$) haben. Aus dem rechten Term der Gleichung, kann man zusätlich Energie gewinnen, falls die Spins S_i parallel zu einem äußeren Magnetfeld h (mit $-h < 0$)ausgerichtet sind. Die Wechselwirkung zwischen den Spins im Gitter setzt jede Menge Wärme frei. Daher benötigt man neben der Energie noch eine Zustandsgröße, die alle thermischen Eigenschaften enthält

$$Z = \sum_{\vec{S}} e^{-H(\vec{S})/k_B T} \tag{2}$$

Z bezeichnet man als *Zustandsumme*, die im Prinzip das Wärmegleichgewicht von wechselwirkenden Teilchen beschreibt. Neben der Energie für jede Spinkonfiguration $\vec{S}$, enthält die Gleichung (2) noch die Temperatur in Kelvin gemessen und die Boltzmann-Konstante mit dem Wert $k_B = 1,380650 \times 10^{-23} JK^{-1}$. Die Summe wird nun über alle Spinkonfigurationen $\vec{S}$ gebildet und jeder Zustand wird entsprechend seiner Energie gewichtet.

[2]Ernst Ising (†1998) war Physiker in Peoria/IL(USA)und das nach ihm benannte Modell gilt in der statistischen Physik als Standardmodell

[3]Idealisierte metallische Festkörper bestehen aus kubischen Gittermodellen

2 Implementierung des Ising-Modells

2.1 Vorüberlegung

Bevor man mit der Implemetierung fortsetzt, muß man einige wichtige Punkte klären. Im Prinzip kann man mit der Zustandsumme jetzt quasi den Zustand für jede Spinkonfiguration simulieren, wenn da nicht die folgenden Probleme wären:

1. **Rechenzeit**: Wenn man ein reales Material simulieren will, braucht man auf dem Gitter ($N \times N = L^2$) etwa $10^7 \times 10^7$ Spinmomente mit zwei Einstellungsmöglichkeiten. Das heißt, wir haben 2^N verschiedene Spinkonfigurationen. Wenn man für N Spins die Zustandsumme ausrechnen müßte, ergebe dies für ein reales Material eine astronomische Zeit.

$$2N\ 2^N = \ Zeit \tag{3}$$

   Um dieses Problem zu lösen, hat man sich das Konzept des *detailed balance* überlegt.Dazu wird eine Übergangswahrscheinlichkeit $W(\vec{S} \rightarrow \vec{S'})$ definiert, die folgendem Gleichgewicht genügt

$$W(\vec{S} \rightarrow \vec{S'})e^{-H(\vec{S})/k_BT} = W(\vec{S'} \rightarrow \vec{S})e^{-H(\vec{S})/k_BT} \tag{4}$$

   Das heißt, wenn sich nach einer gewissen Zeit ein Gleichgewicht einstellt, dann gibt es eine Wahrscheinlichkeit der Form

$$P(\vec{S}) = \frac{1}{Z}e^{-H(\vec{S})/k_BT} \tag{5}$$

   mit der mindestens ein Zustand $\vec{S}$ anzutreffen ist. Damit wird quasi die Physik auf dem Rechner so nachgebildet, daß die Konfigurationen entsprechend ihrem Gewicht erzeugt werden und nicht mit der gleichen Wahrscheinlichkeit vorzufinden sind.

2. **Phasenübergang:** Eine Computersimulation kann nur auf einem endlichen Gitter durchgeführt werden. Das Problem hierbei ist, daß ein Phasenübergang nur für unendlich ausgedehnete Gitter zu beobachten ist. Dazu könnte die Methode des *finite size scaling* hilfreich sein. Diese Methode erlaubt es, aus den numerischen Rechnungen für endliche Gitter auf die Verhältnisse im unendlichen Gitter zu schließen.

3. **Randbedingungen:** Damit an den Gitterrändern keine unerwünschten Überraschungen auftreten, sollte man immer Randbedingungen vereinbaren. Für die Ränder von Gittermodellen nimmt man der Regel periodische Randbedingungen an.

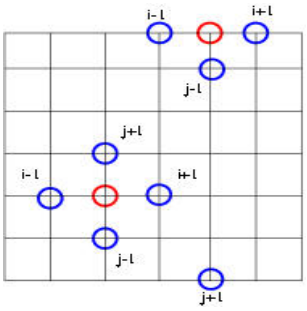

Abbildung 1: Links:Darstellung der vier direkten Nachbarn von S_i (rot) und rechts: Beispiel für das Verhalten an Rändern (siehe Text)

Das heißt, ein Spin am Ort i, j, der sich am oberen Rand des Gitters befindet, hat eines seiner direkten Nachbarn $i, j+1$ auf der selben Gitterlinie aber am anderen Ende des Gitters. Das gleiche gilt auch für die linke und rechte Seite des Gitters. Damit erhält man dann so etwas wie ein Torus.

3 Monte Carlo-Simulation

3.1 Metropolis-Algorithmus

Eins der gängigsten Algorithmen für das Gittermodell, ist der *Metropolis-Algorithmus*.Er fordert, das W nur Übergänge erlauben soll, bei denen höchstens ein Spin geändert wird. Daher sei $\vec{S} = (S_1, S_2, ..., +S_i, ..., S_N)$ und $\vec{S'} = (S_1, S_2, ..., -S_i, ..., S_N)$. Außerdem sollen Übergänge in energertisch günstige Zustände immer erlaubt sein. Das heißt also kurz gefaßt:

$$W(\vec{S} \to \vec{S'}) = \begin{cases} 1 & \text{für } \; H(\vec{S'}) < H(\vec{S}) \\ e^{\Delta E / k_B T} & \text{sonst} \end{cases} \tag{6}$$

Die Energiedifferenz entspricht $\Delta E = H(\vec{S}) - H(\vec{S'})$ und läßt sich mit Hilfe von Gleichung(1) und der obigen Forderung[4], daß $S'_i = \; - S_i$ und $S'_j =$

[4]Diese Forderung ist nicht offensichtlich, da man zunächst eine Symmetrie zwischen $H(\vec{S})$ und $H(\vec{S'})$ zeigen müßte. Näheres dazu kann man in [4] erfahren.

S_j ist,die folgende einfache Beziehung herleiten $\Delta E = -2\ S_i\ h_i$. Für die Programmierung kann man später diese einfache Gleichung für die Energie verwenden.

3.2 Programmierung der Monte Carlo-Simulation

Für das quadratische Gitter braucht man für die Spinmomente S_i eine zweidimensionale Array `s[][]=new int [L][L]`; hier z.B mit `L=512` für die Systemgröße `N=64`. Um das ganze System einfach zu halten, verzichtet man hier auf ein externes Magnetfeld h und setzt für die Kopplungsstärke $J = 1$. Man betrachtet also nur die vier *angrenzenden* Nachbarn von S_i. Für die Übergangswahrscheilichkeit gibt es ebenfalls eine Array `bf[]=new double [3]`, in der später dann die e-Funktion (Boltzmann-Gewicht) gewichtet wird. Für die kritische Temperatur T_c benutzt man die exakte Lösung von Onsager(1944) ,der für das unendlich ausgedehnte Gitter ohne externen Magneten ($h = 0$) den Wert $T_c = 2/ln(\sqrt{2} + 1) = 2.26919\ J/k_B$ bestimmt hat. Nun kann man nach *Metropolis* folgende Schritte formulieren:

1. **Man wähle eine Startkonfiguration** $\vec{S} = S_1, S_2 ... S_N$

   ```
   void ising()
   {
     int i,j;
     for(i=0;i<N;i++)
     for(j=0;j<N;j++)
     {
       if( Math.random() > 0.5 ) s[i][j] = 1;
       else s[i][j] = -1;
   ```

   Wie man aus dem Code-Fragment erkennen kann, wird der Startzustand mit Hilfe einer bedingten Anweisung erzeugt. Ist eine gezogene Zufallszahl größer als `0.5`, so werden die Momente auf 1 gesetzt ansonsten auf -1.

2. **Wähle ein** S_i **zufällig und berechne** ΔE**:**

   ```
   for(i=0;i<10000;i++)
   {
     x=(int)(Math.random()*N);
     y=(int)(Math.random()*N);
   ```

```
e=s[x][y]*(s[(x-1+N)%N][y]+s[(x+1)%N][y]+
  s[x][(y+1)%N]+s[x][(y-1+N)%N]);
```

3. **Wenn $\Delta E \geq 0$, dann drehe den Spin $S_i \to -S_i$. Wenn $\Delta E < 0$, dann ziehe eine gleichverteilte Zufallszahl $r \in [0,1]$. Wenn $r < e^{\Delta E/k_B T}$, dann soll $S_i \to -S_i$ gemacht werden, sonst berücksichtige die alte Kofig.:**

```
if((e<0) || (Math.random()<bf[e/2] ))
{
  s[x][y]=-s[x][y];
        if(s[x][y]==1) g.setColor(black);
```

und schließlich

4. **Iteriere 1. und 2.**

Im Algorithmus werden Zufallszahlen mit dem Boltzmann-Gewicht bei der Temperatur T $\exp \Delta E/k_B T$ verglichen. Um bei der e-Funktion Rechenzeit zu sparen, werden nur die vier angrenzenden Nachbarn von S_i betrachtet. Das heißt, ΔE kann nur die Werte $\pm 4, \pm 2, \pm 1$ und 0 annehmen. Wenn die negativen Werte auftreten, wird S_i immer gedreht. Deshalb benötigt man nur zwei verschiedene Boltzmann-Gewichte.

```
void setT(double T)
    {
temperatur=T;
bf[0]=0.5;
bf[1]=Math.exp(-4.0/temperatur);
bf[2]=Math.exp(-8.0/temperatur);
```

Eigentlich sollte `bf[0]` gleich 1 sein. Um aber mögliche Oszillationen zu unterdrücken, wählt man für den Übergang $W(\vec{S} \to \vec{S'}) = W(\vec{S'} \to \vec{S}) = 1/2$. Als nächstes kann man sich die Randbedigungen vornehmen. Wie schon erwähnt sollen sie periodisch sein. Dazu werden Hilfsfelder erzeugt, die an den Rändern die Bedingungen erfüllen.

```
for (x=0;x<N;x++)
{
  s[0][x]= s[N][x];
  s[N][x]= s[1][x];
  s[x][0]= s[x][N];
  s[x][N]= s[x][1];
```

Physikalisch ist man natürlich noch an der temperaturabhängigen Magnetisierung interessiert. Dafür gilt die Beziehung:

$$M(T) = \frac{1}{N^2} \sum_i S_i \tag{7}$$

Dazu bildet man einfach die Summe aller Spins und teilt einfach durch die Systemgröße. Dies könnte folgendermaßen programmiert werden:

```
Magnetisierung();
summe=0.0;
for(x=0;x<N;x++)
   for(y=0;y<N;y++)
    summe+=s[x][y];
    summe = summe/N/N;
```

4 Ergebnis

Bei Ausführung des Applets, erzeugt das Programm ein 64×64 großen Ising-Ferromagneten. Jedes der flackernden Kästchen stellt nun einen möglichen Spin-Zustand $S_i = -1$ für weiß oder $+1$für schwarz ,dar.

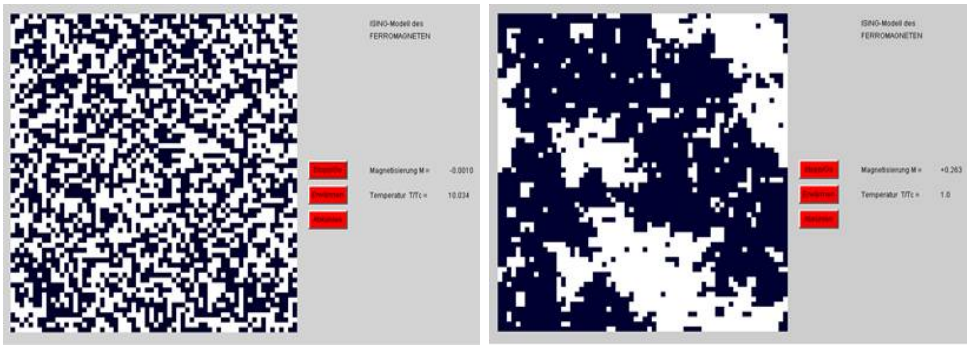

Abbildung 2: Links $T = 10.0 \cdot T_c$ und rechts $T = 1.0 \cdot T_c$

Wenn man jetzt das System abkühlt, so beobachtet man langsam die Bildung von verschiedenen Bezirken, in denen sich die parallel orientierten Spins tummeln. Diese Bezirke nennt man in der Physik auch *wandernde Domänen*.

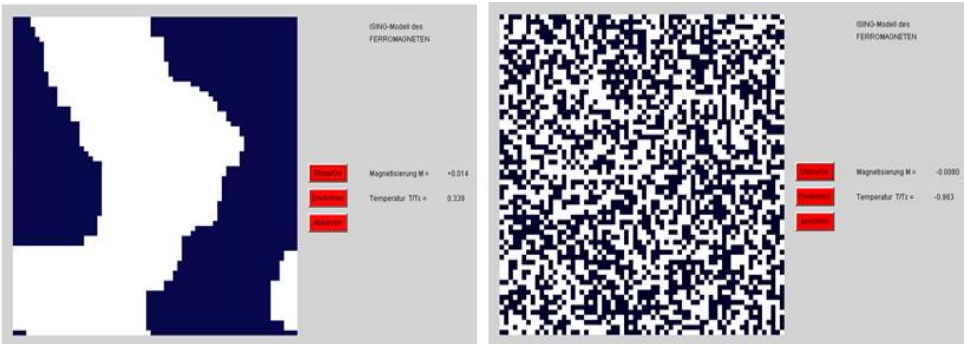

Abbildung 3: Links $T = 0.4 \cdot T_c$ und rechts $T = -1.0 \cdot T_c$

Literatur

[1] W.Nolting: *Theoretische Physik 6,* Springer Verlag, UAP 4336

[2] W.Greiner:*Thermodynamik und statistische Physik 6,* Verlag Harry Deutsch,UAP2075.

[3] K.Binder: *Monte Carlo Simulation in Statistical Physics,* Springer Verlag 1988, naturwiss.Bib. Campus Essen

[4] G.Reents:*Computerphysik,* Spektrum, naturwiss.Bib. Campus Essen

[5] http://www.physik.tu-dresden.de/itp/members/kobe/isingphbi

5 Anhang

Vollständiger Source Code:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;

public class ising_v4 extends Applet implements Runnable,ActionListener
{
int L=512;
int N=64;
int k=8;
int s[][]= new int [L][L];
double temperatur;
double Tc=2.26919;
double summe=0.0;
double bf[]=new double[3];
boolean stop=false;
Thread t=null;
Canvas canvas,canvasPrint;
Button button_Stopp;
Button button_Tplus;
Button button_Tminus;
Graphics g,print;
Color black=new Color(0,0,70);
```

```
public void init()
{
canvas=new Canvas();
canvas.setBackground(Color.white);
canvas.setBounds(20,20,L,L);
canvasPrint=new Canvas();
canvasPrint.setBounds(652,20,248,L);
setLayout(null);
button_Stopp=new Button("Stopp/Go");
button_Stopp.setBackground(Color.red);
button_Stopp.setBounds(L+40,L/2,70,30);
button_Tplus=new Button("Erwärmen");
button_Tplus.setBackground(Color.red);
button_Tplus.setBounds(L+40,L/2+40,70,30);
button_Tminus=new Button("Abkühlen");
button_Tminus.setBackground(Color.red);
button_Tminus.setBounds(L+40,L/2+80,70,30);

add(canvas);
add(canvasPrint);
add(button_Stopp); button_Stopp.addActionListener(this);
add(button_Tplus); button_Tplus.addActionListener(this);
add(button_Tminus); button_Tminus.addActionListener(this);

g=canvas.getGraphics();
print=canvasPrint.getGraphics();
print.setColor(Color.black);
}

public void run()
{
  int e,x,y,i;
  setT(1.0*Tc);
  ising();

  try{Thread.sleep(1);} catch(InterruptedException exc){}

redraw();

  Temperatur();
```

```
while(t!=null)
    {
while(stop)
    {
try{Thread.sleep(20);}
catch(InterruptedException exc){}
    }
for(i=0;i<10000;i++)
     {
x=(int)(Math.random()*N);
y=(int)(Math.random()*N);
e=s[x][y]*(s[(x-1+N)%N][y]+s[(x+1)%N][y]+
                          s[x][(y+1)%N]+s[x][(y-1+N)%N]);

if((e<0) || (Math.random()<bf[e/2] ))
    {
      s[x][y]=-s[x][y];
                              if(s[x][y]==1) g.setColor(black);
      else g.setColor(Color.white);
      g.fillRect(x*k,y*k,k,k);

    }
    }
    for (x=0;x<N;x++)
    {
        s[0][x]= s[N][x];
s[N][x]= s[1][x];
s[x][0]= s[x][N];
s[x][N]= s[x][1];
}
                        Magnetisierung();
        summe=0.0;
        for(x=0;x<N;x++)
        for(y=0;y<N;y++)
summe+=s[x][y];
        summe = summe/N/N;

    }
    }
    void ising()
    {
```

```
int i,j;
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
{
    if( Math.random() > 0.5 ) s[i][j] = 1;
    else s[i][j] = -1;
}
    }


    void redraw()
   {
int i,j;
g.clearRect(0,0,L,L);
g.setColor(black);
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
    if( s[i][j]==1 )
    g.fillRect(i*k,j*k,k,k);

print.clearRect(652,20,248,L);
print.drawString("Magnetisierung M =  ",10,L/2);
print.drawString("Temperatur  T/Tc =  ",10,L/2+40);
print.drawString("ISING-Modell des ",10,20);
print.drawString("FERROMAGNETEN ",10,40);


}

    void Magnetisierung()
    {
String str;
if(summe > 0.0) str = "+"; else str = " ";
print.clearRect(150,0,225,L/2);
print.drawString(str+Math.round(1000*summe)/1000.0,150,L/2);
    }

    void Temperatur()
    {
print.clearRect(150,0,225,L/2+40);
print.drawString(""+Math.round(1000*
```

```
                        temperatur/Tc)/1000.0,150,L/2+40);
    }
     public void actionPerformed(ActionEvent evt)
    {
if(evt.getActionCommand()
           ==button_Stopp.getActionCommand())
            {
stop = !stop;
            }
            if(evt.getActionCommand()
               ==button_Tplus.getActionCommand())
     {
     setT(temperatur+0.05);
     Temperatur();
     }

     if(evt.getActionCommand()==button_Tminus.getActionCommand())
     {
     setT(temperatur-0.05);
     Temperatur();
     }
        }


     void setT(double T)
     {
temperatur=T;
bf[0]=0.5;
bf[1]=Math.exp(-4.0/temperatur);
bf[2]=Math.exp(-8.0/temperatur);
    }
    public void start()
    {
if(t==null)
    {
t=new Thread(this);
t.start();
    }
    }

    public void stop()
```

```
    {
t=null;
    }
}
```

Appletviewer:

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.0 Transitional//EN">
<html><head><title>ISING-Modell</title>
</head><body><applet code="ising_d2.class"
width=900 height=552></applet></body></html>
```