青山学院大学　理工学部
物理学科　卒業論文

パーコレーションの繰り込み群
シミュレーション

酒井寿徳　　著

パーコレーションの繰り込み群シミュレーション

酒井寿徳
羽田野研究室

平成 13 年 3 月 22 日

概 要

２次元の正方格子のサイトパーコレーションを用いて、繰り込み群の操作をコンピューター上で再現する。簡単のため、最大クラスターのみを抜き出してクラスターの周縁と面積を測定し、繰り込み群の方法を適用した。結論として、臨界確率とフラクタル次元が求まり、パーコレーションの浸透確率から求めた臨界確率と最大クラスターの周縁を測り繰り込んだものから求めた臨界確率はほぼ一致した。また、臨界点におけるパーコレーションクラスターが繰り込み群の操作によって変化しないことが確かめられた。

目 次

1　はじめに

誰もが一度はオセロというゲームで遊んだ事があるだろう。格子状に白と黒の石を置き、黒のつながりを白で挟んだらその間を黒とする。また逆に、白のつながりを黒で挟んだらその間を白とする。こんな単純なルールなのに、複雑な展開に皆、頭を悩ませる。先の先はこうだから、あそこをこうしなければ、、、、と。それと問題にする所は違うけれど、似たような形をした物理的モデルに**パーコレーションモデル**というものがある。オセロと違うのは、パーコレーションの科学がつながりを議論する科学である所だ。しかしながら、単純なルールから複雑な問題を引き起こす点ではオセロと同じだ。また、系を大きくしていけばいくほど、複雑な問題が現れてくる。この複雑な現象を解析するための手段として**繰り込み群の方法**と**フラクタル**というものがある。繰り込み群の方法とは、そのような複雑な現象における物理量を段階的に粗視化することによって定量的に解析する方法である。また、フラクタルとは、スケール不変の性質およびそのような性質をもつ図形の事である。一見複雑な現象からもスケール不変性を見つけ出せることが多々ある。

本研究の目的は、繰り込み群を用いて、パーコレーションクラスターのスケール不変性を調べることである。具体的には、まず、パーコレーションモデルにおける最大クラスターの周縁と面積を測り、その系の浸透確率を求める。そして、そのクラスターに繰り込み群の方法を適用し、臨界確率やフラクタル次元を求め、臨界点におけるクラスターの分布を調べることである。本研究を通じて３つの「パーコレーションモデル」「繰り込み群の方法」「フラクタル」という概念が密接につながり合っていることが自ずとわかるだろう。

2　パーコレーションモデルについて

2.1　パーコレーションモデルの説明

パーコレーションモデルとは、ある確率で格子点（サイト）上やつなぎめ（ボンド）上に粒子を置き、クラスターをつくるモデルのことである。日本語に訳せば浸透という意味であり、最初は浸透現象を表すモデルとして考え出された。パーコレーションには２種類ある。サイトパーコレーションは、粒子を格子点上に置くモデルのことで、今回このモデルを用いることにした（図 1)。ボンドパーコレーションは、粒子をつなぎめに置くモデルである。

一見、単なる数学的モデルのように見えるが、物理的モデルとして応用できる範囲は非常に広い。例えば [1, 2]

- 岩石中の液体の浸透
- 不純物が多く存在する物質中の電気伝導

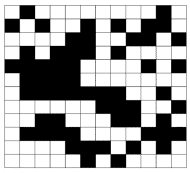

図 1: 12×12 の格子上のサイトパーコレーションの例。

- 森林火災
- 伝染病の伝播
- 要素の成長過程
- 地域間の情報の伝達
- 高分子のゲル化

などに用いられる。他にも現在では遺伝情報の均一化と局在化を理解しようという試みが行われている。

パーコレーションの最も簡単な例として、正方格子の格子点に適当な大きさの石をまったくでたらめに置き、その石のつながり方を考えてみよう。上下または左右方向に隣り合った石同士がつながるものとする。このとき、置いた石の数を変えてつながり方の変化を見てみる。

石の割合が少ないときは、石は小さな塊となって何も置かれていない点に取り囲まれている。このような石の塊のことを**クラスター**といい、一つのクラスターの石の数をそのクラスターの大きさということにする。石の置かれた格子点の割合が１００％に近いと石は一つの大きなクラスターとなり、そのクラスターは正方格子の上下、左右の端から端までつながったものとなる。そのようなクラスターが存在した状態を浸透状態と定義する。

ここで、石の置かれた格子点の割合を $p\,(0 \leq p \leq 1)$ としたとき、p がどれ位になると浸透状態になるのであろうか。系の大きさが有限の系の極限で考えると、無限に広がった大きなクラスターができ始める臨界的な割合（臨界確率）p_c が定義できる。このとき、クラスターは**フラクタル**になると言われている。フラクタルについては後に説明する。p_c の値はいくらだろうか？直感的には５０％のような気がするが、それは正しいのだろうか？また、p_c 近辺でのクラスターの大きさの分布はどの様なものであろうか？そのような問題について考えてみたい。

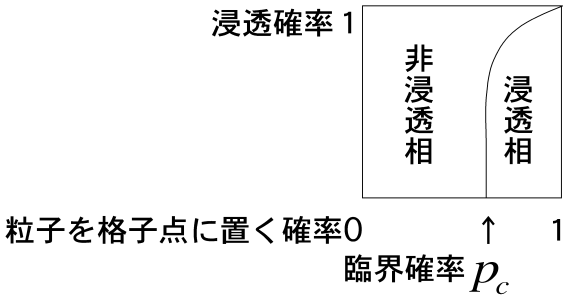

図 2: 浸透確率 P と粒子を格子点に置く確率 p の関係。臨界確率 p_c が必ず存在する。

2.2 パーコレーションモデルにおける臨界確率

前節で述べたように、石を置く確率 p を増やしていくと無限に大きなクラスターが存在しない相から存在する相に相転移する。前者の相を非浸透相と呼び、後者の相を浸透相と呼ぶ。非浸透相から浸透相に相転移する際、その変わり目となる臨界点が存在するはずである。どうやって正確にその臨界点を求めればいいのだろうか？そのために物理量の一つとして**浸透確率**というものを定義する。それは、

$$P(p) = \lim_{L\to\infty} \frac{\text{最大クラスターの大きさ}}{\text{系の大きさ}} \tag{1}$$

である。ここで系を一辺の長さ L の正方形としている。また、p は粒子を格子点に置く確率である。粒子を格子点に置く確率と系全体に対する占有された格子点の割合は厳密には異なるが、$L \to \infty$ の極限では等しいので、どちらと受け取ってもよい。また、浸透確率 P は非浸透相でゼロ、浸透相で有限の値となり図 2 のように振舞うと予想される。つまり、P はパーコレーション相転移の秩序変数である。

ただ、数値計算では系を無限の大きさとするのは不可能なので、近似して

$$P(p, L) \simeq \frac{\text{最大クラスターの大きさ}}{\text{系の大きさ}} \tag{2}$$

で考えることにした。プログラムでは、$kakuritu$ としている。

3 繰り込み群の方法について

3.1 繰り込み群の方法の説明

繰り込み群の方法とは、非常に大きい系の状態を知る手段の 1 つである。その方法とは、系を段階的に粗視化して物理量の変化をとらえることである。粗視化とは、遠くから見たときに大体どのようなものなのか？ということである。つまり、繰り込み群の方法は、微視的な観点ではわからない複雑な現象を理解すると

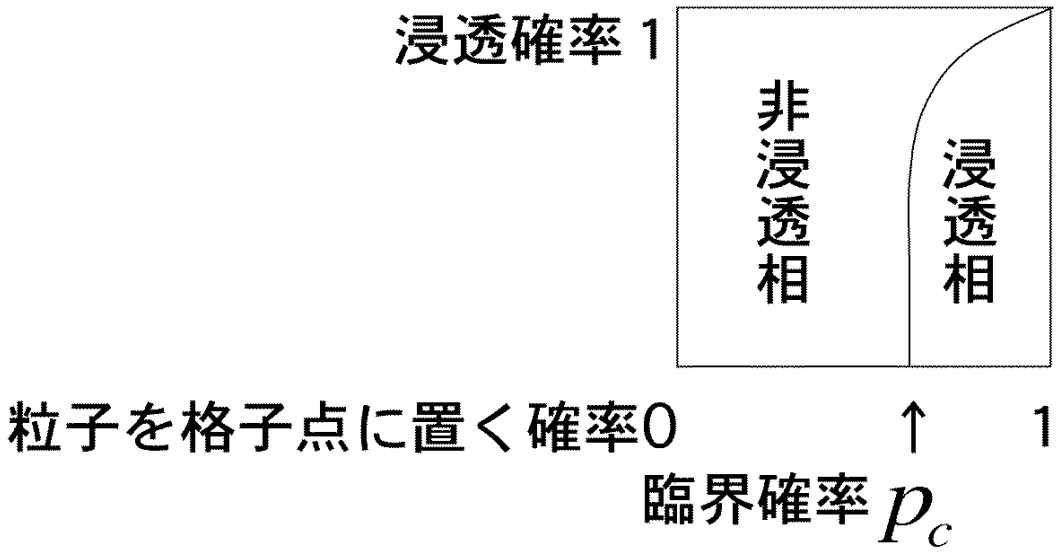

図 3: ３行３列の繰り込みの例。

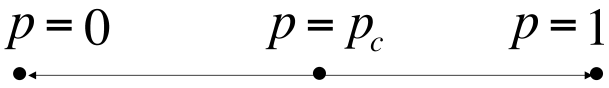

図 4: 繰り込みの流れ図。

きに用いる。具体的には、今回のパーコレーションモデルに適用するにあたって、３行３列で粗視化した。３行３列の９マスのうち、５マス以上黒なら１マスの黒に、５マス以上白なら１マスの白に置き換える（図 3)。

3.2　繰り込み群の方法による臨界確率

上の操作によって、パーコレーションの臨界点が正確に求められる。黒が臨界点より多い相において何度も繰り込んでいくと、最終的には（$L = 3$ になったときには）黒がより多くなっているだろうし、白がその点より多い相において何度も繰り込んでいくと、白がより少なくなっているだろう。プログラムでは、繰り込む前の白黒の配置を $masu[i][j]$、繰り込んだ後の配置を $MASU[m][n]$ としている。

ある臨界点を境目として最終的に黒くなるか、白くなるかが決まる。従って、図 4 のような繰り込みの流れ図が予想される [4]。この流れ図を作ることによって臨界確率を求めることができる。一般に何度粗視化しても状態が変わらない p の値を固定点という。例えば $p = 0(\%)$ のときは、粒子が全くないので何度繰り込んでも $p = 0(\%)$ のままである。$p = 100(\%)$ のときも同様に固定点であることは明らかである。自明でない固定点が $p = p_c$（臨界確率）である。このとき、クラスター

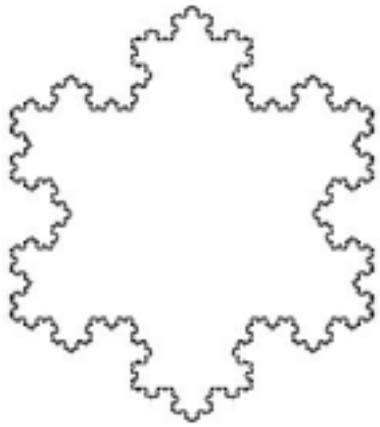

図 5: コッホ曲線。

は自己相似性をもつフラクタル図形になり、次節で説明するように何度粗視化しても変化しない。$p = p_c$ を不安定固定点といい、$p = 0(\%)$ 及び $p = 100(\%)$ の 2 点は安定固定点といわれる。なぜ、臨界確率が不安定固定点なのかといえば、その点よりわずかでもずれれば安定固定点に収束してしまうからである。

4　フラクタルについて

4.1　フラクタルの説明

フラクタルとは、大きなスケールで見ても小さなスケールで見ても同じ物にみえる（スケール不変性をもつ）図形、及びその性質のことである。自然界を見渡せば、そのようなものは結構ある。雪の結晶、珊瑚礁や、海岸線などがそれである。海岸線は厳密にフラクタルであるわけではないが、一部分を拡大すると統計的には元と同じ形になっている所がある。

このような**スケール不変性** が一番わかりやすい説明だが、他にもいろいろな説明の仕方がある。それは、特徴的な長さをもたない、ということや、微分不可能だということだ。例えば、地球は厳密に球や回転楕円体ではないけれど、だいたい特徴的な長さである半径で近似できる。そうすれば、微分可能な点が存在する。

ところが、図 5 のフラクタル図形はそうではない。特徴的な長さを持たないし、あらゆる点において微分不可能である。これは、コッホ曲線という有名なフラクタル図形である。作り方は簡単で、直線を 3 分の 1 にし、真ん中の部分をその 3 分の 1 の長さを 2 つ使って尖らせる。つまり、全体の長さが一度その操作をするたびに 3 分の 4 倍になる。この操作を無限に繰り返したのがこの図形である。少し雪の結晶に似ている。この事からもフラクタルが自然界に深く関係していることがわかるだろう。なぜ、このような形になるのか？一説によれば、フラクタル

であることが最も自然でエネルギー的に安定だと言われている。

4.2　フラクタル次元の定義

フラクタル図形は、次元が非整数になっているという大きな特徴がある。フラクタル図形の次元の定義には様々なものがあるが、一番わかりやすいと言われる相似性次元を用いることにした。相似性次元は、厳密に自己相似性を持つ場合のみ用いられるのが一般的だが、繰り込み群の操作によってフラクタル次元を求める場合には適用することができる。相似性次元 D の定義は、

$$S \propto L^{-D} \tag{3}$$

$$\log S \simeq -D \log L + \text{定数} \tag{4}$$

である。ここで、S は、その次元の測度である。測度とは、わかりやすい例で言えば、1 次元測度が線、2 次元測度が面積、3 次元測度が体積、といった量のことである。また、L は、スケールをどれくらい変えたときに元の測度と等しくなるか、を考えたときのスケールのことである。今回の繰り込みでは、一度繰り込む毎にスケールが 3 分の 1 になるので、k 回繰り込んだ後では $L = 3^{-k}$ となる。従って、$\log_3 S \simeq kD +$ 定数 となる。このような形で結果は図示した。このように分布がフラクタルかどうかを調べるためには、その分布が**べき乗則**に従っているかどうかを問題にする。

相似性次元が整数である場合は、普段、私達が考えている次元のことであり、ユークリッド次元と言われる。一方、整数でない場合はフラクタル次元と言われる。前節で例にあげたコッホ曲線の場合は、スケールを 3 分の 1 にすると同じ形が 4 つできるので、$S = 4$、$L = \frac{1}{3}$ となり、次元はだいたい 1.26 くらいである。

5　パーコレーションの繰り込み群シミュレーション

5.1　シミュレーションの概要

本研究ではパーコレーションクラスターをコンピューター上で発生させ、それを数値的に繰り込んだ。実際の操作の概要は以下の通り。

1. 黒（2.1 章の例で言えば粒子及び石）を置く確率 p と乱数の初期値を決める。ここで、黒を置く確率は p であり、白を置く確率は $1 - p$ である。

2. 各格子点に対して乱数を発生させて 0 か 1 かを決め、格子点上に 1（黒）と 0（白）をばらまく。但し、格子の大きさ L はあらかじめプログラム上で決めておく。

3. クラスターに大体の番号を付ける。

```
1 1 0 1 0 0 0 0 1 1 1 0 0 1 1 0 1 0 0 0 1
1 0 0 1 0 0 0 1 0 1 1 1 1 0 1 1 1 0 1 1 1
1 0 0 0 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1
1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 0 0 1 1 1 1 0 0 0 1 1 0 1 0 1 0
1 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 0 0 1 1 0
1 0 1 0 0 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1
1 1 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 0 0 1 1
1 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 0
1 1 1 1 1 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0
1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 0 1 1
1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 1 1 1 1 1
0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0
1 1 1 0 1 0 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0
1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 1 1 0 1
1 1 0 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 1 1 1
1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 0
0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 0 1
1 1 0 1 0 1 1 1 1 0 0 1 0 0 0 1 0 1 0 1 0
1 1 0 0 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0
```

図 6: L=27 のパーコレーションを作った状態。

4. その番号を厳密に区別する。

5. 最大のクラスターのみ残し（番号を１にする）、それ以外のクラスターは消す（番号を０にする）。

6. 最大クラスターの周縁と面積を測る。また、浸透確率も調べる。

7. 格子が３行３列になるまで周縁と面積と浸透確率を測り繰り込む。

以下で、より詳しく説明する。

5.2　クラスターの判別

まず、乱数を用いてある確率で１と０をばらまく（図 6）。

次に１をクラスターに分類する。つまり、同じクラスターに属す１には同じクラスター番号をつけることを考える。まず第一列目の最初の占有された点のクラスター番号を１とする。右方向へスキャンし、**その点が占有されている場合のみ**次の対処をする。

- 左の点が占有されていれば左の点と同じクラスター番号を、また占有されていなければ新しいクラスター番号を付ける。

```
 1  1  0  2  0  3  3  0  4  0  5  5  5  5  0  6  0  7  7  7  0
 1  0  8  0  9  3  0 10  0 11  5  5  0  5  5  5  0  7  0  7  0
 1  1  1  1  0  3  0 10  0  0  0  5  5  5  5  0  0  7  0  0  0
 1  0  0  1  0  3  0 10 10  0  0  0  5  5  5  0 12  0  0 13  0
 0 14 14  0 15  0 16 10 10 10  0  0  0  5  5  5  5  0 17 13 13
 0 14 14 14 14 14 14 10 10 10  0  0 18  5  5  0  0 19 17 13  0
 0 14  0 14  0 14  0 10  0  0 20 20  0  5  5  0 21 19 17 13  0
 0  0 22 14  0 14 14 10  0 23 20  0  0  0  5  0 21  0 17  0  0
 0 24  0  0 25 14  0 10 10  0  0 26  0 27  0 28  0 29 17  0 30
31 24  0 32 25  0 33 10 10  0 34 26 26 26 26  0 35 29 17 17 17
31 24 24 24  0  0 33  0  0  0 34  0 26 26 26 26  0  0 17 17 17
 0  0  0  0 36 36 33 33 33  0  0 37  0 26  0 26 26  0 17 17 17
38 38 38 38  0 36 33  0  0 39 39  0 40 26  0  0  0  0 17  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 41 17 17  0
42 42 42 42  0 43 43 43  0 44  0 45 45 45 45 45  0 41  0  0 46
 0 42 42  0  0 43  0  0 47 44 44  0 45  0 45  0 48 41 41 41 41
49 42 42  0 50 43 43  0 47 44 44  0  0  0 45  0  0  0 41 41 41
49 42 42 42  0  0  0  0  0  0 44  0 51  0  0 52 52 52  0 41 41
 0  0  0 42 42  0 53 53  0  0 44  0 51  0  0  0  0 52 52 41 41
 0  0 54 42  0 55 53  0  0 56 44 44  0 57 57 57  0  0  0 41  0
58 58 54  0  0  0  0 59 59  0  0  0 60  0 57  0 61 61  0  0 62
```

図 7: 大体のクラスター番号をつけた状態。

第一列目が終ると第二列目を調べる。第二列目をスキャンする場合には、左の点が占有されているか否かを見るだけでなく、上の点も見る必要がある。

- 左と上の両方とも 0 であれば、新しいクラスター番号を付ける。
- 片方が占有されていればその点とおなじクラスター番号をつける。
- 両方とも占有されていれば、それらのクラスター番号のうち小さい方と同じクラスター番号をつける。

このようにして、大体の番号を付けたのが、図 7 である。

しかし、このままでは同じクラスターに属するのに異なるクラスター番号をつけられている場所が随所にある。よって、これらの番号をきちんと判別することを考える。まず、サイト番号 $i, j\,(0 \leq i, j \leq L-1)$ に対する配列 $masu[i][j]$ を用意する。L は、正方格子の系全体の一辺の長さである。ここで、先程の図 7 で与えられた大体のクラスター番号とは、$masu[i][j]$ のことである。そこで、それらを真のクラスター番号にする（きちんと判別する）ために、配列 $relabel[masu[i][j]]$ を作る [5]。端のサイト以外のサイトでは、上下左右全てを調べることができる。また、端のサイトを占有するクラスターがあれば、その一つ内側のサイトをスキャンしたときに端のサイトの番号を変えればよいのだから、**端のサイトを除いてもよい**。

もう一度左上からスキャンし、先程と同様**その点が占有されている場合のみ**次の対処をする。

```
 1  1  0  2  0  3  3  0  4  0  5  5  5  5  0  5  0  7  7  7  0
 1  0  1  0  3  3  0 10  0  5  5  5  0  5  5  5  0  7  0  7  0
 1  1  1  1  0  3  0 10  0  0  0  5  5  5  5  0  0  7  0  0  0
 1  0  0  1  0  3  0 10 10  0  0  0  5  5  5  0  5  0  0 13  0
 0 10 10  0 10  0 10 10 10 10  0  0  0  5  5  5  5  0 13 13 13
 0 10 10 10 10 10 10 10 10 10  0  0  5  5  5  0  0 13 13 13  0
 0 10  0 10  0 10  0 10  0  0 20 20  0  5  5  0 13 13 13 13  0
 0  0 10 10  0 10 10 10  0 20 20  0  0  0  5  0 13  0 13  0  0
 0 10  0  0 10 10  0 10 10  0  0 26  0 26  0 28  0 13 13  0 13
10 10  0 10 10  0 10 10 10  0 26 26 26 26 26  0 13 13 13 13 13
10 10 10 10  0  0 10  0  0  0 26  0 26 26 26 26  0  0 13 13 13
 0  0  0  0 10 10 10 10 10  0  0 37  0 26  0 26 26  0 13 13 13
38 38 38 38  0 10 10  0  0 39 39  0 26 26  0  0  0  0 13  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 13 13 13  0
42 42 42 42  0 43 43 43  0 44  0 45 45 45 45 45  0 13  0  0 13
 0 42 42  0  0 43  0  0 44 44 44  0 45  0 45  0 13 13 13 13 13
42 42 42  0 43 43 43  0 44 44 44  0  0  0 45  0  0  0 13 13 13
42 42 42 42  0  0  0  0  0  0 44  0 51  0  0 13 13 13  0 13 13
 0  0  0 42 42  0 53 53  0  0 44  0 51  0  0  0  0 13 13 13 13
 0  0 42 42  0 53 53  0  0 44 44 44  0 57 57 57  0  0  0 13  0
42 42 42  0  0  0  0 59 59  0  0  0 60  0 57  0 61 61  0  0 62
```

図 8: 完全にクラスター番号を判別できた状態。

1. $masu[i][j]$ を $relabel[masu[i][j]]$ に代入する。
2. $relabel[masu[i][j]]$ を $labelmin$ に代入する。
3. 上下左右で $masu[i][j]$ よりも小さなクラスター番号があれば、$relabel[masu[i][j]]$ に代入する。
4. 上下左右のうち最も小さな番号を $labelmin$ に代入する。
5. $labelmin$ を $relabel[relabel[masu[i][j]]]$ に代入する。
6. 上下左右が占有されていれば同様に $relabel[relabel[masu[i][j]]]$ を代入する。
7. 一つでも $relabel[masu[i][j]]$ よりも $masu[i][j]$ が大きければ 1 から 6 を繰り返す。

図 7 のように、真のクラスター番号にするために、３６番を、３３番に変えて、さらに１０番に変える必要がある場合がある。系が大きくなればなるほど、このような問題が増えてくるので、配列の配列を考えなければならなくなる。さらに、その配列の配列の..... を作ればもっと処理時間が早くなるだろう。とにかく、こうして完全に判別できたクラスターが次の図 8 である。

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

図 9: 最大のクラスターを抜き出した状態。

そして、最大のクラスターを抜き出す。丁度 $relabel[masu[i][j]]$ がクラスターの大きさになっているので、これをもとにどのクラスターが最大か？を考えればよい。最大クラスターの番号は１に、それ以外のサイトの番号は０に変える（図 9)。

最後にこの系に３行３列の繰り込みを適用する（図 10)。これを、最終的に９マスになるまで続ける。他にもクラスターの周縁、面積、系の浸透確率などを求めておく。

5.3 周縁の定義と数え方

クラスターの分布を特徴づける量の一つとして周縁という量がある。まず、一つのクラスターが独立して存在するためには、その周囲の格子点が空、つまりつながりを形成する要素でしめられていないことが必要である。一つのクラスターを指定したときに、必ず空でなければならない格子点のことを**周縁 (ペリメーター)** という。大きなクラスターになると、これらの格子点はクラスターの中の方にも存在することを注意しておく。クラスターの周囲と周縁の違いはまさにこの点にある。

次に、周縁を計算するプログラムの説明に入る。クラスターは１で表されているが、（内側を含む）周囲の格子点を２以上の数で表し、その数を数えることを考

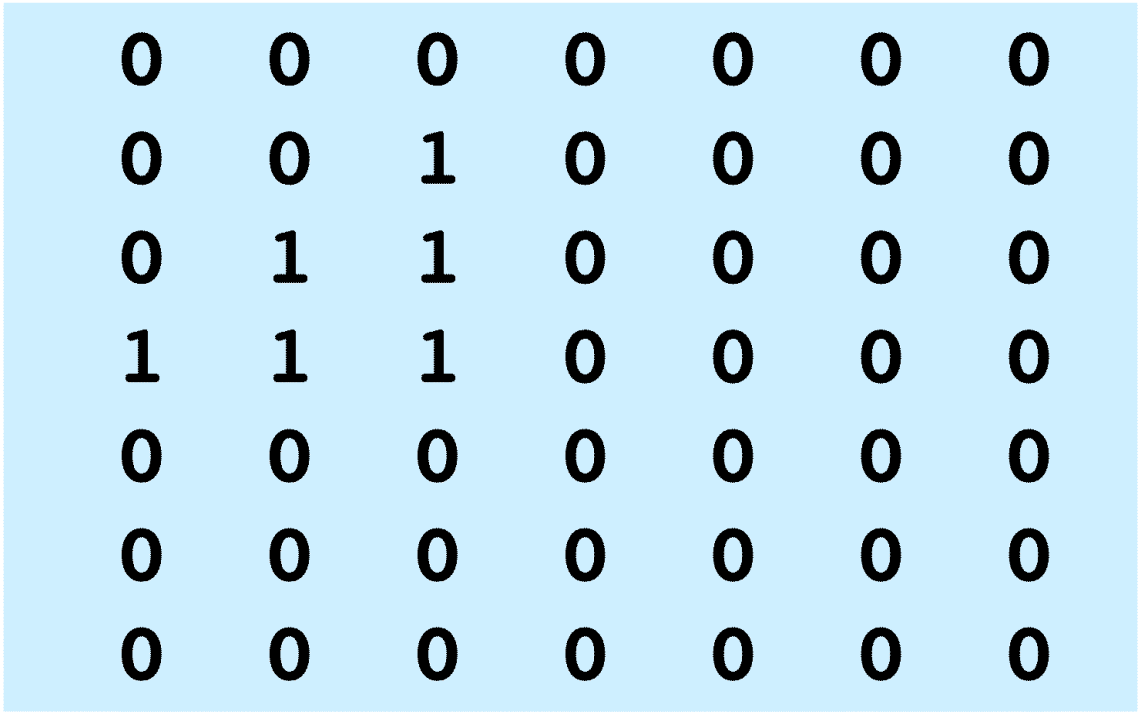

図 10: ３行３列の繰り込みを適用した状態。

える（図 11）。ただし、端のサイトに１がある場合は、それより外側のサイトがないのでその分も勘定に加える。更に、４隅にある場合は、これらも勘定に加える。まず、系の左上端の点から始めて、一列目を右方向に０をスキャンしながら

- 上下左右のどこかに１がある場合、その０を２以上の数に変える。
- そして、その２以上の数の個数を変数 m で数える。

次に１をスキャンして行きながら

- （４隅を含む）端のサイトでは、変数 n を１ずつ増やしていく。
- ４隅のサイトでは、変数 n を１ずつ増やしていく。

つまり４隅では、n を２つ増やしたことになる。最後に、m を n に加える。このとき、n が周縁の数となる。

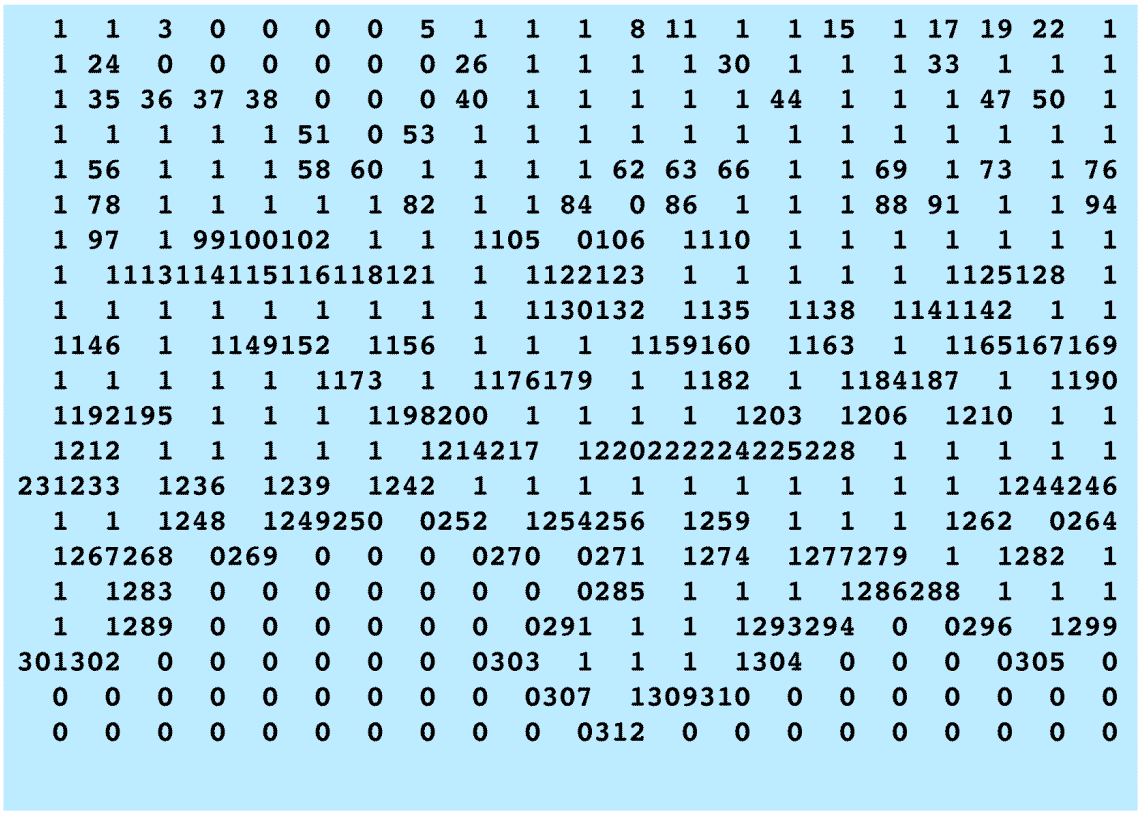

図 11: クラスターの（内側を含む）周囲に２以上の数を埋めた状態。

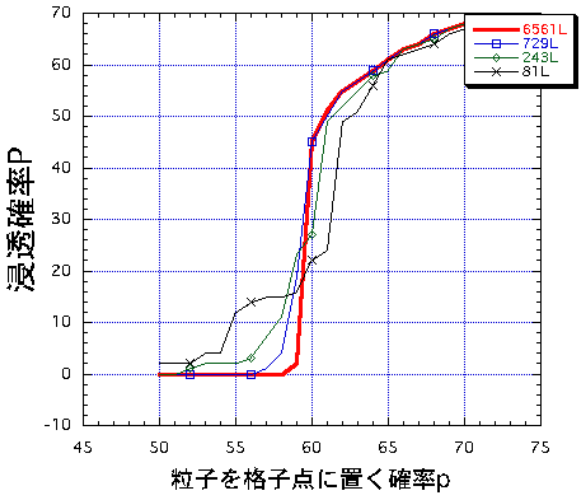

図 12: 浸透確率 P と粒子を格子点に置く確率 p の関係。

6　結果及び考察

6.1　臨界確率とフラクタル次元

以上の計算より臨界確率は５９%、フラクタル次元は 1.76 という結果を得た。これを以下で詳しく説明する。

図 12 は、粒子を格子点に置く確率と浸透確率の関係を示すグラフである。格子が大きくなるにつれて臨界確率が５９%に近付いているのがわかるだろう。

図 13 は、最大クラスターの周縁と繰り込み回数の関係を示すグラフである。注目していただきたいのは、５９%の直線と１００%の直線である。５９%の直線は、前述した不安定固定点でこの点が臨界点になっている。このとき、フラクタル次元は 1.76 となった。また、１００%の直線は安定固定点で、このときのフラクタル次元は、ほぼ 1 になっている。周縁が５９から６０%の辺りを境に段々と減少しているのは、一度浸透状態に達したクラスターにおいてさらに初期確率を大きくしていくと内側に存在する穴（周縁）が埋まるためだと考えられる。

図 14 は、最大クラスターの面積と繰り込み回数の関係を示したグラフである。どの粒子を格子点に置く確率においても次元はほぼ 2 であり、フラクタル次元は得られなかった。今後検討の余地がある。

図 15 は、7 回繰り込んだときの占有されている格子点の数と粒子を格子点に置く確率 p の関係を示したグラフである。確率５９%を臨界確率としてもよいのは一目瞭然である。

これらの結果から、パーコレーションの浸透確率から求めた臨界確率（図 12）と最大クラスターの周縁を測り繰り込んだものから求めた臨界確率（図 13 及び図

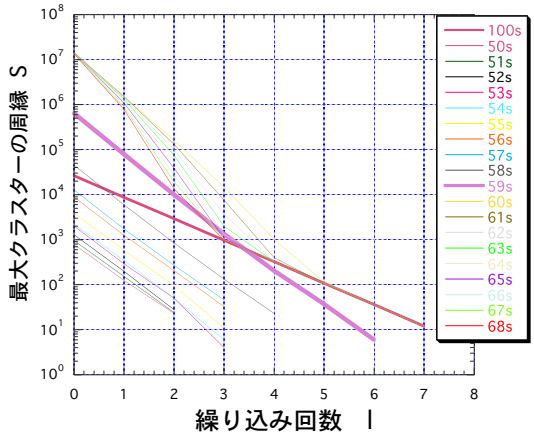

図 13: 最大クラスターの周縁と繰り込み回数の関係。

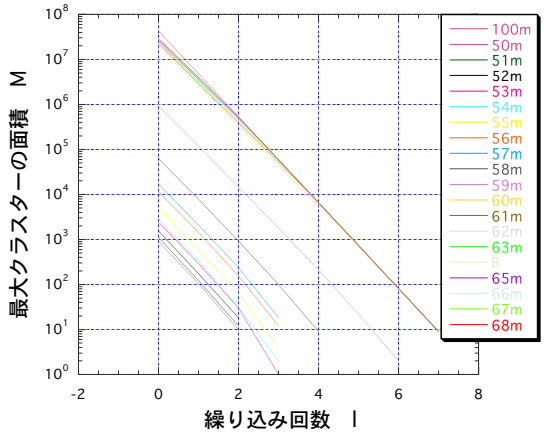

図 14: 最大クラスターの面積と繰り込み回数の関係。

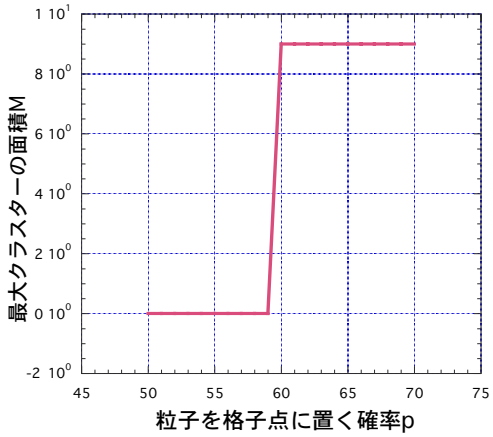

図 15: ７回繰り込んだとき占有されている格子点の数と粒子を格子点に置く確率 p の関係。

15）は一致するものと思われる。５９%というのは、ほぼ正しい結果だと思うが、厳密には５９%から６０%の間に存在するのだろう。プログラムを整数型で作ってしまったことが悔やまれる。

フラクタル次元に関しては、クラスターの重心とその距離による臨界次元から求める方法（繰り込み群を用いない）では、1.90[1] ぐらいであり、この値に一致しないのは厳密な臨界点ではないからだと思われる（図 13)。つまり、５９%から６０%の間にある臨界確率を厳密に求め、その確率で最大クラスターの周縁を測り繰り込めば、1.90 に一致するかもしれないと思う。もう一つの考えられる原因は、フラクタル次元には様々な定義があり [3] ここで用いた相似性次元はその中の一つにすぎないためではないか？ということだ。つまり、他の定義で測ってみる必要があるだろう。

最後に、さらに系を大きくできればなお良かったが、繰り込むことを考えて 3 の倍数でなければならなかったため、このコンピューターでは $L = 6561 = 3^8$ が限界であった（結果では、$L = 6561$ の場合を用いた)。

6.2　繰り込み群の方法による最大クラスターの形状の変化

先程の結果から臨界確率が５９%程度であることがわかった。よって、その辺りのクラスターの形状を調べることにした。ここで例にあげたのは、５５%（図 16–18)、５９%（図 19–21)、６３%（図 22–24）のクラスターの形状である。 前述した通り、臨界点では繰り込んでもクラスターの形状はほぼ変化しない。それより初期確率が小さい場合は繰り込むとなくなってしまうし、それより初期確率

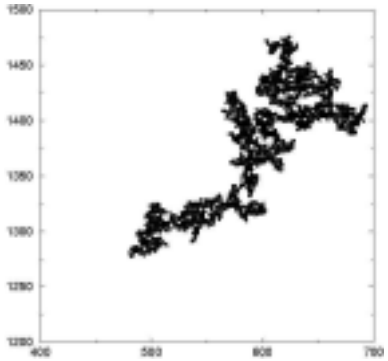

図 16: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 55\%$ で、１度も繰り込んでない状態。

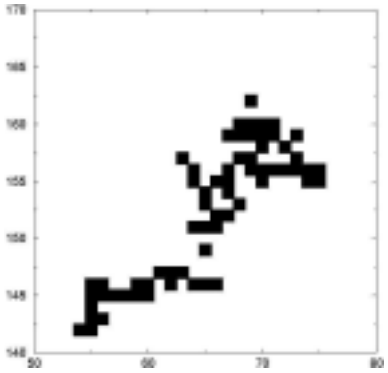

図 17: 系の一辺の長さ $L = 6561$ が、粒子を格子点に置く確率 $p = 55\%$ で、２回繰り込んだ状態。黒の割合が減っている（白の割合が増えている）。

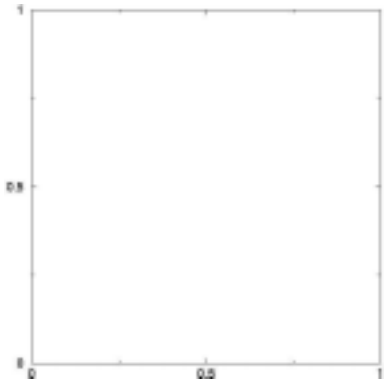

図 18: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 55\%$ で、４回繰り込んだ状態。完全に真っ白になっている。

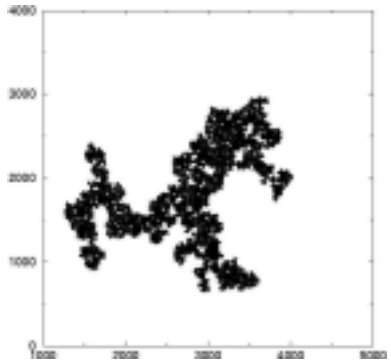

図 19: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 59\%$ で、１度も繰り込んでない状態。

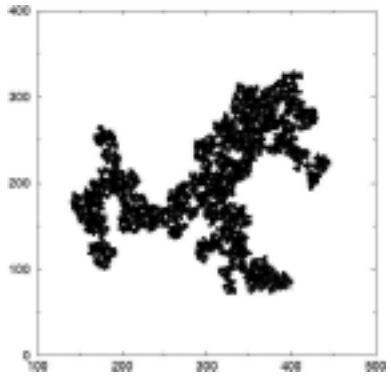

図 20: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 59\%$ で、２回繰り込んだ状態。形状はほぼ、変化してない状態。

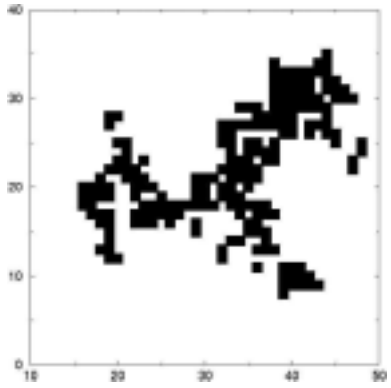

図 21: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 59\%$ で、４回繰り込んだ状態。形状は、ほぼ変化していない。

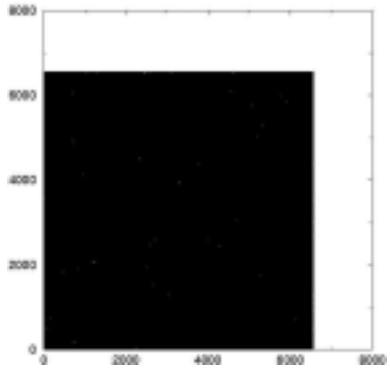

図 22: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 63\%$ で、１度も繰り込んでない状態。真っ黒のようにみえるが、それはドットが多いためで、実際は全部うまっていない。面積を数値で見てみれば正しいことがわかる。

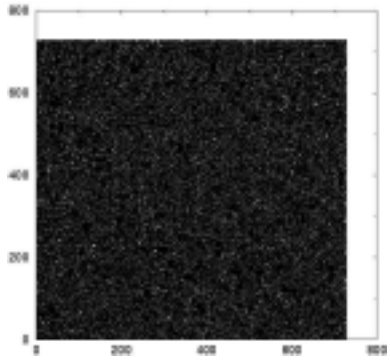

図 23: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 63\%$ で、２回繰り込んだ状態。黒の割合が増えている。

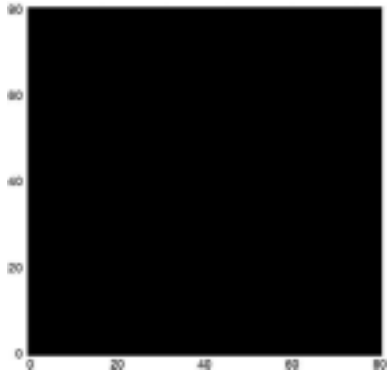

図 24: 系の一辺の長さ $L = 6561$、粒子を格子点に置く確率が $p = 63\%$ で、４回繰り込んだ状態。完全に真っ黒になっている。

繰り込みの回数	最大クラスターの面積	浸透確率（%）
0	24885246	57.8
2	458297	86.2
4	6561	100

表 1: 図 22–24 における最大クラスターの面積と浸透確率。

が大きい場合は繰り込むと格子全体がクラスターになってしまう。そうなるはずなのだが、６３%のクラスターのおいて初期確率が大きい場合はドットが多過ぎるため全部埋まっているかのように見えてしまった。実際、面積の数値を調べてみると段々と（浸透）確率が増えているのがわかる。その数値を表 1 に示す。

7　謝辞

羽田野先生には、この研究をするにあたっていろいろとアドバイス頂き、大変お世話になりました。この場をかりて感謝いたします。特に、自分にこうしたいという思いはあってもその通りのプログラムが書けずに苦労してる時や、自分だけ研究室内で違う輪講をさせていただいた時、研究以外でも進路で悩んでいる時にいろいろと相談にのっていただいたのは、大変助けになりました。また、同室の先輩方、同級生の方々にも御礼を言いたいと思います。どうもありがとうございました。

参考文献

[1] 小田垣孝『パーコレーションの科学』（裳華房、１９９３）p.103 p.111,p.39

[2] 今野紀雄『図解雑学複雑系』（ナツメ社、１９９８）p.156

[3] 高安秀樹『フラクタル』（朝倉書店、１９８６）p.161 p.165

[4] John Cardy『Scaling and Renomalization in Statistical Physics』(Cambridge University Press,1996) p.38

[5] David P.Landau & Kurt Binder『A Guide to Monte Carlo Simulations in Statistical Physics』(Cambridge University Press,2000) p.59

APPENDIX

A パーコレーションモデルのプログラム

```
#include <stdio.h>
#include <stdlib.h>
#define _L 6561

int main ()
{
  static int i,j,k=1,l,m=2,n=0,p,q,Irelabel;
  static int masu[_L][_L],relabel[_L*_L],size[_L*_L/2];
  static int syuen=0,goukei=0,kakuritu=0,maxsize=0,maxmasu,labelmin,s=_L*_L;
  FILE *datafile1,*datafile2,*datafile3,*datafile4,*datafile5,
       *outputp,*outputs;

  /* Creates the percolation */
  printf("syokiti=");
  scanf("%d",&q);
  srand(q);
  printf("kakuritu(%)=");
  scanf("%d",&p);

  for(i=0;i<_L;i++)
      {
        for(j=0;j<_L;j++)
  {
     if(p>rand()%100)
       {
 masu[i][j]=1;
       }
      else
masu[i][j]=0;
  }
      }

  /* First labeling */
```

```
  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  if(masu[i][j]==1)
    {
              if((i==0)&&(j==0))
  {
    masu[0][0]=k;
    k++;
  }

     else if((i==0)&&(j!=0))
{
  if(masu[i][j-1]==0)
    {
      masu[i][j]=k;
      k++;
    }
  else{
    masu[i][j]=masu[i][j-1];}
}
    if((i!=0)&&(j==0))
{
  if(masu[i-1][j]==0)
    {
      masu[i][j]=k;
      k++;
    }
                  else{
    masu[i][j]=masu[i-1][j];}
}

      else if((i!=0)&&(j!=0))
      {
if((masu[i-1][j]==0)&&(masu[i][j-1]==0))
  {
    masu[i][j]=k;
    k++;
```

```
  }
else if ((masu[i-1][j]>0)&&(masu[i][j-1]==0))
  {
    masu[i][j]=masu[i-1][j];
  }
else if((masu[i-1][j]==0)&&(masu[i][j-1]>0))
  {
    masu[i][j]=masu[i][j-1];
  }
else if((masu[i-1][j]>0)&&(masu[i][j-1]>0))
{
  if(masu[i-1][j]>masu[i][j-1])
    {
      masu[i][j]=masu[i][j-1];
    }

                  else if(masu[i-1][j]<=masu[i][j-1])
    {
      masu[i][j]=masu[i-1][j];
    }
}
      }
    }
}
    }

  datafile2=fopen("perc21.dat","w");

  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  fprintf(datafile2,"%3d",masu[i][j]);
}
      fprintf(datafile2,"\n");
    }

  fclose(datafile2);
```

```
  /* Corrects labeling */
do  {

  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  relabel[masu[i][j]]=masu[i][j];
}
    }

 for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  if(masu[i][j]>0)
    {
      labelmin=relabel[masu[i][j]];
      if(i!=0 && masu[i-1][j]>0 && relabel[masu[i-1][j]]<labelmin)
{
  labelmin=relabel[masu[i-1][j]];
}
      if(j!=0 && masu[i][j-1]>0 && relabel[masu[i][j-1]]<labelmin)
{
  labelmin=relabel[masu[i][j-1]];
}
      if(i!=_L-1 && masu[i+1][j]>0 && relabel[masu[i+1][j]]<labelmin)
{
  labelmin=relabel[masu[i+1][j]];
}
      if(j!=_L-1 && masu[i][j+1]>0 && relabel[masu[i][j+1]]<labelmin)
{
  labelmin=relabel[masu[i][j+1]];
}

      relabel[relabel[masu[i][j]]]=labelmin;
      if(i!=0 && masu[i-1][j]>0)
{
  relabel[relabel[masu[i-1][j]]]=labelmin;
```

```
}
      if(j!=0 && masu[i][j-1]>0)
{
  relabel[relabel[masu[i][j-1]]]=labelmin;
}
      if(i!=_L-1 && masu[i+1][j]>0)
{
  relabel[relabel[masu[i+1][j]]]=labelmin;
}
      if(j!=_L-1 && masu[i][j+1]>0)
{
  relabel[relabel[masu[i][j+1]]]=labelmin;
}
    }
}
    }

 /* relabeling */
 Irelabel=0;
 for(i=0;i<_L;i++)
   {
     for(j=0;j<_L;j++)
       {
          while(masu[i][j]>relabel[masu[i][j]])
            {
              masu[i][j]=relabel[masu[i][j]];
              Irelabel++;
    }
       }
   }

} while(Irelabel>0);

  datafile3=fopen("perc22.dat","w");

  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
```

```
  fprintf(datafile3,"%3d",masu[i][j]);
}
      fprintf(datafile3,"\n");
    }

  fclose(datafile3);

 /* Finds the maximum cluster */
 for(i=0;i<_L*_L/2;i++) size[i]=0;

 for(i=0;i<_L;i++)
  {
    for(j=0;j<_L;j++)
      {
if(masu[i][j]>0){
  size[masu[i][j]]++;}
      }
  }

 for(i=0;i<_L;i++)
  {
    for(j=0;j<_L;j++)
      {
if(size[masu[i][j]]>maxsize)
  {
    maxsize=size[masu[i][j]];
    maxmasu=masu[i][j];
  }
      }
  }

 for(i=0;i<_L;i++)
   {
     for(j=0;j<_L;j++)
       {
          if(masu[i][j]==maxmasu)
    {
       masu[i][j]=1;
    }
```

```
  else
    {
       masu[i][j]=0;
    }
       }
   }

 /* Outputs the cluster form */
  datafile4=fopen("perc3.dat","w");

  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  fprintf(datafile4,"%d\n",masu[i][j]);
  /*printf("%3d",masu[i][j]);*/
}
      /* printf("\n");*/
    }

  fclose(datafile4);

  /* Outputs the cluster position */
  datafile5=fopen("perc4.dat","w");

  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  if(masu[i][j]>0)
    {
      goukei++;
      fprintf(datafile5,"%d %d\n",i,j);
    }
}
    }

  fclose(datafile5);
```

```
  /* Finds syuen around the cluster */
  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  if(masu[i][j]==0)
    {
      if(i==0 && 0<j && j<_L-1 &&(masu[i][j-1]==1 || masu[i][j+1]==1
                 || masu[i+1][j]==1)){
masu[i][j]=m; m++;}

                 if(i==_L-1 && 0<j && j<_L-1 &&(masu[i][j-1]==1 || masu[i][j+1]==1
                    || masu[i-1][j]==1)){
masu[i][j]=m; m++;}

                 if(j==0 && 0<i && i<_L-1 &&(masu[i-1][j]==1 || masu[i+1][j]==1
                    || masu[i][j+1]==1)){
masu[i][j]=m; m++;}

                 if(j==_L-1 && 0<i && i<_L-1 &&(masu[i-1][j]==1 || masu[i+1][j]==1
                    || masu[i][j-1]==1)){
masu[i][j]=m; m++;}

      if(i!=0 && masu[i-1][j]==1){
masu[i][j]=m; m++;}

      if(j!=0 && masu[i][j-1]==1){
masu[i][j]=m; m++;}

      if(i!=_L-1 && masu[i+1][j]==1){
masu[i][j]=m; m++;}

      if(j!=_L-1 && masu[i][j+1]==1){
masu[i][j]=m; m++;}
    }

  if(masu[i][j]==1)
    {
      if(i==0 || i==_L-1 || j==0 || j==_L-1){
```

```
n++;}

      if((i==0 && j==0)||(i==_L-1 && j==0)||(i==0 && j==_L-1)
                ||(i==_L-1 && j==_L-1)){
n++;}
    }
}
    }

  /* Outputs the syuen cluster form */
  outputs=fopen("syuen0.dat","w");

  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  fprintf(outputs,"%3d",masu[i][j]);
}
      fprintf(outputs,"\n");
    }

  fclose(outputs);

  for(i=0;i<_L;i++)
    {
      for(j=0;j<_L;j++)
{
  if(masu[i][j]>1){
    masu[i][j]==0; n++;}
}
    }

  /* Outputs the cluster syuen & kakuritu */
  syuen=n;
  kakuritu=(float)goukei/s*100;
  printf("s=%d k=%d\n",syuen,kakuritu);

  outputp=fopen("kakuritu0.dat","w");
  fprintf(outputp,"%d %d\n",p,kakuritu);
```

```
  fclose(outputp);

}
```

B　繰り込み群の方法のプログラム（３行３列）

```
#include <stdio.h>
#define _L 6561

int main()
{
  int i,j,k=0,l=_L,m=2,n=0,s,syuen=0,goukei=0,kakuritu=0;
  static int masu[_L][_L],MASU[_L][_L],sum[_L][_L];
  FILE *input,*outputs,*outputm,*outputr,*outputk;
  char filename[100],fileno[10];

  /* Inputs the cluster form */
  input=fopen("perc3.dat","r");

  for(i=0;i<l;i++)
    {
      for(j=0;j<l;j++)
{
  fscanf(input,"%d",&masu[i][j]);
}
    }

  fclose(input);


  outputs=fopen("syuen.dat","w");
  outputm=fopen("menseki.dat","w");
  outputk=fopen("kakuritu.dat","w");

  for(k=0;l>1;k++){
  /* Syokika */
    m=2; n=0; s=l*l;syuen=0; goukei=0; kakuritu=0;

  /* Finds syuen around the cluster */
  for(i=0;i<l;i++)
    {
      for(j=0;j<l;j++)
```

```
{
  if(masu[i][j]==0)
    {
      if(i==0 && 0<j && j<l-1 &&(masu[i][j-1]==1 || masu[i][j+1]==1
                || masu[i+1][j]==1)){
masu[i][j]=m; m++;}

              if(i==l-1 && 0<j && j<l-1 &&(masu[i][j-1]==1 || masu[i][j+1]==1
                || masu[i-1][j]==1)){
masu[i][j]=m; m++;}

              if(j==0 && 0<i && i<l-1 &&(masu[i-1][j]==1 || masu[i+1][j]==1
                || masu[i][j+1]==1)){
masu[i][j]=m; m++;}

              if(j==l-1 && 0<i && i<l-1 &&(masu[i-1][j]==1 || masu[i+1][j]==1
                || masu[i][j-1]==1)){
masu[i][j]=m; m++;}

      if(i!=0 && masu[i-1][j]==1){
masu[i][j]=m; m++;}

      if(j!=0 && masu[i][j-1]==1){
masu[i][j]=m; m++;}

      if(i!=l-1 && masu[i+1][j]==1){
masu[i][j]=m; m++;}

      if(j!=l-1 && masu[i][j+1]==1){
masu[i][j]=m; m++;}
    }

  if(masu[i][j]==1)
    {
      if(i==0 || i==l-1 || j==0 || j==l-1){
n++;}

      if((i==0 && j==0)||(i==l-1 && j==0)||(i==0 && j==l-1)
                ||(i==l-1 && j==l-1)){
```

```
n++;}
    }
}
    }

  for(i=0;i<l;i++)
    {
      for(j=0;j<l;j++)
{
  if(masu[i][j]>1){
    n++; masu[i][j]=0;}
}
    }

  /* Outputs the cluster position number */
  strcpy(filename,"renoma");
  sprintf(fileno,"%d",k);
  strcat(filename,fileno);
  strcat(filename,".dat");

  outputr=fopen(filename,"w");

  for(i=0;i<l;i++)
    {
      for(j=0;j<l;j++)
{
  if(masu[i][j]==1)
    {
      fprintf(outputr,"%d %d\n",i,j);
      goukei++;
    }
}
    }

  fclose(outputr);

  kakuritu=(float)goukei/s*100;

  /* Outputs the cluster syuen */
```

```
  syuen=n;
  if(syuen==0){
    fprintf(outputs,"# ");}
  fprintf(outputs,"%d %d\n",k,syuen);

  /* Outputs the cluster menseki */
  if(goukei==0){
    fprintf(outputm,"# ");}
  fprintf(outputm,"%d %d\n",k,goukei);

  /* Outputs ther cluster kakuritu */
  if(kakuritu==0){
    fprintf(outputk,"# ");}
  fprintf(outputk,"%d %d\n",k,kakuritu);

 /* Renomalizes the cluster */
  m=0; n=0;
  for(i=0;i<l;i++)
    {
      for(j=0;j<l;j++)
{
  if(i%3==0 && j%3==0)
    {
      m=i/3;
      n=j/3;

      sum[m][n]=masu[i][j]+masu[i][j+1]+masu[i][j+2]
                          +masu[i+1][j]+masu[i+1][j+1]+masu[i+1][j+2]
                          +masu[i+2][j]+masu[i+2][j+1]+masu[i+2][j+2];

      if(sum[m][n]>4)
{
  MASU[m][n]=1;
}
              else
MASU[m][n]=0;
    }
}
    }
```

```
  /* okikae */
  l=l/3;
  for(i=0;i<l;i++)
    {
      for(j=0;j<l;j++)
{
  masu[i][j]=MASU[i][j];
}
    }
  }

  fclose(outputs);
  fclose(outputm);
  fclose(outputk);

}
```