1. Introductory Examples

We introduce the concept of the deterministic and stochastic simulation methods. Two problems are provided to explain the methods: the percolation problem, providing an example for the application of a stochastic method and the simulation of a particle in a force field, providing an example for the application of a deterministic method. We also work out three constraints that one encounters: Finiteness of the system, finiteness of the sample size and the accuracy.

1.1 Percolation

A problem lending itself naturally to a computer simulation approach is that of percolation. Consider a lattice, which we take, for simplicity, as a two-dimensional square lattice. Each lattice site can be either occupied or unoccupied. A site is occupied with a probability $p \in [0,1]$ and unoccupied with a probability $1-p$. If p is small than only a tiny fraction of the sites is occupied and only small isolated patches of occupied sites that are close to each other exist. On the other hand, if p is large, than large patches of near lying occupied sites exist. We may at this point speculate that for p less than a certain probability p_c only finite clusters exist on the lattice. We define more precisely what we mean by patches or clusters by saying: A cluster is a collection of occupied sites connected by nearest-neighbour distances (see Figure 1.1). For p larger than or equal to p_c, there exists a large cluster (for an infinite lattice, i.e., in the thermodynamic limit) such that for an infinite lattice the fraction of sites belonging to the largest cluster is zero below p_c, and non zero above p_c. For $d = 1$ the situation is straight-forward: $p_c = 1$. For $d > 1$ the computation of p_c is non-trivial, and analytic results for the percolation threshold p_c are only available for certain dimensions or special lattices. From the point of view of computer simulation we are interested in the calculation of the percolation probability $P_\infty(p)$ (defined as the number of occupied sites in the largest cluster divided by the lattice size) as a function of p to determine the percolation threshold p_c.

A simple enumeration of all the possible configurations, computing the fraction of sites that belong to the largest cluster of a configuration, is impossible. The number of configurations is 2^N where $N = L^d$, L the linear

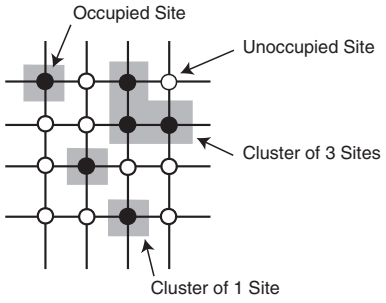

Fig. 1.1. Percolation problem: Shown is part of a simple two-dimensional lattice with occupied and unoccupied lattice sites. Shown is also a cluster of nearest-neighbour sites

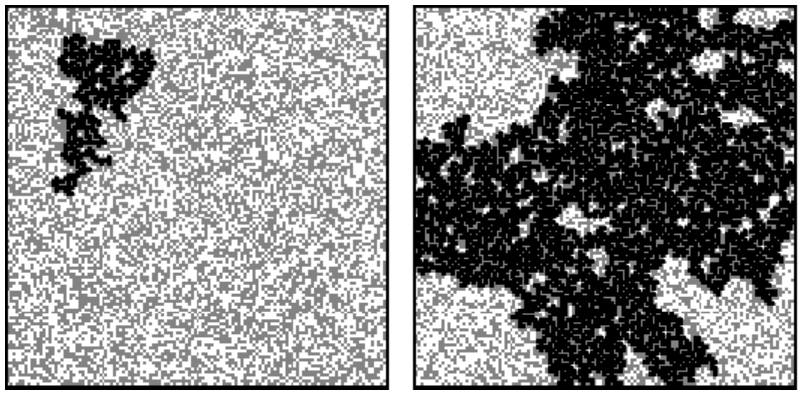

Fig. 1.2. Configurations generated by a stochastic computer simulation of the percolation problem. The left figure shows a configuration generated with occupation probability $p = 0.5$. The configuration shown in the right part of the picture was generated using $p = 0.6$. The largest cluster is marked

system size and d the space dimension. The question arises whether one can obtain an approximation for the percolation threshold by computer simulation. Since we can not enumerate the configurations, we need to *sample* the space of all possible configurations, i.e. draw a finite subset of configurations $|S| = n$. We observe that all configurations have the same weight, and we are allowed to draw from the ensemble of all possible configurations a subset at random.

To keep the computer simulation approach transparent we stay with the two-dimensional square lattice. By its nature the problem suggests a stochastic approach. Suppose one could generate a lattice filled with a given probability and check whether a percolating structure occurs. To be sure that one is definitely above or below the percolation threshold an average over many

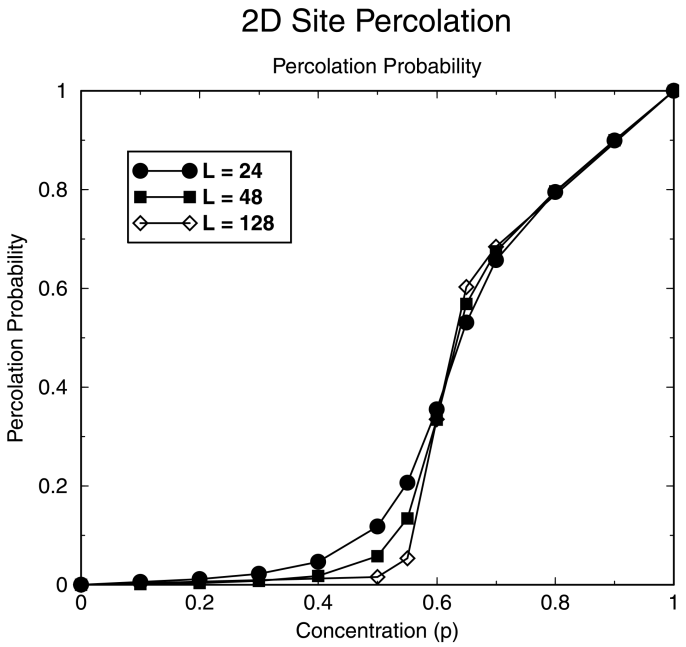

Fig. 1.3. Site percolation on the simple square lattice. Shown is the percolation probability P_∞ as a function of the occupation probability p

such samples must be taken. Running through a range of p values the percolation threshold is narrowed down until sufficient accuracy is established.

Algorithmically the problem can be treated as follows: We set up a two-dimensional array to hold the occupied or unoccupied sites. All sites of the lattice are visited by going successively through all rows (columns). For each element of the array a uniformly distributed random number $r \in [0, 1]$ is drawn. If r is less than p, then the element is set to one (occupied) otherwise 0 (un-occupied). After having visited all elements, one realization or configuration is generated. (For a full program see the online supplement.)

Examples of percolation configurations are shown in Fig 1.2. For a value of p equal to 0.5 (Fig 1.2 left) there are only scattered finite clusters. Choosing p equal to 0.6 (Fig 1.1 right) a large cluster exists which occupies a substantial fraction of the lattice.

Performing an analysis of a configuration for an infinite cluster two possibilities arise: either a cluster exists that occupies a finite fraction of the lattice, as in the case $p = 0.6$, then p is a candidate for being greater than or equal to the percolation threshold, or the opposite case is true. To obtain an average value for the percolation probability $P_\infty(p)$ the generation of a configuration with the analysis must be performed n-times. To see where some of

the difficulties lie in connection with such simulations, we look at the results for the percolation probability P_∞ $(d = 2)$ (Fig 1.3).

The first difficulty is the size dependence of the results for the percolation probability. Intuitively we expect that it should be easier to generate clusters that occupy a finite fraction of the lattice for small lattices. This would shift the percolation threshold to smaller p values. Indeed, the results on the percolation probability for the three-dimensional lattice displayed in Fig 1.3 show this behaviour. We note further that no sharp transition occurs for the finite small lattices. The percolation threshold is smeared out and difficult to estimate.

The second difficulty is the number of samples. For an accurate determination of p_c, quite a large number of samples have to be taken to reduce the statistical uncertainty. This holds true for other such direct simulations (simple sampling, as well as for more sophisticated sampling methods). The third difficulty concerns the random numbers and thus the inherent accuracy of the method. A problem arises if the random numbers have some built-in correlations. Such correlations are extremely dangerous since they bias the results and can only be detected if some aspects of the problem are known from different methods or the results show some extreme anomaly.

The approach described above to determine the percolation threshold is an example of a stochastic simulation method, in particular of a Monte Carlo simulation. As the name suggests, such simulations are intricately connected with random numbers.

1.2 A One-Particle Problem

A particle moving in a spring potential, i.e., a one-dimensional harmonic oscillator, supplies another illustrative example where a solution is obtainable by computer simulation. Although the system is trivial to solve analytically, it nevertheless provides insight into possible ways to attack problems involving a collection of interacting particles not readily solvable with analytic methods. The nature of the problem is deterministic, in the sense that we start from a Hamiltonian description of the particle moving under the influence of the force exerted by the spring (see Figure 1.4), i.e.,

$$\mathcal{H} = \frac{p^2}{2m} + \frac{1}{2}kx^2 \quad , \tag{1.1}$$

where p is the momentum, m the mass of the particle, k the spring constant and x the position. In addition to the Hamiltonian, we need to specify the initial conditions $(x(0), p(0))$. There is no coupling to an external system, and the energy E is a conserved quantity. The particle will follow a trajectory on a surface of constant energy given by

$$\frac{p^2}{2mE} + \frac{kx^2}{2E} = 1 \quad . \tag{1.2}$$

Having written down the Hamiltonian we have some options as to the form of the equations of motion. We may reformulate the problem in terms of a Lagrangian and derive the equation of motion or cast the problem in the form of the Newtonian equation. The algorithm for a numerical solution and its properties depend on this choice. Here we take the Hamiltonian form of the equation of motion

$$\frac{dx}{dt} = \frac{\partial \mathcal{H}}{\partial p} = \frac{p}{m}, \quad \frac{dp}{dt} = -\frac{\partial \mathcal{H}}{\partial x} = -kx \quad . \tag{1.3}$$

We would like to compute properties of the system as it moves along a path $(x(t), p(t))$ in phase space. In general, the complexity of the equations of motion defies an analytic treatment, and one resorts to a numerical integration. We go about solving the problem by approximating the continuous path by a polygon, i.e., to first order the differential is taken as

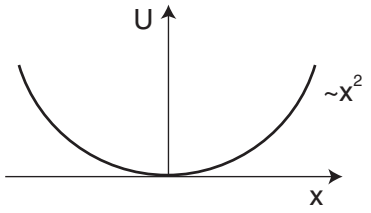

Fig. 1.4. Potential for a harmonic spring

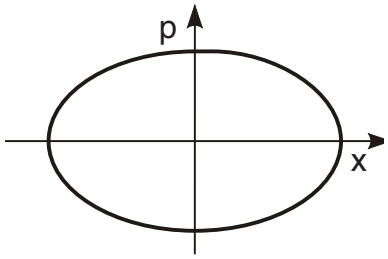

Fig. 1.5. Shown in phase space is the surface of constant energy for the one dimensional spring

$$\frac{df}{dt} = \frac{1}{h}\left[f\left(t+h\right) - f\left(t\right)\right] \quad . \tag{1.4}$$

h being the basic time step. With such an approximation a solution of the equations of motion can be obtained only at times which are multiples of the basic unit h. Note that if the basic time step is finite there will always be a certain error, i.e., the generated path will deviate from the true one. Inserting the discretization for the differential into the equations of motion, we obtain the following recursion formulae for the position and the momentum:

$$\frac{dx}{dt} \cong \frac{1}{h}\left[x\left(t+h\right) - x\left(t\right)\right] = \frac{p\left(t\right)}{m}, \quad \frac{dp}{dt} \cong \frac{1}{h}\left[p\left(t+h\right) - p\left(t\right)\right] = -kx\left(x\right) \tag{1.5}$$

$$x\left(t+h\right) = x\left(t\right) + \frac{hp\left(t\right)}{m}, \quad p\left(t+h\right) = p\left(t\right) - hkx\left(t\right) \tag{1.6}$$

Given an initial position $x(0)$ and momentum $p(0)$ consistent with a given energy, the trajectory of the particle is simulated. Starting from time zero, the position and momentum at time h are computed via the above equations; then at $t = 2h, 3h, ...$ etc. Any property one is interested in can be computed along the trajectory that is generated by the recursion relation. An example of a trajectory based on the polygon method is shown in Fig 1.6. The path is a spiral, indicating that energy dissipated.

Our algorithm is an iteration

$$(x,p)_{i+1} = G((x,p)_i), \qquad (x,y)_0 = \text{initial value} \quad , \tag{1.7}$$

where

$$x_{i+1} = x_i + hp_i \tag{1.8}$$

$$p_{i+1} = p_i - hkx_i \quad . \tag{1.9}$$

The fixpoint for this iteration is at

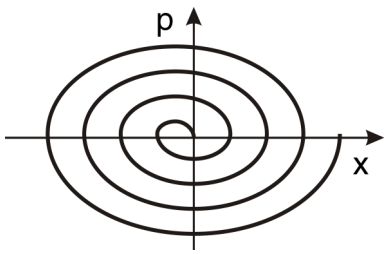

Fig. 1.6. Trajectories in phase space for the spring problem as generated with a simple algorithm.

$$x = 0 \tag{1.10}$$
$$p = 0 \tag{1.11}$$

Due to the linear approximation that we made for the differential we find the following equation

$$\begin{pmatrix} x_{i+1} \\ p_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & h \\ -hkx & 1 \end{pmatrix} \begin{pmatrix} x_i \\ p_i \end{pmatrix} \tag{1.12}$$

Let λ_1, λ_2 be the eigenvalues of the matrix A with the eigenvectors u_1, u_2. We can write $z_n = (x, y)_n$ as a linear combination

$$z_i = \alpha_i u_1 + \beta_i u_2 \quad , \tag{1.13}$$

and by induction we find

$$z_i = \alpha_0 \lambda_1^i u_1 + \beta_0 \lambda_2^i u_2 \quad , \tag{1.14}$$

and note that the eigenvalues for this problem are given by

$$\lambda_{1/2} = 1 \pm ih\sqrt{k} \tag{1.15}$$
$$= r(\cos\theta \pm i \sin\theta) \tag{1.16}$$

and most important

$$r = \sqrt{1 + h^2 k} \quad . \tag{1.17}$$

Even if we try very hard and set the time step h to a very small value we never get the correct solution. The solution we obtain is always a spiral and not an ellipse.

For a collection of particles we are faced with similar problems. Due to the finite step size the trajectory departs from the true one. Also we must ensure that the alogrithm that we are developing conserves phase space volume.

We are interested in developing algorithms of high order in h to keep the error as small as possible. The basic time unit determines the simulated real time. A very small time step may have the consequence that the system does not reach equilibrium during a run. The kind of approximation made to solve the spring problem numerically on a computer is but one type. In the following, more sophisticated schemes will be presented for deterministic simulations. Such methods are known as molecular dynamics. In many cases, computer simulation methods offer a surprisingly simple and elegant approach to the solution of a problem. In the following chapters we study some most commonly used methods.

1.3 A little help for the percolation problem

1.3.1 From the algorithm to the program

In the percolation problem we pose the question what the probability is with which we must occupy lattice points such that we can reach the opposite lattice edge over connected nearest neighbour sites. Nearest neighbour sites of the lattice are connected if both sites are occupied. If we choose 1 for the probability for the occupation of the lattice sites then we always will have connectivity between opposite edges of lattice. On the other hand if we choose zero for the probability of occupation then there will be no connectivity between opposite edges. Is there a probability (percolation threshold) $p_c > 0$ such that there is no connectivity below p_c and connectivity above p_c?

1.3.2 The structure of the program

The basis for the computation of the percolation threshold is a Monte Carlo algorithm that makes use of random numbers. Here successively generated configurations will be independent and averaging is done over these independent configurations.

Our task is to work out a program for the simulation of the site percolation problem. With other applications in mind we will not make use of some features of the problem that would allow us to write very compact code. Rather we deliberately make the program as general as possible for it to serve as a starting point for other projects. The basic parts of the program will be

- The lattice that will hold the occupation variables, i.e., whether a site is occupied or not.
- Fixing the percolation probability.
- A loop for the generation of configurations.
- A loop to cycle through the lattice.
- A random number generator that delivers uniformly distributed numbers in the interval $[0, 1]$.
- Analysis of the generated configuration.

Let us start from the infamous hello-world program. The first task is to declare the lattice for the occupation variable. Since the lattice site can only be occupied or unoccupied we only need a declaration for binary data. However, we make to program more general right from the start. We declare the lattice of type integer. Next we fix the percolation probability p_B. p_B must be a real variable in the interval between 0 and 1

The task is to generate many configurations to gather statistics. We write down a loop so that we generate mcsMax configurations. Within the loop two tasks have to be handled: the generation of a configuration and the analysis of the configuration.

A configuration is obtained by setting a lattice either to 0 or to 1 depending on the outcome of the result of a drawing from the set of uniformly distributed random numbers r. The random number is compared to the fixed probability p_B. Suppose p_B is set to 1. No matter what the outcome of the drawing is, we need to set the occupation variable to 1. Hence comparing the r to p_B we always have to set the lattice site to 1 if r is less than p_B. If p_B is set to 0 then indeed no r will be less than p_B.

The question at this point is how to run through the lattice. Since there are no dependencies between the sites on the lattice we can run through the lattice which ever way we like. The simplest choice is to cycle through the lattice.

Next we need to check for the presence of a cluster that spans the lattice either from top to bottom or from left to right. We assign this problem to a function that returns the results as either 0 in the case of no connectivity or 1 in the case of connectivity.

Finally we need to perform the averaging over the independent results of the sampling. Remember that due to the finiteness of the lattice we can expect to find configurations that have a connectivity even though in the thermodynamic limit and averaging of over all possible configurations this would not the case.

1.3.3 Testing the program

Finally we must test our program. Generally we have no results available that would help us to compare our simulation result to and find out whether the program is correct or still carries a bug. We must check the program by looking at special cases to make reasonably sure that the program is working correctly.

We assume that the random number generator has already been checked (see remarks to this point in the later sections!). The first obvious checks are the special cases of $p_B = 0$ and $p_B = 1$. Our averaged result must always be 0 ($p_B = 0$) and 1 ($p_B = 1$). Next we can generate specific configurations by explicitly setting lattice sites commenting out the setting of lattice using the random numbers.

1.4 Problems

1. Use the supplied computer program for the two-dimensional site-percolation problem to compute the cluster size distribution as a function of p. Note that the moments of the cluster size distribution are related to observables as the cluster susceptibility $\chi = \sum_s{}' sn(s)$, where the prime denotes that the largest cluster is to be omitted from the summation.

2. **Bond Percolation**
   In this chapter we were concerned with the site percolation problem. Equally well we can study the bond percolation problem. Instead of occupying the lattice sites we connect sites by choosing a probability p and if the outcome of a comparison with a random number is positive, we occupy the bond between the selected two nearest-neighbour sites. Revise your program from the previous exercise for this problem. Do you think that the site and the bond percolation problem are equivalent?
3. **Random Walk**
   Consider a lattice with coordination number q on which a particle can move. The particle is only allowed to go from one site to the nearest neighbour site. Start from a site which is considered the origin. Choose a random number which determines to which of the q nearest neighbour sites the particle moves. Continue to do so for n moves. To establish the diffusion law, develop an algorithm which generates walks of length n. Determine the mean-square displacement for each walk and average the result. Plot the mean-square displacement as a function of n.
4. **Growth of Clusters (Eden cluster [?])**
   One starts from one occupied site on a lattice as a seed for a cluster. At every "time step" one additional randomly selected perimeter site is occupied. A perimeter site is an empty neighbour of an already occupied site. Write a computer program to simulate the growth of the cluster as a function of "time".
5. **Aggregation**
   A nice example of a direct method is the generation of aggregates. Take a lattice with one or more sites of the lattice designated as aggregation sites. Draw a wide circle or sphere around the potential aggregation sites. Introduce a new particle at a randomly picked site on the circle or sphere. The particle is now performing a random walk (cf. previous exercise) until it walks outside the valid region, or moves to one of the surface sites of the aggregation site. If the particle has come to the surface site of the aggregation site, it sticks, and the potential aggregation surface has grown. Inject a new particle into the valid region and continue until the aggregate has grown substantially. The valid region is then changed to give room for the walker to move. Why is this a direct simulation of aggregation? Can you invent variations of the problem?