Computational Techniques in Theoretical Physics
Lecture Notes, January 1998

Lecturer: Drs Wang Jian-Sheng/Chen Kan/Zhang Fei — Computational Science

25 lectures, Tue/Fri 11am, Rm S16-07-31 — 6 tutorials (Wed 1 pm) + 6 labs (Wed morning)

Contents

Chapter 1 Introduction

It is widely believed now that physics consists of three major branches: theoretical, experimental, and computational. The computational aspect becomes more and more important. Computational physics has the flavors of both theoretical physics and experimental physics. One must have a very good theoretical background to studied a subject by computational methods. A good computational method often comes from a thorough theoretical analysis. On the other hand, the analyses of results are not much different from analyzing experimental data. Computational methods become advantageous when (1) the problem at hand is too difficult to do analytically; (2) an approximate theoretical result may not be reliable, and it is necessary to check with a different method; (3) an experiment is not feasible or expensive.

Computational physics is a very large field – any calculations using computers extensively could be called computational physics. In this course, we'll concentrate on a few well established methods and a few well studied areas, in particular of condensed matter physics. However, some of our materials are also active research topics. As for the contents of computational physics, we can divide it into roughly two areas, that of numerical analysis and that of computer simulation. Numerical methods include solving linear equations, eigenvalue problems, solving differential equations and partial differential equations, etc. They are very useful and important in many areas of physics. Computer simulations are methods that try to model the physical world directly, rather than solving the equations that governing the physical processes. Our course will be mostly computer simulations. A large class of problems is of stochastic in nature. They are most suited for computer simulations. The topics of percolation, and Monte Carlo methods in statistical physics problems are of this type. The division of the two types of methods is not that clear cut. We could say that molecular dynamics is just solving a set of ordinary differential equations, so the method is a deterministic numerical method. But looked it globally, it is a simulation, since we are interested in the general behavior of the system, rather than individual particles. It just happens that the underline particles obey Newton's law deterministically. In this course, we'll concentrate on physical concepts and simulation techniques. And basic knowledge of numerical methods is assumed.

In this course, I require that you do your tutorial assignment before the tutorial class. You will have a strong impression by doing the problems and I can get feedback promptly from you.

Chapter 2 Percolation and Other Simulation Models

Percolation problem is one of the few simple problems ideal for computer simulation studies. Before going into percolation theory, we first review methods of generating random numbers.

1. Random Number Generators

[Mostly excerpt from Knuth, *the Art of Computer Programming*, Vol 2].

Numbers that are "chosen at random" are useful in many different kinds of applications. In computer simulation of physics problems, we often need uniformly distributed random variables or random numbers.

A truly random variable cannot be generated by computer using deterministic methods. But we can generate random number sequences which are fairly close to the properties of a random variable for most practical applications. Reference 2 and 3 give a much detailed discussions on random number generators.

In a sense, there is no such thing as a random number; for example, is 2 a random number? Rather, we speak of a *sequence of independent random numbers* with a specified *distribution*, and this means loosely that each number was obtained merely by chance, having nothing to do with other numbers of the sequence, and that each number has a specified probability of falling in any given range of values.

A *uniform* distribution on a finite set of numbers (integers) is one in which each possible number is equally probable. A distribution is generally understood to be uniform unless some other distribution is specifically mentioned.

Each of the ten digits 0 through 9 will occur about $\frac{1}{10}$ of the time in a (uniform) sequence of random digits. Each pair of two successive digits should occur about $\frac{1}{100}$ of the time. Yet if we take a truly random sequence of a million digits, it will not always have exactly 100,000 zeros, 100,000 ones, etc. In fact, chances of this are quite slim; a *sequence* of such sequences will have this character on the average.

Any specified sequence of a million digits is equally as probable as the sequence consisting of a million *zeros*. Thus, if we are choosing a million digits at random and if the first 999,999 of them happen to come out to be zero, the chance that the final digit is zero is till exactly $\frac{1}{10}$, in a truly random situation. These statements seem paradoxical to many people, but there is really no contradiction involved.

The history of generating random numbers is interesting. At first, people who needed random numbers in their scientific work would draw balls out of a "well-stirred urn" or would roll dice or deal out cards. A table of over 40,000 random digits, "taken at random from census reports," was published in 1927 by L. H. C. Tippett. Since then, a number of devices have been built to generate random numbers mechanically; the first such machine was used in 1939 by M. G. Kendall and B. Babington-Smith to produce a table of 100,000 random digits, and in 1955 the RAND Corporation published a widely used table of a million random digits obtained with the help of another special device. A famous random-number machine called ERNIE has been used to pick the winning numbers in the British Premium Bonds lottery.

Shortly after computers were introduced, people began to search for efficient ways to obtain random numbers within computer programs. A table can be used, but this method is of limited utility because of the memory space and input time requirement, because the table may be too short, and because it is a bit of nuisance to prepare and maintain the table. Machines such as ERNIE might be attached to the computer, but this would be unsatisfactory since it would be impractical to reproduce calculations exactly a second time when checking out a program; and such machines have tended to suffer from malfunctions that are difficult to detect.

The inadequacy of these methods led to an interest in the production of random numbers using the arithmetic operations of a computer. John von Neumann first suggested this approach in about 1946, using the "middle-square" method. His idea was to take the square of the previous random number and to extract the middle digits; for example, if we are generating 10-digit numbers and the previous value as 5772156649, we square it to get

$$33317792380594909201,$$

and the next number is therefore 7923805949.

There is a fairly obvious objection to this technique: how can a sequence generated in such a way be random, since each number is completely determined by its predecessor? The answer is that this sequence *isn't* random, but it appears to be. In typical applications the actual relationship between one number and its successor has no physical significance; hence the nonrandom character is not really undesirable. Intuitively, the middle square seems to be fairly good scrambling of the previous number.

Sequences generated in a deterministic way such as this are usually called *pseudo-random* or *quasi-random* sequences in the highbrow technical literature, but here we shall simply call them random sequences, with the understanding that they only *appear* to be random. Being "apparently random" is perhaps all that can be said about any random sequence anyway. Random numbers generated deterministically on computers have worked quite well in nearly every application, provided that a suitable method has been carefully selected. Of course, deterministic sequences aren't always the answer; they certainly shouldn't replace ERNIE for the lotteries.

Von Neumann's original "middle-square method" has actually proved to be a comparatively poor source of random numbers. The danger is that the sequence tends to get into a rut, a short cycle of repeating elements. For example, if zero ever appears as a number of the sequence, it will continually perpetuate itself.

Many random number generators in use today are not very good. There is a tendency for people to avoid learning anything about such subroutines; quite often we find that some old method that is comparatively unsatisfactory has blindly been passed down from one programmer to another, and today's users have no understanding of its limitations. We shall see that it is not difficult to learn the most important facts about random number generators and their proper use.

1–1. Linear congruential method

By far the most popular random number generators in use today are special cases of the following scheme. We choose four "magic numbers":

$$\begin{array}{rll} m, & \text{the modulus;} & m > 0. \\ a, & \text{the multiplier;} & 0 < a < m. \\ c, & \text{the increment;} & 0 \le c < m. \\ X_0, & \text{the starting value;} & 0 \le X_0 < m. \end{array}$$

The desired sequence of random numbers X_n is then obtained by setting

$$X_{n+1} = (aX_n + c) \bmod m, \qquad n \ge 0, \tag{1}$$

where a, c, and m are properly chosen integer constants. This is called a *linear congruential sequence*. Taking the remainder mod m is somewhat like determining where a ball will land in a spinning roulette wheel.

For example, the sequence obtained when $m = 10$ and $X_0 = a = c = 7$ is

$$7, 6, 9, 0, 7, 6, 9, 0, \ldots$$

As this example shows, the sequence is not always "random" for all choices of m, a, c, and X_0; the principles of choosing the magic numbers appropriately will be outlined.

The above example illustrates the fact that the congruential sequences always "get into a loop"; i.e., there is ultimately a cycle of numbers that is repeated endlessly. This property is common to all sequences having the general form $X_{n+1} = f(X_n)$. The repeating cycle is called the *period*; the above sequence has a period of length 4. A useful sequence will of course have a relatively long period.

The special case $c = 0$ deserves explicit mention, since the number generation process is a little faster when $c = 0$ than it is when $c \neq 0$. The restriction $c = 0$ cuts down the length of the period of the sequence but it is still possible to make the period reasonably long.

Choice of modulus. Our current goal is to find good values for the parameters that define a linear congruential sequence. Let us first consider the proper choice of the number m. We want m to be rather large, since the period cannot have more than m elements. Even if a person wants to generate only random zeros and ones, he should *not* take $m = 2$, for then the sequence would at best have the form ..., 0, 1, 0, 1, 0, 1, ...!

Another factor that influence our choice of m is speed of generation: We want to pick a value so that the computation of $(aX_n + c) \bmod m$ is fast. The modulo operation is one of the basic operation in C or Fortran programming language, but a much faster way is to avoid the modulo operation all together. This is possible if we take m to be a value that is especially convenient, such as the word size of our computer.

Let w be the computer's word size, namely, 2^e on an e-bit binary computer ($e = 16$ on Intel 8086 processor, $e = 32$ on Silicon Graphics and 64 on Cray). The result of an addition operation is usually given modulo m; and multiplication mod w is also quite simple, since the desired result is the lower half of the product. Hence the C statement

```
x = a * x + c;
```

is the same as $a\,x + c$ modulo 2^e, assuming that `a`, `x`, and `c` are declared as `unsigned int`. Quick tricks for modulo $w \pm 1$ also exist if one writes in assembly languages.

Choice of multiplier. In this subsection we shall show how to choose the multiplier a so as to give the period of *maximum length*. A long period is essential for any sequence that is to be used as a source of random numbers; indeed, we would hope that the period contains considerably more numbers than will ever be used in a single application. Therefore we shall concern ourselves in this section with the question of period length. The reader should keep in mind, however, that a long period is only one desirable criterion for the randomness of our sequence. For example, when $a = c = 1$, the sequence is simply $X_{n+1} = (X_n + 1) \bmod m$, and this obviously has a period of length m, yet it is anything but random. Other considerations affecting the choice of a multiplier will be given later in this chapter.

Since only m different values are possible, the period surely cannot be longer than m. Can we achieve the maximum length, m? The example above shows that it is always possible, although the choice $a = c = 1$ does not yield a desirable sequence. Let us investigate all possible choices of a, c and X_0 that give a period of length m. It turns out that all such values of the parameters can be characterized very simply; when m is the product of distinct primes, only $a = 1$ will produce the full period, but when m is divisible by a high power of some prime there is considerable latitude in the choice of a. The following theorem makes it easy to tell if the maximum period is achieved.

Theorem A. The linear congruential sequence defined by m, a, c, and X_0 has period length m if and only if

i) c is relatively prime to m;

ii) $b = a - 1$ is a multiple of p, for every prime p dividing m;

iii) b is a multiple of 4, if m is a multiple of 4.

Because of the strong condition for obtaining the period m, the random number generators in use typically have period few times smaller than m. The details on the period length are quite complicated. For these of you interested in the theory of random numbers, please refer to Knuth's book. Here we introduce some commonly used parameters for linear congruential generators. For the random number generators with period 2^{30}, it takes only a hour to finish the sequence (the full period) on our workstations.

a	m	c	*period*
7^5	$2^{31} - 1$	0	$2^{31} - 2$
1664525	2^{32}	1013904223	2^{32}
69069	2^{32}	0	2^{30}
6364136223846793005	2^{64}	1	2^{64}

As indicated earlier, Eq. (1) can be efficiently implemented if $m = 2^e$ on an e-bit computer. In such a case, the modulo operation is really unnecessary (and slow) since it is automatic. In multiplication, if the result is larger than the maximum possible value in a computer, the higher order bits are truncated. For example, in C we could write two lines of code

```
x = 1664525U * x + 1013904223U;
r = x * 2.328306437e-10;
```

each time a random number r in the range $(0, 1)$ is needed. The variable `x` must be `unsigned` and `r` could be `double` or `float`. It is more efficient to use the uniformly distributed random integer x between 0 and $2^{32} - 1$ directly.

C compilers supply a standard random integer function, `rand()` (header file `<stdlib.h>`). Many implementations give only a 16-bit random number, which is not acceptable for Monte Carlo calculation, and should be avoided. A better choice is the `drand48()` which returns a `double` and is generated with 48-bit integers. The 64-bit random number generator above can be implemented with `long long int` type in SGI workstations.

1–2. Tausworthe method

Another class of random number generators uses several previous random numbers to generate the next one.

$$X_n = X_{n-p} \oplus X_{n-q}, \tag{2}$$

where the symbol $\oplus$ is for logical sum, or exclusive OR. And X_n can be a bit. In this case, the above formula generates a sequence of random bit. Or X_n can be an integer. The operation $\oplus$ is performed bitwise. This generates random integers. Possible choice of the pair (p, q) is (250, 103), or (1279, 1063). The first random number generator is called R250, because it needs 250 random numbers to begin with. It was believed that Tausworthe method is better than linear congruential method, because of the extremely long period $2^p - 1$. However, recently people discovered short correlations in this random number generator. A good implementation of R250 is the following code

```
k = (k + 1) & 255;
x[k] = x[(k-103) & 255] ^ x[(k-250) & 255];
```

The random numbers are stored in an array with 256 elements, indexed from 0 to 255. Each time the value of index `k` is changed, it is masked with 255 (the least significant byte). This is a trick to keep the array indices inside bound. The value `k` after the first statement is the index of current, newly generated random number. The new number `x[k]` should be used in calculation. The initial 256 random integers must be generated with some other random number generators with care. These random number seeds together with the index `k` can be stored in a file. The file is read in when the program starts, and is overwritten back when the program finishes. Because of this extra book-keeping, it is not as convenient as the simple linear congruential method.

Other operations like multiplication or addition are also used instead of exclusive OR. However, the properties of the Tausworthe sequence and its generalization are less understood.

1–3. Statistical Tests

Our main purpose is to obtain sequences that behave as if they are random. So far we have seen how to make the period of the a sequence so long that for practical purposes it never will repeat; this is an important criterion, but it by no means guarantees that the sequence will be useful in applications. How then are we to decide whether a sequence is sufficiently random?

If we were to give some man a pencil and paper and ask him to write down 100 random decimal digits, chances are very slim that he will give a satisfactory result. People tend to avoid things that seem nonrandom, such as pairs of equal adjacent digits (although about one out of every 10 digits should equal its predecessor). And if we would show someone a table of truly random digits, he would quite probably tell us they are not random at all; his eye would spot certain apparent regularities.

According to Dr. I. J. Matrix, “Mathematicians consider the decimal expansion of π a random series, but to a modern numerologist it is rich with remarkable patterns.” Dr. Matrix has pointed out, for example, that the first repeated two-digit number in π’s expansion is 26, and its second appearance comes in the middle of a curious repetition pattern:

$$3.14159265358979323846264338327950$$

After listing a dozen or so further properties of these digits, he observed that π, when correctly interpreted, conveys the entire history of human race!

We all notice patterns in our telephone numbers, license numbers, etc., as aids to memory. The point of these remarks is that we cannot be trusted to judge by ourselves whether a sequence of numbers is random or not. Some unbiased mechanical tests must be applied.

The theory of statistics provides us with some quantitative measures for randomness. If a sequence behaves randomly with respect to test $T_1, T_2, \ldots, T_n$, we cannot be sure in general that it will not be a miserable failure when it is subjected to a further test T_{n+1}; yet each test gives us more and more confidence in the randomness of the sequence. In practice, we apply about half a dozen different kinds of statistical tests to a sequence, and if it passes these satisfactorily we consider it to be random—it is then presumed innocent until proven guilty.

A general test procedure called chi-square test is perhaps the best known of all statistical tests. For the linear congruential random number generators, they must pass the spectral test. One of the shortcoming of linear congruential random number generators is that the successive points (X_n, X_{n+1}) plotted in two dimensions form a grid of lines. Similarly, triplets (X_n, X_{n+1}, X_{n+2}) fall on families of parallel planes. Spectral test can be used to check the seriesness of such effect. Detail methods of tests will be omitted here. Please refer to Knuth if your are interested in testing random numbers.

1–4. Summary

We have covered some topics on random numbers. Perhaps the main question in many reader's minds will be, "What is a simple, virtuous generator I can use in my programs in order to have a reliable source of random numbers?"

The detailed investigation suggest that the following procedure gives the "nicest" and "simplest" random number generator for the machine language of most computers: At the beginning of the program, set an integer variable X to some value X_0. This variable X is to be used only for the purpose of random number generation. Whenever a new random number is required by the program, set

$$X \leftarrow (aX + c) \bmod m,$$

and use the new value of X as the random value. It is necessary to choose X_0, a, c and m properly, and to use the random numbers wisely, according to the following principles:

i) The "seed" number X_0 many be chosen arbitrarily. If the program is run several times and a different source of random numbers is desired each time, set X_0 to the last value attained by X on the preceding run; or (if more convenient) set X_0 to the current date and time. If the program may need to be rerun later with the *same* random numbers (e.g., when debugging), be sure to print out X_0 if it isn't otherwise known.

ii) The number m should be large, say at least 2^{30}. It may conveniently be taken as the computer's word size, since this makes the computation of $(aX+c) \bmod m$ quite efficient. The computation of $(aX+c) \bmod m$ must be done exactly, with no roundoff error.

iii) If m is a power of 2 (i.e., if a binary computer is being used), pick a so that $a \bmod 8 = 5$. This choice of a together with the choice of c given below ensures that the random number generator will produce all m different possible values of X before it starts to repeat.

iv) The multiplier a should preferably be chosen between $0.01m$ and $0.99m$, and its binary or decimal digits should not have a simple, regular pattern. Various testing should be done if the random number generator is to be used extensively.

v) The value of c is immaterial when a is a good multiplier, except that c must have no factor in common with m. Thus we may choose $c = 1$ or $c = a$.

vi) The least significant (right-hand) digits of X are not very random, so decisions based on the number X should always be influenced primarily by the most significant digits. It is generally best to think of X as a random fraction X/m between 0 and 1, that is, to visualize X with a decimal point at its left, rather than to regard X as a random integer between 0 and $m-1$. To compute a random integer between 0 and $k-1$, one should multiply by k and truncate the result.

vii) One should always aware of the limitation of random numbers generated by recursion relation.

TUTORIAL PROBLEM SET 1.

1. Just for Fun

Suppose that you wish to obtain a decimal digit at random, not using a computer. Which of the following methods would be suitable?

a) Open a telephone directory to a random place (i.e., stick your finger in it somewhere) and use the units digit of the first number found on the selected page.

b) Same as (a), but use the units digit of the *page* number.

c) Roll a die that is in the shape of a regular icosahedron, whose twenty faces have been labeled with the digits 0, 0, 1, 1, ..., 9, 9. Use the digit that appears on top, when the die comes to rest. (A felt table with a hard surface is recommended for rolling dice.)

d) Expose a geiger counter to a source of radioactivity for one minute (shielding yourself) and use the units digit of the resulting count. Assume that the geiger counter displays the number of counts in decimal notation, and that the count is initially zero.

e) Glance at your wristwatch; and if the position of the second-hand is between $6n$ and $6(n+1)$ second, choose the digit n.

f) Ask a friend to think of a random digit, and use the digit he names.

g) Ask an enemy to think of a random digits, and use the digit he names.

h) Assume that 10 horses are entered in a race and that you know nothing whatever about their qualifications. Assign to these horses the digits 0 to 9, in arbitrary fashion, and after the race use the winner's digit.

2. Theorem A

Check each of the random number generators on page 4 against the Theorem A, assert for each case whether the period is m.

3. Prove a Formula

Given the linear congruential sequence defined by

$$X_{n+1} = (aX_n + c) \bmod m, \qquad n \geq 0,$$

show

$$X_{n+k} = \big(a^k X_n + (a^k - 1)c/(a-1)\big) \bmod m, \qquad k \geq 0, \quad n \geq 0.$$

The equation expresses the $(n+k)$th term directly in terms of the nth term. It follows that the subsequence consisting of every kth term of X_n is another linear congruential sequence, having the multiplier $a^k \bmod m$ and the increment $\big((a^k-1)c/(a-1)\big) \bmod m$.

4. Interesting Fact

If the modulus $m = w = 2^e$, the low-order bits are much less random than the high-order bits. More precisely, the least significant bit is either constant or strictly alternating. The last two bits cannot have a period of more than 4; and the low-order four bits has a period of length 16 or less. To demonstrate this mathematically, let define

$$Y_n = X_n \bmod d,$$

where d is a divisor of m, show that

$$Y_{n+1} = (aY_n + c) \bmod d.$$

That is Y_n is also a linear congruential sequence with modulus d, multiplier a, and increment c.

LABORATORY EXERCISE SET 1.

1. MIDDLE-SQUARE METHOD

Make a complete examination of the middle-square method in the case of two digit decimal numbers. (a) We might start the process out with any of the 100 possible values 00, 01, ..., 99. How many of these values lead ultimately to the repeating cycle 00, 00, ...? [Example: Starting with 43, we obtain the sequence 43, 84, 05, 02, 00, 00, 00, ...] (b) How long is the longest cycle? (c) What starting value or values will give the largest number of distinct elements before the sequence repeats? Of course, you want to write a program to answer these questions.

2. LINEAR CONGRUENTIAL RANDOM NUMBER GENERATORS

Implement in C the linear congruential random number generators with (a) $a = 7^5 = 16807, m = 2^{31} - 1$, $c = 0$; (b) $a = 6364136223846793005$, $m = 2^{64}$, and $c = 1$. Print out the first five numbers starting from the seed $X_0 = 1$. Check that your code is correct by comparing your results with other methods of computations (e.g. with Mathematica or Maple results). A tricky point is how to do 64-bit computations on the 32-bit SGI workstations. Determine the speed (in terms of μsec per number) of your random number generators. Submit only the codes of your random numbers and first 5 numbers each. Your program should be correct, well-documented, simple, and fast.

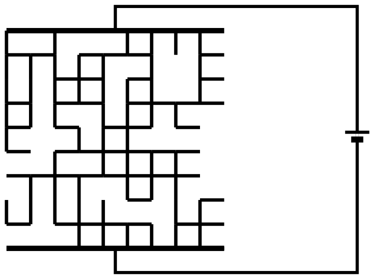
3. READING

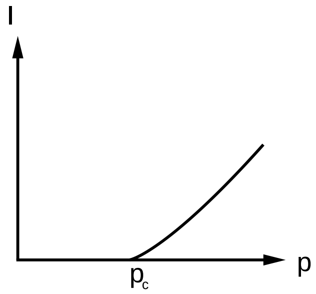
Read pages 274–286, Chapter 7, Random Numbers, in *Numerical Recipes in C*, W. H. Press, etc. Write a half page summary to aid your understanding of the text. Don't waste time typesetting; just scribble down by hand. Submit the notes.

2. What Is Percolation

Percolation arises from a number of problems in condensed matter physics. Here we present few examples.

(1) Consider a mixture of metallic and non-metallic material. A voltage is applied on the sample, and we want to know the current passing through the sample. We imagine that the mixture consists of small grains. For simplicity, we model the material as small conducting segments distributed among non-conducting background. We also assume that the conducting segments can only sit on a regular lattice, e.g., a two-dimensional square lattice.





Our question is how physical quantity like current or conductivity varies with the concentration of the conducting component. The current as a function of the fraction p of conducting bonds will qualitatively look like the Figure shown above.

(2) The second example is dilute magnet at low temperatures. A ferromagnetic material shows magnetization at low enough temperatures. When the material is diluted with nonmagnetic material, the magnetization may disappear. Here the question is how magnetization depends on the concentration of dilution.

(3) The third example is the origin of the word *percolation*. Consider a fluid trying to pass through a porous material. The porous media (e.g., rock) have many small random channels. The question is whether the fluid can flow through the material. The answer, of course, depends on the concentration and nature of the channels. If there are few open short channels distributed at random, the fluid cannot pass through, or in other words, the system does not percolate. When more channels are there, the fluid can pass through the solid – we say that the system is percolating. At some particular concentration the system changes its behavior, that particular concentration or situation is called percolation threshold.

3. Mathematical Model for Percolation

The spirit of physics (or perhaps more prevailing in condensed matter physics) is to simplify the problem as far as possible, retaining only the essential feature. We'll propose a mathematical model for percolation which takes into account only the most important feature, neglecting non-essential details. It turns out that certain aspects of the problem do not depend on the details. This is quite remarkable and is called universality.

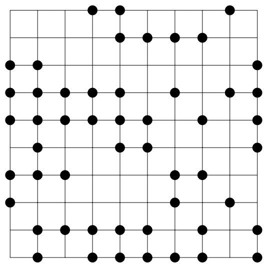
3–1. Site percolation

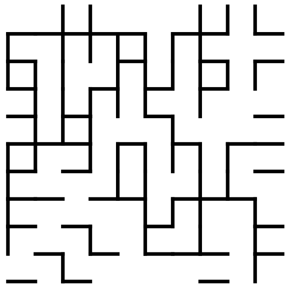
Let us consider a two-dimensional problem. Three or higher dimensional cases are similar. Consider a system of two-dimensional square lattice. For computer simulation, we also work on a finite lattice. Let's take the size of the system to be $L \times L$. Each site may be occupied or empty. We may denote this by an occupation number $n(i, j) = 0$, or 1, where $0 < i, j \leq L$ specifies the location of a lattice site. The occupation number zero indicates an empty site and one for an occupied site. The collection $\{n(i, j)\}$ denotes one particular configuration. In a computer program, we can use a two-dimensional array taking values zero and one to represent a configuration. Since only lattice sites are involved, this is called a site percolation model.

How do we decide which sites to occupy? It is simple. We decide at random. More precisely, we go through each site once and occupy the site with probability p, and consequently, leave the site empty with probability $1 - p$.

3–2. Bond percolation

In bond percolation, instead of sites, bonds are occupied with probability p. A bond connects two neighboring sites. The right figure below is a bond percolation configuration.





3–3. Statistical description

Since probability enters into our description of the site (or bond) percolation, our problem is statistical in nature. We'll consider site percolation in the following. First of all, we want to ask what is the probability that a particular configuration $\{n(i,j)\}$ appears. Since each appearance of an occupied site has probability p independent of other sites, each appearance of an empty site has probability $1-p$, the whole configuration appears with probability

$$p^m(1-p)^{L^d-m}, \tag{3}$$

where $d=2$ is the dimension, $m=\sum n(i,j)$ is the total number of occupied sites. Later on, we'll mention quite often taking averages. For site percolation, the average of certain quantity $A(\{n(i,j)\})$ which depends on the configuration means:

$$\langle A\rangle = \sum_{all\ conf.} A\big(\{n(i,j)\}\big)p^m\big(1-p\big)^{L^d-m}, \tag{4}$$

where the summation is over all possible configurations. We have 2^{L^d} of them. The average can be estimated approximately by Monte Carlo method.

4. Percolation Threshold

One of the quantities that we want to know is the percolation threshold p_c. The percolation threshold is defined as the value of probability p below which there are only finite size clusters in an infinitely large system, above which there is at least one cluster which is infinitely large. A cluster is defined as a set of sites which are connected through nearest neighbors. The size s of a cluster is the number of sites in the cluster.

The percolation threshold value depends on the type of lattice and dimensions. Here are selected percolation thresholds for various lattices (from Ref. 1). 'Site' refers to site percolation and 'bond' to bond percolation. Some of the results are exact, others are obtained approximately.

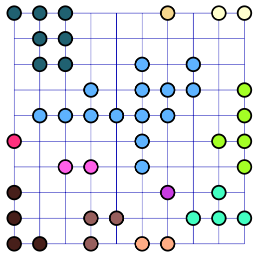
Lattice	Site	Bond
Honeycomb	0.6962	$1-2\sin(\pi/18)$
Square	0.592746	1/2
Triangular	1/2	$2\sin(\pi/18)$
Diamond	0.43	0.388
Simple cubic	0.3116	0.2488
BCC	0.246	0.1803
FCC	0.198	0.119
$d=4$ hypercubic	0.197	0.1601
$d=5$ hypercubic	0.141	0.1182
$d=6$ hypercubic	0.107	0.0942
$d=7$ hypercubic	0.089	0.0787

5. Important Physical Quantities in Percolation

We'll introduce and define several quantities that will be useful in the simulation and analysis of percolation problem.

5–1. Clusters and cluster number n_s

The definition of a cluster in a configuration is somewhat arbitrary. But the following definition is commonly used. A cluster is a collection of sites connected by bonds (for bond percolation) or a collection of nearest neighbor sites (for site percolation). When the sites are at the boundary, we wrap the lattice (like a torus) to decide connectivity. The size s of a cluster is the number of sites in the cluster. Here is a site percolation configuration and clusters.



Site Pecolation at p = 0.5 with Size L = 10

Cluster number is probably the most important quantity conceptually since many other quantities can be derived from it. The cluster number n_s is the *average* number of clusters having exactly size s, normalized by the system size L^d. The cluster number n_s depends on the probability p as well as system size L. But the L-dependence quickly approaches a limit as $L \to \infty$. In the following, we'll discuss the properties of the cluster number in the infinity lattice size limit. That is, we consider a very large system. This is not something artificial. This is physical. The mathematical sites in our percolation model represent atoms or groups of atoms. Since the typical size of atoms or molecules is of order 10^{-8} centimeter, and typical size of a laboratory sample is of order of few centimeter. So, there are typically 10^{23} atoms, which is very large.

5–2. Percolation probability P

The lowercase letter p denotes the probability of occupying a site (concentration of occupied sites). We'll use the uppercase P to denote *percolation probability*. This is also called order parameter of the percolation problem or strength of the infinite network. The percolation probability P is defined as the probability that the origin or any other arbitrarily selected site belongs to an infinite cluster. The infinite cluster is called a percolating cluster since it spans the whole lattice. Clearly for $p < p_c$, the percolation probability $P = 0$, because there exists no infinite cluster. For $p > p_c$, the quantity P becomes positive, and approaches 1 as $p \to 1$. The quantities n_s, p, and P are related:

$$P + \sum_{s=1,2,3,\cdots} n_s s = p. \tag{5}$$

The equation can be understood as follows: a site has probability p being occupied; if it is occupied, it is either on the infinitely large cluster or on a finite size cluster. The probability that the site belongs to the infinite cluster is P, and the probability on a finite cluster of size s is $n_s s$. The later probability can be understood this way – we have $L^d n_s$ clusters of size s; out of total of L^d lattice sites, there are $L^d n_s s$ sites belonging to clusters of size s. If we pick a site at random, we have the probability $n_s s$ picking the sites belonging to a cluster of size s. The probability of the site being occupied is p and is the sum of probabilities on the infinite cluster and on finite clusters of size $s = 1, 2, 3, \cdots$.

5–3. Average cluster size S

The average cluster size is defined through the equation

$$S = \frac{\sum_{s=1}^{\infty} n_s s^2}{\sum_{s=1}^{\infty} n_s s}. \tag{6}$$

The summation runs to infinity. But it does not include contribution from infinitely large percolating cluster. The interpretation is left as an exercise.

5–4. Pair correlation function $g(r)$

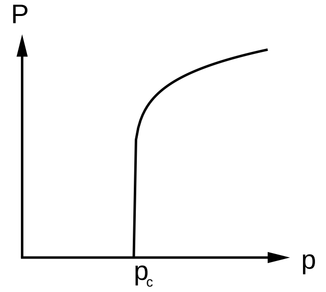
Pair correlation function $g(r)$ measures how two points are related. It is defined as the probability that a site distance r away from an occupied site belongs to the same cluster. This quantity is related to S by

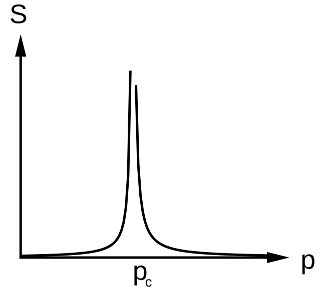
$$\sum_r g(r) = S, \qquad (p < p_c). \tag{7}$$

6. Critical Behavior and Critical Exponents

6–1. Critical exponents

We just introduced various quantities. The qualitative behaviors of the percolation probability P and average cluster size S look like the figures.





An interesting thing happens at the percolation threshold, p_c; this point is also referred as critical point, and the phenomena associated with it are called critical phenomena. The modern theory of critical phenomena is a well-developed theory and a great success of statistical mechanics. K. Wilson won his Nobel Prize for his renormalization group theory of critical phenomena. We'll deal with critical phenomena again in molecular dynamics and Monte Carlo methods later in the course.

Near the critical point p_c the two quantities can be described empirically as

$$P \propto \begin{cases} 0, & p < p_c; \\ (p - p_c)^\beta, & p > p_c,\, p \to p_c. \end{cases} \tag{8}$$

$$S \propto |p - p_c|^{-\gamma}, \quad p \to p_c. \tag{9}$$

Where β and γ are some constants called critical exponents. The proportionality sign $\propto$ takes the usual meaning – the ratio of the left-hand side and the right-hand side approaches a constant as p approaches p_c. This kind of equation or laws that the quantity P or S obeys is called asymptotic law since it takes this form only when p is close to the critical value p_c.

One of the most important result of the theory of critical phenomena (from renormalization group theory) is that the critical exponents like β and γ are universal. More precisely, the value β is the same for a whole class of models and experimental samples independent of the details of the models or samples. The value depends only on the dimensionality of the problem. Two dimensional percolation is different from three dimensional percolation. It depends whether the percolation is short ranged or long ranged. But it does not depend whether the lattice is a square lattice or triangular lattice. The critical exponents are the same on both lattices. The universality of exponents justifies the study of the simplest possible model, and the result also applies to more general situations. One of the tasks of this study is to calculate the critical exponents by computer simulation.

6–2. Scaling assumption of the cluster number

The percolation probability P and average cluster size S have a simple power-law behavior near the critical point p_c. What about the cluster number n_s? This quantity is more complicated because it depends on two variables, the cluster size s and occupation probability p. So we should write more precisely as $n_s(p)$. Except in one dimension and a special lattice (Bethe lattice), there are no exact solutions for the percolation problems. Based on many examples and experience in critical phenomena, the following functional form has been assumed

$$n_s(p) = s^{-\tau} f[(p - p_c)s^\sigma], \qquad p \to p_c,\ s \to \infty. \tag{10}$$

The above equation is called the scaling assumption for the cluster number. The exact values for the exponents τ, σ and the exact form for the function $f(z)$ are not known in general. They must be determined by (computer) experiment or by other methods. In fact, the actual assumption must be tested. People believe that the assumption indeed holds. There has been no rigorous proof of the assumption.

6–3. Exponent relations

We have introduced four different exponents β, γ, τ, and σ. Since P and S can be derived from $n_s(p)$, the exponents are not independent. They are related. This is one of the consequence of the scaling assumption. A test of the exponent equality is also a test for the scaling assumption. From the equation

$$P = p - \sum_{s=1,2,3,\cdots} n_s(p) s, \tag{11}$$

together with the scaling assumption, we have

$$P = p - \sum_{s=1}^{\infty} s^{1-\tau} f\big((p-p_c)s^{\sigma}\big). \tag{12}$$

We'll use a trick to write the equation in the variable $p - p_c$. Since $P = 0$ at $p = p_c$, setting $p = p_c$ and subtracting it from the above equation, we have

$$P = (p-p_c) - \sum_{s=1}^{\infty} s^{1-\tau}\Big[f\big((p-p_c)s^{\sigma}\big) - f(0)\Big]. \tag{13}$$

We'll convert the summation to integral. The error introduced is small near the percolation threshold.

$$P = (p-p_c) - \int_{s=s_0}^{\infty} s^{1-\tau}\Big[f\big((p-p_c)s^{\sigma}\big) - f(0)\Big]\, ds. \tag{14}$$

Only the upper limit matters. The lower limit does not influence the finite result very much. So we set it at some arbitrary value s_0. By changing the integration variable $s = \big(x/(p-p_c)\big)^{1/\sigma}$, we found

$$P = (p-p_c) - (p-p_c)^{\frac{\tau-2}{\sigma}} c \propto (p-p_c)^{\beta}. \tag{15}$$

Comparing the exponents, using the fact that $\beta < 1$, (neglecting $(p-p_c)$ term), we have the first relation

$$\beta = \frac{\tau - 2}{\sigma}. \tag{16}$$

Now we calculate the average cluster size. This quantity diverges at the percolation threshold. The divergence comes from the numerator $\sum s^2 n_s$ since the denominator $\sum s n_s$ simply approaches a constant p_c. To leading order of the divergence term, we can approximate it by p_c and

$$\begin{aligned} S =& \frac{\sum s^2 n_s(p)}{\sum s n_s(p)} \approx \sum \frac{s^2 n_s(p)}{p_c} \\ \propto& \int s^2 n_s(p)\, ds \propto \int s^{2-\tau} f\big((p-p_c)s^{\sigma}\big)\, ds \\ \propto& |p-p_c|^{(\tau-3)/\sigma} \propto |p-p_c|^{-\gamma}. \end{aligned}$$

Again we have changed integration variable, and the value of the integral is just some number independent of p. Thus

$$\gamma = \frac{3-\tau}{\sigma}. \tag{17}$$

Note that in order that both β and γ are positive we need $2 < \tau < 3$.

Here is a list of percolation exponents for $d = 2$, 3, 4, 5, $6-\epsilon$, and for the Bethe lattice.

Exponent	$d=2$	$d=3$	$d=4$	$d=5$	$d=6-\epsilon$	Bethe
β	5/36	0.41	0.64	0.84	$1-\epsilon/7$	1
γ	43/18	1.80	1.44	1.18	$1+\epsilon/7$	1
ν	4/3	0.88	0.68	0.57	$1/2+5\epsilon/84$	1/2
σ	36/91	0.45	0.48	0.49	$1/2+O(\epsilon^2)$	1/2
τ	187/91	2.18	2.31	2.41	$5/2-3\epsilon/14$	5/2
D	91/48	2.53	3.06	3.54	$4-10\epsilon/21$	4

The dimension $d = 6$ is called upper critical dimension, above which the exponents become independent of the dimensionality and take the values of the Bethe lattice results (We'll discuss Bethe lattice in a tutorial). The exponent ν is called correlation length exponent, defined in the next section. And D is the fractal dimension of clusters at the percolation threshold.

7. Cluster Structure

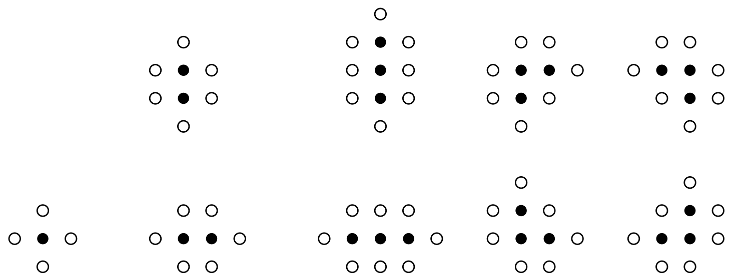
In the above section, we studied global features of percolation problems. In this section, we'll concentrate on the individual clusters.

7–1. Cluster perimeter

We'll consider site percolation. Each individual cluster is called a *lattice animal*. The empty sites immediately adjacent to the cluster (empty neighbors) are called perimeter sites. Let's use t for the number of perimeter sites of a given lattice animal, and s is the total number of occupied sites of the lattice animal. And let g_{st} denote the number of lattice animals (cluster configurations) with size s and perimeter t, then

$$n_s(p) = \sum_t g_{st}\, p^s (1-p)^t, \tag{18}$$

is the average number of s clusters per lattice site. If we have an analytic formula for g_{st}, then we should have solved the percolation problem exactly. Unfortunately, this is not the case except in one dimension. However, we can find n_s for the first few small values of s. For $s = 1$ and dimension $d = 2$, $n_1 = p(1-p)^4$, $g_{1,4} = 1$, all other $g_{st} = 0$. For $s = 2$, $n_2 = 2p^2(1-p)^6$, $g_{2,6} = 2$. And $n_3 = 2p^3(1-p)^8 + 4p^3(1-p)^7$, $g_{3,8} = 2$, $g_{3,7} = 4$. The meaning of the formula correlates closely with the figures shown. With the help from computer, people were able to obtain g_{st} for s upto 30.



The average perimeter t of a cluster is not proportional to the surface area of the cluster. This is because there are lot of holes in a percolation cluster. If we have one hole for, say, every thirty sites we have a perimeter proportional to the number of sites. Thus

$$t \propto s, \qquad (s \to \infty). \tag{19}$$

this result can be proved more rigorously.

7–2. Correlation length

We have introduced the correlation function $g(r)$, which is the probability that a site at distance r from a given occupied site is also occupied and belongs to the same cluster. The function behaves at large distance in the form

$$g(r) \propto e^{-r/\xi}, \qquad p < p_c. \tag{20}$$

For $p > p_c$ it approaches a constant P^2. The quantity ξ is called correlation length. The correlation length can also be defined as

$$\xi^2 = \frac{\sum_r r^2 g(r)}{\sum_r g(r)}. \tag{21}$$

The correlation length measures the typical size of clusters. Near the percolation threshold, clusters become very large, and the correlation length diverges,

$$\xi \propto |p - p_c|^{-\nu}, \tag{22}$$

where ν is called correlation length exponent.

7–3. Cluster radius and fractal dimension

The correlation length is a global quantity for the whole system. Cluster radius measures the dimension of clusters having exactly s sites. Since the clusters are rather irregular, we can only give an average quantity which measures the size of the clusters. The *radius of gyration* R_s is used for this purpose.

$$R_s^2 = \frac{1}{s}\sum_{i=1}^{s} |\mathbf{r}_i - \mathbf{r}_0|^2, \tag{23}$$

where

$$\mathbf{r}_0 = \frac{1}{s}\sum_{i=1}^{s} \mathbf{r}_i \tag{24}$$

is the position of the center of mass of the cluster, and vector $\mathbf{r}_i$ is the position of the i-th site in the cluster. The above formula can also be rewritten as (after a long calculation)

$$R_s^2 = \frac{1}{2s^2}\sum_{i=1}^{s}\sum_{j=1}^{s} |\mathbf{r}_i - \mathbf{r}_j|^2, \tag{25}$$

that is, R_s is proportional to the root-mean-square distance of sites in a cluster.

The clusters at the percolation threshold are fractals, in the sense that the total size s or mass (imagine each occupied site has certain weight) is related to its size as

$$s \propto R_s^D, \qquad p = p_c,\ s \gg 1. \tag{26}$$

D is called the fractal dimension of the clusters. This fractal behavior for large but finite clusters also holds for the largest cluster in a finite system. If L is the linear size of the system with L^d lattice sites, the mass M (or total number of sites) of the largest cluster varies with the system size as

$$M \propto L^D. \tag{27}$$

According to scaling theory, D can be related to other exponents as

$$D = d - \frac{\beta}{\nu}, \quad d \le 6. \tag{28}$$

8. Exact Solution in One Dimension

[Excerpt from Stauffer, *Introduction to Percolation Theory*]

Like so many other problems of theoretical physics, the percolation problem can be solved exactly in one dimension, and some aspects of that solution seem to be valid also for higher dimensions.

Let us study site percolation on an infinitely long linear chain, where 'lattice' sites are placed in fixed distances. Each of these lattice sites is randomly occupied with probability p. A cluster is a group of neighboring occupied sites containing no empty site in between. A single empty site would split the group into two different clusters. In order that the cluster is separated from the other clusters, the site neighboring the left end of the cluster must be empty; and the same is true for the right end of the cluster. Thus for the central cluster consisting of five occupied sites, we need these five sites occupied and their two neighbors empty.

The probability of each site being occupied is p. Since all sites are occupied randomly, the probability of two arbitrary sites being occupied is p^2, for three being occupied is p^3, and for five being occupied is p^5. (This product property of the probabilities is valid only for statistically independent events, as for random percolation). The probability of one end having an empty neighbor is $(1-p)$, and again the two ends are statistically independent. Therefore the total probability, that a fixed lattice site is the left end of a five cluster is $p^5(1-p)^2$.

How many clusters of size five do we have in our chain, if the total chain length is L, with $L \to \infty$, much larger than the cluster length? Every site has the probability $p^5(1-p)^2$ of being the left hand end of such a cluster, and there are L such sites (when we ignore the small number of sites on the end of the whole chain for which the situation is different since there no place is left for five occupied and two empty sites). Thus the total number of five-clusters, apart from effects from the chain ends, is $Lp^5(1-p)^2$. We see that it is practical to talk about the number of clusters per lattice site, which is the total number divided by L

and thus $p^5(1-p)^2$. This normalized cluster number is thus independent of the lattice size L and equals the probability that a fixed site is the end of a cluster.

For clusters containing s sites, we define n_s as the number of such s-clusters per lattice site. In our one-dimensional case, the above consideration for five-clusters is easily generalized to

$$n_s = p^s(1-p)^2, \tag{29}$$

This normalized cluster number is crucial for many of our discussions in two or three dimensions. It equals the probability, in an infinite chain, of an arbitrary site being the left hand end of the cluster. For $p < 1$, the cluster number goes exponentially to zero if the cluster size s goes to infinity.

The probability that an arbitrary site is part of an s-cluster, and not only its left end, is larger by a factor s; for now that site can be any of the s cluster sites. Thus that probability is $n_s s$. Many authors prefer to work with the probability $n_s s$, instead of with the cluster number n_s. To avoid confusion with the probability p we will not introduce a special symbol for $n_s s$ and will work with the cluster number. This cluster number is also more natural quantity if one counts all clusters in a lattice of a fixed large size by a computer simulation.

Where is the percolation threshold? For $p = 1$, all sites of the chain are occupied, and the whole chain constitutes one single cluster. For every p smaller than unity, there will be some holes in the chain where a site is not occupied. Thus a chain of length L will have on average $(1-p)L$ empty sites. For L going to infinity at fixed p, this number is also increasing to infinity. Thus there will be at least one empty site somewhere in the chain, and that means that there is no continuous row of occupied sites, i.e., no one-dimensional cluster, connecting the two ends. In other words, there is no percolating cluster for p below unity. Thus the percolation threshold is unity:

$$p_c = 1. \tag{30}$$

Therefore it is not possible to observe the region $p > p_c$ in one dimension. Only one side of the phase transition is accessible since at least these authors cannot occupy a site with a probability $p > 1$. Nevertheless, this somewhat unusual phase transition has some similarities with percolation in higher dimensions, and also with certain aggregation processes. Thus we will try to squeeze out some more information form this simple result.

The probability that a site belongs to a cluster of size s is $n_s s$, as discussed above. Every occupied site must belong to one cluster since single occupied sites surrounded by empty neighbors are also clusters of size unity. The probability that an arbitrary site belongs to any cluster is therefore equal to the probability p that it is occupied. Thus

$$\sum_{s=1}^{\infty} n_s s = p, \qquad (p < p_c). \tag{31}$$

This law can also be checked directly from expression for n_s and the formula for the geometric series:

$$\begin{aligned}
\sum_{s=1}^{\infty} p^s(1-p)^2 s &= (1-p)^2 \sum_{s=1}^{\infty} p\frac{dp^s}{dp} \\
&= (1-p)^2 p \frac{d\left(\sum_{s=1}^{\infty} p^s\right)}{dp} \\
&= (1-p)^2 p \frac{d\big(p/(1-p)\big)}{dp} \\
&= p.
\end{aligned}$$

For higher dimension Eq. (31) is also valid except that one has to take into account the sites in the infinite cluster separately, if one does not include them in the sum over all cluster sizes. There Eq. (31) is restricted to $p < p_c$; even in one dimension at $p = p_c = 1$ there is only one cluster covering the whole lattice, thus $s = \infty$ and $n_s = 0$, which makes the equation undefined at $p = 1$. (The above trick to calculate a sum by expressing it as a derivative is also useful in other parts of statistical physics.)

How large on average is the cluster we are hitting if we point randomly to a lattice site which is part of a finite cluster? There is a probability $n_s s$ that an arbitrary site (occupied or not) belongs to an s-cluster and a probability $\sum_s n_s s$ that it belongs to any finite cluster. Thus $w_s = n_s s / \sum_s n_s s$ is the probability that the cluster to which an arbitrary occupied site belongs contains exactly s sites. The average cluster size S

which we are measuring in this process of randomly hitting some cluster site is therefore

$$S = \sum_s w_s s = \frac{\sum n_s s^2}{\sum n_s s}. \tag{32}$$

Although we will learn later that different types of averages exist, the term *mean cluster size* is in widespread use for S and will also be used here. (For example, $\sum n_s s / \sum n_s$ is the average cluster size of every cluster, and not every site as in Eq. (32) is selected with equal probability.) We have defined S here in such a way that Eq. (32) is also our definition for higher dimensions provided that the infinite cluster is excluded from the sums.

Let us now calculate this mean cluster size explicitly. The denominator is simply p, as previously shown. The numerator is

$$(1-p)^2 \sum_{s=1}^{\infty} s^2 p^s = (1-p)^2 \left(p\frac{d}{dp}\right)^2 \sum_{s=1}^{\infty} p^s, \tag{33}$$

where the trick from our previous derivation is applied twice in order to calculate sum by using suitable derivatives of easier sums. Thus

$$S = \frac{1+p}{1-p}, \qquad (p < p_c). \tag{34}$$

The mean cluster size diverges if we approach the percolation threshold. We will obtain similar results in more than one dimension. This divergence is very plausible, for if there is an infinite cluster present above the percolation threshold, then slightly below the threshold one already has very large (though finite) clusters. Thus a suitable average over these sizes is also getting very large, if one is only slightly below the threshold.

We may define the *correlation function* or pair connectivity $g(r)$ as the probability that a site a distance r apart from an occupied site belongs to the same cluster. For $r = 0$ that probability $g(0)$ equals unity, of course. For $r = 1$ the neighboring site belongs to the same cluster if and only if it is occupied; this is the case with probability p. For a site at distance r, this site and the $(r-1)$ sites in between this site and the origin at $r = 0$ must be occupied without exception, which happens with probability p^r. Thus

$$g(r) = p^r, \tag{35}$$

for all p and r. For $p < 1$ this correlation function, which is also called a connectivity function goes to zero exponentially if the distance r goes to infinity:

$$g(r) = \exp\left(\frac{-r}{\xi}\right),$$

where

$$\xi = -\frac{1}{\ln p} \approx \frac{1}{p_c - p}. \tag{36}$$

The last equality in Eq. (36) is valid only for p close to $p_c = 1$ and uses the expansion $\ln(1-x) = -x$ for small x. The quantity ξ is called the correlation length and we see that it also diverges at the threshold. We will see in higher dimensions that the correlation length is proportional to a typical cluster diameter. This relation is quite obvious here. The length of a cluster with s sites is $(s-1)$, not much different form s if s is large. Thus the average length ξ varies as the average cluster size S:

$$S \propto \xi, \qquad (p \to p_c). \tag{37}$$

Unfortunately, this relation becomes more complicated in higher dimensions. Rather more generally valid is a relation between the sum over all distances r of the correlation function, and the mean cluster size:

$$\sum_r g(r) = S. \tag{38}$$

The reader who wants to check this and has difficulties should keep in mind that the sum in Eq. (38) not only includes $r = 0, 1, 2, \ldots$, i.e., the neighbors to the right, but also the neighbors to the left. They cannot be treated through $r = -1, -2$, however, since r is a non-negative distance. Thus one should calculate the sum over the right neighbors and the center, $r = 0,\ 1,\ 2,\ \ldots$, multiply it by 2 to take into acount the left part of the lattice, and subtract the contribution unity from the center, which otherwise would be counted twice.

We see from this exact solution for one dimension, that certain quantities diverge at the percolation threshold, and that the divergence can be described by simple power laws like $1/(p_c - p)$, at least asymptotically close to p_c. The same seems true in higher dimensions where the problems have not been solved exactly.

The quantities S and ξ have counterparts for thermal phase transitions. In fluids near their critical point, critical opalescence is observed in light-scattering experiments, since the compressibility (analogous to S) and the correlation length ξ diverge there. The van der Waals equation for fluids in his thesis of 1873 was the first successful theory to describe aspects of such thermal phase transitions.

One may utilize one-dimensional percolation further by calculating the cluster numbers in finite one-dimensional chains. Then one can check the general concepts of finite-size scaling and universality.

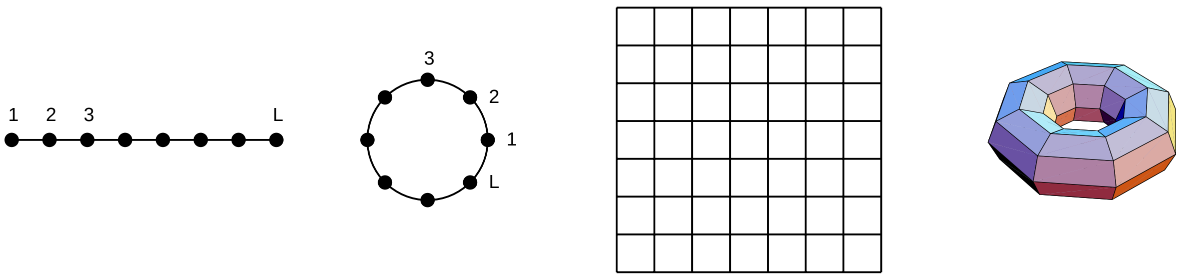
The one-dimensional case is now solved exactly, whereas for the d-dimensional case only small clusters will be treated exactly. There is another case with an exactly known solution, the Bethe lattice.

9. Computer Simulation of Site Percolation

In this section we outline the technique for a computer simulation of the percolation problem, using site percolation as an example. It is almost always true that for a given physical problem, the most important part is physical idea and insight, and good computer algorithms. The difficult part is how to analyze and interpret data. Writing a program and generating a lot of data are the easy part. Very often, one should keep from generating too many data and being flooded by them. Thus a good design on how to handle data and what to compute is essential. The purpose of computer simulations in physics is to understand the underline principle, or discover new principle, or check theoretical prediction. Getting numerical value by itself is not the goal. One must interpret them.

9–1. Periodic boundary condition

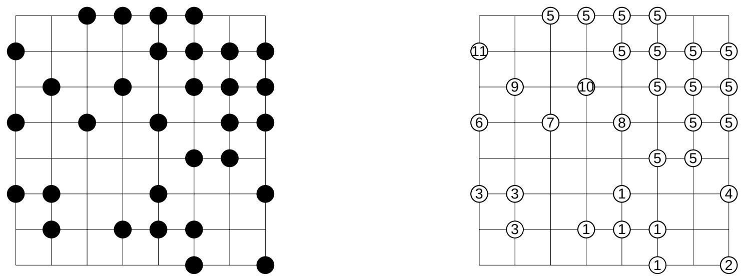
Periodic boundary condition is very often used when dealing with a finite system. Periodic boundary condition usually reduces the influence of finite system. But it can not really eliminate the effect of finite sizes. In one dimension, let us label the sites as 1, 2, 3, $\cdots$, L. Then the left neighbor of site 1 is site L, and the right neighbor of L is 1. You can view the system as on a ring, rather than on a finite straight line segment. In two or higher dimensions, each direction is treated just like that of one dimension. In two dimensions, the topology due to periodic boundary conditions is that of a torus. See figures below.



9–2. Cluster labeling algorithms

Percolation deals with clusters in a random pattern. Given a configuration which specifies the occupation of sites, we need to know which set of sites forms a single cluster. If we count the clusters of size s, we can estimate n_s. The maximum size s can be used to estimate P. Let's first concentrate on the problem of labeling the cluster. We are given a configuration or state stored in a one-dimensional array `s[N]` with indices varying from 0 to $N-1$. N is some fixed number, which is the total number of sites of the system.

Our task is to give a label to each site, such that each cluster has a unique label, as indicated in the figure.



This is more or less a standard problem of graph traversal. There are at least two basic algorithms to the problem. The first algorithm uses recursive function calls. Recursive program tends to be short and elegant. However, it is less efficient both in computer time and memory, because of the overhead in handling the stack. This may not be a problem. But if you want to have the world record on percolation calculations, you'll want your program to be executed as fast as possible, and the system size as large as possible. The efficiency becomes a major concern.

Algorithm 1: Cluster identification with recursive function.

```
#define Z 4
#define L 16
#define N (L*L)
```

the coordination number is twice the dimension
the system linear size
the total number of sites (two-dimensional)

```
void label(int s[], int number[])
{
   int i, inc = 0;

   for(i = 0; i < N; ++i)
      number[i] = 0;
   for(i = 0; i < N; ++i)
      if(s[i] == 1 && number[i] == 0)
         color(s, number, i, ++inc);
}
```

This function goes over each site and checks whether the site has been labeled. If it is not and is not empty, it calls `color` to label the site and those sites connected to it.

```
void color(int s[], int number[], int i, int inc)
{
   int nn[Z], j;

   if (s[i] == 1 && number[i] == 0 ) {
      number[i] = inc;
      neighbor(i, nn);
      for(j = 0; j < Z; ++j)
         color(s, number, nn[j], inc);
   }
}
```

The `color` function labels the site `i` if it is occupied, and labels its neighbors as well. The function is used recursively. Empty sites have label zero.

```
void neighbor(int i, int nn[ ])
{
   int j, r, p, q;

   r = i;
   p = 1 - L;
   q = 1;

   for(j = 0; j < Z; j += 2) {
      nn[j] = (r + 1) % L == 0 ? i + p : i + q;
```

Find the neighbors of site `i`, Sites are named by number from 0 to `N-1`. This function works for any hypercubic lattice. The `Z` neighbors of site `i` are returned in the array `nn`.

```
        nn[j+1]        = r % L == 0 ? i - p : i - q;
        r = r/L;
        p *= L;
        q *= L;
    }
}
```

Algorithm 2: Cluster identification with a temporary list.

The algorithm of Hosen and Campellman is more efficient. This algorithm uses list to representing connectivity. It also calculates the sizes of the clusters at the same time. For detail, see Ref. 1.

9–3. Quantities to calculate and finite-size effects

What to calculate in a typical simulation is a difficult decision you have to make. On the one hand, you don't want to handle too many data. On the other hand, very often, you don't know which quantity is the most appropriate to the problem at hand. Thus you have to make a compromise.

For the site percolation problem, let's say we want to calculate the cluster number function n_s, the percolation probability P, the average cluster size S. Sometimes, we might want to calculate the pair-correlation function $g(r)$. By now you should know that n_s is the average of the number of clusters having exactly size s, normalized by the system size L^d.

P is the probability that the origin or any given site is on the infinitely large percolating cluster. Since we cannot simulate an infinitely large system on a finite memory computer in finite time, we must do some extrapolation – calculating everything on a finite system, observing the changes of quantities as the system size changes, and finally trying to predict the value at infinite-size limit. The finite-size scaling theory deals with this problem. Fortunately, many quantities tend to their infinite-size limits very quickly. We can choose a large system (say $L = 100$ or even 1000) to overcome the finite-size effect. But often, simply taking a large system does not work.

What is a good estimator for P from a computational point of view. We can approximate P as

$$P_L = \langle s_{max} \rangle \frac{1}{L^d} \tag{39}$$

where s_{max} is the size of the largest cluster in the system. The percolation probability is $P = \lim_{L\to\infty} P_L$. For $p < p_c$ we have $P = 0$, and P_L approaches zero as $L^{-d/2}$. At the percolation threshold $p = p_c$, the finite-size scaling theory predicts $P_L \propto L^{-\beta/\nu} \to 0$. Above the percolation threshold $p > p_c$, P_L approaches the infinite size limit P probably exponentially fast, $P - P_L \propto \exp(-cL^d)$??

The average cluster size can be calculated from

$$S = \frac{1}{(p - P)L^d} \sum_i{}' c_i^2, \tag{40}$$

where c_i is the size of i-th cluster in the system. The summation is carried out over all clusters except the largest, indicated by the prime $'$. Note that $P = 0$ for $p < p_c$.

9–4. Monte Carlo steps and statistical errors

A simulation goes as follows. We first generate a random percolation configuration by assigning 1 for occupation, and 0 for empty at each site. The site is occupied with probability p. This is achieved by comparing p with a uniformly distributed random number r. If $r < p$, we occupy the site; otherwise we leave it empty. Then we go on to the next site to do the same thing.

When all of the L^d sites on the lattice have assigned values, we start to analyze the configuration. This is one Monte Carlo step. When the analysis of the configuration is finished, we forget or erase the old configuration and generate a new configuration and repeat the process. In each Monte Carlo step, we obtain some value A_i. Our method is in fact a Monte Carlo method in its simplest form, namely a naïve simple sampling method (as oppose to importance sampling method, e.g. Metropolis algorithm).

The average of any quantity you calculate is

$$\langle A \rangle = \frac{1}{N} \sum_{i=1}^{N} A_i, \tag{41}$$

after N Monte Carlo steps.

Why do we bother to generate many configurations. The reason is accuracy. The Monte Carlo method is a lousy method in terms of accuracy. In order that the numerical estimate $\langle A \rangle$ approaches the exact value A, you should do *infinite* number of steps, $N \to \infty$. Unfortunately, we don't have infinite amount of computer time, and we cannot afford to wait. Thus the result you obtained is accurate upto an error of order $N^{-1/2}$, or

$$A = \langle A \rangle \pm \frac{\sigma}{\sqrt{N}}, \qquad N \gg 1, \tag{42}$$

where we should have written $\langle A \rangle$ as $\langle A \rangle_N$ since it is the estimate using N samples. The quantity σ^2 is the variance of the random variable A_i.

$$\sigma^2 = \langle A^2 \rangle - \langle A \rangle^2. \tag{43}$$

The actual value depends on the nature of A_i. The above formulae are the basic results of probability theory. It tells us that if you want to improve your result by one more decimal digit, you'd better increase your computer efforts by a 100-fold.

It appears that Monte Carlo method is a very inefficient method. We agree. But this method sometimes is the only method practical. Taking our percolation problem again as an example, we can calculate the quantity A without error by the formula (assuming no roundoff error in your calculation)

$$\langle A \rangle = \sum_{all\ conf.} A\big(\{n(i,j)\}\big) p^m (1-p)^{L^d - m}. \tag{44}$$

Unfortunately, there are just too many terms in the summation. Since each site has two states, either empty or occupied, the total number of possible configurations is 2^{L^d}. If we have a system of 10×10, the number of configurations is $2^{100} = (2^{10})^{10} \approx (10^3)^{10} = 10^{30}$. Suppose that your program can run at a speed of one microsecond for each site, (including generating the state and performing multiplication – our Silicon Graphics is probably slower than that), we can finish one configuration in 10^{-4} second. It takes 10^{26} seconds to finish the sum over all configurations. Since a year is about 3×10^7 seconds, it takes about 10^{18} years. The age of the Universe is believed to be 10^{10} years. If you had already started your calculation when the Universe was born, you would only have finished $1/10^8$-th of the whole project by now! Thus, we cannot even try a small size like 10×10 by a brute force exact enumeration. The largest size you can do this way is probably 4×4 or 5×5. Because of the finite-size effect, small size results are not very useful anyway. Now you should believe that Monte Carlo method is very powerful.

9–5. Debugging

Your simulation result is meaningless if your program does not work correctly. So, it is absolutely important that your program is bug free. Beside studying your code carefully, you can test in various ways. (1) Test it against known result. If the problem under study has exact solution in certain cases, use the exact result to test your program. (2) Try to calculate exact result on small systems (like 4×4), and compare it with simulation result. (3) look for special limit at which series-expansion solution is easily obtainable. Use analytical result (may be approximate) to compare with simulation result. (4) As a last resort, write a new, independent simulation program in a way to avoid bugs; that is, use simplest possible method, and straightforward algorithm. In a typical production program, efficiency is a major concern. But here we sacrifice efficiency for correctness of the program. (5) Try different random number generators.

Another technique that you might want to use is to write your code defensively. More concretely, add a line of statement

`assert(`*condition*`);`

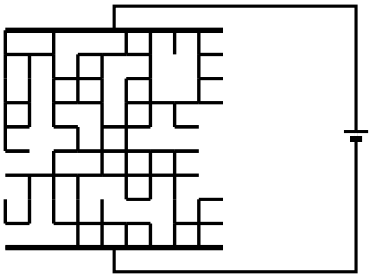
where *condition* is some logical condition you want to check. For example, index i should be greater than or equal to zero but smaller than L. `assert()` is a macro definition in `<assert.h>`. The assertion code can be turned off if you `#define NDEBUG` at the beginning of your program.

10. Random Resistor Network

Random resistor network is a good laboratory to understand percolation. Consider mixture of metallic and ceramic powder. The powder can form solid sample under high pressure. We are concerned with the conducting property of the sample. If you add only small fraction of metallic powder, the material is an insulator. As the fraction of metal powder increases, at some point p_c the system suddenly becomes a conductor, the conductivity increases as you mix more metallic powder.

10–1. Model

We'll study this problem theoretically. Let's first introduce a model for it. For simplicity, we assume that our sample is two-dimensional. We'll use metallic bar (bond) with resistance r as the basic conducting units. The bonds connect neighboring sites on a square lattice, see figure. This is a bond percolation problem.



10–2. Conductivity

All the top sites are connected by a good conductor like silver. So are all the bottom sites. The left and right sites are corresponding neighbors. That is, we use periodic boundary condition in this direction. We subject the system to a direct current. The voltage at the top is V, and that at the bottom is 0. Our aim is to calculate the current I passing through the random network. Or physically more significant quantity, the conductivity of the material. The conductivity Σ of a piece of material is defined through the equation

$$R = \frac{1}{\Sigma}\frac{L}{A}, \tag{45}$$

where R is the resistance, L is the length of the sample, and A is cross-section area of the sample. We'll generalize this equation to any number of dimensions. So the cross-section 'area' is $A = L^{d-1}$, for a hypercube in d dimensions. Since the resistance is related to voltage and current by Ohm's law

$$I = \frac{V}{R}, \tag{46}$$

we can calculate the conductivity by

$$\Sigma = \frac{I}{VL^{d-2}}. \tag{47}$$

The conductivity is a number that characterizes the material. It is independent of the shape of the material as well as the voltage applied to the sample. In two dimensions, we see that the conductivity is equal in magnitude to the current when a unit voltage is applied to the square sample.

The voltage at each site V_i is related by Kirchhoff's law

$$I_i = \sum_j (V_j - V_i)\sigma_{ij}, \tag{48}$$

where I_i is the external current going into the site i, it is zero for all the internal sites. The summation is over the nearest neighbors of i. The conductance between site i and its neighbor site j is σ_{ij}, which is zero if there is no bond between them and is $1/r$ if there is one. Since the top and bottom voltage are fixed at V and 0, respectively, the equations are linear equations with V_i as unknown of the standard form

$$\sum_{j=1}^{L^d-2L} a_{ij}x_j = b_i, \quad i = 1, 2, \cdots, L^d - 2L. \tag{49}$$

Since a site connects only to its neighbor sites, the matrix a has a lot of zeros. It is a sparse matrix. We'll discuss the methods of solving the set of equations in the next subsection. Once you have solved the linear equations, the total current I can be obtained by the Kirchhoff equation, applying to the top or bottom row of sites.

In an actual computation, we seldom use quantities with dimensions (that is, units). Typically we normalize the quantity with characteristic dimension, and the numerical values stored in computer are dimensionless. This is also equivalent to set the model parameter to some convenient value. In our problem, we'll set $r = 1$. In other word, the resistance of the system will be measured in units of the resistance of each individual bond. We also set $V = 1$, namely, applying a unit voltage on the system.

10–3. Algorithms to find voltage

Algorithm 1: We can use the standard Gauss-Jordan elimination or LU decomposition method. Unfortunately, for large matrices, we'll have difficulty due to roundoff errors and computational efficiency. If the system size is $N = L \times L$, we'll have a matrix of $N \times N$. The time complexity goes as N^3. We wouldn't be able to deal with large system.

Algorithm 2: Iterative (relaxational) methods. Looking at the equation carefully, you notice that it is like a discretized Laplace's equation with random coefficient σ_{ij} in each direction. This is not surprising, since the voltage (or potential) satisfies Laplace's equation. Much more sophisticated methods exist (over-relaxation, multigrid method), but we'll just introduce the most elementary one. Starting with boundary values $V_i = 1$ at the top row and $V_i = 0$ at the bottom row and some good guess for the internal site, e.g., linear dependence, $V_i = l/L$, where l is distance from the bottom sites. The actual initial values should not matter very much. Then we go through the sites one by one in some order, and update the value V_i according to

$$V_i = \frac{1}{n} \sum_j V_j, \tag{50}$$

where n is the number of bonds connecting to the site i. The summation is over the neighbor sites j which are connected to the current site i. In other words, the value of the center site is equal to the average of the connecting neighboring sites. Disconnected sites do not contribute in the average. This calculation is carried out at each site as one iteration. One performs as many iterations as necessary to obtain desired accuracy (e.g., 50 iterations). It is better to use the newly obtained value of V_i as soon as it is available. It is called Gauss-Seidel method.

The relaxation method works fine, but it has disadvantages. (1) The method is approximate. We have to make the decision when the solution is accurate enough. (2) the convergence becomes slow on large systems near the percolation threshold p_c, where the physics gets interesting and it is precisely the place we like to have more accurate values. This phenomenon is called critical slowing down. We'll also see such phenomena in other problems later in the course.

Algorithm 3: Transformation method (Fogelholm, 1980, Ref. 4). This method probably is the state-of-the-art method for this problem. It is one of these methods that we occasionally present, which are of new research development in journals, rather than those of well developed methods in textbooks.

You may have learned the Y–Δ transformation in electric circuit theory. Let's assume that we have four nodes 0, 1, 2, 3. The node 0 is connected to 1, 2, and 3 with conductances σ_1, σ_2, and σ_3, respectively. Then the Y-shaped circuit is electrically equivalent to the Δ-shaped circuit, with center node 0 eliminated, and connecting other nodes 1, 2, and 3 to each other, see figures below.

the conductances between these nodes are now

$$\sigma_{ij} = \frac{\sigma_i \sigma_j}{\sum_{k=1}^{3} \sigma_k}. \tag{51}$$

It turns out that this formula works for the center node 0 connected to any number of other nodes 1, 2, $\cdots$, n. One simply runs the index k from 1 to n.

The idea of transformation method is to eliminate the internal nodes completely by repeated application of the above transformation. Eventually, we have just two terminal nodes. The conductance between those nodes is the conductance Σ. It is clear that one needs complicated data structure (graphs) to represent the circuit. The computational (time) complexity is N^2 (??). So it is very efficient comparing with other methods. With this method, it is possible to have system as large as 1000×1000.

11. Other Simulation Models in Physics

We'll just mention briefly some other models studied in condensed matter physics. Fractal patterns were studied extensively during the past decades. They appear in many physical processes, e.g, growth process. Models such as Eden cluster growth, diffusion limited aggregation, ballistic deposition can be studied easily by computer simulation.

Cellular automata is another branch of simulation. Instead of solving differential or partial differential equations, people invent simple evolution rules to simulate real processes. For example, we can use cellular automata to mimic hydrodynamic behavior of fluids under certain conditions. Neuron network is in the same spirit, and it is a very hot research topic.

References

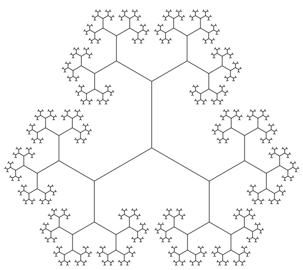
1. D. Stauffer and A. Aharony, *Introduction to Percolation Theory*, 2nd ed., (Taylor & Francis, 1992).
2. D. K. Knuth, *The Art of Computer Programming*, Vol. 2, (Addison-Wesley, 1981).
3. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed., (1992), Chapter 7.
4. R. Fogelholm, J. Phys. C, **13**, L571 (1980).

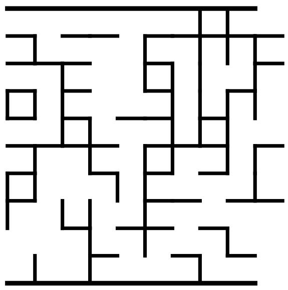
TUTORIAL PROBLEM SET 2.

1. PERCOLATION ON BETHE LATTICE

Bethe lattice (or Cayley tree) is the lattice shown below on the left (which looks like cauliflowers). In fact, it is three identical binary trees joining together at the roots. If we look at only the topology of the Bethe lattice, and if the lattice is infinitely large, every site is equivalent to other sites. So it is really a lattice in this sense. However, in order to draw the lattice so that the bond lengths are the same everywhere, we must put the lattice in a dimension $d = \infty$. So, Bethe lattice is said a system with infinite dimension.

We want to solve the percolation problem on Bethe lattice. More precisely, we want to find an expression for the percolation probability P. The quantity P is the probability that a given site, say the center site, is on an infinitely large network. Remember that $P = 0$ if the system does not percolate, and $P > 0$ if the system percolates. To find P, let us define Q to be the probability that an arbitrarily occupied site is not connected to infinite through one fixed branch orignating from this site. In other words, take a branch, which is a tree, Q is the probability that you can not traverse down to infinite in the tree. The connectivity is defined just like site percolation: two nodes are connected if both of the nodes are occupied. Your can find an equation for Q because of the recursive nature of the tree (Since the tree is supposed to be infinitely large, a subtree looks exactly like the original tree). Solving this equation gives Q. Find a relation between P and Q. Refer to Stauffer's book (*Introduction to Percolation*) if you cannot work out the answer after some efforts.





2. QUASI-ONE-DIMENSIONAL PERCOLATION

Consider site percolation on a $2 \times \infty$ lattice (two rows of one-dimensional lattices). In considering the connectivity between sites in two different rows, free boundary condition will be used.

(a) We know that the one-dimensional site percolation has the percolation threshold $p_c = 1$. Is the percolation threshold for the $2 \times \infty$ lattice $p_c < 1$ or $p_c = 1$? Explain. What about systems of $n \times \infty$, for any finite n?

(b) Write down the cluster number n_s for the first few terms $s = 1$, 2, and 3, as a function of the occupation probability p.

(c) State the definition of correlation function (or sometimes called pair connectivity) $g(\mathbf{r})$. Calculate the correlation function $g(x, y)$ for $(x, y) = (0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$.

3. RANDOM RESISTOR NETWORK

In the problem of two-dimensional random resistor network of bonds, many bonds do not carry current (they have the same voltage on both ends). Examples are isolated islands, dangling bonds, and clusters that do not span the top and bottom rows, see figure above. Give an algorithm that would identify the zero-current bonds on the lattice, based on the geometry of the percolation configuration. The algorithm should be efficient so that you need not solve the voltage problem. Do not write codes, just describe how you do it.

LABORATORY EXERCISE 2.

1. SITE PERCOLATION

We want to count the cluster numbers n_s and to determine the critical exponent τ in $n_s \propto s^{-\tau}$ at $p = p_c \approx 0.592746$ on the two-dimensional square lattice. Develop your program with a small system size, say $L = 4$ for easy debugging. You may use my labelling function `label.c` in the directory `/usr1/public/cttp`. For production runs, take $L = 100$ or even 500, and $p = p_c$, compute n_s. Use `xmgr` to find the slope (by a linear least-squares fit) of the curve $\ln n_s$ vs. $\ln s$. Submit your codes, report your estimate of τ, and also quote error in τ, and system sizes and Monte Carlo steps used.

2. RANDOM RESISTOR NETWORK

Our random resistor network consists of a two-dimensional lattice of $L \times L$ sites. The left and right sites are connected by periodic boundary conditions (wrapped around like a cylinder). All the top or bottom sites are connected together by very good conducting metal (zero resistance). In the middle, some of the neighbor sites are connected by not-so-good conductors with resistance $r = 1$; other neighboring sites do not connect at all (zero conductivity). The conducting bonds appear at random with probability p, forming bond percolation configurations.

This exercise asks you to compute the conductivity using the LU decomposition method for linear equations, for a particular set of configurations given by a function in the `rrn.c` program. Use the outline program `/usr1/public/cttp/rrn.c` to do your problem. Consult page 15–23, 32–48, of *Numerical Recipes in C* regarding the use of `ludcmp.c`, `lubksb.c` and `nrutil.c`.

Submit only the codes for the two functions `coeff()` and `current()`. Report your results for linar size $L = 4$, $L = 8$, and $L = 16$.

Discuss the limitation of the LU method.