Monte Carlo methods

1

Starting with this chapter, we embark on a journey into the fascinating realms of statistical mechanics and computational physics. We set out to study a host of classical and quantum problems, all of value as models and with numerous applications and generalizations. Many computational methods will be visited, by choice or by necessity. Not all of these methods are, however, properly speaking, computer algorithms. Nevertheless, they often help us tackle, and understand, properties of physical systems. Sometimes we can even say that computational methods give numerically exact solutions, because few questions remain unanswered.

Among all the computational techniques in this book, one stands out: the Monte Carlo method. It stems from the same roots as statistical physics itself, it is increasingly becoming part of the discipline it is meant to study, and it is widely applied in the natural sciences, mathematics, engineering, and even the social sciences. The Monte Carlo method is the first essential stop on our journey.

In the most general terms, the Monte Carlo method is a statistical—almost experimental—approach to computing integrals using random[1] positions, called samples,[1] whose distribution is carefully chosen. In this chapter, we concentrate on how to obtain these samples, how to process them in order to approximately evaluate the integral in question, and how to get good results with as few samples as possible. Starting with very simple example, we shall introduce to the basic sampling techniques for continuous and discrete variables, and discuss the specific problems of high-dimensional integrals. We shall also discuss the basic principles of statistical data analysis: how to extract results from well-behaved simulations. We shall also spend much time discussing the simulations where something goes wrong.

The Monte Carlo method is extremely general, and the basic recipes allow us—in principle—to solve any problem in statistical physics. In practice, however, much effort has to be spent in designing algorithms specifically geared to the problem at hand. The design principles are introduced in the present chapter; they will come up time and again in the real-world settings of later parts of this book.

[1] "Random" comes from the old French word *randon* (to run around); "sample" is derived from the Latin *exemplum* (example).

Children randomly throwing pebbles into a square, as in Fig. 1.1, illustrate a very simple direct-sampling Monte Carlo algorithm that can be adapted to a wide range of problems in science and engineering, most of them quite difficult, some of them discussed in this book. The basic principles of Monte Carlo computing are nowhere clearer than where it all started: on the beach, computing π.

Fig. 1.1 Children computing the number π on the Monte Carlo beach.

1.1 Popular games in Monaco

The concept of sampling (obtaining the random positions) is truly complex, and we had better get a grasp of the idea in a simplified setting before applying it in its full power and versatility to the complicated cases of later chapters. We must clearly distinguish between two fundamentally different sampling approaches: direct sampling and Markov-chain sampling.

1.1.1 Direct sampling

Direct sampling is exemplified by an amusing game that we can imagine children playing on the beaches of Monaco. In the sand, they first draw a large circle and a square exactly containing it (see Fig. 1.1). They then randomly throw pebbles.[2] Each pebble falling inside the square constitutes a trial, and pebbles inside the circle are also counted as "hits".

By keeping track of the numbers of trials and hits, the children perform a direct-sampling Monte Carlo calculation: the ratio of hits to trials is close to the ratio of the areas of the circle and the square, namely $\pi/4$. The other day, in a game of 4000 trials, they threw 3156 pebbles inside the circle (see Table 1.1). This means that they got 3156 hits, and obtained the approximation $\pi \simeq 3.156$ by just shifting the decimal point.

Let us write up the children's game in a few lines of computer code (see Alg. 1.1 (`direct-pi`)). As it is difficult to agree on language and dialect, we use the universal *pseudocode* throughout this book. Readers can then translate the general algorithms into their favorite programming language, and are strongly encouraged to do so. Suffice it to say here that calls to the function $\texttt{ran}(-1,1)$ produce uniformly distributed real random numbers between -1 and 1. Subsequent calls yield independent numbers.

procedure `direct-pi`
$N_{\text{hits}} \leftarrow 0$ (initialize)
for $i = 1, \ldots, N$ **do**
  $x \leftarrow \texttt{ran}(-1,1)$
  $y \leftarrow \texttt{ran}(-1,1)$
  if $(x^2 + y^2 < 1)$ $N_{\text{hits}} \leftarrow N_{\text{hits}} + 1$
output N_{hits}

Algorithm 1.1 `direct-pi`. Using the children's game with N pebbles to compute π.

The results of several runs of Alg. 1.1 (`direct-pi`) are shown in Table 1.1. During each trial, $N = 4000$ pebbles were thrown, but the ran-

[2]The Latin word for "pebble" is *calculus*.

Table 1.1 Results of five runs of Alg. 1.1 (`direct-pi`) with $N = 4000$

Run	N_{hits}	Estimate of π
1	3156	3.156
2	3150	3.150
3	3127	3.127
4	3171	3.171
5	3148	3.148

dom numbers differed, i.e. the pebbles landed at different locations in each run.

We shall return later to this table when computing the statistical errors to be expected from Monte Carlo calculations. In the meantime, we intend to show that the Monte Carlo method is a powerful approach for the calculation of integrals (in mathematics, physics, and other fields). But let us not get carried away: none of the results in Table 1.1 has fallen within the tight error bounds already known since Archimedes from comparing a circle with regular n-gons:

$$3.141 \simeq 3\frac{10}{71} < \pi < 3\frac{1}{7} \simeq 3.143. \qquad (1.1)$$

The children's value for π is very approximate, but improves and finally becomes exact in the limit of an infinite number of trials. This is Jacob Bernoulli's weak law of large numbers (see Subsection 1.3.2). The children also adopt a very sensible rule: they decide on the total number of throws before starting the game. The other day, in a game of "N=4000" they had at some point 355 hits for 452 trials—this gives a very nice approximation to the book value of π. Without hesitation, they went on until the 4000th pebble was cast. They understand that one must not stop a stochastic calculation simply because the result is just right, nor should one continue to play because the result is not close enough to what we think the answer should be.

$$\frac{355}{452} = \frac{355}{4 \times 113} = \tfrac{1}{4} \times 3.14159292\ldots$$
$$\pi/4 = \tfrac{1}{4} \times 3.14159265\ldots$$

1.1.2 Markov-chain sampling

In Monte Carlo, it is not only children who play at pebble games. We can imagine that adults, too, may play their own version at the local heliport, in the late evenings. After stowing away all their helicopters they wander around the square-shaped landing pad (Fig. 1.2), which looks just like the area in the children's game, only bigger.

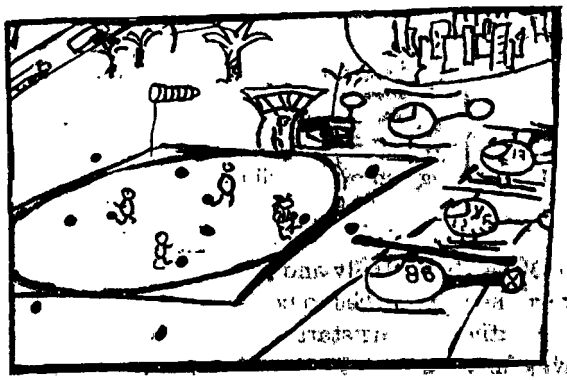

Fig. 1.2 Adults computing the number π at the Monte Carlo helip

The playing field is much wider than before. Therefore, the game must be modified. Each player starts at the clubhouse, with their expensive designer handbags filled with pebbles. With closed eyes, they throw the first little stone in a random direction, and then they walk to where this stone has landed. At that position, a new pebble is fetched from the handbag, and a new throw follows. As before, the aim of the game is to sweep out the heliport square evenly in order to compute the number π, but the distance players can cover from where they stand is much smaller than the dimensions of the field. A problem arises whenever there is a rejection, as in the case of a lady with closed eyes at a point c near the boundary of the square-shaped domain, who has just thrown a pebble to a position outside the landing pad. It is not easy to understand whether she should simply move on, or climb the fence and continue until, by accident, she returns to the heliport.

What the lady should do, after a throw outside the heliport, is very surprising: where she stands, there is already a pebble on the ground. She should now ask someone to bring her the "outfielder", place it on top of the stone already on the ground, and use a new stone to try another fling. If this is again an "outfielder", she should have it fetched and increase the pile by one again, etc. Eventually, the lady moves on, visits other areas of the heliport, and also gets close to the center, which is without rejections.

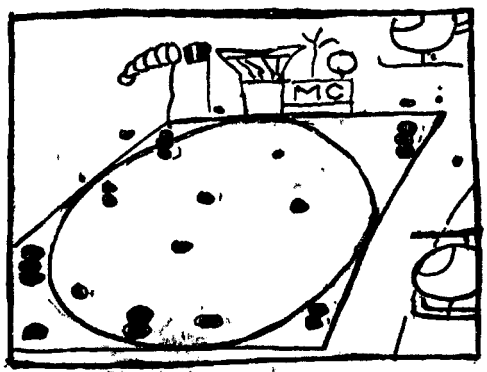

Fig. 1.3 Landing pad of the heliport at the end of the game.

The game played by the lady and her friends continues until the early morning, when the heliport has to be prepared for the day's takeoffs and landings. Before the cleaning starts, a strange pattern of pebbles on the ground may be noticed (see Fig. 1.3): far inside the square, there are only single stones, because from there, people do not throw far enough to reach the outfield. However, close to the boundaries, and especially in the corners, piles of several stones appear. This is quite mind-boggling,

but does not change the fact that π comes out as four times the ratio hits to trials.

Those who hear this story for the first time often find it dubious. Th observe that perhaps one should not pile up stones, as in Fig. 1.3, if t aim is to spread them out evenly. This objection places these mode critics in the illustrious company of prominent physicists and matl maticians who questioned the validity of this method when it was fi published in 1953 (it was applied to the hard-disk system of Chapter Letters were written, arguments were exchanged, and the issue was s tled only after several months. Of course, at the time, helicopters a heliports were much less common than they are today.

A proof of correctness and an understanding of this method, call the Metropolis algorithm, will follow later, in Subsection 1.1.4. He we start by programming the adults' algorithm according to the abc prescription: go from one configuration to the next by following a randc throw:

$$\Delta_x \leftarrow \texttt{ran}\,(-\delta, \delta)\,,$$
$$\Delta_y \leftarrow \texttt{ran}\,(-\delta, \delta)$$

(see Alg. 1.2 (`markov-pi`)). Any move that would take us outside t pad is rejected: we do not move, and count the configuration a seco time (see Fig. 1.4).

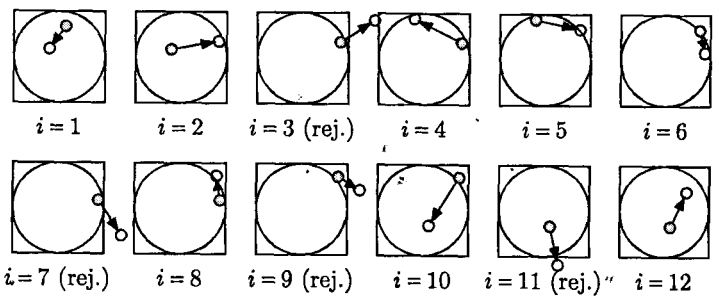

Fig. 1.4 Simulation of Alg. 1.2 (`markov-pi`). A rejection leaves the con figuration unchanged (see frames $i = 3, 7, 9, 11$).

Table 1.2 Results of five runs of Alg. 1.2 (`markov-pi`) with $N = 4000$ and a throwing range $\delta = 0.3$

Run	N_{hits}	Estimate of π
1	3123	3.123
2	3118	3.118
3	3040	3.040
4	3066	3.066
5	3263	3.263

Table 1.2 shows the number of hits produced by Alg. 1.2 (`markov-`) in several runs, using each time no fewer than $N = 4000$ digital pebb taken from the lady's bag. The results scatter around the number π 3.1415 . . . , and we might be more inclined to admit that the idea piling up pebbles is probably correct, even though the spread of data, for an identical number of pebbles, is much larger than for direct-sampling method (see Table 1.1).

In Alg. 1.2 (`markov-pi`), the throwing range δ, that is to be kept fi throughout the simulation, should not be made too small: for $\delta \gtrsim 0$, acceptance rate is high, but the path traveled per step is small. On other hand, if δ is too large, we also run into problems: for a large ra $\delta \gg 1$, most moves would take us outside the pad. Now, the accepta

```
procedure markov-pi
N_hits ← 0; {x, y} ← {1, 1}
for i = 1, ..., N do
    Δ_x ← ran(−δ, δ)
    Δ_y ← ran(−δ, δ)
    if (|x + Δ_x| < 1 and |y + Δ_y| < 1) then
        x ← x + Δ_x
        y ← y + Δ_y
    if (x² + y² < 1) N_hits ← N_hits + 1
output N_hits
```

Algorithm 1.2 `markov-pi`. Markov-chain Monte Carlo algorithm for computing π in the adults' game.

rate is small and on average the path traveled per iteration is again small, because we almost always stay where we are. The time-honored rule of thumb consists in choosing δ neither too small, nor too large—such that the acceptance rate turns out to be of the order of $\frac{1}{2}$ (half of the attempted moves are rejected). We can experimentally check this "one-half rule" by monitoring the precision and the acceptance rate of Alg. 1.2 (`markov-pi`) at a fixed, large value of N.

Algorithm 1.2 (`markov-pi`) needs an initial condition. One might be tempted to use random initial conditions

$$x \leftarrow \texttt{ran}(-1, 1),$$
$$y \leftarrow \texttt{ran}(-1, 1)$$

(obtain the initial configuration through direct sampling), but this is unrealistic because Markov-chain sampling comes into play precisely when direct sampling fails. For simplicity, we stick to two standard scenarios: we either start from a more or less arbitrary initial condition whose only merit is that it is legal (for the heliport game, this is the clubhouse, at $(x, y) = (1, 1)$), or start from where a previous simulation left off. In consequence, the Markov-chain programs in this book generally omit the outer loop, and concentrate on that piece which leads from the configuration at iteration i to the configuration at $i + 1$. The core heliport program then resembles Alg. 1.3 (`markov-pi(patch)`). We note that this is what defines a Markov chain: the probability of generating configuration $i + 1$ depends only on the preceding configuration, i, and not on earlier configurations.

The Monte Carlo games epitomize the two basic approaches to sampling a probability distribution $\pi(\mathbf{x})$ on a discrete or continuous space: direct sampling and Markov-chain sampling. Both approaches evaluate an observable (a function) $\mathcal{O}(\mathbf{x})$, which in our example is 1 inside the

```
procedure markov-pi(patch)
input {x, y} (configuration i)
Δ_x ← ...
Δ_y ← ...
⋮
output {x, y} (configuration i + 1)
```

Algorithm 1.3 `markov-pi(patch)`. Going from one configuration to the next, in the Markov-chain Monte Carlo algorithm.

circle and 0 elsewhere (see Fig. 1.5). In both cases, one evaluates

$$\underbrace{\frac{N_{\text{hits}}}{\text{trials}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{O}_i}_{\text{sampling}} \simeq \underbrace{\langle\mathcal{O}\rangle = \frac{\int_{-1}^{1}\mathrm{d}x\ \int_{-1}^{1}\mathrm{d}y\ \pi(x,y)\mathcal{O}(x,y)}{\int_{-1}^{1}\mathrm{d}x\ \int_{-1}^{1}\mathrm{d}y\ \pi(x,y)}}_{\text{integration}}. \tag{1.2}$$

The probability distribution $\pi(x, y)$ no longer appears on the left: rather than being evaluated, it is sampled. This is what defines the Monte Carlo method. On the left of eqn (1.2), the multiple integrals have disappeared. This means that the Monte Carlo method allows the evaluation of high-dimensional integrals, such as appear in statistical physics and other domains, if only we can think of how to generate the samples.

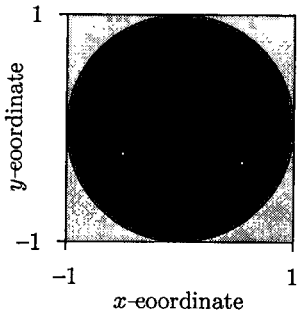

Fig. 1.5 Probability density ($\pi = 1$ inside square, zero outside) and observable ($\mathcal{O} = 1$ inside circle, zero outside) in the Monte Carlo games.

Direct sampling, the approach inspired by the children's game, is like pure gold: a subroutine provides an independent hit at the distribution function $\pi(\mathbf{x})$, that is, it generates vectors $\mathbf{x}$ with a probability proportional to $\pi(\mathbf{x})$. Notwithstanding the randomness in the problem, direct sampling, in computation, plays a role similar to exact solutions in analytical work, and the two are closely related. In direct sampling, there is no throwing-range issue; no worrying about initial conditions (the clubhouse), and a straightforward error analysis—at least if $\pi(\mathbf{x})$ and $\mathcal{O}(\mathbf{x})$

are well behaved. Many successful Monte Carlo algorithms contain exact sampling as a key ingredient.

Markov-chain sampling, on the other hand, forces us to be much more careful with all aspects of our calculation. The critical issue here is the correlation time, during which the pebble keeps a memory of the starting configuration, the clubhouse. This time can become astronomical. In the usual applications, one is often satisfied with a handful of independent samples, obtained through week-long calculations, but it can require much thought and experience to ensure that even this modest goal is achieved. We shall continue our discussion of Markov-chain Monte Carlo methods in Subsection 1.1.4, but want to first take a brief look at the history of stochastic computing.

1.1.3 Historical origins

The idea of direct sampling was introduced into modern science in the late 1940s by the mathematician Ulam, not without pride, as one can find out from his autobiography *Adventures of a Mathematician* (Ulam (1991)). Much earlier, in 1777, the French naturalist Buffon (1707–1788) imagined a legendary needle-throwing experiment, and analyzed it completely. All through the eighteenth and nineteenth centuries, royal courts and learned circles were intrigued by this game, and the theory was developed further. After a basic treatment of the Buffon needle problem, we shall describe the particularly brilliant idea of Barbier (1860), which foreshadows modern techniques of variance reduction.

The Count is shown in Fig. 1.6 randomly throwing needles of length a onto a wooden floor with cracks a distance b apart. We introduce

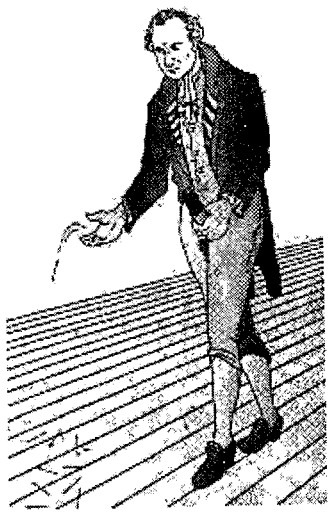

Fig. 1.6 Georges Louis Leclerc, Count of Buffon (1707–1788), performing the first recorded Monte Carlo simulation, in 1777. (Published with permission of *Le Monde*.)

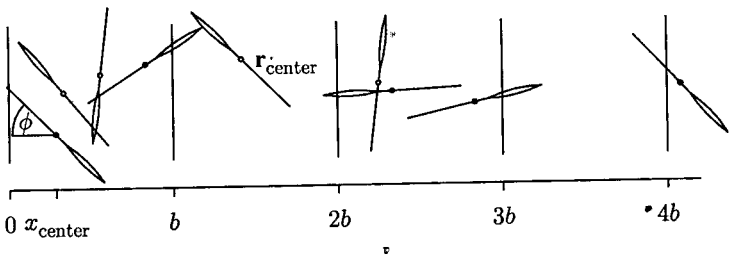

Fig. 1.7 Variables x_{center} and ϕ in Buffon's needle experiment. The needles are of length a.

coordinates $\mathbf{r}_{\text{center}}$ and ϕ as in Fig. 1.7, and assume that the needles' centers $\mathbf{r}_{\text{center}}$ are uniformly distributed on an infinite floor. The needles do not roll into cracks, as they do in real life, nor do they interact with each other. Furthermore, the angle ϕ is uniformly distributed between 0 and 2π. This is the mathematical model for Buffon's experiment.

All the cracks in the floor are equivalent, and there are symmetries $x_{\text{center}} \leftrightarrow b - x_{\text{center}}$ and $\phi \leftrightarrow -\phi$. The variable y is irrelevant to the

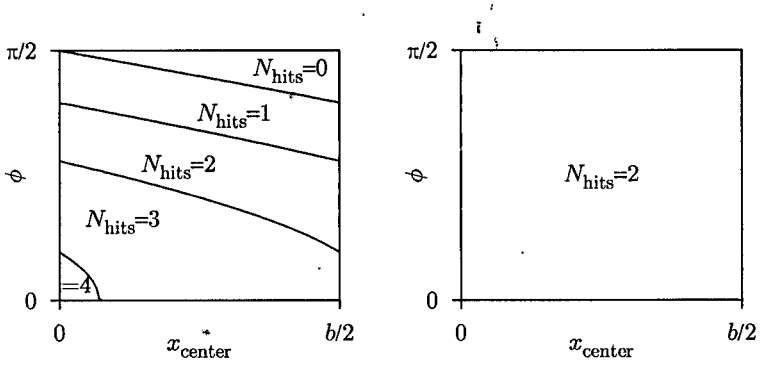

Fig. 1.15 "Landing pads" for the Buffon needle experiment with $a = \pi b$. *Left*: straight needles. *Right*: crazy cobbler's needles.

differ, and only the mean numbers of hits (the mean of N_{hits} over the whole pad) agree.

Barbier's trick is an early example of variance reduction, a powerful strategy for increasing the precision of Monte Carlo calculations. It comes in many different guises and shows that there is great freedom in finding the optimal setup for a computation.

1.1.4 Detailed balance

We left the lady and the heliport, in Subsection 1.1.2, without clarifying why the strange piles in Fig. 1.3 had to be built. Instead of the heliport game, let us concentrate on a simplified discrete version, the 3×3 pebble game shown in Fig. 1.17. The pebble can move in at most four directions: up, down, left, and right. In this subsection, we perform a complete analysis of Markov-chain algorithms for the pebble game, which is easily generalized to the heliport.

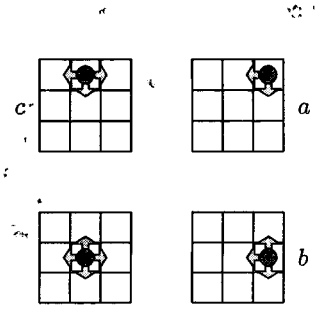

Fig. 1.17 Discrete pebble game. The corner configuration a is in contact with configurations b and c.

We seek an algorithm for moving the pebble one step at a time such

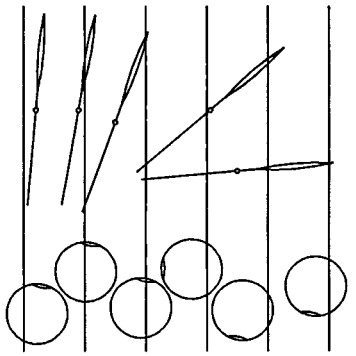

Fig. 1.16 Straight needles of length πb, with between zero and four hits, and round (crazy cobbler's) needles, which always hit twice.

that, after many iterations, it appears with the same probability in each of the fields. Anyone naive who had never watched ladies at heliports would simply chuck the pebble a few times in a random direction, i.e. one of four directions from the center, one of three directions from the edges, or one of two directions from the corners. But this natural algorithm is wrong. To understand why we must build piles, let us consider the corner configuration a, which is in contact with the configurations b and c (see Fig. 1.17). Our algorithm (yet to be found) must generate the configurations a, b, and c with prescribed probabilities $\pi(a)$, $\pi(b)$, and $\pi(c)$, respectively, which we require to be equal. This means that we want to create these configurations with probabilities

$$\{\pi(a), \pi(b), \ldots\} : \left\{ \begin{array}{c} \text{stationary probability} \\ \text{for the system to be at } a, b, \text{ etc.} \end{array} \right\}, \tag{1.9}$$

with the help of our Monte Carlo algorithm, which is nothing but a set of transition probabilities $p(a \to b)$ for moving from one configuration to the other (from a to b),

$$\{p(a \to b), p(a \to c), \ldots\} : \left\{ \begin{array}{c} \text{probability of the algorithm} \\ \text{to move from } a \text{ to } b, \text{ etc.} \end{array} \right\}.$$

Furthermore, we enforce a normalization condition which tells us that the pebble, once at a, can either stay there or move on to b or c:

$$p(a \to a) + p(a \to b) + p(a \to c) = 1. \tag{1.10}$$

The two types of probabilities can be linked by observing that the configuration a can only be generated from b or c or from itself:

$$\pi(a) = \pi(b)p(b \to a) + \pi(c)p(c \to a) + \pi(a)p(a \to a), \tag{1.11}$$

which gives

$$\pi(a)[1 - p(a \to a)] = \pi(b)p(b \to a) + \pi(c)p(c \to a).$$

Writing eqn (1.10) as $1 - p(a \to a) = p(a \to b) + p(a \to c)$ and introducing it into the last equation yields

$$\pi(a)\underbrace{p(a \to b) + \pi(a)\overbrace{p(a \to c) = \pi(c)}p(c \to a) + \pi(b)}p(b \to a).$$

This equation can be satisfied by equating the braced terms separately, and thus we arrive at the crucial condition of detailed balance,

$$\left\{ \begin{array}{c} \text{detailed} \\ \text{balance} \end{array} \right\} : \begin{array}{l} \pi(a)p(a \to b) = \pi(b)p(b \to a) \\ \pi(a)p(a \to c) = \pi(c)p(c \to a) \end{array} \quad \text{etc.} \tag{1.12}$$

This rate equation renders consistent the Monte Carlo algorithm (the probabilities $\{p(a \to b)\}$) and the prescribed stationary probabilities $\{\pi(a), \pi(b), \ldots\}$.

In the pebble game, detailed balance is satisfied because all probabilities for moving between neighboring sites are equal to 1/4, and the

probabilities $p(a \to b)$ and the return probabilities $p(b \to a)$ are trivially identical. Now we see why the pebbles have to pile up on the sides and in the corners: all the transition probabilities to neighbors have to be equal to 1/4. But a corner has only two neighbors, which means that half of the time we can leave the site, and half of the time we must stay, building up a pile.

Of course, there are more complicated choices for the transition probabilities which also satisfy the detailed-balance condition. In addition, this condition is sufficient but not necessary for arriving at $\pi(a) = \pi(b)$ for all neighboring sites a and b. On both counts, it is the quest for simplicity that guides our choice.

To implement the pebble game, we could simply modify Alg. 1.2 (`markov-pi`) using integer variables $\{k_x, k_y\}$, and installing the moves of Fig. 1.17 with a few lines of code. Uniform integer random numbers $\texttt{nran}(-1, 1)$ (that is, random integers taking values $\{-1, 0, 1\}$, see Subsection 1.2.2) would replace the real random numbers. With variables $\{k_x, k_y, k_z\}$ and another contraption to select the moves, such a program can also simulate three-dimensional pebble games.

Table 1.3 Neighbor table for the 3×3 pebble game

Site k	Nbr(..., k) 1	2	3	4
1	2	4	0	0
2	3	5	1	0
3	0	6	2	0
4	5	7	0	1
5	6	8	4	2
6	0	9	5	3
7	8	0	0	4
8	9	0	7	5
9	0	0	8	6

A smart device, the neighbor table, lets us simplify the code in a decisive fashion: this addresses each site by a number (see Fig. 1.18) rather than its Cartesian coordinates, and provides the orientation (see Table 1.3). An upward move is described, not by going from $\{k_x, k_y\}$ to $\{k_x, k_y + 1\}$, but as moving from a site k to its second neighbor $(\text{Nbr}(2, k))$. All information about boundary conditions, dimensions, lattice structure, etc. can thus be outsourced into the neighbor table, whereas the core program to simulate the Markov chain remains unchanged (see Alg. 1.6 (`markov-discrete-pebble`)). This program can be written in a few moments, for neighbor relations as in Table 1.3. It visits the nine sites equally often if the run is long enough.

We have referred to $\pi(a)$ as a stationary probability. This simple concept often leads to confusion because it involves ensembles. To be completely correct, we should imagine a Monte Carlo simulation simultaneously performed on a large number of heliports, each with its own

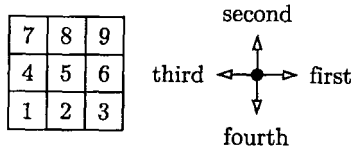

Fig. 1.18 Numbering and neighbor scheme for the 3×3 pebble game. The first neighbor of site 5 is site 6, etc.