Computergestütztes wissenschaftliches Rechnen
SoSe 2004

Alexander K. Hartmann, Universität Göttingen

14. Juli 2004

10.2 Cluster

Repräsentation eines d-dim Systems:
z.B. durch ein d-im Array `site`, z.B. 3-dim `site[x][y][z]`. Auch höhere Dimensionen von theoretischem Interesse, also `site [x1][x2][x3][x4][x5][x6][x7]`
→ zu komplizert, zu unflexibel (andere Gitterstrukturen).

Lösung: verwende 1-dimensionales (!!) Array `site`:
`site[t]=1` falls Gitterplatz `i` besetzt.
Realisierung der Gitterstruktur Variable '`num_n`' (=Anzahl der Nachbarn) und durch Array `next`.
Jeder Gitterplatz 't hat hier `num_n` Nachbarn, in `next[t*num_n]`... `next[(t+1)*num_n-1]` gespeichert.
Reihenfolge: +x,-x,-y,-y,... Richtung.
Zugriff bequem über Makros:

```
#define INDEX(t, r, nn) ((t)*(nn)+r)
#define NEXT(t, r) next[INDEX(t, r, num_n)]
```

Achtung: immer wenn das Makro verwendet wird, müssen die Variablen `next` und `num_n` mit genau diesen Namen verügbar sein. Kann man aber immer sicherstellen, wenn es nur lokal verwendet wird. Durch die Makros ist es ein wenig unflexibler, aber viel bequemer und besser lesbar.

Man braucht das Array `next[]` nur einmal am Anfang zu initialisieren, und kann es dann immer verwenden. Hier: periodische Randbedingungen:
Beispiel: $L \times L$ Quadratgitter für $L = 10$. Gitterplatz $i = 48$ hat die Nachbarn $i + 1 = 49$ (+x), $i - 1 = 47$ (-x), $i + L = 58$ (+y) und $i - L = 38$. Gitterplatz $i = 1$ hat die Nachbarn 2 (+x), 10 (-x), 11 (+y), 91 (-y).
Untersuchung der Perkolation: Man bestimmt zuerst die *Cluster* = zusammenhängende Bereiche, s.u.. Dann:

a) Direktes Nachschauen, ob es einen Cluster gibt, der von einer Seite zur anderen geht.

b) Wenn das System perkoliert, muss der Größte Cluster linear mit der Systemgröße wachsen →: Anteil der Punkte am größten Cluster ausrechnen (für verschiedene Größen).

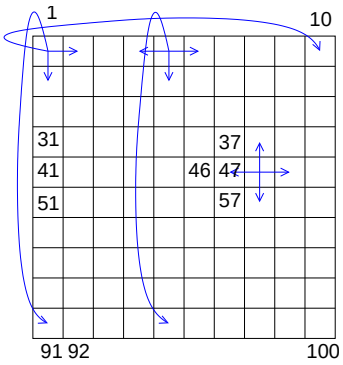


Abbildung 1: Ein 10x10 Quadratgitter mit Periodischen Randbedingungen. Die Pfeile zeigen zu den Nachbarn.

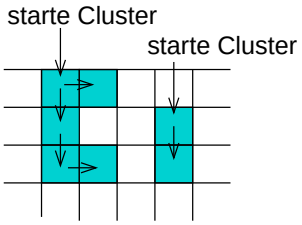
Hier: Methode b), weil einfacher zu realisieren.

Idee für Bestimmung der Cluster:
Ausgangspunkt: ein besetzter Gitterpunkt.
Alle besetzten Nachbarn gehören zum gleichen Cluster.
Alle deren besetzte Nachbarn gehören auch zum gleichen Cluster etc.
Realisierung: Die Nachbarn werden in einem Stack gespeichert und dann nach und nach verarbeitet. Man muss nur dafür Sorgen tragen, dass kein Gitterplatz doppelt behandelt wird.

```
algorithm Cluster
begin
   while es gibt unbehandelten Gitterplatz t do
   begin
      lege t auf Stack
      Starte neuen Cluster mit t
      while Stack ist nicht leer do
      begin
         nehme einen Gitterplatz c vom Stack
         for alle Nachbarn n von c do
            if n noch nicht behandelt then
               Legen n auf Stack und füge zum Cluster hinzu
      end
   end
end
```

Da jeder Gitterplatz nur einmal behandelt wird: Laufzeit $O(N)$.

Beispiel:



Unterprogramm für Clusterbestimmung:

```
/****************** percol_cluster() ******************/
/** Calculates the connected clusters of the lattice   **/
/** 'site'. Occupied sites have 'site[i]=1' (i=1..N).  **/
/** Neigbours of occupied sites form clusters.         **/
/** For each site, in 'cluster[i]' the id of the       **/
/** cluster (starting at 0) is stored                  **/
/** PARAMETERS: (*)= return-paramter                   **/
/**       num_n: number of neighbours                  **/
/**           N: number of sites                       **/
/**        next: gives neighbours (0..N)x(0..num_n-1)  **/
/**              0 not used here. Use NEXT() to access **/
/**        site: tells whether site is occupied        **/
/** (*) cluster: id of clusters sites are contained in **/
/** RETURNS: number of clusters                        **/
/******************************************************/
int percol_cluster(int num_n, int N, int *next,
                   short int *site, int *cluster)
{
  int num_clusters=0;
  int t, r;       /* loop counters over sites, directions */
  int current, neighbour;                        /* sites */
  lstack_t *members;      /* stack of members for cluster */
  for(t=1; t<=N; t++)    /* initialise all clusters empty */
    cluster[t] = -1;
  members = lstack_new(N, sizeof(int));
  for(t=1; t<=N; t++)              /* loop over all sites */
  {
    if((site[t] == 1)&&(cluster[t]==-1)) /* new cluster ? */
    {
      lstack_push(members, &t);          /* start cluster */
      cluster[t] = num_clusters;
      while(!lstack_is_empty(members))  /* extend cluster */
      {
        lstack_pop(members, &current);
        for(r=0; r<num_n; r++)   /* loop over neighbours */
        {
          neighbour = NEXT(current, r);
          if((site[neighbour]==1)&&(cluster[neighbour]==-1))
          {          /* neighbour belongs to same cluster */
            lstack_push(members, &neighbour);
            cluster[neighbour] = num_clusters;
          }
        }
      }
      num_clusters++;
    }
  }
  lstack_delete(members);
  return(num_clusters);
}
```

Beispiellauf für 2 Dimensionen, Seitenlänge $L = 10$, $p = 0.5$). Ausgabe der Nummern der resultierenden Cluster:

```
  0  0  0  0  0  0  0  0     0
  0                 0     0  0  0
                    0        0
     1           0  0  0  0
     1     2           0  0     3
     1     2     4           5
        6     4  4  4  4
  0                          0  0
  0  0                    7     0  0
  0  0     0  0  0     0  0  0
```

10.3 Ergebnisse

Viele Programmaufrufe mittels Script `run_percol.scr`:

```
#!/bin/bash
L=$1
for p in 0.1 0.2 0.3 0.4 0.45 0.5 0.52 0.54 0.56 0.57 0.58 0.59 \
         0.60 0.61 0.62 0.64 0.66 0.68 0.7 0.8 0.9 1.0
do
  percol $L $p
done
```

Ggf. muss das Script per Hand ausführbar gemacht werden: `chmod u+x run_percol.scr`. Ergebnis für 2 Dimensionen, $L = 10, \ldots, 200$:

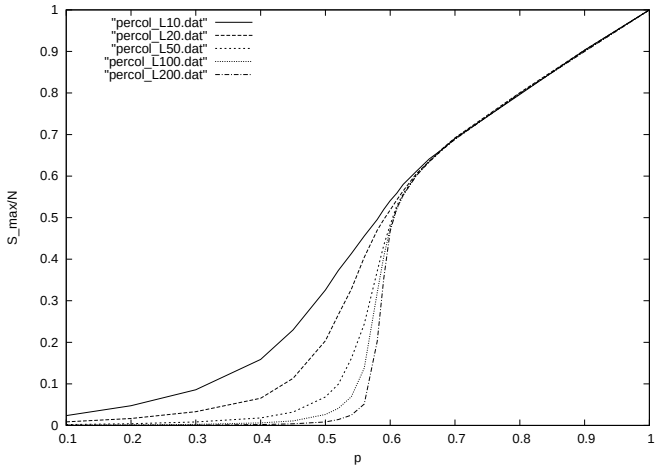


Abbildung 2: Relative Größe des maximalen Clusters als Funktion der Wahrscheinlichkeit p für Quadratgitter.

11 Datenanalyse II

11.1 Binder Kumulante

Bisher: Mittelwerte. Mehr Informationen stecken in den ganzen Verteilungen, also in höheren Momenten. Daraus neue Messgrößen. Häufig verwendet: *Binder*

Kumulante g [1]:

$$g = 0.5 \left(3 - \frac{\overline{x^4}}{(\overline{x^2})^2} \right) \tag{1}$$

Hier, für $x =$ Größe der größten Komponente. Ergebnis:

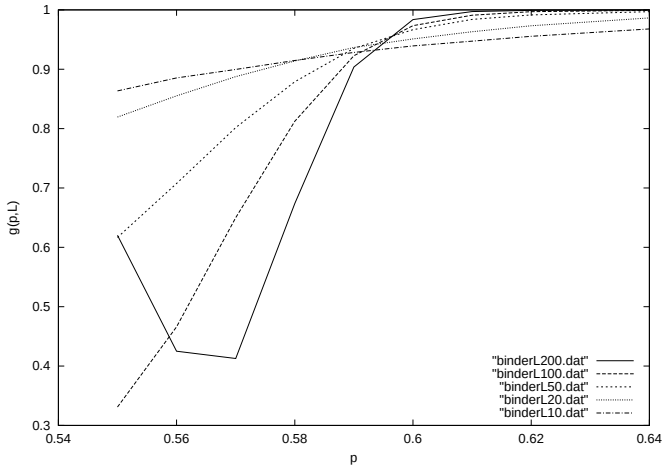


Abbildung 3: Binder Parameter für die relative Größe des maximalen Clusters als Funktion der Wahrscheinlichkeit p für Quadratgitter.

Die Kurven schneiden sich am Phasenübergangspunkt (Ausnahme: zu kleine Systeme). Das ist typisch für Phasenübergänge zweiter Ordnung. → leichte/genaue Bestimmung der Phasenübergangspunkte. Mehr/Begründung: Statistische Physik II

11.2 Resampling

Gegeben N unkorrelierte Messwerte x_i.
Wiederholung: Schätzer für Mittelwert $\langle x \rangle$:

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2}$$

Schätzwert für Fehlerbalken (65%):

$$\sigma_{\overline{x}} = \sqrt{[\overline{x^2} - \overline{x}^2]/(N-1)} \tag{3}$$

Problem: Kombinierte Größen $g(x_1, \ldots, x_N)$ wie der Binder Kumulante (1).
Wie berechnet man die Fehlerbalken? (Fehlerfortpflanzung überschätzt den Fehler, weil $\overline{x^2}$ und $\overline{x^4}$ nicht statistisch unabhängig sind.)
Lösung (ohne Beweise hier): Jackknife Methode [2]. *Definiere*

$$g_i^J = g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N) \tag{4}$$

also der Mittelwert, so dass Datenpunkt i weggelassen wird.
Man kann zeigen, dass

$$[g]_J \equiv \frac{1}{N} \sum_i g_i^J \to \langle g \rangle \tag{5}$$

Der Schätzwert für den Fehlerbalken ist:

$$\sigma_{g^J} = \sqrt{(N-1)\left([g^2]_J - [g]_J^2\right)} \tag{6}$$

Anmerkung: Üblicherweise reicht es aus die Daten in K Blöcke zu unterteilen und jedes mal einen Block wegzulassen.

Literatur

[1] K. Binder, Z. Phys. B **43**, 119 (1981)

[2] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*, SIAM 1982