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 `ran` $(-1, 1)$ produce uniformly distributed real random numbers between -1 and 1. Subsequent calls yield independent numbers.

procedure `direct-pi`
$N_{\rm hits} \leftarrow 0$ (initialize)
for $i = 1, \ldots, N$ **do**
$\left\{ \begin{array}{l} x \leftarrow$ `ran` $(-1, 1) \\ y \leftarrow$ `ran` $(-1, 1) \\$ **if** $(x^2 + y^2 < 1)$ $N_{\rm hits} \leftarrow N_{\rm hits} + 1 \end{array} \right.$
output $N_{\rm hits}$

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-

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

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

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

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:

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

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.

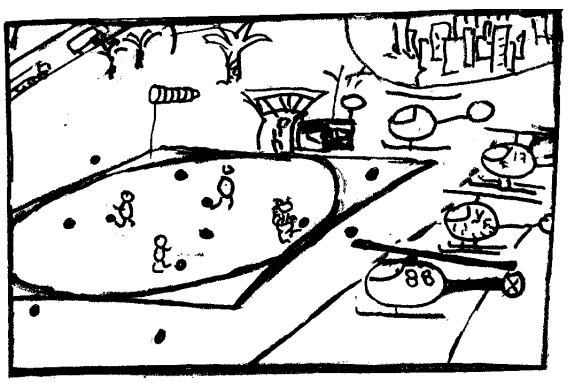

Fig. 1.2 Adults computing the number π at the Monte Carlo heliport.

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.

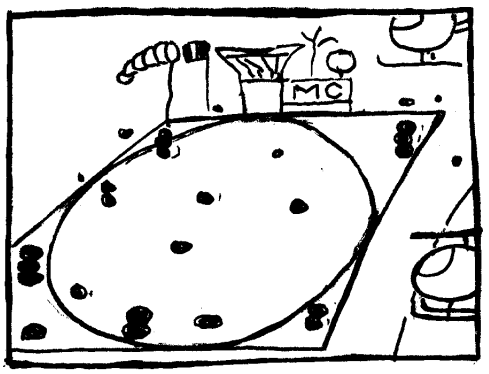

Fig. 1.3 Landing pad of the heliport at the end of the game.

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 of hits to trials.

Those who hear this story for the first time often find it dubious. They observe that perhaps one should not pile up stones, as in Fig. 1.3, if the aim is to spread them out evenly. This objection places these modern critics in the illustrious company of prominent physicists and mathematicians who questioned the validity of this method when it was first published in 1953 (it was applied to the hard-disk system of Chapter 2). Letters were written, arguments were exchanged, and the issue was settled only after several months. Of course, at the time, helicopters and heliports were much less common than they are today.

A proof of correctness and an understanding of this method, called the Metropolis algorithm, will follow later, in Subsection 1.1.4. Here, we start by programming the adults' algorithm according to the above prescription: go from one configuration to the next by following a random 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 the pad is rejected: we do not move, and count the configuration a second time (see Fig. 1.4).

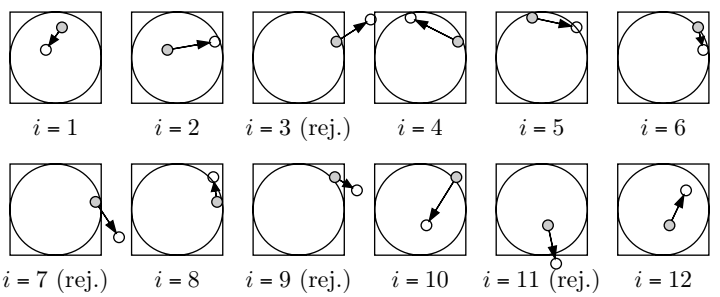

Fig. 1.4 Simulation of Alg. 1.2 (`markov-pi`). A rejection leaves the configuration unchanged (see frames $i = 3, 7, 9, 11$).

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

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

```
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 (x^2 + y^2 < 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)
$\Delta_x \leftarrow \ldots$
$\Delta_y \leftarrow \ldots$
$\vdots$
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

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

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.

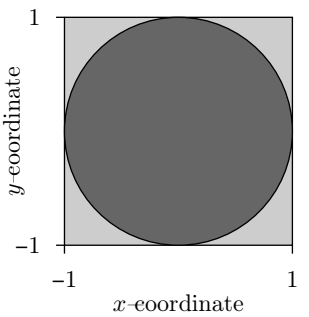

Fig. 1.5 Probability density ($\pi = 1$ inside square, zero outside) and observable ($\mathcal{O} = 1$ inside circle, zero outside) in the Monte Carlo games.

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.

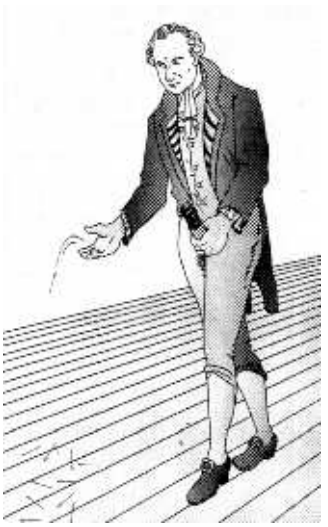

Fig. 1.6 Georges Louis Leclerc, Count of Buffon (1707–1788), performing the first recorded Monte Carlo simulation, in 1777. (Published with permission of *Le Monde*.)

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

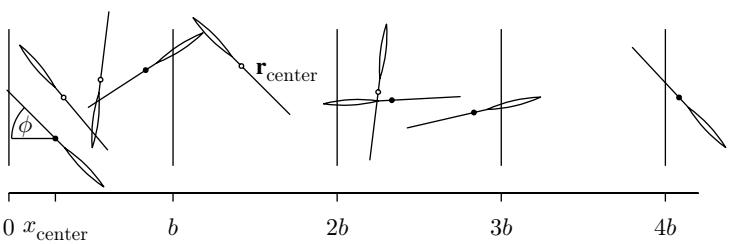

Fig. 1.7 Variables x_{center} and ϕ in Buffon's needle experiment. The needles are of length a.

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

problem. We may thus consider a reduced "landing pad", in which

$$0 < \phi < \frac{\pi}{2}, \tag{1.3}$$

$$0 < x_{\text{center}} < \frac{b}{2}. \tag{1.4}$$

The tip of each needle is at an x-coordinate $x_{\text{tip}} = x_{\text{center}} - (a/2)\cos\phi$; and every $x_{\text{tip}} < 0$ signals a hit on a crack. More precisely, the observable to be evaluated on this landing pad is (writing x for x_{center})

$$N_{\text{hits}}(x,\phi) = \left\{ \begin{array}{c} \#\text{ of hits of needle centered at } x, \\ \text{with orientation } \phi \end{array} \right\}$$
$$= \begin{cases} 1 & \text{for } x < a/2 \text{ and } |\phi| < \arccos\left[x/(a/2)\right] \\ 0 & \text{otherwise} \end{cases}.$$

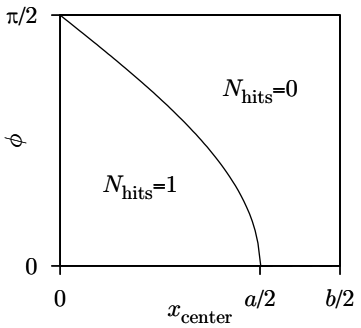

Fig. 1.8 "Landing pad" for the Buffon needle experiment for $a < b$.

The mean number of hits of a needle of length a on cracks is given, finally, by the normalized integral of the function N_{hits} over the landing pad from Fig. 1.8:

$$\left\{ \begin{array}{c} \text{mean number} \\ \text{of hits per needle} \end{array} \right\} = \langle N_{\text{hits}} \rangle = \frac{\int_0^{b/2} \mathrm{d}x \int_0^{\pi/2} \mathrm{d}\phi \; N_{\text{hits}}(x,\phi)}{\int_0^{b/2} \mathrm{d}x \int_0^{\pi/2} \mathrm{d}\phi}. \tag{1.5}$$

Integrating (over ϕ) a function which is equal to one in a certain interval and zero elsewhere yields the length of that interval ($\arccos\left[x/(a/2)\right]$), and we find, with a suitable rescaling of x,

$$\langle N_{\text{hits}} \rangle = \frac{a/2}{(b/2)(\pi/2)} \int_0^1 \mathrm{d}x \;\; \arccos x = \ldots .$$

We might try to remember how to integrate $\arccos x$ and, in passing, marvel at how Buffon—an eighteenth-century botanist—might have done it, until it becomes apparent to us that in eqn (1.5) it is wiser to first integrate over x, and then over ϕ, so that the "$\phi = \arccos\; x$" turns into "$x = \cos\phi$":

$$\ldots = \frac{a}{b}\frac{2}{\pi} \int_0^{\pi/2} \mathrm{d}\phi \;\; \cos\phi = \frac{a}{b} \cdot \frac{2}{\pi} \quad (a \le b). \tag{1.6}$$

For a needle as long as the floorboards are wide ($a = b$), the mean number of crossings is $2/\pi$. We should also realize that a needle shorter than the distance between cracks ($a \le b$) cannot hit two of them at once. The number of hits is then either 0 or 1. Therefore, the probability for a needle to hit a crack is the same as the mean number of hits:

$$\left\{ \begin{array}{c} \text{probability of} \\ \text{hitting a crack} \end{array} \right\} = \pi(N_{\text{hits}} \ge 1),$$
$$\left\{ \begin{array}{c} \text{mean number} \\ \text{of hits} \end{array} \right\} = \pi(N_{\text{hits}} = 1) \cdot 1 + \underbrace{\pi(N_{\text{hits}} = 2)}_{=0} \cdot 2 + \ldots .$$

We can now write a program to do the Buffon experiment ourselves, by simply taking x_{center} as a random number between 0 and $b/2$ and ϕ as a random angle between 0 and $\pi/2$. It remains to check whether or not the tip of the needle is on the other side of the crack (see Alg. 1.4 (`direct-needle`)).

procedure `direct-needle`
$N_{N_{\text{hits}}} \leftarrow 0$
for $i = 1, \ldots, N$ **do**
$\left\{ \begin{array}{l} x_{\text{center}} \leftarrow \texttt{ran}\,(0, b/2) \\ \phi \leftarrow \texttt{ran}\,(0, \pi/2) \\ x_{\text{tip}} \leftarrow x_{\text{center}} - (a/2)\texttt{cos}\ \phi \\ \textbf{if}\ (x_{\text{tip}} < 0)\ N_{\text{hits}} \leftarrow N_{\text{hits}} + 1 \end{array} \right.$
output N_{hits}

Algorithm 1.4 `direct-needle`. Implementing Buffon's experiment for needles of length a on the reduced pad of eqn (1.4) ($a \leq b$).

On closer inspection, Alg. 1.4 (`direct-needle`) is inelegant, as it computes the number π but also uses it as input (on line 5). There is also a call to a nontrivial cosine function, distorting the authenticity of our implementation. Because of these problems with π and $\cos\phi$, Alg. 1.4 (`direct-needle`) is a cheat! Running it is like driving a vintage automobile (wearing a leather cap on one's head) with a computer-controlled engine just under the hood and an airbag hidden inside the wooden steering wheel. To provide a historically authentic version of Buffon's experiment, stripped down to the elementary functions, we shall adapt the children's game and replace the pebbles inside the circle by needles (see Fig. 1.9). The pebble–needle trick allows us to sample a random angle $\phi = \texttt{ran}\,(0, 2\pi)$ in an elementary way and to compute $\sin\phi$ and $\cos\phi$ without actually calling trigonometric functions (see Alg. 1.5 (`direct-needle(patch)`)).

procedure `direct-needle(patch)`
$x_{\text{center}} \leftarrow \texttt{ran}\,(0, b/2)$
1 $\Delta_x \leftarrow \texttt{ran}\,(0, 1)$
$\Delta_y \leftarrow \texttt{ran}\,(0, 1)$
$\Upsilon \leftarrow \sqrt{\Delta_x^2 + \Delta_y^2}$
if $(\Upsilon > 1)$ **goto** 1
$x_{\text{tip}} \leftarrow x_{\text{center}} - (a/2)\Delta_x/\Upsilon$
$N_{\text{hits}} \leftarrow 0$
if $(x_{\text{tip}} < 0)\ N_{\text{hits}} \leftarrow 1$
output N_{hits}

Algorithm 1.5 `direct-needle(patch)`. Historically authentic version of Buffon's experiment using the pebble–needle trick.

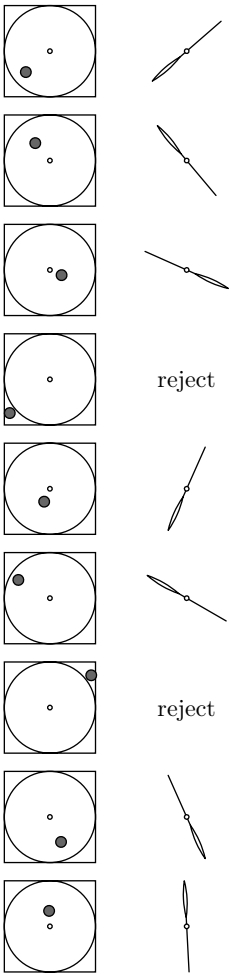

Fig. 1.9 The pebble–needle trick samples a random angle ϕ and allows us to compute $\sin\phi$ and $\cos\phi$.

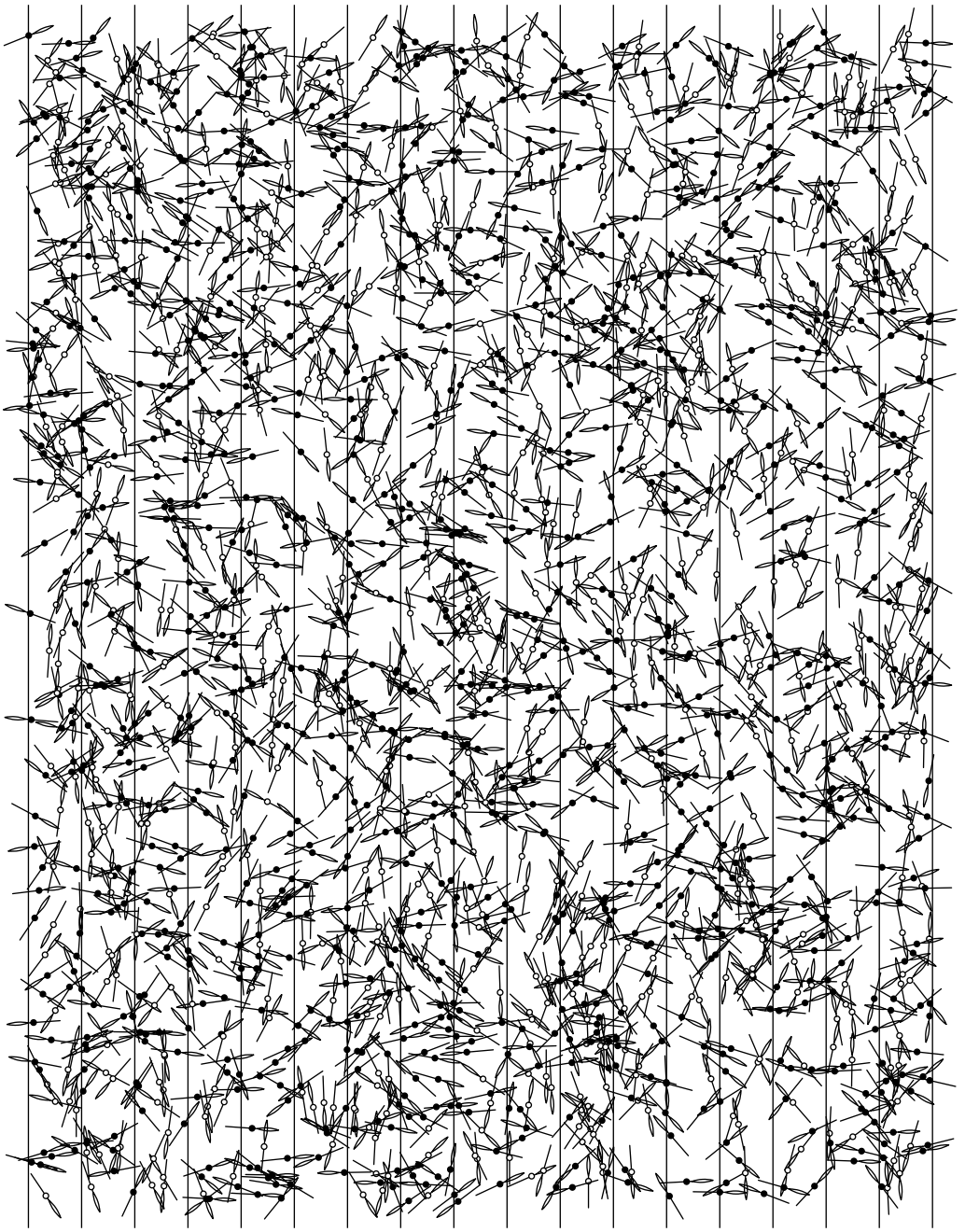

Fig. 1.10 Buffon's experiment with 2000 needles ($a = b$).

The pebble–needle trick is really quite clever, and we shall revisit it several times, in discussions of isotropic samplings in higher-dimensional hyperspheres, Gaussian random numbers, and the Maxwell distribution of particle velocities in a gas.

We can now follow in Count Buffon's footsteps, and perform our own needle-throwing experiments. One of these, with 2000 samples, is shown in Fig. 1.10.

Looking at this figure makes us wonder whether needles are more likely to intersect a crack at their tip, their center, or their eye. The full answer to this question, the subject of the present subsection, allows us to understand the factor $2/\pi$ in eqn (1.6) without any calculation.

Mathematically formulated, the question is the following: a needle hitting a crack does so at a certain value l of its interior coordinate $0 \le l \le a$ (where, say, $l = 0$ at the tip, and $l = a$ at the end of the eye). The mean number of hits, $N_{\rm hits}$, can be written as

$$\langle N_{\rm hits} \rangle = \int_0^a {\rm d}l \ \langle N_{\rm hits}(l) \rangle \, .$$

We are thus interested in $\langle N_{\rm hits}(l) \rangle$, the histogram of hits as a function of the interior coordinate l, so to speak. A probabilistic argument can be used (see Aigner and Ziegler (1992)). More transparently, we may analyze the experimental gadget shown in Fig. 1.11: two needles held together by a drop of glue.

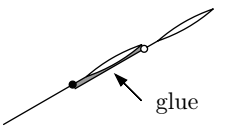

Fig. 1.11 Gadget No. 1: a white-centered and a black-centered needle, glued together.

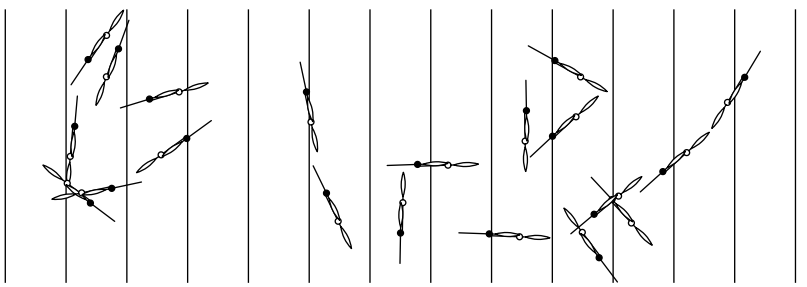

Fig. 1.12 Buffon's experiment performed with Gadget No. 1. It is impossible to tell whether black or white needles were thrown randomly.

In Fig. 1.12, we show the result of dropping this object—with its white and dark needles—on the floor. By construction (glue!), we know that

$$\langle N_{\rm hits}(a/2) \rangle_{\rm white\ needle} = \langle N_{\rm hits}(a) \rangle_{\rm black\ needle} \, .$$

However, by symmetry, both needles are deposited isotropically (see Fig. 1.12). This means that

$$\langle N_{\rm hits}(a) \rangle_{\rm black\ needle} = \langle N_{\rm hits}(a) \rangle_{\rm white\ needle} \, ,$$

and it follows that for the white needle, $\langle N_{\rm hits}(a/2) \rangle = \langle N_{\rm hits}(a) \rangle$. Gluing the needles together at different positions allows us to prove analogously

that $\langle N_{\rm hits}(l)\rangle$ is independent of l. The argument can be carried even further: clearly, the total number of hits for the gadget in Fig. 1.12 is 3/2 times that for a single needle, or, more generally,

$$\left\{ \begin{matrix} \text{mean number} \\ \text{of hits} \end{matrix} \right\} = \Upsilon \cdot \left\{ \text{length of needle} \right\}. \tag{1.7}$$

The constant Υ (Upsilon) is the same for needles of any length, smaller or larger than the distance between the cracks in the floor (we have computed it already in eqn (1.6)).

Gadgets and probabilistic arguments using them are not restricted to straight needles. Let us glue a bent cobbler's (shoemaker's) needle (see Fig. 1.13) to a straight one. We see from Fig. 1.14 that the mean number of hits where the two needles touch must be the same.

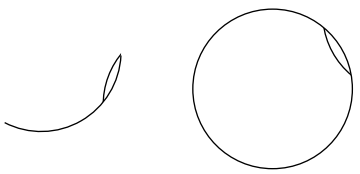

Fig. 1.13 A cobbler's needle (*left*) and a crazy cobbler's needle (*right*).

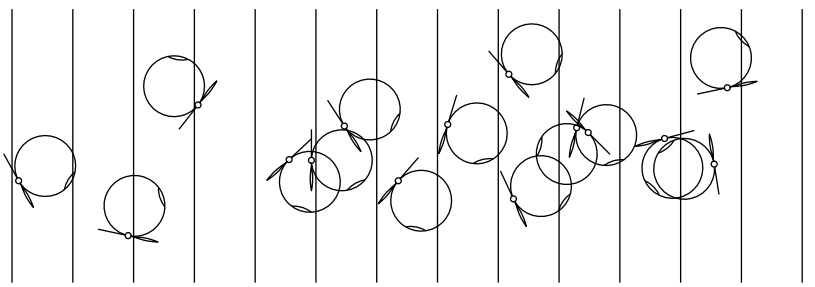

Fig. 1.14 Buffon's experiment performed with Gadget No. 2, a straight needle and a crazy cobbler's needle glued together.

This leads immediately to a powerful generalization of eqn (1.7):

$$\left\{ \begin{matrix} \text{mean} \\ \text{number of hits} \end{matrix} \right\} = \Upsilon \cdot \left\{ \begin{matrix} \text{length of needle} \\ \text{(of any shape)} \end{matrix} \right\}. \tag{1.8}$$

The constant Υ in eqn (1.8) is the same for straight needles, cobbler's needles, and even crazy cobbler's needles, needles that are bent into full circles. Remarkably, crazy cobbler's needles of length $a = \pi b$ always have two hits (see Fig. 1.16). Trivially, the mean number of hits is equal to 2 (see Fig. 1.16). This gives $\Upsilon = 2/(\pi b)$ without any calculation, and clarifies why the number π appears in Buffon's problem (see also Fig. 1.8). This ingenious observation goes back to Barbier (1860).

Over the last few pages, we have directly considered the mean number of hits $\langle N_{\rm hits}\rangle$, without speaking about probabilities. We can understand this by looking at what generalizes the square in the Monte Carlo games, namely a two-dimensional rectangle with sides $b/2$ and $\pi/2$ (see Fig. 1.15). On this generalized landing pad, the observable $\mathcal{O}(x, \phi)$, the number of hits, can take on values between 0 and 4, whereas for the crazy cobbler's needles of length πb, the number of hits is always two (see Fig. 1.15). Evidently, throwing straight needles is not the same as throwing crazy cobblers' needles—the probability distributions $\pi(N_{\rm hits})$

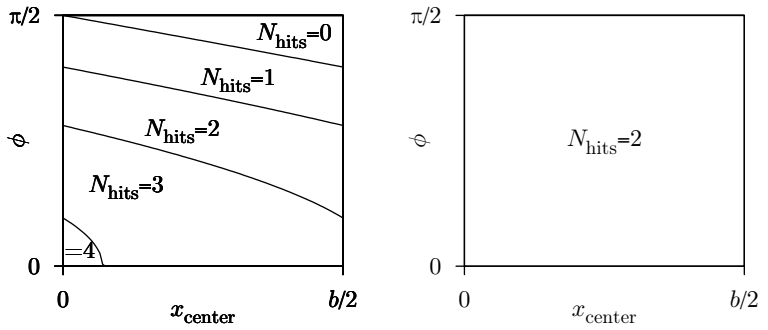

Fig. 1.15 "Landing pads" for the Buffon needle experiment with $a = \pi b$. *Left*: straight needles. *Right*: crazy cobbler's needles.

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.

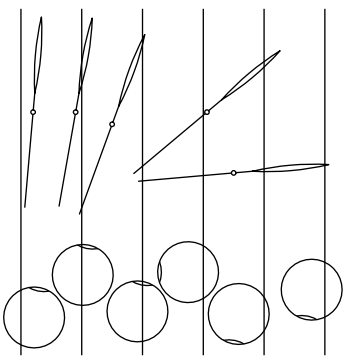

Fig. 1.16 Straight needles of length πb, with between zero and four hits, and round (crazy cobbler's) needles, which always hit twice.

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.

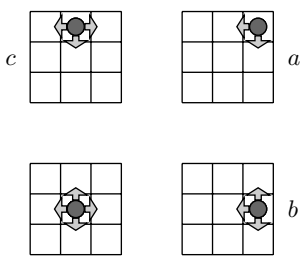

Fig. 1.17 Discrete pebble game. The corner configuration a is in contact with configurations b and c.

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

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 `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	Nbr$(\ldots, k)$			
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

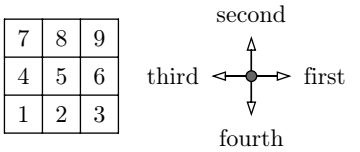

Fig. 1.18 Numbering and neighbor scheme for the 3×3 pebble game. The first neighbor of site 5 is site 6, etc.

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 (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

```
procedure markov-discrete-pebble
input k (position of pebble)
n ← nran(1, 4)
if (Nbr(n, k) ≠ 0) then (see Table 1.3)
  { k ← Nbr(n, k)
output k (next position)
```

Algorithm 1.6 `markov-discrete-pebble`. Discrete Markov-chain Monte Carlo algorithm for the pebble game.

pebble-throwing lady. In this way, we can make sense of the concept of the probability of being in configuration a at iteration i, which we have implicitly used, for example in eqn (1.11), during our derivation of the detailed-balance condition. Let us use the 3×3 pebble game to study this point in more detail. The ensemble of all transition probabilities between sites can be represented in a matrix, the system's transfer matrix P:

$$P = \{p(a \to b)\} = \begin{bmatrix} p(1 \to 1) & p(2 \to 1) & p(3 \to 1) & \dots \\ p(1 \to 2) & p(2 \to 2) & p(3 \to 2) & \dots \\ p(1 \to 3) & p(2 \to 3) & p(3 \to 3) & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{1.13}$$

The normalization condition in eqn (1.10) (the pebble must go somewhere) implies that each column of the matrix in eqn (1.13) adds up to one.

With the numbering scheme of Fig. 1.18, the transfer matrix is

$$\{p(a \to b)\} = \begin{bmatrix} \boxed{\frac{1}{2}} & \frac{1}{4} & \cdot & \frac{1}{4} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{1}{4} & \boxed{\frac{1}{4}} & \frac{1}{4} & \cdot & \frac{1}{4} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \frac{1}{4} & \boxed{\frac{1}{2}} & \cdot & \cdot & \frac{1}{4} & \cdot & \cdot & \cdot \\ \frac{1}{4} & \cdot & \cdot & \boxed{\frac{1}{4}} & \frac{1}{4} & \cdot & \frac{1}{4} & \cdot & \cdot \\ \cdot & \frac{1}{4} & \cdot & \frac{1}{4} & \boxed{0} & \frac{1}{4} & \cdot & \frac{1}{4} & \cdot \\ \cdot & \cdot & \frac{1}{4} & \cdot & \frac{1}{4} & \boxed{\frac{1}{4}} & \cdot & \cdot & \frac{1}{4} \\ \cdot & \cdot & \cdot & \frac{1}{4} & \cdot & \cdot & \boxed{\frac{1}{2}} & \frac{1}{4} & \\ \cdot & \cdot & \cdot & \cdot & \frac{1}{4} & \cdot & \frac{1}{4} & \boxed{\frac{1}{4}} & \frac{1}{4} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \frac{1}{4} & \cdot & \frac{1}{4} & \boxed{\frac{1}{2}} \end{bmatrix}, \tag{1.14}$$

where the symbols "$\cdot$" stand for zeros. All simulations start at the clubhouse, site 9 in our numbering scheme. For the ensemble of Monte Carlo simulations, this means that the probability vector at iteration $i = 0$ is

$$\{\pi^0(1), \dots, \pi^0(9)\} = \{0, \dots, 0, 1\}.$$

After one iteration of the Monte Carlo algorithm, the pebble is at the clubhouse with probability 1/2, and at positions 6 and 8 with probabilities 1/4. This is mirrored by the vector $\{\pi^{i=1}(1), \ldots, \pi^{i=1}(9)\}$ after one iteration, obtained by a matrix–vector multiplication

$$\pi^{i+1}(a) = \sum_{b=1}^{9} p(b \to a)\pi^i(b) \tag{1.15}$$

for $i = 0$, and $i + 1 = 1$. Equation (1.15) is easily programmed (see Alg. 1.7 (`transfer-matrix`); for the matrix in eqn (1.13), the eqn (1.15) corresponds to a matrix–vector multiplication, with the vector to the right). Repeated application of the transfer matrix to the initial probability vector allows us to follow explicitly the convergence of the Monte Carlo algorithm (see Table 1.4 and Fig. 1.19).

procedure `transfer-matrix`
input $\{p(a \to b)\}$ (matrix in eqn (1.14))
input $\{\pi^i(1), \ldots, \pi^i(9)\}$
for $a = 1, \ldots, 9$ **do**
  $\pi^{i+1}(a) \leftarrow 0$
  for $b = 1, \ldots, 9$ **do**
    $\pi^{i+1}(a) \leftarrow \pi^{i+1}(a) + p(b \to a)\pi^i(b)$
output $\{\pi^{i+1}(1), \ldots, \pi^{i+1}(9)\}$

Algorithm 1.7 `transfer-matrix`. Computing pebble-game probabilities at iteration $i + 1$ from the probabilities at iteration i.

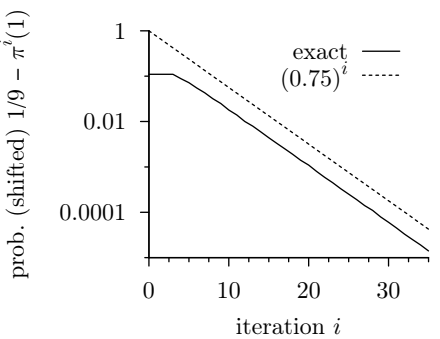

Fig. 1.19 Pebble-game probability of site 1, shifted by 1/9 (from Alg. 1.7 (`transfer-matrix`); see Table 1.4).

To fully understand convergence in the pebble game, we must analyze the eigenvectors $\{\boldsymbol{\pi}_e^1, \ldots, \boldsymbol{\pi}_e^9\}$ and the eigenvalues $\{\lambda_1, \ldots, \lambda_9\}$ of the transfer matrix. The eigenvectors $\boldsymbol{\pi}_e^k$ are those vectors that essentially reproduce under the application of P:

$$P\boldsymbol{\pi}_e^k = \lambda_k \boldsymbol{\pi}_e^k.$$

Table 1.4 Input/output of Alg. 1.7 (`transfer-matrix`), initially started at the clubhouse

Prob.	Iteration i				
	0	1	2	...	∞
$\pi^i(1)$	0	0	0	...	1/9
$\pi^i(2)$	0	0	0	...	1/9
$\pi^i(3)$	0	0	0.062	...	1/9
$\pi^i(4)$	0	0	0	...	1/9
$\pi^i(5)$	0	0	1/8	...	1/9
$\pi^i(6)$	0	1/4	0.188	...	1/9
$\pi^i(7)$	0	0	0.062	...	1/9
$\pi^i(8)$	0	1/4	0.188	...	1/9
$\pi^i(9)$	1	1/2	0.375	...	1/9

Writing a probability vector $\boldsymbol{\pi} = \{\pi(1), \ldots, \pi(9)\}$ in terms of the eigenvectors, i.e.

$$\boldsymbol{\pi} = \alpha_1 \boldsymbol{\pi}_e^1 + \alpha_2 \boldsymbol{\pi}_e^2 + \cdots + \alpha_9 \boldsymbol{\pi}_e^9 = \sum_{k=1}^{9} \alpha_k \boldsymbol{\pi}_e^k,$$

allows us to see how it is transformed after one iteration,

$$P\boldsymbol{\pi} = \alpha_1 P\boldsymbol{\pi}_e^1 + \alpha_2 P\boldsymbol{\pi}_e^2 + \cdots + \alpha_9 P\boldsymbol{\pi}_e^9 = \sum_{k=1}^{9} \alpha_k P\boldsymbol{\pi}_e^k$$
$$= \alpha_1 \lambda_1 \boldsymbol{\pi}_e^1 + \alpha_2 \lambda_2 \boldsymbol{\pi}_e^2 + \cdots + \alpha_9 \lambda_9 \boldsymbol{\pi}_e^9 = \sum_{k=1}^{9} \alpha_k \lambda_k \boldsymbol{\pi}_e^k,$$

or after i iterations,

$$P^i\boldsymbol{\pi} = \alpha_1 \lambda_1^i \boldsymbol{\pi}_e^1 + \alpha_2 \lambda_2^i \boldsymbol{\pi}_e^2 + \cdots + \alpha_9 \lambda_9^i \boldsymbol{\pi}_e^9 = \sum_{k=1}^{9} \alpha_k (\lambda_k)^i \boldsymbol{\pi}_e^k.$$

Only one eigenvector has components that are all nonnegative, so that it can be a vector of probabilities. This vector must have the largest eigenvalue λ_1 (the matrix P being positive). Because of eqn (1.10), we have $\lambda_1 = 1$. Other eigenvectors and eigenvalues can be computed explicitly, at least in the 3×3 pebble game. Besides the dominant eigenvalue λ_1, there are two eigenvalues equal to 0.75, one equal to 0.5, etc. This allows us to follow the precise convergence towards the asymptotic equilibrium solution:

$$\{\pi^i(1), \ldots, \pi^i(9)\}$$
$$= \underbrace{\{\tfrac{1}{9}, \ldots, \tfrac{1}{9}\}}_{\substack{\text{first eigenvector} \\ \text{eigenvalue } \lambda_1 = 1}} + \alpha_2 \cdot (0.75)^i \underbrace{\{-0.21, \ldots, 0.21\}}_{\substack{\text{second eigenvector} \\ \text{eigenvalue } \lambda_2 = 0.75}} + \cdots .$$

In the limit $i \to \infty$, the contributions of the subdominant eigenvectors disappear and the first eigenvector, the vector of stationary probabilities in eqn (1.9), exactly reproduces under multiplication by the transfer matrix. The two are connected through the detailed-balance condition, as discussed in simpler terms at the beginning of this subsection.

The difference between $\{\pi^i(1), \ldots, \pi^i(9)\}$ and the asymptotic solution is determined by the second largest eigenvalue of the transfer matrix and is proportional to

$$(0.75)^i = \mathrm{e}^{i \cdot \log 0.75} = \exp\left(-\frac{i}{3.476}\right). \tag{1.16}$$

The data in Fig. 1.19 clearly show the $(0.75)^i$ behavior, which is equivalent to an exponential $\propto \mathrm{e}^{-i/\Delta_i}$ where $\Delta_i = 3.476$. Δ_i is a timescale, and allows us to define short times and long times: a short simulation

has fewer than Δ_i iterations, and a long simulation has many more than that.

In conclusion, we see that transfer matrix iterations and Monte Carlo calculations reach equilibrium only after an infinite number of iterations. This is not a very serious restriction, because of the existence of a timescale for convergence, which is set by the second largest eigenvalue of the transfer matrix. To all intents and purposes, the asymptotic equilibrium solution is reached after the convergence time has passed a few times. For example, the pebble game converges to equilibrium after a few times 3.476 iterations (see eqn (1.16)). The concept of equilibrium is far-reaching, and the interest in Monte Carlo calculations is rightly strong because of this timescale, which separates fast and slow processes and leads to exponential convergence.

1.1.5 The Metropolis algorithm

In Subsection 1.1.4, direct inspection of the detailed-balance condition in eqn (1.12) allowed us to derive Markov-chain algorithms for simple games where the probability of each configuration was either zero or one. This is not the most general case, even for pebbles, which may be less likely to be at a position a on a hilltop than at another position b located in a valley (so that $\pi(a) < \pi(b)$). Moves between positions a and b with arbitrary probabilities $\pi(a)$ and $\pi(b)$, respecting the detailed-balance condition in eqn (1.12), are generated by the Metropolis algorithm (see Metropolis *et al.* (1953)), which accepts a move $a \to b$ with probability

$$p(a \to b) = \min\left[1, \frac{\pi(b)}{\pi(a)}\right]. \tag{1.17}$$

In the heliport game, we have unknowingly used eqn (1.17): for a and b both inside the square, the move was accepted without further tests ($\pi(b)/\pi(a) = 1$, $p(a \to b) = 1$). In contrast, for a inside but b outside the square, the move was rejected ($\pi(b)/\pi(a) = 0$, $p(a \to b) = 0$).

Table 1.5 Metropolis algorithm represented by eqn (1.17): detailed balance holds because the second and fourth rows of this table are equal

Case	$\pi(a) > \pi(b)$	$\pi(b) > \pi(a)$
$p(a \to b)$	$\pi(b)/\pi(a)$	1
$\pi(a)p(a \to b)$	$\pi(b)$	$\pi(a)$
$p(b \to a)$	1	$\pi(a)/\pi(b)$
$\pi(b)p(b \to a)$	$\pi(b)$	$\pi(a)$

To prove eqn (1.17) for general values of $\pi(a)$ and $\pi(b)$, one has only to write down the expressions for the acceptance probabilities $p(a \to b)$ and $p(b \to a)$ from eqn (1.17) for the two cases $\pi(a) > \pi(b)$ and $\pi(b) > \pi(a)$ (see Table 1.5). For $\pi(a) > \pi(b)$, one finds that $\pi(a)p(a \to b) =$

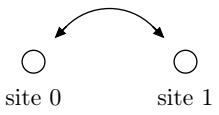

Fig. 1.20 Two-site problem. The probabilities to be at site 0 and site 1 are proportional to $\pi(0)$ and $\pi(1)$, respectively.

$\pi(b)p(b \to a) = \pi(b)$. In this case, and likewise for $\pi(b) > \pi(a)$, detailed balance is satisfied. This is all there is to the Metropolis algorithm.

Let us implement the Metropolis algorithm for a model with just two sites: site 0, with probability $\pi(0)$, and site 1, with $\pi(1)$, probabilities that we may choose to be arbitrary positive numbers (see Fig. 1.20). The pebble is to move between the sites such that, in the long run, the times spent on site 0 and on site 1 are proportional to $\pi(0)$ and $\pi(1)$, respectively. This is achieved by computing the ratio of statistical weights $\pi(1)/\pi(0)$ or $\pi(0)/\pi(1)$, and comparing it with a random number $\texttt{ran}\,(0,1)$, a procedure used by almost all programs implementing the Metropolis algorithm (see Fig. 1.21 and Alg. 1.8 (`markov-two-site`)).

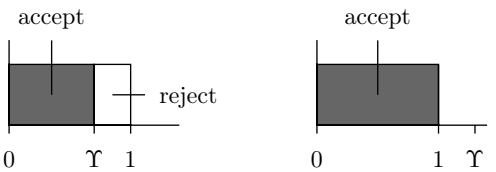

Fig. 1.21 Accepting a move with probability $\min(1, \Upsilon)$ with the help of a random number $\texttt{ran}\,(0,1)$.

We may run this program for a few billion iterations, using the output of iteration i as the input of iteration $i+1$. While waiting for the output, we can also clean up Alg. 1.8 (`markov-two-site`) a bit, noticing that if $\Upsilon > 1$, its comparison with a random number between 0 and 1 makes no sense: the move will certainly be accepted. For $\pi(l) > \pi(k)$, we should thus work around the calculation of the quotient, the generation of a random number and the comparison with that number.

procedure `markov-two-site`
input k (either 0 or 1)
if $(k = 0)$ $l \leftarrow 1$
if $(k = 1)$ $l \leftarrow 0$
$\Upsilon \leftarrow \pi(l)/\pi(k)$
if $(\texttt{ran}\,(0,1) < \Upsilon)$ $k \leftarrow l$
output k (next site)

Algorithm 1.8 `markov-two-site`. Sampling sites 0 and 1 with stationary probabilities $\pi(0)$ and $\pi(1)$ by the Metropolis algorithm.

1.1.6 A priori probabilities, triangle algorithm

On the heliport, the moves Δ_x and Δ_y were restricted to a small square of edge length 2δ, the throwing range, centered at the present position (see Fig. 1.22(A)). This gives an example of an a priori probability distribution, denoted by $\mathcal{A}(a \to b)$, from which we sample the move $a \to b$,

that is, which contains all possible moves in our Markov-chain algorithm, together with their probabilities.

The small square could be replaced by a small disk without bringing in anything new (see Fig. 1.22(B)). A much more interesting situation arises if asymmetric a priori probabilities are allowed: in the triangle algorithm of Fig. 1.22(C), we sample moves from an oriented equilateral triangle centered at a, with one edge parallel to the x-axis. This extravagant choice may lack motivation in the context of the adults' game, but contains a crucial ingredient of many modern Monte Carlo algorithms.

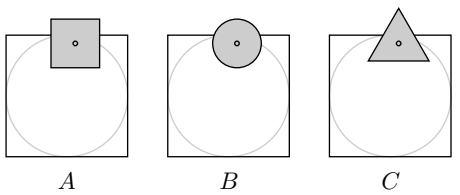

Fig. 1.22 Throwing pattern in Alg. 1.2 (`markov-pi`) (A), with variants. The triangle algorithm (C) needs special attention.

In fact, detailed balance can be reconciled with any a priori probability $\mathcal{A}(a \to b)$, even a triangular one, by letting the probability $\mathcal{P}(a \to b)$ for moving from a to b be composite:

$$\mathcal{P}(a \to b) = \underbrace{\mathcal{A}(a \to b)}_{\text{consider } a \to b} \cdot \underbrace{p(a \to b)}_{\text{accept } a \to b} .$$

The probability of moving from a to b must satisfy $\pi(a)\mathcal{P}(a \to b) = \pi(b)\mathcal{P}(b \to a)$, so that the acceptance probabilities obey

$$\frac{p(a \to b)}{p(b \to a)} = \frac{\pi(b)}{\mathcal{A}(a \to b)} \frac{\mathcal{A}(b \to a)}{\pi(a)}.$$

This leads to a generalized Metropolis algorithm

$$p(a \to b) = \min \left[1, \frac{\pi(b)}{\mathcal{A}(a \to b)} \frac{\mathcal{A}(b \to a)}{\pi(a)} \right], \tag{1.18}$$

also called the Metropolis–Hastings algorithm. We shall first check the new concept of an a priori probability with the familiar problem of the heliport game with the small square: as the pebble throw $a \to b$ appears with the same probability as the return throw $b \to a$, we have $\mathcal{A}(a \to b) = \mathcal{A}(b \to a)$, so that the generalized Metropolis algorithm is the same as the old one.

The triangle algorithm is more complicated: both the probability of the move $a \to b$ and that of the return move $b \to a$ must be considered in order to balance the probabilities correctly. It can happen, for example, that the probability $\mathcal{A}(a \to b)$ is finite, but that the return probability $\mathcal{A}(b \to a)$ is zero (see Fig. 1.23). In this case, the generalized Metropolis algorithm in eqn (1.18) imposes rejection of the original pebble throw

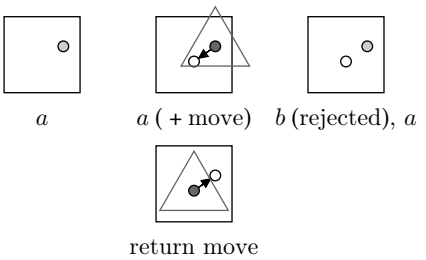

Fig. 1.23 Rejected move $a \to b$ in the triangle algorithm.

from a to b. (Alg. 7.3 (`direct-triangle`) allows us to sample a random point inside an arbitrary triangle).

The triangle algorithm can be generalized to an arbitrary a priori probability $\mathcal{A}(a \to b)$, and the generalized Metropolis algorithm (eqn (1.18)) will ensure that the detailed-balance condition remains satisfied. However, only good choices for $\mathcal{A}(a \to b)$ have an appreciable acceptance rate (the acceptance probability of each move averaged over all moves) and actually move the chain forward. As a simple example, we can think of a configuration a with a high probability ($\pi(a)$ large), close to configurations b with $\pi(b)$ small. The original Metropolis algorithm leads to many rejections in this situation, slowing down the simulation. Introducing a priori probabilities to propose configurations b less frequently wastes less computer time with rejections. Numerous examples in later chapters illustrate this point.

A case worthy of special attention is $\mathcal{A}(a \to b) = \pi(b)$ and $\mathcal{A}(b \to a) = \pi(a)$, for which the acceptance rate in eqn (1.18) of the generalized Metropolis algorithm is equal to unity: we are back to direct sampling, which we abandoned because we did not know how to put it into place. However, no circular argument is involved. A priori probabilities are crucial when we can almost do direct sampling, or when we can almost directly sample a subsystem. A priori probabilities then present the computational analogue of perturbation theory in theoretical physics.

1.1.7 Perfect sampling with Markov chains

The difference between the ideal world of children (direct sampling) and that of adults (Markov-chain sampling) is clear-cut: in the former, direct access to the probability distribution $\pi(\mathbf{x})$ is possible, but in the latter, convergence towards $\pi(\mathbf{x})$ is reached only in the long-time limit. Controlling the error from within the simulation poses serious difficulties: we may have the impression that we have decorrelated from the clubhouse, without suspecting that it is—figuratively speaking—still around the corner. It has taken half a century to notice that this difficulty can sometimes be resolved, within the framework of Markov chains, by producing

perfect chain samples, which are equivalent to the children's throws and guaranteed to be totally decorrelated from the initial condition.

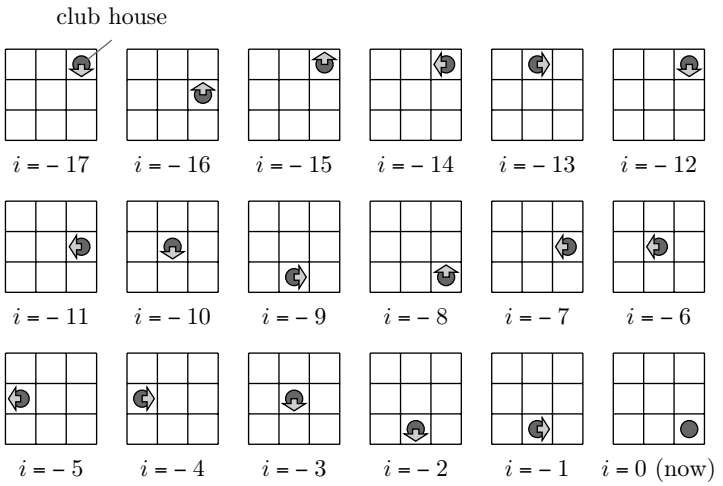

Fig. 1.24 A 3×3 pebble game starting at the clubhouse at iteration $i = -17$, arriving at the present configuration at $i = 0$ (now).

For concreteness, we discuss perfect sampling in the context of the 3×3 pebble game. In Fig. 1.24, the stone has moved in 17 steps from the clubhouse to the lower right corner. As the first subtle change in the setting, we let the simulation start at time $i = -17$, and lead up to the present time $i = 0$. Because we started at the clubhouse, the probability of being in the lower right corner at $i = 0$ is slightly smaller than $1/9$. This correlation goes to zero exponentially in the limit of long running times, as we have seen (see Fig. 1.19).

The second small change is to consider random maps rather than random moves (see Fig. 1.25: from the upper right corner, the pebble must move down; from the upper left corner, it must move right; etc.). At each iteration i, a new random map is drawn. Random maps give a consistent, alternative definition of the Markov-chain Monte Carlo method, and for any given trajectory it is impossible to tell whether it was produced by random maps or by a regular Monte Carlo algorithm (in Fig. 1.26, the trajectory obtained using random maps is the same as in Fig. 1.24).

In the random-map Markov chain of Fig. 1.26, it can, furthermore, be verified explicitly that any pebble position at time $i = -17$ leads to the lower right corner at iteration $i = 0$. In addition, we can imagine that $i = -17$ is not really the initial time, but that the simulation has been going on since $i = -\infty$. There have been random maps all along the way, and Fig. 1.26 shows only the last stretch. The pebble position at $i = 0$ is the same for any configuration at $i = -17$: it is also the outcome of an infinite simulation, with an initial position at $i = -\infty$, from which it has

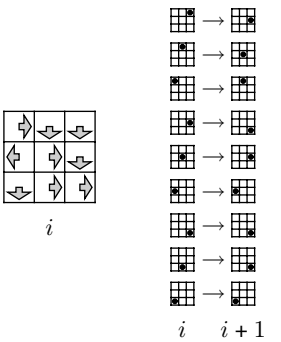

Fig. 1.25 A random map at iteration i and its action on all possible pebble positions.

decorrelated. The $i = 0$ pebble position in the lower right corner is thus a direct sample—obtained by a Markov-chain Monte Carlo method.

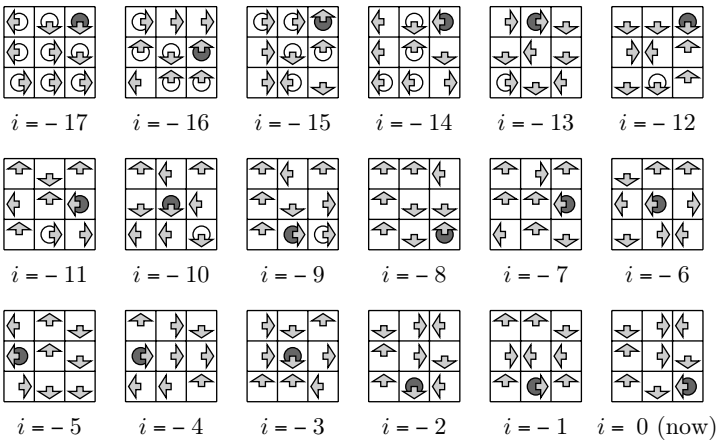

Fig. 1.26 Monte Carlo dynamics using time-dependent random maps. All positions at $i = -17$ give an identical output at $i = 0$.

The idea of perfect sampling is due to Propp and Wilson (1996). It mixes a minimal conceptual extension of the Monte Carlo method (random maps) with the insight that a finite amount of backtracking (called coupling from the past) may be sufficient to figure out the present state of a Markov chain that has been running forever (see Fig. 1.27).

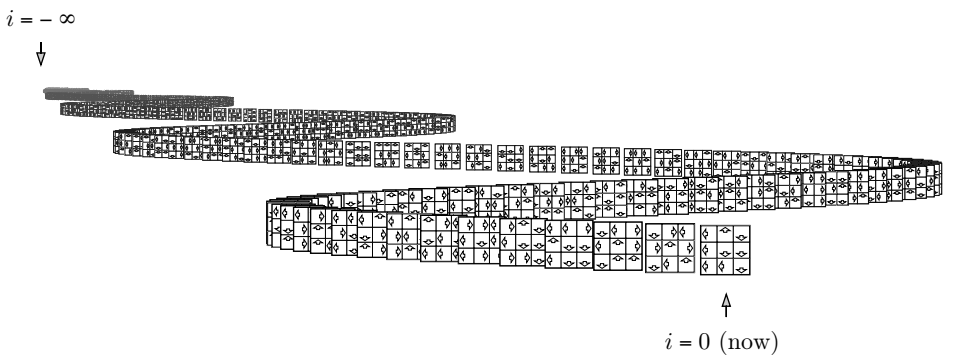

Fig. 1.27 A random-map Markov chain that has been running since $i = -\infty$.

Producing direct samples for a 3×3 pebble game by Markov chains is a conceptual breakthrough, but not yet a great technical achievement. Later on, in Chapter 5, we shall construct direct samples (using Markov chains) with $2^{100} = 1\,267\,650\,600\,228\,229\,401\,496\,703\,205\,376$ configurations. Going through all of them to see whether they have merged is

out of the question, but we shall see that it is sometimes possible to squeeze all configurations in between two extremal ones: if those two configurations have come together, all others have merged, too.

Understanding and running a coupling-from-the-past program is the ultimate in Monte Carlo style—much more elegant than walking around a heliport, well dressed and with a fancy handbag over one's shoulder, waiting for the memory of the clubhouse to more or less fade away.

1.2 Basic sampling

On several occasions already, we have informally used uniform random numbers x generated through a call $x \leftarrow \texttt{ran}\,(a, b)$. We now discuss the two principal aspects of random numbers. First we must understand how random numbers enter a computer, a fundamentally deterministic machine. In this first part, we need little operational understanding, as we shall always use routines written by experts in the field. We merely have to be aware of what can go wrong with those routines. Second, we shall learn how to reshape the basic building block of randomness—$\texttt{ran}\,(0, 1)$—into various distributions of random integers and real numbers, permutations and combinations, N-dimensional random vectors, random coordinate systems, etc. Later chapters will take this program much further: $\texttt{ran}\,(0, 1)$ will be remodeled into random configurations of liquids and solids, boson condensates, and mixtures, among other things.

1.2.1 Real random numbers

Random number generators (more precisely, pseudorandom number generators), the subroutines which produce $\texttt{ran}\,(0, 1)$, are intricate deterministic algorithms that condense into a few dozen lines a lot of clever number theory and probabilities, all rigorously tested. The output of these algorithms looks random, but is not: when run a second time, under exactly the same initial conditions, they always produce an identical output. Generators that run in the same way on different computers are called "portable". They are generally to be preferred. Random numbers have many uses besides Monte Carlo calculations, and rely on a solid theoretical and empirical basis. Routines are widely available, and their writing is a mature branch of science. Progress has been fostered by essential commercial applications in coding and cryptography. We certainly do not have to conceive such algorithms ourselves and, in essence, only need to understand how to test them for our specific applications.

Every modern, good $\texttt{ran}\,(0, 1)$ routine has a flat probability distribution. It has passed a battery of standard statistical tests which would have detected unusual correlations between certain values $\{x_i, \ldots, x_{i+k}\}$ and other values $\{x_j, \ldots, x_{j+k'}\}$ later down the sequence. Last but not least, the standard routines have been successfully used by many people before us. However, all the meticulous care taken and all the endorsement by others do not insure us against the small risk that the particular random number generator we are using may in fact fail in our particular

problem. To truly convince ourselves of the quality of a complicated calculation that uses a given random number generator, it remains for us (as end users) to replace the random number generator *in the very simulation program we are using* by a second, different algorithm. By the definition of what constitutes randomness, this change of routine should have no influence on the results (inside the error bars). Therefore, if changing the random number generator in our simulation program leads to no systematic variations, then the two generators are almost certainly OK for our application. There is nothing more we can do and nothing less we should do to calm our anxiety about this crucial ingredient of Monte Carlo programs.

Algorithm 1.9 (`naive-ran`) is a simple example—useful for study, but unsuited for research—of linear congruential[3] random number generators, which are widely installed in computers, pocket calculators, and other digital devices. Very often, such generators are the building blocks of good algorithms.

Table 1.6 Repeated calls to Alg. 1.9 (`naive-ran`). Initially, the seed was set to idum ← 89053.

#	idum	ran
1	123456	0.91819
2	110651	0.82295
3	55734	0.41451
4	65329	0.48588
5	1844	0.01371
6	78919	0.58695
...	...	...
134457	123456	...
134458	110651	...
...	...	...

procedure `naive-ran`
$m \leftarrow 134456$
$n \leftarrow 8121$
$k \leftarrow 28411$
input idum
idum $\leftarrow$ `mod`(idum $\cdot\, n + k, m$)
`ran` $\leftarrow$ idum/`real`(m)
output idum, `ran`

Algorithm 1.9 `naive-ran`. Low-quality portable random number generator, `naive-ran`$(0, 1)$, using a linear congruential method.

In Alg. 1.9 (`naive-ran`), the parameters $\{m, n, k\}$ have been carefully adjusted, whereas the variable {idum}, called the seed, is set at the beginning, but never touched again from the outside during a run of the program. Once started, the sequence of pseudorandom numbers unravels. Just like the sequence of any other generator, even the high-quality ones, it is periodic. In Alg. 1.9 (`naive-ran`), the periodicity is 134 456 (see Table 1.6); in good generators, the periodicity is much larger than we shall ever be able to observe.

Let us denote real random numbers, uniformly distributed between values a and b, by the abstract symbol `ran`(a, b), without regard for initialization and the choice of algorithm (we suppose it to be perfect). In the printed routines, repeated calls to `ran`(a, b), such as

$$\begin{aligned} x &\leftarrow \texttt{ran}\,(-1, 1)\,, \\ y &\leftarrow \texttt{ran}\,(-1, 1)\,, \end{aligned} \tag{1.19}$$

generate statistically independent random values for x and y. Later, we shall often use a concise vector notation in our programs. The two

[3]Two numbers are *congruent* if they agree with each other, i.e. if their difference is divisible by a given modulus: 12 is congruent to 2 (modulo 5), since $12 - 2 = 2 \times 5$.

variables $\{x, y\}$ in eqn (1.19), for example, may be part of a vector $\mathbf{x}$, and we may assign independent random values to the components of this vector by the call

$$\mathbf{x} \leftarrow \{\mathtt{ran}\,(-1,1)\,, \mathtt{ran}\,(-1,1)\}.$$

(For a discussion of possible conflicts between vectors and random numbers, see Subsection 1.2.6.)

Depending on the context, random numbers may need a little care. For example, the logarithm of a random number between 0 and 1, $x \leftarrow \log \mathtt{ran}\,(0,1)$, may have to be replaced by

```
1  ϒ ← ran(0,1)
   if (ϒ = 0) goto 1 (reject number)
   x ← log ϒ
```

to avoid overflow ($\Upsilon = 0$, $x = -\infty$) and a crash of the program after a few hours of running. To avoid this problem, we might define the random number $\mathtt{ran}\,(0,1)$ to be always larger than 0 and smaller than 1. However this does not get us out of trouble: a well-implemented random number generator between 0 and 1, always satisfying

$$0 < \mathtt{ran}\,(0,1) < 1,$$

might be used to implement a routine $\mathtt{ran}\,(1,2)$. The errors of finite-precision arithmetic could lead to an inconsistent implementation where, owing to rounding, $1 + \mathtt{ran}\,(0,1)$ could turn out to be exactly equal to one, even though we want $\mathtt{ran}\,(1,2)$ to satisfy

$$1 < \mathtt{ran}\,(1,2) < 2.$$

Clearly, great care is called for, in this special case but also in general: Monte Carlo programs, notwithstanding their random nature, are extremely sensitive to small bugs and irregularities. They have to be meticulously written: under no circumstance should we accept routines that need an occasional manual restart after a crash, or that sometimes produce data which has to be eliminated by hand. Rare problems, for example logarithms of zero or random numbers that are equal to 1 but should be strictly larger, quickly get out of control, lead to a loss of trust in the output, and, in short, leave us with a big mess ....

1.2.2 Random integers, permutations, and combinations

Random variables in a Monte Carlo calculation are not necessarily real-valued. Very often, we need uniformly distributed random integers m, between (and including) k and l. In this book, such a random integer is generated by the call $m \leftarrow \mathtt{nran}\,(k,l)$. In the implementation in Alg. 1.10 (`nran`), the **if ( )** statement (on line 4) provides extra protection against rounding problems in the underlying $\mathtt{ran}\,(k, l+1)$ routine.

```
      procedure nran
      input {k, l}
1     m ← int(ran(k, l + 1))
      if (m > l) goto 1
      output m
```

Algorithm 1.10 `nran`. Uniform random integer `nran`(k, l) between (and including) k and l.

The next more complicated objects, after integers, are permutations of K distinct objects, which we may take to be the integers $\{1, \ldots, K\}$. A permutation P can be written as a two-row matrix[4]

$$P = \begin{pmatrix} P_1 & P_2 & P_3 & P_4 & P_5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}. \tag{1.20}$$

We can think of the permutation in eqn (1.20) as balls labeled $\{1, \ldots, 5\}$ in the order $\{P_1, \ldots, P_5\}$, on a shelf (ball P_k is at position k). The order of the columns is without importance in eqn (1.20), and P can also be written as $P = \left(\begin{smallmatrix} P_1 & P_3 & P_4 & P_2 & P_5 \\ 1 & 3 & 4 & 2 & 5 \end{smallmatrix}\right)$: information about the placing of balls is not lost, and we still know that ball P_k is in the kth position on the shelf. Two permutations P and Q can be multiplied, as shown in the example below, where the columns of Q are first rearranged such that the lower row of Q agrees with the upper row of P. The product PQ consists of the lower row of P and the upper row of the rearranged Q:

$$\overbrace{\left(\begin{smallmatrix} 1&4&3&2&5 \\ 1&2&3&4&5 \end{smallmatrix}\right)}^{P}\overbrace{\left(\begin{smallmatrix} 1&3&2&4&5 \\ 1&2&3&4&5 \end{smallmatrix}\right)}^{Q} = \overbrace{\left(\begin{smallmatrix} 1&4&3&2&5 \\ 1&2&3&4&5 \end{smallmatrix}\right)}^{P}\ \overbrace{\left(\begin{smallmatrix} 1&4&2&3&5 \\ 1&4&3&2&5 \end{smallmatrix}\right)}^{Q\ \text{(rearranged)}} = \overbrace{\left(\begin{smallmatrix} 1&4&2&3&5 \\ 1&2&3&4&5 \end{smallmatrix}\right)}^{PQ}. \tag{1.21}$$

On the shelf, with balls arranged in the order $\{P_1, \ldots, P_5\}$, the multiplication of P from the right by another permutation $Q = \left(\begin{smallmatrix} Q_1 & Q_2 & Q_3 & Q_4 & Q_5 \\ 1 & 2 & 3 & 4 & 5 \end{smallmatrix}\right)$ replaces the ball k with Q_k (or, equivalently, the ball P_k by Q_{P_k}).

The identity $\left(\begin{smallmatrix} 1 & 2 & \cdots & K \\ 1 & 2 & \cdots & K \end{smallmatrix}\right)$ is a special permutation. Transpositions are the same as the identity permutation, except for two elements which are interchanged ($P_k = l$ and $P_l = k$). The second factor in the product in eqn (1.21) is a transposition. Any permutation of K elements can be built up from at most $K - 1$ transpositions.

Permutations can also be arranged into disjoint cycles

$$P = \left(\underbrace{\begin{matrix} P_2 & P_3 & P_4 & P_1 \\ P_1 & P_2 & P_3 & P_4 \end{matrix}}_{\text{first cycle}} \underbrace{\begin{matrix} P_6 & P_7 & P_8 & P_9 & P_5 \\ P_5 & P_6 & P_7 & P_8 & P_9 \end{matrix}}_{\text{second cycle}} \underbrace{\begin{matrix} . & . & . & . & . \\ . & . & . & . & . \end{matrix}}_{\text{other cycles}} \right), \tag{1.22}$$

which can be written in a cycle representation as

$$P = (P_1, P_2, P_3, P_4)(P_5, \ldots, P_9)(\ldots)(\ldots). \tag{1.23}$$

[4]Anticipating later applications in quantum physics, we write permutations "bottom-up" as $\left(\begin{smallmatrix} P_1 & \cdots & P_K \\ 1 & \ldots & K \end{smallmatrix}\right)$ rather than "top-down" $\left(\begin{smallmatrix} 1 & \cdots & K \\ P_1 & \cdots & P_K \end{smallmatrix}\right)$, as is more common.

In this representation, we simply record in one pair of parentheses that P_1 is followed by P_2, which is in turn followed by P_3, etc., until we come back to P_1. The order of writing the cycles is without importance. In addition, each cycle of length k has k equivalent representations. We could, for example, write the permutation P of eqn (1.23) as

$$P = (P_5, \ldots, P_9)(P_4, P_1, P_2, P_3)(\ldots)(\ldots).$$

Cycle representations will be of central importance in later chapters; in particular, the fact that every cycle of k elements can be reached from the identity by means of $k-1$ transpositions. As an example, we can see that multiplying the identity permutation $\left(\begin{smallmatrix}1&2&3&4\\1&2&3&4\end{smallmatrix}\right)$ by the three transpositions of $(1,2)$, $(1,3)$, and $(1,4)$ gives

$$\underbrace{\left(\begin{smallmatrix}1&2&3&4\\1&2&3&4\end{smallmatrix}\right)}_{\text{identity}}\underbrace{\left(\begin{smallmatrix}2&1&3&4\\1&2&3&4\end{smallmatrix}\right)}_{1\leftrightarrow 2}\underbrace{\left(\begin{smallmatrix}2&3&1&4\\2&1&3&4\end{smallmatrix}\right)}_{1\leftrightarrow 3}\underbrace{\left(\begin{smallmatrix}2&3&4&1\\2&3&1&4\end{smallmatrix}\right)}_{1\leftrightarrow 4} = \left(\begin{smallmatrix}2&3&4&1\\1&2&3&4\end{smallmatrix}\right) = (1,2,3,4),$$

a cycle of four elements. More generally, we can now consider a permutation P of K elements containing n cycles, with $\{k_1, \ldots, k_n\}$ elements. The first cycle, which has k_1 elements, is generated from the identity by $k_1 - 1$ transpositions, the second cycle by $k_2 - 1$, etc. The total number of transpositions needed to reach P is $(k_1 - 1) + \cdots + (k_n - 1)$, but since $K = k_1 + \cdots + k_n$, we can see that the number of transpositions is $K - n$. The sign of a permutation is positive if the number of transpositions from the identity is even, and odd otherwise (we then speak of even and odd permutations). We see that

$$\text{sign } P = (-1)^{K-n} = (-1)^{K+n} = (-1)^{\#\text{ of transpositions}}. \tag{1.24}$$

We can always add extra transpositions and undo them later, but the sign of the permutation remains the same. Let us illustrate the crucial relation between the number of cycles and the sign of a permutation by an example:

$$P = \left(\begin{smallmatrix}4&2&5&7&6&3&8&1\\1&2&3&4&5&6&7&8\end{smallmatrix}\right) = (1,4,7,8)(2)(3,5,6).$$

This is a permutation of $K = 8$ elements with $n = 3$ cycles. It must be odd, because of eqn (1.24). To see this, we rearrange the columns of P to make the elements of the same cycle come next to each other:

$$P = \begin{pmatrix} 4 & 7 & 8 & 1 & 2 & 5 & 6 & 3 \\ \underbrace{1 & 4 & 7 & 8}_{\text{first cycle}} & 2 & \underbrace{3 & 5 & 6}_{\text{third cycle}} \end{pmatrix}. \tag{1.25}$$

In this representation, it is easy to see that the first cycle is generated in three transpositions from $\left(\begin{smallmatrix}1&4&7&8\\1&4&7&8\end{smallmatrix}\right)$; the second cycle, consisting of the element 2, needs no transposition; and the third cycle is generated in two transpositions from $\left(\begin{smallmatrix}3&5&6\\3&5&6\end{smallmatrix}\right)$. The total number of transpositions is five, and the permutation is indeed odd.

Let us now sample random permutations, i.e. generate one of the $K!$ permutations with equal probability. In our picture of permutations as

balls on a shelf, it is clear that a random permutation can be created by placing all K balls into a bucket, and randomly picking out one after the other and putting them on the shelf, in their order of appearance. Remarkably, one can implement this procedure with a single vector of length K, which serves both as the bucket and as the shelf (see Alg. 1.11 (`ran-perm`); for an example, see Table 1.7). We may instead stop the

procedure `ran-perm`
$\{P_1, \ldots, P_K\} \leftarrow \{1, \ldots, K\}$
for $k = 1, \ldots, K-1$ **do**
$\left\{ \begin{array}{l} l \leftarrow \texttt{nran}\,(k, K) \\ P_l \leftrightarrow P_k \end{array} \right.$
output $\{P_1, \ldots, P_K\}$

Algorithm 1.11 `ran-perm`. Generating a uniformly distributed random permutation of K elements.

process after M steps, rather than K ($M < K$), to sample a random combination (see Alg. 1.12 (`ran-combination`)).

procedure `ran-combination`
$\{P_1, \ldots, P_K\} \leftarrow \{1, \ldots, K\}$
for $k = 1, \ldots, M$ **do**
$\left\{ \begin{array}{l} l \leftarrow \texttt{nran}\,(k, K) \\ P_l \leftrightarrow P_k \end{array} \right.$
output $\{P_1, \ldots, P_M\}$

Algorithm 1.12 `ran-combination`. Generating a uniformly distributed random combination of M elements from K.

Table 1.7 Example run of Alg. 1.11 (`ran-perm`). In each step k, the numbers k and l are underlined.

#	P_1	P_2	P_3	P_4	P_5
	$\underline{1}$	2	3	$\underline{4}$	5
1					
	4	$\underline{2}$	$\underline{3}$	1	5
2					
	4	3	$\underline{\underline{2}}$	1	5
3					
	4	3	2	$\underline{1}$	$\underline{5}$
4					
	4	3	2	5	1

These two programs are the most basic examples of random combinatorics, a mature branch of mathematics, where natural, easy-to-prove algorithms are right next to tough ones. For illustration, we consider a closely related problem of (a few) black dots and many white dots

$$(\bullet\ \bullet\ \bullet\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ)\,,$$

which we want to mix, as in

$$(\circ\ \bullet\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \bullet\ \circ\ \circ\ \circ\ \circ\ \bullet\ \circ\ \circ\ \circ)\,.$$

The fastest algorithm for mixing the three black dots with the white ones is a slightly adapted version of Alg. 1.12 (`ran-combination`): first swap the black dot at position 1 with the element at position $j = \texttt{nran}\,(1, 18)$, then exchange whatever is at position 2 with the element at $\texttt{nran}\,(2, 18)$, and finally swap the contents of the positions 3 and $\texttt{nran}\,(3, 18)$. The resulting configuration of dots is indeed a random mixture, but it takes a bit of thought to prove this.

1.2.3 Finite distributions

The sampling of nonuniform finite distributions has an archetypal example in the Saturday night problem. We imagine K possible evening activities that we do not feel equally enthusiastic about: study ($k = 1$, probability $\pi_1 \gtrsim 0$), chores ($k = 2$, probability $\pi_2 \lll 1$), cinema, book writing, etc. The probabilities π_k are all known, but we may still have trouble deciding what to do. This means that we have trouble in sampling the distribution $\{\pi_1, \ldots, \pi_K\}$. Two methods allow us to solve the Saturday night problem: a basic rejection algorithm, and tower sampling, a rejection-free approach.

```
     procedure reject-finite
     π_max ← max_{k=1}^{K} π_k
1    k ← nran(1, K)
     ϒ ← ran(0, π_max)
     if (ϒ > π_k) goto 1
     output k
```

Algorithm 1.13 `reject-finite`. Sampling a finite distribution $\{\pi_1, \ldots, \pi_K\}$ with a rejection algorithm.

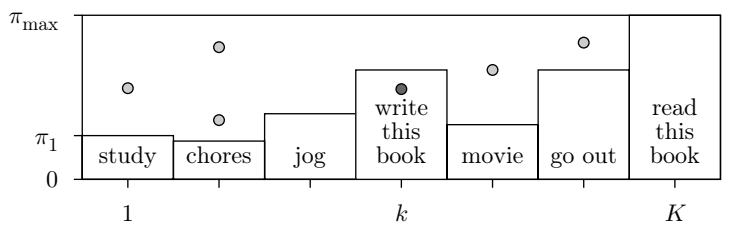

Fig. 1.28 Saturday night problem solved by Alg. 1.13 (`reject-finite`).

In the rejection method (see Alg. 1.13 (`reject-finite`)), pebbles are randomly thrown into a big frame containing boxes for all the activities, whose sizes represent their probabilities. Eventually, one pebble falls into one of them and makes our choice. Clearly, the acceptance rate of the algorithm is given by the ratio of the sum of the volumes of all boxes to the volume of the big frame and is equal to $\langle \pi \rangle / \pi_{\max}$, where the mean probability is $\langle \pi \rangle = \sum_k \pi_k / K$. This implies that on average we have to throw $\pi_{\max} / \langle \pi \rangle$ pebbles before we have a hit. This number can easily become so large that the rejection algorithm is not really an option.

Tower sampling is a vastly more elegant solution to the Saturday night problem. Instead of placing the boxes next to each other, as in Fig. 1.28, we pile them up (see Fig. 1.29). Algorithm 1.14 (`tower-sample`) keeps track of the numbers $\Pi_1 = \pi_1, \Pi_2 = \pi_1 + \pi_2$, etc. With a single random number $\mathtt{ran}(0, \Pi_K)$, an activity k is then chosen. There is no rejection.

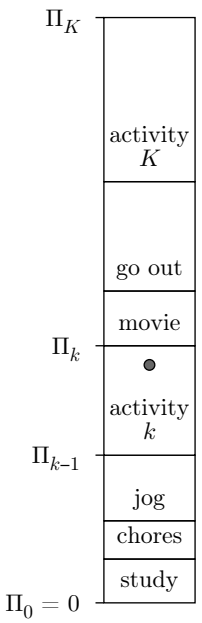

Fig. 1.29 Saturday night problem solved by tower sampling.

procedure `tower-sample`
input $\{\pi_1, \ldots, \pi_K\}$
$\Pi_0 \leftarrow 0$
for $l = 1, \ldots, K$ **do** $\Pi_l \leftarrow \Pi_{l-1} + \pi_l$
$\Upsilon \leftarrow$ `ran` $(0, \Pi_K)$
$*$ find k with $\Pi_{k-1} < \Upsilon < \Pi_k$
output k

Algorithm 1.14 `tower-sample`. Tower sampling of a finite distribution $\{\pi_1, \ldots, \pi_K\}$ without rejections.

Tower sampling can be applied to discrete distributions with a total number K in the hundreds, thousands, or even millions. It often works when the naive rejection method of Fig. 1.28 fails because of too many rejections. Tower sampling becomes impracticable only when the probabilities $\{\pi_1, \ldots, \pi_K\}$ can no longer be listed.

In Alg. 1.14 (`tower-sample`), we must clarify how we actually find the element k, i.e. how we implement the line marked by an asterisk. For small K, we may go through the ordered table $\{\Pi_0, \ldots, \Pi_K\}$ one by one, until the element k, with $\Pi_{k-1} < \Upsilon \leq \Pi_k$, is encountered. For large K, a bisection method should be implemented if we are making heavy use of tower sampling: we first check whether Υ is smaller or larger than $\Pi_{K/2}$, and then cut the possible range of indices k in half on each subsequent iteration. Algorithm 1.15 (`bisection-search`) terminates in about $\log_2 K$ steps.

procedure `bisection-search`
input $\Upsilon, \{\Pi_0, \Pi_1, \ldots, \Pi_K\}$ (ordered table with $\Pi_k \geq \Pi_{k-1}$)
$k_{\min} \leftarrow 0$
$k_{\max} \leftarrow K + 1$
for $i = 1, 2, \ldots$ **do**
  $k \leftarrow (k_{\min} + k_{\max})/2$ (integer arithmetic)
  if $(\Pi_k < \Upsilon)$ **then**
    $k_{\min} \leftarrow k$
  else if $(\Pi_{k-1} > \Upsilon)$ **then**
    $k_{\max} \leftarrow k$
  else
    output k
    exit

Algorithm 1.15 `bisection-search`. Locating the element k with $\Pi_{k-1} < \Upsilon < \Pi_k$ in an ordered table $\{\Pi_0, \ldots, \Pi_K\}$.

1.2.4 Continuous distributions and sample transformation

The two discrete methods of Subsection 1.2.3 remain meaningful in the continuum limit. For the rejection method, the arrangement of boxes in Fig. 1.28 simply becomes a continuous curve $\pi(x)$ in some range $x_{\min} < x < x_{\max}$ (see Alg. 1.16 (`reject-continuous`)). We shall often use a refinement of this simple scheme, where the function $\pi(x)$, which we want to sample, is compared not with a constant function $\pi_{\max}$ but with another function $\tilde{\pi}(x)$ that we know how to sample, and is everywhere larger than $\pi(x)$ (see Subsection 2.3.4).

```
     procedure reject-continuous
1    x ← ran(x_min, x_max)
     ϒ ← ran(0, π_max)
     if (ϒ > π(x)) goto 1 (reject sample)
     output x
     ——
```

Algorithm 1.16 `reject-continuous`. Sampling a value x with probability $\pi(x) < \pi_{\max}$ in the interval $[x_{\min}, x_{\max}]$ with the rejection method.

For the continuum limit of tower sampling, we change the discrete index k in Alg. 1.14 (`tower-sample`) into a real variable x:

$$\{k, \pi_k\} \longrightarrow \{x, \pi(x)\}$$

(see Fig. 1.30). This gives us the transformation method: the loop in the third line of Alg. 1.14 (`tower-sample`) turns into an integral formula:

$$\overbrace{\Pi_k \leftarrow \Pi_{k-1} + \pi_k}^{\text{in Alg. 1.14 (tower-sample)}} \longrightarrow \overbrace{\Pi(x) = \int_{-\infty}^{x} \mathrm{d}x'\ \pi(x')}^{\Pi(x)=\Pi(x-\mathrm{d}x)+\pi(x)\mathrm{d}x}. \tag{1.26}$$

Likewise, the line marked by an asterisk in Alg. 1.14 (`tower-sample`) has an explicit solution:

$$\overbrace{\text{find } k \text{ with } \Pi_{k-1} < \Upsilon < \Pi_k}^{\text{in Alg. 1.14 (tower-sample)}} \longrightarrow \overbrace{\text{find } x \text{ with } \Pi(x) = \Upsilon}^{\text{i.e. } x=\Pi^{-1}(\Upsilon)}, \tag{1.27}$$

where Π^{-1} is the inverse function of Π.

As an example, let us sample random numbers $0 < x < 1$ distributed according to an algebraic function $\pi(x) \propto x^\gamma$ (with $\gamma > -1$) (see Fig. 1.30, which shows the case $\gamma = -\frac{1}{2}$). We find

$$\begin{aligned} \pi(x) &= (\gamma+1)x^\gamma \text{ for } 0 < x < 1, \\ \Pi(x) &= \int_0^x \mathrm{d}x\ \pi(x') = x^{\gamma+1} = \texttt{ran}\,(0,1)\,, \\ x &= \texttt{ran}\,(0,1)^{1/(\gamma+1)}\,. \end{aligned} \tag{1.28}$$

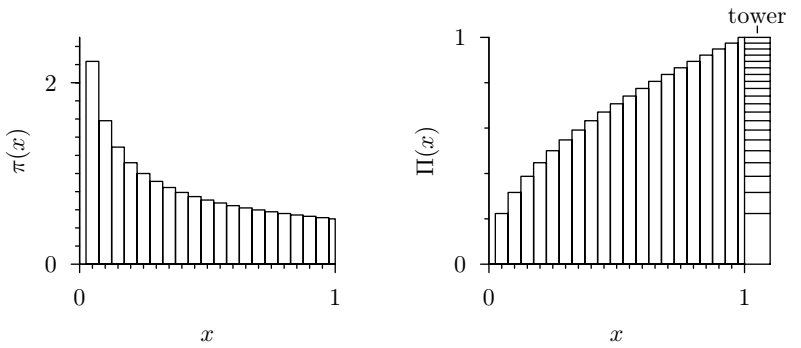

Fig. 1.30 Transformation method as the continuum limit of tower sampling.

The transformation method as laid out in eqns (1.26) and (1.27) can be interpreted as a sample transformation, stressing the unity between integration and sampling: any change of variables in an integral can be done directly with random numbers, i.e. with samples. Indeed, in the above example of an algebraic function, we can transform the integral over a flat distribution into the integral of the target distribution:

$$\int_0^1 \mathrm{d}\Upsilon \xrightarrow[\text{transform}]{\text{integral}} \text{const} \int_0^1 \mathrm{d}x\ x^\gamma .$$

The same transformation works for samples:

$$\left\{ \begin{array}{c} \text{sample } \Upsilon \\ \Upsilon = \texttt{ran}\,(0,1) \end{array} \right\} \xrightarrow[\text{transform}]{\text{sample}} \left\{ \begin{array}{c} \text{sample } x \\ \text{with } \pi(x) \propto x^\gamma \end{array} \right\} .$$

We now seek the transformation between x and Υ:

$$\mathrm{d}\Upsilon \ = \text{const} \cdot \mathrm{d}x\ x^\gamma .$$

The sample $\Upsilon = \texttt{ran}\,(0,1)$ is thus transformed as follows:

$$\texttt{ran}\,(0,1) = \Upsilon = \text{const}' \cdot x^{\gamma+1} + \text{const}'' .$$

Finally (checking that the bounds of $\texttt{ran}\,(0,1)$ correspond to $x = 0$ and $x = 1$), this results in

$$x = \texttt{ran}\,(0,1)^{1/(\gamma+1)} , \tag{1.29}$$

in agreement with eqn (1.28). (In Subsection 1.4.2, we shall consider algebraic distributions for x between 1 and ∞.)

As a second example of sample transformation, we consider random numbers that are exponentially distributed, so that $\pi(x) \propto \mathrm{e}^{-\lambda x}$ for $x \geq 0$. As before, we write

$$\int_0^1 \mathrm{d}\Upsilon \ = \text{const} \int_0^\infty \mathrm{d}x\ \mathrm{e}^{-\lambda x} \tag{1.30}$$

and seek a transformation of a flat distribution of Υ in the interval $[0,1]$ into the target distribution of x:

$$\begin{aligned} \mathrm{d}\Upsilon &= \text{const} \cdot \mathrm{d}x\ \mathrm{e}^{-\lambda x}, \\ \texttt{ran}\,(0,1) = \Upsilon &= \text{const}' \cdot \mathrm{e}^{-\lambda x} + \text{const}''. \end{aligned}$$

Checking the bounds $x = 0$ and $x = \infty$, this leads to

$$-\frac{1}{\lambda} \log \texttt{ran}\,(0,1) = x. \tag{1.31}$$

In this book, we shall often transform samples under the integral sign, in the way we have seen in the two examples of the present subsection.

1.2.5 Gaussians

In many applications, we need Gaussian random numbers y distributed with a probability

$$\pi(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y - \langle y \rangle)^2}{2\sigma^2}\right].$$

(The parameter $\langle y \rangle$ is the mean value of the distribution, and σ is the standard deviation.) One can always change variables using $x = (y - \langle y \rangle)/\sigma$ to obtain normally distributed variables x with a distribution

$$\pi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right). \tag{1.32}$$

Inversely, the normally distributed variables x can be rescaled into $y = \sigma x + \langle y \rangle$.

Naively, to sample Gaussians such as those described in eqn (1.32), one can compute the sum of a handful of independent random variables, essentially $\texttt{ran}\,(-1,1)$, and rescale them properly (see Alg. 1.17 (`naive-gauss`), the factor $1/12$ will be derived in eqn (1.55)). This program illustrates the power of the central limit theorem, which will be discussed further in Subsection 1.3.3. Even for $K = 3$, the distribution takes on the characteristic Gaussian bell shape (see Fig. 1.31).

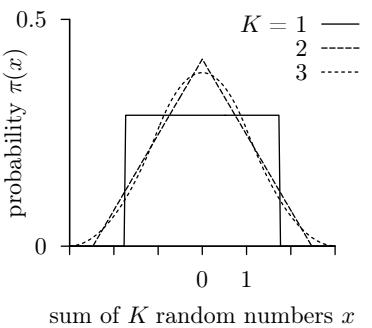

Fig. 1.31 Probability for the sum of K random numbers $\texttt{ran}\left(-\frac{1}{2}, \frac{1}{2}\right)$ for small K, rescaled as in Alg. 1.17 (`naive-gauss`).

procedure `naive-gauss`
$\sigma \leftarrow \sqrt{K/12}$
$\Sigma \leftarrow 0$
for $k = 1, \ldots, K$ **do**
  $\{\ \Sigma \leftarrow \Sigma + \texttt{ran}\left(-\frac{1}{2}, \frac{1}{2}\right)$
$x \leftarrow \Sigma/\sigma$
output x

Algorithm 1.17 `naive-gauss`. An approximately Gaussian random number obtained from the rescaled sum of K uniform random numbers.

With Alg. 1.17 (`naive-gauss`), we shall always worry whether its parameter K is large enough. For practical calculations, it is preferable to sample Gaussian random numbers without approximations. To do so, we recall the trick used to evaluate the error integral

$$\int_{-\infty}^{\infty} \frac{\mathrm{d}x}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2} = 1. \tag{1.33}$$

We square eqn (1.33)

$$\left[\int_{-\infty}^{\infty} \frac{\mathrm{d}x}{\sqrt{2\pi}} \exp\left(-x^2/2\right) \right]^2 = \int_{-\infty}^{\infty} \frac{\mathrm{d}x}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2} \int_{-\infty}^{\infty} \frac{\mathrm{d}y}{\sqrt{2\pi}} \mathrm{e}^{-y^2/2} \tag{1.34}$$

$$= \int_{-\infty}^{\infty} \frac{\mathrm{d}x\,\mathrm{d}y}{2\pi} \exp\left[-(x^2+y^2)/2\right], \tag{1.35}$$

introduce polar coordinates ($\mathrm{d}x\,\mathrm{d}y = r\,\mathrm{d}r\,\mathrm{d}\phi$),

$$\ldots = \int_0^{2\pi} \frac{\mathrm{d}\phi}{2\pi} \int_0^{\infty} r\,\mathrm{d}r\ \exp\left(-r^2/2\right),$$

and finally substitute $r^2/2 = \Upsilon$ ($r\,\mathrm{d}r = \mathrm{d}\Upsilon$)

$$\ldots = \underbrace{\int_0^{2\pi} \frac{\mathrm{d}\phi}{2\pi}}_{1} \underbrace{\int_0^{\infty} \mathrm{d}\Upsilon\ \mathrm{e}^{-\Upsilon}}_{1}. \tag{1.36}$$

Equation (1.36) proves the integral formula in eqn (1.33). In addition, the two independent integrals in eqn (1.36) are easily sampled: ϕ is uniformly distributed between 0 and 2π, and Υ is exponentially distributed and can be sampled through the negative logarithm of $\mathtt{ran}\,(0,1)$ (see eqn (1.31), with $\lambda = 1$). After generating $\Upsilon = -\log \mathtt{ran}\,(0,1)$ and $\phi = \mathtt{ran}\,(0,2\pi)$,

procedure gauss
input σ
$\phi \leftarrow \mathtt{ran}\,(0,2\pi)$
$\Upsilon \leftarrow -\mathtt{log}\ \mathtt{ran}\,(0,1)$
$r \leftarrow \sigma\sqrt{2\Upsilon}$
$x \leftarrow r\mathtt{cos}\ \phi$
$y \leftarrow r\mathtt{sin}\ \phi$
output $\{x, y\}$

Algorithm 1.18 `gauss`. Two independent Gaussian random numbers obtained by sample transformation. See Alg. 1.19 (`gauss(patch)`).

we transform the sample, as discussed in Subsection 1.2.4: a crab's walk leads from eqn (1.36) back to eqn (1.34) ($r = \sqrt{2\Upsilon}$, $x = r\cos\phi$, $y = r\sin\phi$). We finally get two normally distributed Gaussians, in one of the nicest applications of multidimensional sample transformation (see Alg. 1.18 (`gauss`)).

Algorithm 1.18 (`gauss`) can be simplified further. As discussed in Subsection 1.1.3, we can generate uniform random angles by throwing pebbles into the children's square and retaining those inside the circle (see Fig. 1.9). This pebble–needle trick yields a random angle ϕ, but allows us also to determine $\sin\phi$ and $\cos\phi$ without explicit use of trigonometric tables (see Alg. 1.19 (`gauss(patch)`)). In the printed routine, $x/\sqrt{\Upsilon'} = \cos\phi$, etc. Moreover, the variable $\Upsilon' = x^2 + y^2$, for $\{x, y\}$ inside the circle, is itself uniformly distributed, so that Υ' in Alg. 1.19 (`gauss(patch)`) can replace the `ran`$(0, 1)$ in the original Alg. 1.18 (`gauss`). This Box–Muller algorithm is statistically equivalent to but marginally faster than Alg. 1.18 (`gauss`).

```
      procedure gauss(patch)
      input σ
1     x ← ran(−1, 1)
      y ← ran(−1, 1)
      Υ′ ← x² + y²
      if (Υ′ > 1 or Υ′ = 0) goto 1 (reject sample)
      Υ ← −log Υ′
      Υ″ ← σ√(2Υ/Υ′)
      x ← Υ″x
      y ← Υ″y
      output {x, y}
```

Algorithm 1.19 `gauss(patch)`. Gaussian random numbers, as in Alg. 1.18 (`gauss`), but without calls to trigonometric functions.

1.2.6 Random points in/on a sphere

The pebble–needle trick, in its recycled version in Subsection 1.2.5, shows that any random point inside the unit disk of Fig. 1.1 can be transformed into two independent Gaussian random numbers:

$$\left\{\begin{array}{c}\text{2 Gaussian}\\ \text{samples}\end{array}\right\} \Longleftarrow \left\{\begin{array}{c}\text{random pebble in}\\ \text{unit disk}\end{array}\right\}.$$

This remarkable relation between Gaussians and pebbles is unknown to most players of games in Monaco, short or tall. The unit disk is the same as the two-dimensional unit sphere, and we may wonder whether a random point inside a d-dimensional unit sphere could be helpful for generating d Gaussians at a time. Well, the truth is exactly the other way around: sampling d Gaussians provides a unique technique for obtaining a random point in a d-dimensional unit sphere:

$$\left\{\begin{array}{c}d\text{ Gaussian}\\ \text{samples}\end{array}\right\} \Longrightarrow \left\{\begin{array}{c}\text{random point in}\\ d\text{-dimensional unit sphere}\end{array}\right\}. \tag{1.37}$$

This relationship is closely linked to the Maxwell distribution of velocities in a gas, which we shall study in Chapter 2.

In higher dimensions, the sampling of random points inside the unit sphere cannot really be achieved by first sampling points in the cube surrounding the sphere and rejecting all those positions which have a radius in excess of one (see Alg. 1.20 (`naive-sphere`)). Such a modified children's algorithm has a very high rejection rate, as we shall show now before returning to eqn (1.37).

```
     procedure naive-sphere
1    Σ ← 0
     for k = 1, ..., d do
       { x_k ← ran(−1, 1)
       { Σ ← Σ + x_k^2
       { if (Σ > 1) goto 1 (reject sample)
     output {x_1, ..., x_d}
```

Algorithm 1.20 `naive-sphere`. Sampling a uniform random vector inside the d-dimensional unit sphere.

The acceptance rate of Alg. 1.20 (`naive-sphere`) is related to the ratio of volumes of the unit hypersphere and a hypercube of side length 2:

$$\left\{ \begin{array}{c} \text{volume} \\ \text{of unit sphere} \\ \text{in } d \text{ dim.} \end{array} \right\} = \left\{ \begin{array}{c} \text{volume of} \\ d\text{-dim. cube} \\ \text{of length 2} \end{array} \right\} \left\{ \begin{array}{c} \text{acceptance rate of} \\ \text{Alg. 1.20} \end{array} \right\}. \quad (1.38)$$

The volume $V_d(R)$ of a d-dimensional sphere of radius R is

$$V_d(R) = \int_{x_1^2+\cdots+x_d^2 \le R^2} \mathrm{d}x_1 \ \ldots \mathrm{d}x_d \ = \underbrace{\left[\frac{\pi^{d/2}}{\Gamma(d/2+1)} \right]}_{V_d(1)} R^d, \quad (1.39)$$

where $\Gamma(x) = \int_0^\infty \mathrm{d}t \; t^{x-1}\mathrm{e}^{-t}$, the gamma function, generalizes the factorial. This function satisfies $\Gamma(x+1) = x\Gamma(x)$. For an integer argument n, $\Gamma(n) = (n-1)!$. The value $\Gamma(1/2) = \sqrt{\pi}$ allows us to construct the gamma function for all half-integer x.

In this book, we have a mission to derive all equations explicitly, and thus need to prove eqn (1.39). Clearly, the volume of a d-dimensional sphere of radius R is proportional to R^d (the area of a disk goes as the radius squared, the volume of a three-dimensional sphere goes as the cube, etc.), and we only have to determine $V_d(1)$, the volume of the unit sphere. Splitting the d-dimensional vector $\mathbf{R} = \{x_1, \ldots, x_d\}$ as

$$\mathbf{R} = \{\underbrace{x_1, x_2}_{\mathbf{r}}, \underbrace{x_3, , \ldots, , x_d}_{\mathbf{u}}\}, \quad (1.40)$$

where $R^2 = r^2 + u^2$, and writing $\mathbf{r}$ in two-dimensional polar coordinates

$\mathbf{r} = \{r, \phi\}$, we find

$$\underline{V_d(1)} = \int_0^1 r\ \mathrm{d}r \int_0^{2\pi} \mathrm{d}\phi \overbrace{\int_{x_3^2+\cdots+x_d^2 \le 1-r^2} \mathrm{d}x_3\ \ldots \mathrm{d}x_d}^{(d-2)\text{-dim. sphere of radius } \sqrt{1-r^2}} \tag{1.41}$$

$$= 2\pi \int_0^1 \mathrm{d}r\ r V_{d-2}(\sqrt{1-r^2})$$

$$= 2\pi V_{d-2}(1) \int_0^1 \mathrm{d}r\ r \left(\sqrt{1-r^2}\right)^{d-2}$$

$$= \pi V_{d-2}(1) \int_0^1 \mathrm{d}u\ u^{d/2-1} = \underline{\frac{\pi}{d/2} V_{d-2}(1).} \tag{1.42}$$

The relation in eqns (1.41) and (1.42) sets up a recursion for the volume of the d-dimensional unit sphere that starts with the elementary values $V_1(1) = 2$ (line) and $V_2(1) = \pi$ (disk), and immediately leads to eqn (1.39).

It follows from eqn (1.39) that, for large d,

$$\left\{ \begin{array}{c} \text{volume of} \\ \text{unit hypersphere} \end{array} \right\} \ll \left\{ \begin{array}{c} \text{volume of} \\ \text{hypercube of side 2} \end{array} \right\},$$

and this in turn implies (from eqn (1.38)) that the acceptance rate of Alg. 1.20 (`naive-sphere`) is close to zero for $d \gg 1$ (see Table 1.8). The naive algorithm thus cannot be used for sampling points inside the d-dimensional sphere for large d.

Table 1.8 Volume $V_d(1)$ and acceptance rate in d dimensions for Alg. 1.20 (`naive-sphere`)

d	$V_d(1)$	Acceptance rate
2	π	$\pi/4 = 0.785$
4	4.9348	0.308
10	2.55016	0.002
20	0.026	2.46×10^{-8}
60	3.1×10^{-18}	2.7×10^{-36}

Going back to eqn (1.37), we now consider d independent Gaussian random numbers $\{x_1, \ldots, x_d\}$. Raising the error integral in eqn (1.33) to the dth power, we obtain

$$1 = \int \ldots \int \mathrm{d}x_1 \ldots \mathrm{d}x_d \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{1}{2}(x_1^2 + \cdots + x_d^2)\right], \tag{1.43}$$

an integral which can be sampled by d independent Gaussians in the same way that the integral in eqn (1.34) is sampled by two of them.

The integrand in eqn (1.43) depends only on $r^2 = (x_1^2 + \cdots + x_d^2)$, i.e. on the square of the radius of the d-dimensional sphere. It is therefore appropriate to write it in polar coordinates with the $d-1$ angular variables; the solid angle Ω is taken all over the unit sphere, the radial variable is r, and the volume element is

$$\mathrm{d}V\ = \mathrm{d}x_1 \ldots \mathrm{d}x_d = r^{d-1}\, \mathrm{d}r\, \mathrm{d}\Omega.$$

Equation (1.43), the integral sampled by d Gaussians, becomes

$$1 = \left(\frac{1}{\sqrt{2\pi}}\right)^d \int_0^\infty \mathrm{d}r\ r^{d-1} \exp\left(-r^2/2\right) \int \mathrm{d}\Omega. \tag{1.44}$$

This is almost what we want, because our goal is to sample random points in the unit sphere, that is, to sample, in polar coordinates, the integral

$$V_d(1) = \int_0^1 \mathrm{d}r\ r^{d-1} \int \mathrm{d}\Omega \quad \text{(unit sphere)}. \tag{1.45}$$

```
procedure direct-sphere
Σ ← 0
for k = 1, ..., d do
  { x_k ← gauss(σ)
  { Σ ← Σ + x_k^2
ϒ ← ran(0,1)^{1/d}
for k = 1, ..., d do
  { x_k ← ϒ x_k / √Σ
output {x_1, ..., x_d}
```

Algorithm 1.21 `direct-sphere`. Uniform random vector inside the d-dimensional unit sphere. The output is independent of σ.

The angular parts of the integrals in eqns (1.44) and (1.45) are identical, and this means that the d Gaussians sample angles isotropically in d dimensions, and only get the radius wrong. This radius should be sampled from a distribution $\pi(r) \propto r^{d-1}$. From eqn (1.29), we obtain the direct distribution of r by taking the dth root of a random number $\texttt{ran}\,(0,1)$. A simple rescaling thus yields the essentially rejection-free Alg. 1.21 (`direct-sphere`), one of the crown jewels of the principality of Monte Carlo.

```
procedure direct-surface
σ ← 1/√d
Σ ← 0
for k = 1, ..., d do
  { x_k ← gauss(σ)
  { Σ ← Σ + x_k^2
for k = 1, ..., d do
  { x_k ← x_k / √Σ
output {x_1, ..., x_d}
```

Algorithm 1.22 `direct-surface`. Random vector on the surface of the d-dimensional unit sphere. For large d, Σ approaches one (see Fig. 1.32).

The above program illustrates the profound relationship between sampling and integration, and we can take the argument leading to Alg. 1.21 (`direct-sphere`) a little further to rederive the volume of the unit sphere. In eqns (1.45) and (1.44), the multiple integrals over the angular variables $\mathrm{d}\Omega$ are the same, and we may divide them out:

$$V_d(1) = \frac{\text{eqn (1.45)}}{\text{eqn (1.44)}} = \frac{\left(\sqrt{2\pi}\right)^d \int_0^1 \mathrm{d}r\; r^{d-1}}{\int_0^\infty \mathrm{d}r\; r^{d-1} \exp\left(-r^2/2\right)} = \frac{\pi^{d/2}}{(d/2)\Gamma(d/2)}. \quad (1.46)$$

This agrees with the earlier expression in eqn (1.39). The derivation in eqn (1.46) of the volume of the unit sphere in d dimensions is lightning-

The integral in the denominator of eqn (1.46), where $r^2/2 = u$ and $r\,\mathrm{d}r = \mathrm{d}u$, becomes

$$2^{d/2-1} \underbrace{\int_0^\infty \mathrm{d}u\; u^{d/2-1} \mathrm{e}^{-u}}_{\Gamma(d/2)}.$$

fast, a little abstract, and as elegant as Alg. 1.21 (`direct-sphere`) itself. After sampling the Gaussians $\{x_1, \ldots, x_d\}$, we may also rescale the vector to unit length. This amounts to sampling a random point on the surface of the d-dimensional unit sphere, not inside its volume (see Alg. 1.22 (`direct-surface`)).

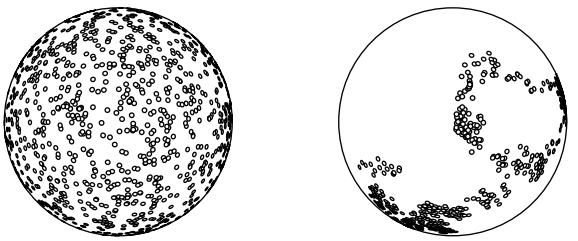

Fig. 1.32 Random pebbles on the surface of a sphere (from Alg. 1.22 (direct sampling, *left*) and Alg. 1.24 (Markov-chain sampling, *right*)).

The sampling algorithm for pebbles on the surface of a d-dimensional sphere can be translated into many computer languages similarly to the way Alg. 1.22 (`direct-surface`) is written. However, vector notation is more concise, not only in formulas but also in programs. This means that we collect the d components of a vector into a single symbol (as already used before)

$$\mathbf{x} = \{x_1, \ldots, x_d\}.$$

The Cartesian scalar product of two vectors is

$$(\mathbf{x} \cdot \mathbf{x}') = x_1 x_1' + \cdots + x_d x_d',$$

and the square root of the scalar product of a vector with itself gives the vector's norm,

$$|\mathbf{x}| = \sqrt{(\mathbf{x} \cdot \mathbf{x})} = \sqrt{x_1^2 + \cdots + x_d^2}.$$

procedure `direct-surface(vector notation)`
$\sigma \leftarrow 1/\sqrt{d}$
$\mathbf{x} \leftarrow \{\texttt{gauss}\,(\sigma)\,, \ldots, \texttt{gauss}\,(\sigma)\}$
$\mathbf{x} \leftarrow \mathbf{x}/|\mathbf{x}|$
output $\mathbf{x}$

Algorithm 1.23 `direct-surface(vector notation)`. Same program as Alg. 1.22, in vector notation, with a d-dimensional vector $\mathbf{x}$.

The difference between Alg. 1.22 and Alg. 1.23 is purely notational. The actual implementation in a computer language will depends on how vectors can be addressed. In addition, it depends on the random number generator whether a line such as $\mathbf{x} \leftarrow \{\texttt{gauss}\,(\sigma)\,, \ldots, \texttt{gauss}\,(\sigma)\}$ can

be implemented as a vector instruction or must be broken up into a sequential loop because of possible conflicts with the seed-and-update mechanism of the random number generator.

For later use, we also present a Markov-chain algorithm on the surface of a d-dimensional unit sphere, written directly in vector notation (see Alg. 1.24 (`markov-surface`)). The algorithm constructs a random vector $\boldsymbol{\epsilon}$ from Gaussians, orthogonalizes it, and normalizes it with respect to the input vector $\mathbf{x}$, the current position of the Markov chain. The step taken is in the direction of this reworked $\boldsymbol{\epsilon}$, a random unit vector in the hyperplane orthogonal to $\mathbf{x}$.

procedure `markov-surface`
input $\mathbf{x}$ (unit vector $|\mathbf{x}| = 1$)
$\boldsymbol{\epsilon} \leftarrow \{\texttt{gauss}\,(\sigma), \ldots, \texttt{gauss}\,(\sigma)\}$ (d independent Gaussians)
$\Sigma \leftarrow (\boldsymbol{\epsilon} \cdot \mathbf{x})$
$\boldsymbol{\epsilon} \leftarrow \boldsymbol{\epsilon} - \Sigma\mathbf{x}$
$\boldsymbol{\epsilon} \leftarrow \boldsymbol{\epsilon}/|\boldsymbol{\epsilon}|$
$\Upsilon \leftarrow \texttt{ran}\,(-\delta, \delta)$ (δ: step size)
$\mathbf{x} \leftarrow \mathbf{x} + \Upsilon\boldsymbol{\epsilon}$
$\mathbf{x} \leftarrow \mathbf{x}/|\mathbf{x}|$
output $\mathbf{x}$

Algorithm 1.24 `markov-surface`. Markov-chain Monte Carlo algorithm for random vectors on the surface of a d-dimensional unit sphere.

1.3 Statistical data analysis

In the first sections of this book, we familiarized ourselves with sampling as an ingenious method for evaluating integrals which are unsolvable by other methods. However, we stayed on the sunny side of the subject, avoiding the dark areas: statistical errors and the difficulties related to their evaluation. In the present section, we concentrate on the description and estimation of errors, both for independent variables and for Markov chains. We first discuss the fundamental mechanism which connects the probability distribution of a single random variable to that of a sum of independent variables, then discuss the moments of distributions, especially the variance, and finally review the basic facts of probability theory—Chebyshev's inequality, the law of large numbers, and the central limit theorem—that are relevant to statistical physics and Monte Carlo calculations. We then study the problem of estimating mean values for independent random variables (direct sampling) and for those coming from Markov-chain simulations.

1.3.1 Sum of random variables, convolution

In this subsection, we discuss random variables and sums of random variables in more general terms than before. The simplest example of a

random variable, which we shall call ξ_i, is the outcome of the ith trial in the children's game. This is called a Bernoulli variable. It takes the value one with probability θ and the value zero with probability $(1-\theta)$. ξ_i can also stand for the ith call to a random number generator $\texttt{ran}(0,1)$ or $\texttt{gauss}(\sigma)$, etc., or more complicated quantities. We note that in our algorithms, the index i of the random variable ξ_i is hidden in the seed-and-update mechanism of the random number generator.

The number of hits in the children's game is itself a random variable, and we denote it by a symbol similar to the others, ξ:

$$\xi = \xi_1 + \cdots + \xi_N.$$

ξ_1 takes values k_1, ξ_2 takes values k_2, etc., and the probability of obtaining $\{k_1, \ldots, k_N\}$ is, for independent variables,

$$\pi(\{k_1, \ldots, k_N\}) = \pi(k_1) \cdot \cdots \cdot \pi(k_N).$$

The sum random variable ξ takes the values $\{0, \ldots, N\}$ with probabilities $\{\pi_0, \ldots, \pi_N\}$, which we shall now calculate. Clearly, only sets $\{k_1, \ldots, k_N\}$ that produce k hits contribute to π_k:

$$\underbrace{\pi_k}_{\substack{N \text{ trials} \\ k \text{ hits}}} = \sum_{\substack{k_1=0,1,\ldots,k_N=0,1 \\ k_1+\cdots+k_N=k}} \overbrace{\pi(k_1)\pi(k_2)\cdots\pi(k_N)}^{\pi(k_1,\ldots,k_N)}.$$

The k hits and $N-k$ nonhits among the values $\{k_1, \ldots, k_N\}$ occur with probabilities θ and $(1-\theta)$, respectively. In addition, we have to take into account the number of ways of distributing k hits amongst N trials by multiplying by the combinatorial factor $\binom{N}{k}$. The random variable ξ, the number of hits, thus takes values k with a probability distributed according to the binomial distribution

$$\pi_k = \binom{N}{k} \theta^k (1-\theta)^{N-k} \quad (0 \le k \le N). \tag{1.47}$$

In this equation, the binomial coefficients are given by

$$\binom{N}{k} = \frac{N!}{k!(N-k)!} = \frac{(N-k+1)(N-k+2)\cdots N}{1 \times 2 \times \cdots \times k}. \tag{1.48}$$

We note that Alg. 1.1 (`direct-pi`)—if we look only at the number of hits—merely samples this binomial distribution for $\theta = \pi/4$.

The explicit formula for the binomial distribution (eqn (1.47)) is inconvenient for practical evaluation because of the large numbers entering the coefficients in eqn (1.48). It is better to relate the probability distribution π' for $N+1$ trials to that for N trials: the $N+1$th trial is independent of what happened before, so that the probability π'_k for k hits with $N+1$ trials can be put together from the independent probabilities π_{k-1} for $k-1$ hits and π_k for k hits with N trials, and the Bernoulli distribution for a single trial:

$$\underbrace{\pi'_k}_{\substack{N+1 \text{ trials} \\ k \text{ hits}}} = \underbrace{\pi_k}_{\substack{N \text{ trials} \\ k \text{ hits}}} \cdot \underbrace{(1-\theta)}_{\text{no hit}} + \underbrace{\pi_{k-1}}_{\substack{N \text{ trials} \\ k-1 \text{ hits}}} \cdot \underbrace{\theta}_{\text{hit}}, \tag{1.49}$$

(see Alg. 1.25 (`binomial-convolution`)).

procedure `binomial-convolution`
input $\{\pi_0, \ldots, \pi_N\}$ (N trials)
$\pi'_0 \leftarrow (1-\theta)\pi_0$ (θ: probability of hit)
for $k = 1, \ldots, N$ **do**
$\{\ \pi'_k \leftarrow \theta\pi_{k-1} + (1-\theta)\pi_k$
$\pi'_{N+1} \leftarrow \theta\pi_N$
output $\{\pi'_0, \ldots, \pi'_{N+1}\}$ ($N+1$ trials)

Algorithm 1.25 `binomial-convolution`. Probabilities of numbers of hits for $N+1$ trials obtained from those for N trials.

Table 1.9 Probabilities $\{\pi_0, \ldots, \pi_N\}$ in Alg. 1.25 (`binomial-convolution`) for small values of N, with $\theta = \pi/4$

N	π_0	π_1	π_2	π_3
1	0.215	0.785	.	.
2	0.046	0.337	0.617	.
3	0.010	0.109	0.397	0.484

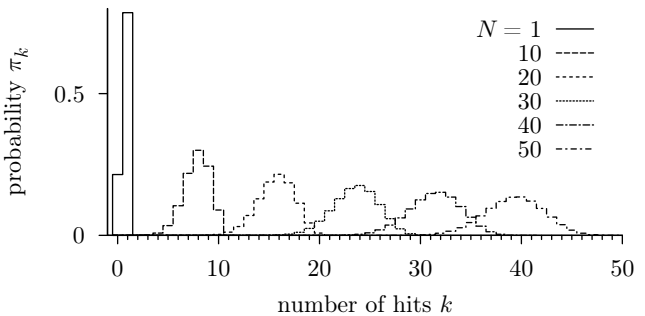

Fig. 1.33 Probabilities for the number of hits in the children's game with N trials (from Alg. 1.25 (`binomial-convolution`), using $\theta = \pi/4$).

In evaluating the binomial distribution using eqn (1.49), we implicitly piece together the binomial coefficients in eqn (1.47) through the relations

$$\binom{N}{0} = \binom{N}{N} = 1, \quad \binom{N}{k-1} + \binom{N}{k} = \binom{N+1}{k} \ (1 \le k \le N).$$

Replacing θ and $(1-\theta)$ by 1 in Alg. 1.25 (`binomial-convolution`) generates Pascal's triangle of binomial coefficients.

Algorithm 1.25 (`binomial-convolution`) implements a convolution[5] of the distribution function for N trials with the Bernoulli distribution of one single trial, and gives the binomial distribution (see Fig. 1.33 and Table 1.9).

Convolutions are a powerful tool, as we show now for the case where ξ_i is uniformly distributed in the interval $[0, 1]$. As before, we are interested in the distribution $\pi(x)$ of the sum variable

$$\xi = \xi_1 + \cdots + \xi_N,$$

[5]From the Latin *convolvere*: to roll together, fold together, or intertwine.

which takes values x in the interval $[0, N]$. Again thinking recursively, we can obtain the probability distribution $\pi'(x)$ for the sum of $N+1$ random numbers by convoluting $\pi(y)$, the distribution for N variables, with the uniform distribution $\pi^1(y-x)$ in the interval $[0,1]$ for a single variable ($\pi^1(x) = 1$ for $0 < x < 1$). The arguments y and $x-y$ have been chosen so that their sum is equal to x. To complete the convolution, the product of the probabilities must be integrated over all possible values of y:

$$\underbrace{\pi'(x)}_{\substack{x\text{: sum} \\ \text{of } N+1}} = \int_{x-1}^{x} \mathrm{d}y \ \underbrace{\pi(y)}_{\substack{y\text{: sum} \\ \text{of } N}} \underbrace{\pi^1(x-y)}_{\substack{x-y\text{:} \\ \text{one term}}}. \tag{1.50}$$

Here, the integration generalizes the sum over two terms used in the binomial distribution. The integration limits implement the condition that $\pi^1(x-y)$ is different from zero only for $0 < x-y < 1$. For numerical calculation, the convolution in eqn (1.50) is most easily discretized by cutting the interval $[0,1]$ into l equal segments $\{x_0 = 0, x_1, \ldots, x_{l-1}, x_l = 1\}$ so that $x_k \equiv k/l$. In addition, weights $\{\pi_0, \ldots, \pi_l\}$ are assigned as shown in Alg. 1.26 (`ran01-convolution`). The program allows us to compute the probability distribution for the sum of N random numbers, as sampled by Alg. 1.17 (`naive-gauss`) (see Fig. 1.34). The distribution of the sum of N uniform random numbers is also analytically known, but its expression is very cumbersome for large N.

procedure `ran01-convolution`
$\{\pi^1_0, \ldots, \pi^1_l\} \leftarrow \{\frac{1}{2l}, \frac{1}{l}, \ldots, \frac{1}{l}, \frac{1}{2l}\}$
input $\{\pi_0, \ldots, \pi_{Nl}\}$ (probabilities for sum of N variables)
for $k = 0, \ldots, Nl + l$ **do**
$\left\{ \ \pi'_k \leftarrow \sum_{m=\max(0,k-Nl)}^{\min(l,k)} \pi_{k-m}\pi^1_m \right.$
output $\{\pi'_0, \ldots, \pi'_{Nl+l}\}$ (probabilities for $N+1$ variables)

Algorithm 1.26 `ran01-convolution`. Probabilities for the sum of $N+1$ random numbers obtained from the probabilities for N numbers.

Equations (1.49) and (1.50) are special cases of general convolutions for the sum $\xi + \eta$ of two independent random variables ξ and η, taking values x and y with probabilities $\pi_\xi(x)$ and $\pi_\eta(y)$, respectively. The sum variable $\xi + \eta$ takes values x with probability

$$\pi_{\xi+\eta}(x) = \int_{-\infty}^{\infty} \mathrm{d}y \ \pi_\xi(y)\pi_\eta(x-y).$$

Again, the arguments of the two independent variables, y and $x-y$, have been chosen such that their sum is equal to x. The value of y is arbitrary, and must be integrated (summed) over. In Subsection 1.4.4, we shall revisit convolutions, and generalize Alg. 1.26 (`ran01-convolution`) to distributions which stretch out to infinity in such a way that $\pi(x)$ cannot be cut off at large arguments x.

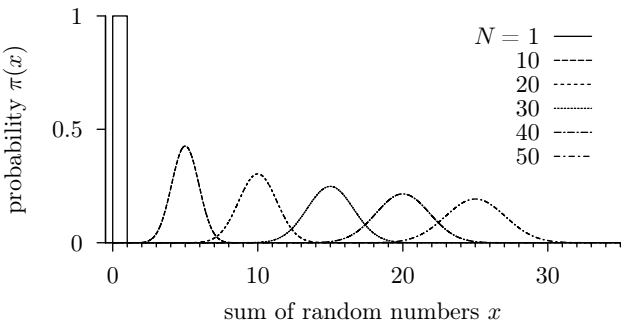

Fig. 1.34 Probability distribution for the sum of N random numbers `ran` $(0, 1)$ (from Alg. 1.26 (`ran01-convolution`), with $l = 100$).

1.3.2 Mean value and variance

In this subsection, we discuss the moments of probability distributions, and the connection between the moments of a single variable and those of a sum of variables. The variance—the second moment—stands out: it is additive and sets a scale for an interval containing a large fraction of the events (through the Chebyshev inequality). These two properties will be discussed in detail. The variance also gives a necessary and sufficient condition for the convergence of a sum of identically distributed random variables to a Gaussian, as we shall discuss in Subsection 1.3.3.

The mean value (also called the expectation value) of a distribution for a discrete variable ξ is given by

$$\langle \xi \rangle = \sum_k k \pi_k,$$

where the sum runs over all possible values of k. For a continuous variable, the mean value is given, analogously, by

$$\langle \xi \rangle = \int \mathrm{d}x \; x \pi(x). \tag{1.51}$$

The mean value of the Bernoulli distribution is θ, and the mean of the random number $\xi =$ `ran` $(0, 1)$ is obviously $1/2$.

The mean value of a sum of N random variables is equal to the sum of the means of the variables. In fact, for two variables ξ_1 and ξ_2, and for their joint probability distribution $\pi(x_1, x_2)$, we may still define the probabilities of x_1 and x_2 by integrating over the other variable:

$$\pi_1(x_1) = \int \mathrm{d}x_2 \; \pi(x_1, x_2), \quad \pi_2(x_2) = \int \mathrm{d}x_1 \; \pi(x_1, x_2),$$

with $\langle \xi_1 \rangle = \int \mathrm{d}x_1 \; x_1 \pi_1(x_1)$, etc. This gives

$$\underline{\langle \xi_1 + \xi_2 \rangle} = \int \mathrm{d}x_1 \; \mathrm{d}x_2 \; (x_1 + x_2) \pi(x_1, x_2) = \underline{\langle \xi_1 \rangle + \langle \xi_2 \rangle}.$$

The additivity of the mean value holds for variables that are not independent (i.e. do not satisfy $\pi(x_1, x_2) = \pi_1(x_1)\pi_2(x_2)$). This was naively assumed in the heliport game: we strove to make the stationary distribution function $\pi(x_1, x_2)$ uniform, so that the probability of falling into the circle was equal to θ. It was natural to assume that the mean value of the sum of N trials would be the same as N times the mean value of a single trial.

Among the higher moments of a probability distribution, the variance

$$\mathrm{Var}\,(\xi) = \left\{ \begin{array}{c} \text{average squared distance} \\ \text{from the mean value} \end{array} \right\} = \left\langle (\xi - \langle \xi \rangle)^2 \right\rangle \tag{1.54}$$

is quintessential. It can be determined, as indicated in eqn (1.54), from the squared deviations from the mean value. For the Bernoulli distribution, the mean value is θ, so that

$$\mathrm{Var}\,(\xi) = \underbrace{\theta^2}_{(0-\langle\xi\rangle)^2} \underbrace{(1-\theta)}_{\pi(0)} + \underbrace{(1-\theta)^2}_{(1-\langle\xi\rangle)^2} \underbrace{\theta}_{\pi(1)} = \theta(1-\theta).$$

It is usually simpler to compute the variance from another formula, obtained from eqn (1.54) by writing out the square:

$$\underline{\mathrm{Var}\,(\xi)} = \left\langle (\xi - \langle \xi \rangle)^2 \right\rangle = \left\langle \xi^2 \right\rangle - 2 \underbrace{\langle \xi \cdot \langle \xi \rangle \rangle}_{\langle \xi \rangle \langle \xi \rangle} + \langle \xi \rangle^2 = \underline{\left\langle \xi^2 \right\rangle - \langle \xi \rangle^2}.$$

This gives the following for the Bernoulli distribution:

$$\mathrm{Var}\,(\xi) = \underbrace{\left\langle \xi^2 \right\rangle}_{0^2 \cdot (1-\theta) + 1^2 \cdot \theta} - \underbrace{\langle \xi \rangle^2}_{\theta^2} = \theta(1-\theta).$$

The variance of the uniform random number $\xi = \mathtt{ran}\,(0, 1)$ is

$$\mathrm{Var}\,(\xi) = \int_0^1 \mathrm{d}x \ \underbrace{\pi(x)}_{=1} x^2 - \left[\int_0^1 \mathrm{d}x \ \underbrace{\pi(x)}_{=1} x \right]^2 = \frac{1}{12}, \tag{1.55}$$

which explains the factor 1/12 in Alg. 1.17 (`naive-gauss`). The variance of a Gaussian random number $\mathtt{gauss}(\sigma)$ is

$$\int_{-\infty}^{\infty} \frac{\mathrm{d}x}{\sqrt{2\pi}\sigma} x^2 \exp\left(-\frac{x^2}{2\sigma^2} \right) = \sigma^2.$$

The variance (the mean square deviation) has the dimensions of a squared length, and it is useful to consider its square root, the root mean square deviation (also called the standard deviation):

$$\left\{ \begin{array}{c} \text{root mean square} \\ \text{(standard) deviation} \end{array} \right\} : \sqrt{\langle (\xi - \langle \xi \rangle)^2 \rangle} = \sqrt{\mathrm{Var}\,(\xi)} = \sigma.$$

This should be compared with the mean absolute deviation $\langle |\xi - \langle \xi \rangle| \rangle$, which measures how far on average a random variable is away from

The fundamental relations

$$\langle a\xi + b \rangle = a \langle \xi \rangle + b, \tag{1.52}$$

$$\mathrm{Var}\,(a\xi + b) = a^2 \mathrm{Var}\,(\xi), \tag{1.53}$$

are direct consequences of the definitions in eqns (1.51) and (1.54).

its mean value. For the Gaussian distribution, for example, the mean absolute deviation is

$$\left\{\begin{matrix}\text{mean absolute}\\ \text{deviation}\end{matrix}\right\}: \quad \int_{-\infty}^{\infty} \frac{\mathrm{d}x}{\sqrt{2\pi}\sigma}|x| \exp\left(-\frac{x^2}{2\sigma^2}\right) = \underbrace{\sqrt{\frac{2}{\pi}}}_{0.798}\sigma, \qquad (1.56)$$

which is clearly different from the standard deviation.

The numerical difference between the root mean square deviation and the mean absolute deviation is good to keep in mind, but represents little more than a subsidiary detail, which depends on the distribution. The great importance of the variance and the reason why absolute deviations play no role reside in the way they can be generalized from a single variable to a sum of variables. To see this, we must first consider the correlation function of two independent variables ξ_i and ξ_j, taking values x_i and x_j (where $i \neq j$):

$$\int \mathrm{d}x_i \int \mathrm{d}x_j \ \pi(x_i)\pi(x_j)x_i x_j = \left[\int \mathrm{d}x_i \ \pi(x_i)x_i\right]\left[\int \mathrm{d}x_j \ \pi(x_j)x_j\right],$$

which is better written—and remembered—as

$$\langle \xi_i \xi_j \rangle = \begin{cases} \langle \xi_i \rangle \langle \xi_j \rangle & \text{for } i \neq j \\ \langle \xi_i^2 \rangle & \text{for } i = j \end{cases} \quad \left\{\begin{matrix}\text{independent}\\ \text{variables}\end{matrix}\right\}.$$

This factorization of correlation functions implies that the variance of a sum of independent random variables is additive. In view of eqn (1.53), it suffices to show this for random variables of zero mean, where we find

$$\begin{aligned}\underline{\mathrm{Var}\,(\xi_1 + \cdots + \xi_N)} &= \langle (\xi_1 + \cdots + \xi_N)^2 \rangle = \left\langle \left(\sum_i \xi_i\right)\left(\sum_j \xi_j\right)\right\rangle \\ &= \sum_i^N \langle \xi_i^2 \rangle + \sum_{i \neq j} \underbrace{\langle \xi_i \rangle \langle \xi_j \rangle}_{=0} = \underline{\mathrm{Var}\,(\xi_1) + \cdots + \mathrm{Var}\,(\xi_N)}. \qquad (1.57)\end{aligned}$$

The additivity of variances for the sum of independent random variables is of great importance. No analogous relation exists for the mean absolute deviation.

Independent random variables ξ_i with the same finite variance satisfy, from eqns (1.52) and (1.53):

$$\begin{aligned}\mathrm{Var}\,(\xi_1 + \cdots + \xi_N) &= N\mathrm{Var}\,(\xi_i)\,, \\ \mathrm{Var}\left(\frac{\xi_1 + \cdots + \xi_N}{N}\right) &= \frac{1}{N}\mathrm{Var}\,(\xi_i)\,.\end{aligned}$$

For concreteness, we shall now apply these two formulas to the children's game, with $N_{\mathrm{hits}} = \xi_1 + \cdots + \xi_N$:

$$\mathrm{Var}\,(N_{\mathrm{hits}}) = \left\langle \left(N_{\mathrm{hits}} - \frac{\pi}{4}N\right)^2 \right\rangle = N\mathrm{Var}\,(\xi_i) = N\overbrace{\theta(1-\theta)}^{0.169},$$

$$\mathrm{Var}\left(\frac{N_{\mathrm{hits}}}{N}\right) = \left\langle \left(\frac{N_{\mathrm{hits}}}{N} - \frac{\pi}{4}\right)^2 \right\rangle = \frac{1}{N}\mathrm{Var}\left(\xi_i\right) = \frac{\overbrace{\theta(1-\theta)}^{0.169}}{N}. \quad (1.58)$$

In the initial simulation in this book, the number of hits was usually of the order of 20 counts away from 3141 (for $N = 4000$ trials), because the variance is $\mathrm{Var}\left(N_{\mathrm{hits}}\right) = N \cdot 0.169 = 674.2$, so that the mean square deviation comes out as $\sqrt{674} = 26$. This quantity corresponds to the square root of the average of the last column in Table 1.10 (which analyzes the first table in this chapter). The mean absolute distance $\langle|\Delta N_{\mathrm{hits}}|\rangle$, equal to 20, is smaller than the root mean square difference by a factor $\sqrt{2/\pi}$, as in eqn (1.56), because the binomial distribution is virtually Gaussian when N is large and θ is of order unity.

Table 1.10 Reanalysis of Table 1.1 using eqn (1.58) ($N = 4000, \theta = \pi/4$)

#	N_{hits}	$\lvert\Delta N_{\mathrm{hits}}\rvert$	$(\Delta N_{\mathrm{hits}})^2$
1	3156	14.4	207.6
2	3150	8.4	70.7
3	3127	14.6	212.9
4	3171	29.4	864.8
5	3148	6.4	41.1
...	...	...	
$\langle\rangle$	3141.	20.7	674.2

The variance not only governs average properties, such as the mean square deviation of a random variable, but also allows us to perform interval estimates. Jacob Bernoulli's weak law of large numbers is of this type (for sums of Bernoulli-distributed variables, as in the children's game). It states that for any interval $[\pi/4-\epsilon, \pi/4+\epsilon]$, the probability for N_{hits}/N to fall within this interval goes to one as $N \to \infty$. This law is best discussed in the general context of the Chebyshev inequality, which we need to understand only for distributions with zero mean:

$$\mathrm{Var}\left(\xi\right) = \int_{-\infty}^{\infty} \mathrm{d}x\ x^2\pi(x) \geq \int_{|x|>\epsilon} \mathrm{d}x\ x^2\pi(x) \geq \epsilon^2 \underbrace{\int_{|x|>\epsilon} \mathrm{d}x\ \pi(x)}_{\substack{\text{prob. that} \\ |x-\langle x\rangle|>\epsilon}}.$$

This gives

$$\left\{\begin{matrix}\text{Chebyshev} \\ \text{inequality}\end{matrix}\right\}: \quad \left\{\begin{matrix}\text{probability that} \\ |x - \langle x\rangle\,| > \epsilon\end{matrix}\right\} < \frac{\mathrm{Var}\left(\xi\right)}{\epsilon^2}. \quad (1.59)$$

In the children's game, the variance of the number of hits, $\mathrm{Var}\left(N_{\mathrm{hits}}/N\right)$, is smaller than $1/(4N)$ because, for $\theta \in [0,1]$, $\theta(1-\theta) \leq 1/4$. This allows us to write

$$\left\{\begin{matrix}\text{weak law of} \\ \text{large numbers}\end{matrix}\right\}: \quad \left\{\begin{matrix}\text{probability that} \\ |N_{\mathrm{hits}}/N - \pi/4| < \epsilon\end{matrix}\right\} > 1 - \frac{1}{4\epsilon^2 N}.$$

In this equation, we can keep the interval parameter ϵ fixed. The probability inside the interval approaches 1 as $N \to \infty$. We can also bound the interval containing, say, 99% of the probability, as a function of N. We thus enter 0.99 into the above inequality, and find

$$\left\{\begin{matrix}\text{size of interval containing} \\ 99\% \text{ of probability}\end{matrix}\right\}: \quad \epsilon < \frac{5}{\sqrt{N}}.$$

Chebyshev's inequality (1.59) shows that a (finite) variance plays the role of a scale delimiting an interval of probable values of x: whatever the distribution, it is improbable that a sample will be more than a few standard deviations away from the mean value. This basic regularity

property of distributions with a finite variance must be kept in mind in practical calculations. In particular, we must keep this property separate from the central limit theorem, which involves an $N \to \infty$ limit. Applied to the sum of independent variables, Chebyshev's inequality turns into the weak law of large numbers and is the simplest way to understand why N-sample averages must converge (in probability) towards the mean value, i.e. why the width of the interval containing any fixed amount of probability goes to zero as $\propto 1/\sqrt{N}$.

1.3.3 The central limit theorem

We have discussed the crucial role played by the mean value of a distribution and by its variance. If the variance is finite, we can shift any random variable by the mean value, and then rescale it by the standard deviation, the square root of the variance, such that it has zero mean and unity variance. We suppose a random variable ξ taking values y with probability $\pi(y)$. The rescaling is done so that ξ_{resc} takes values $x = (y - \langle \xi \rangle)/\sigma$ with probability

$$\pi_{\text{resc}}(x) = \sigma \pi(\underbrace{\sigma x + \langle \xi \rangle}_{y}) \tag{1.60}$$

and has zero mean and unit variance. As an example, we have plotted in Fig. 1.35 the probability distribution of a random variable ξ corresponding to the sum of 50 random numbers $\texttt{ran}\,(0,1)$. This distribution is characterized by a standard deviation $\sigma = \sqrt{50} \times \sqrt{1/12} = 2.04$, and mean $\langle \xi \rangle = 25$. As an example, $y = 25$, with $\pi(y) = 0.193$, is rescaled to $x = 0$, with $\pi_{\text{resc}}(0) = 2.04 \times 0.193 = 0.39$ (see Fig. 1.35 for the rescaled distributions for the children's game and for the sum of N random numbers $\texttt{ran}\,(0,1)$).

The central limit theorem[6] states that this rescaled random variable is Gaussian in the limit $N \to \infty$ if ξ itself is a sum of independent random variables $\xi = \xi_1 + \cdots + \xi_N$ of finite variance.

In this subsection, we prove this fundamental theorem of probability theory under the simplifying assumption that all moments of the distribution are finite:

$$\begin{array}{ccccccc} \langle \xi_i \rangle & & \langle \xi_i^2 \rangle & & \langle \xi_i^3 \rangle & \cdots & \langle \xi_i^k \rangle \\ \shortparallel & & \shortparallel & & \shortparallel & \cdots & \shortparallel \\ 0 & & 1 & & \text{all moments finite} & & \end{array} . \tag{1.61}$$

(Odd moments may differ from zero for asymmetric distributions.) For independent random variables with identical distributions, the finite variance by itself constitutes a necessary and sufficient condition for convergence towards a Gaussian (see Gnedenko and Kolmogorov (1954)). The finiteness of the moments higher than the second renders the proof

[6] To be pronounced as *central* limit *theorem* rather than *central limit* theorem. The expression was coined by mathematician G. Polya in 1920 in a paper written in German, where it is clear that the theorem is central, not the limit.

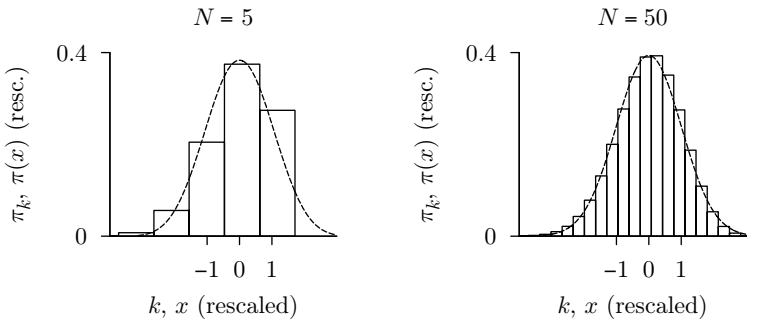

Fig. 1.35 Distribution functions corresponding to Figs 1.33 and 1.34, rescaled as in eqn (1.60).

trivial: in this case, two distributions are identical if all their moments agree. We thus need only compare the moments of the sum random variable, rescaled as

$$\xi = \frac{1}{\sqrt{N}}(\xi_1 + \cdots + \xi_N), \tag{1.62}$$

with the moments of the Gaussian to verify that they are the same. Let us start with the third moment:

$$\left\langle \left(\frac{\xi_1 + \cdots + \xi_N}{\sqrt{N}} \right)^3 \right\rangle = \frac{1}{N^{3/2}} \sum_{ijk=1}^{N} \langle \xi_i \xi_j \xi_k \rangle$$
$$= \frac{1}{N^{3/2}} (\underbrace{\langle \xi_1 \xi_1 \xi_1 \rangle}_{\langle \xi_1^3 \rangle} + \underbrace{\langle \xi_1 \xi_1 \xi_2 \rangle}_{\langle \xi_1^2 \rangle \langle \xi_2 \rangle = 0} + \cdots + \underbrace{\langle \xi_2 \xi_2 \xi_2 \rangle}_{\langle \xi_2^3 \rangle} + \cdots).$$

In this sum, only N terms are nonzero, and they remain finite because of our simplifying assumption in eqn (1.61). We must divide by $N^{3/2}$; therefore $\langle \xi^3 \rangle \to 0$ for $N \to \infty$. In the same manner, we scrutinize the fourth moment:

$$\left\langle \left(\frac{\xi_1 + \cdots + \xi_N}{\sqrt{N}} \right)^4 \right\rangle = \frac{1}{N^2} \sum_{ijkl=1}^{N} \langle \xi_i \xi_j \xi_k \xi_l \rangle$$
$$= \frac{1}{N^2} [\underbrace{\langle \xi_1 \xi_1 \xi_1 \xi_1 \rangle}_{\langle \xi_1^4 \rangle} + \underbrace{\langle \xi_1 \xi_1 \xi_1 \xi_2 \rangle}_{\langle \xi_1^3 \rangle \langle \xi_2 \rangle = 0} + \cdots + \underbrace{\langle \xi_1 \xi_1 \xi_2 \xi_2 \rangle}_{\langle \xi_1^2 \rangle \langle \xi_2^2 \rangle} + \cdots]. \tag{1.63}$$

Only terms not containing a solitary index i can be different from zero, because $\langle \xi_i \rangle = 0$: for $N = 4$, there are 40 such terms (see Table 1.11). Of these, 36 are of type "$iijj$" (and permutations) and four are of type "$iiii$". For general N, the total correlation function is

$$\langle \xi^4 \rangle = \frac{1}{N^2} \left[3N(N-1) \langle \xi_i^2 \rangle^2 + N \langle \xi_i^4 \rangle \right].$$

Table 1.11 The 40 choices of indices $\{i, j, k, l\}$ for $N = 4$ (see eqn (1.63)) for which $\langle \xi_i \xi_j \xi_k \xi_l \rangle$ is different from zero

#	i	j	k	l
1	1	1	1	1
2	1	1	2	2
3	1	1	3	3
4	1	1	4	4
5	1	2	1	2
6	1	2	2	1
7	1	3	1	3
8	1	3	3	1
9	1	4	1	4
10	1	4	4	1
11	2	1	1	2
12	2	1	2	1
13	2	2	1	1
14	2	2	2	2
15	2	2	3	3
16	2	2	4	4
17	2	3	2	3
18	2	3	3	2
19	2	4	2	4
20	2	4	4	2
21	3	1	1	3
22	3	1	3	1
23	3	2	2	3
24	3	2	3	2
25	3	3	1	1
26	3	3	2	2
27	3	3	3	3
28	3	3	4	4
29	3	4	3	4
30	3	4	4	3
31	4	1	1	4
32	4	1	4	1
33	4	2	2	4
34	4	2	4	2
35	4	3	3	4
36	4	3	4	3
37	4	4	1	1
38	4	4	2	2
39	4	4	3	3
40	4	4	4	4

In the limit of large N, where $N \simeq N-1$, we have $\langle \xi^4 \rangle = 3$.

In the same way, we can compute higher moments of ξ. Odd moments approach zero in the large-N limit. For example, the fifth moment is put together from const$\cdot N^2$ terms of type "$iijjj$", in addition to the N terms of type "$iiiii$". Dividing by $N^{5/2}$ indeed gives zero. The even moments are finite, and the dominant contributions for large N come, for the sixth moment, from terms of type "$iijjkk$" and their permutations. The total number of these terms, for large N is $\simeq N^3 \cdot 6!/3!/2^3 = N^3 \times 1 \times 3 \times 5$. For arbitrary k, the number of ways is

$$\left\{ \begin{array}{c} \text{moments} \\ \text{in eqn (1.62)} \end{array} \right\} \left\{ \begin{array}{l} \langle \xi^{2k} \rangle \to 1 \times 3 \times \cdots \times (2k-1) \\ \langle x^{2k-1} \rangle \to 0 \end{array} \right. \quad \text{for } N \to \infty. \quad (1.64)$$

We now check that the Gaussian distribution also has the moments given by eqn (1.64). This follows trivially for the odd moments, as the Gaussian is a symmetric function. The even moments can be computed from

$$\underbrace{\int_{-\infty}^{\infty} \frac{\mathrm{d}x}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2} + xh\right) = \exp\left(\frac{h^2}{2}\right)}_{\text{because } \int \frac{\mathrm{d}x}{\sqrt{2\pi}} \exp\left[-(x-h)^2/2\right] = 1} = 1 + \frac{h^2}{2} + \frac{(h^2/2)^2}{2!} + \cdots .$$

We may differentiate this equation $2k$ times (under the integral on the left, as it is absolutely convergent) and then set $h = 0$. On the left, this gives the $(2k)$th moment of the Gaussian distribution. On the right, we obtain an explicit expression:

$$\begin{aligned} \langle x^{2k} \rangle &= \frac{\partial^{2k}}{\partial h^{2k}} \int_{-\infty}^{\infty} \frac{\mathrm{d}x}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2} + xh\right)\Big|_{h=0} \\ &= \frac{\partial^{2k}}{\partial h^{2k}} \left[1 + \frac{h^2}{2} + \frac{(h^2/2)^2}{2!} + \frac{(h^2/2)^3}{3!} + \cdots\right]_{h=0} \\ &= \frac{\partial^{2k}}{\partial h^{2k}} \left(\frac{h^{2k}}{k! 2^k}\right) = \frac{(2k)!}{k! 2^k} = 1 \times 3 \times \cdots \times (2k-1). \end{aligned}$$

This is the same as eqn (1.64), and the distribution function of the sum of N random variables, in the limit $N \to \infty$, has no choice but to converge to a Gaussian, so that we have proven the central limit theorem (for the simplified case that all the moments of the distribution are finite). The Gaussian has 68% of its weight concentrated within the interval $[-\sigma, \sigma]$ and 95% of its weight within the interval $[-2\sigma, 2\sigma]$ (see Fig. 1.36). These numbers are of prime importance for statistical calculations in general, and for Monte Carlo error analysis in particular. The probability of being more than a few standard deviations away from the mean value drops precipitously (see Table 1.12, where the error function is $\mathrm{erf}(x) = (2/\sqrt{\pi}) \int_0^t \mathrm{d}t \ \exp\left(-t^2\right)$).

In practical calculations, we are sometimes confronted with a few exceptional sample averages that are many (estimated) standard deviations away from our (estimated) mean value. We may, for example,

Table 1.12 Probability of being outside a certain interval for any distribution (from the Chebyshev inequality) and for a Gaussian distribution.

Excluded interval	Probability of being $\notin$ interval	
	Chebyshev	Gaussian
$[-\sigma, \sigma]$	Less than 100%	32%
$[-2\sigma, 2\sigma]$	Less than 25%	5%
$[-3\sigma, 3\sigma]$	Less than 11%	0.3%
$[-4\sigma, 4\sigma]$	Less than 6%	0.006%
$[-k\sigma, k\sigma]$	Less than $\frac{1}{k^2}$	$1 - \mathrm{erf}(k/\sqrt{2})$

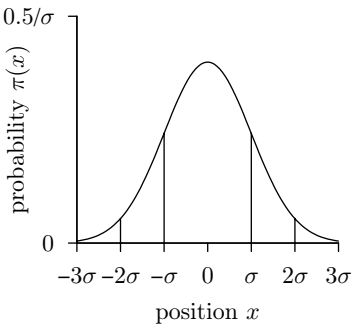

Fig. 1.36 The Gaussian distribution. The probability of being outside the interval $[-\sigma, \sigma]$ is 0.32, etc. (see Table 1.12)

compute an observable (say a magnetization) as a function of an external parameter (say the temperature). A number of values lie nicely on our favorite theoretical curve, but the exceptional ones are 10 or 15 standard deviations off. With the central limit theorem, such an outcome is extremely unlikely. In this situation, it is a natural tendency to think that the number of samples might be too small for the central limit theorem to apply. This reasoning is usually erroneous because, first of all, this limit is reached extremely quickly if only the distribution function of the random variables ξ_i is well behaved. Secondly, the Chebyshev inequality applies to arbitrary distributions with finite variance. It also limits deviations from the mean value, even though less strictly (see Table 1.12). In the above case of data points which are very far off a theoretical curve, it is likely that the estimated main characteristics of the distribution function are not at all what we estimate them to be, or (in a Markov-chain Monte Carlo calculation) that the number of independent data sets has been severely overestimated. Another possibility is that our favorite theoretical curve is simply wrong.

1.3.4 Data analysis for independent variables

Experiments serve no purpose without data analysis, and Monte Carlo calculations are useful only if we can come up with an estimate of the observables, such as an approximate value for the mathematical constant π, or an estimate of the quark masses, a value for a condensate fraction, an approximate equation of state, etc. Our very first simulation in this book generated 3156 hits for 4000 trials (see Table 1.1). We shall now see what this result tells us about π, at the most fundamental level of understanding. Hits and nonhits were generated by the Bernoulli distribution:

$$\xi_i = \begin{cases} 1 & \text{with probability } \theta \\ 0 & \text{with probability } (1-\theta) \end{cases} , \tag{1.65}$$

but the value of $\pi/4 = \theta = \langle \xi_i \rangle$ is supposed unknown. Instead of the original variables ξ_i, we consider random variables η_i shifted by this

unknown mean value:

$$\eta_i = \xi_i - \theta.$$

The shifted random variables η_i now have zero mean and the same variance as the original variables ξ_i (see eqns (1.52) and (1.53)):

$$\langle \eta_i \rangle = 0, \quad \mathrm{Var}\,(\eta_i) = \mathrm{Var}\,(\xi_i) = \theta(1-\theta) \leq \frac{1}{4}.$$

Without invoking the limit $N \to \infty$, we can use the Chebyshev inequality eqn (1.59) to obtain an interval around zero containing at least 68% of the probability:

$$\left\{ \begin{matrix} \text{with 68\%} \\ \text{probability} \end{matrix} \right\} : \quad \underbrace{\left| \frac{1}{N} \sum_{i=1}^{N} \eta_i \right|}_{\text{see eqn (1.66)}} < \frac{1.77\sigma}{\sqrt{N}} < \frac{1.77}{2\sqrt{4000}} = 0.014.$$

In our example, the realizations of the η_i satisfy

$$\frac{1}{N} \sum_{i=1}^{N} \eta_i = \underbrace{\frac{3156}{4000}}_{0.789} - \frac{\pi}{4}. \qquad (1.66)$$

This has implications for the difference between our experimental result, 0.789, and the mathematical constant π. The difference between the two, with more than 68% chance, is smaller than 0.014:

$$\frac{\pi}{4} = 0.789 \pm 0.014 \Leftrightarrow \pi = 3.156 \pm 0.056, \qquad (1.67)$$

where the value 0.056 is an upper bound for the 68% confidence interval that in physics is called an error bar. The quite abstract reasoning leading from eqn (1.65) to eqn (1.67)—in other words from the experimental result 3156 to the estimate of π with an error bar—is extremely powerful, and not always well understood. To derive the error bar, we did not use the central limit theorem, but the more general Chebyshev inequality. We also used an upper bound for the variance. With these precautions we arrived at the following result. We know with certainty that among an infinite number of beach parties, at which participants would play the same game of 4000 as we described in Subsection 1.1.1 and which would yield Monte Carlo results analogous to ours, more than 68% would hold the mathematical value of π inside their error bars. In arriving at this result, we did not treat the number π as a random variable—that would be nonsense, because π is a mathematical constant.

We must now relax our standards. Instead of reasoning in a way that holds for all N, we suppose the validity of the central limit theorem. The above 68% confidence limit for the error bar now leads to the following estimate of the mean value:

$$\langle \xi \rangle = \frac{1}{N} \sum_{i=1}^{N} \xi_i \pm \frac{\sigma}{\sqrt{N}}$$

(see Table 1.12; the 68% confidence interval corresponds to one standard deviation of the Gaussian distribution).

In addition, we must also give up the bound for the variance in favor of an estimate of the variance, through the expression

$$\mathrm{Var}\,(\xi_i) = \mathrm{Var}\,(\eta_i) \simeq \frac{1}{N-1} \left[\sum_{j=1}^{N} (\xi_j - \frac{1}{N} \sum_{i=1}^{N} \xi_i)^2 \right]. \qquad (1.68)$$

The mean value on the right side of eqn (1.68) is equal to the variance for all N (and one therefore speaks of an unbiased estimator). With the replacement $\xi_j \to \eta_j$, we obtain

$$\left\langle \frac{1}{N-1}\left[\sum_{j=1}^{N}\left(\eta_j - \frac{1}{N}\sum_{i=1}^{N}\eta_i\right)^2\right]\right\rangle$$
$$= \frac{1}{N-1}\left[\sum_{jk}^{N}\underbrace{\langle\eta_j\eta_k\rangle}_{\mathrm{Var}(\eta_j)\delta_{jk}} - \frac{2}{N}\sum_{ij=1}^{N}\langle\eta_i\eta_j\rangle + \frac{1}{N^2}\sum_{ijk}\langle\eta_i\eta_k\rangle\right] = \mathrm{Var}\,(\eta_i)$$

(a more detailed analysis is possible under the conditions of the central limit theorem). In practice, we always replace the denominator $\frac{1}{N-1} \to \frac{1}{N}$ in eqn (1.68) (the slight difference plays no role at all). We arrive, with eqn (1.69), at the standard formulas for the data analysis of independent random variables:

$$\langle\xi\rangle = \frac{1}{N}\sum_{i=1}^{N}\xi_i \pm \text{error},$$

where

$$\text{error} = \frac{1}{\sqrt{N}}\sqrt{\frac{1}{N}\sum_i \xi_i^2 - \left(\frac{1}{N}\sum_i \xi_i\right)^2}.$$

The error has a prefactor $1/\sqrt{N}$ and not $1/N$, because it goes as the standard deviation, itself the square root of the variance. It should be noted that the replacement of the variance by an estimate is far from innocent. This replacement can lead to very serious problems, as we shall discuss in an example, namely the γ-integral of Section 1.4. In Markov-chain Monte Carlo applications, another problem arises because the number of samples, N, must be replaced by an effective number of independent samples, which must itself be estimated with care. This will be taken up in Subsection 1.3.5.

We now briefly discuss the Bayesian approach to statistics, not because we need it for computing error bars, but because of its close connections with statistical physics (see Section 2.2). We stay in the imagery of the children's game. Bayesian statistics attempts to determine the probability that our experimental result (3156 hits) was the result of a certain value π_{test}. If π_{test} was very large ($\pi_{\text{test}} \lesssim 4$, so that $\theta \lesssim 1$), and also if $\pi_{\text{test}} \gtrsim 0$, the experimental result, 3156 hits for 4000 trials, would be very unlikely. Let us make this argument more quantitative and suppose that the test values of π are drawn from an a priori probability distribution (not to be confused with the a priori probability of the generalized Metropolis algorithm). The test values give hits and nonhits with the appropriate Bernoulli distribution and values of N_{hits} with their binomial distribution (with parameters $\theta = \pi_{\text{test}}/4$ and $N = 4000$), but there is a great difference between those test values that give 3156 hits, and those that do not. To illustrate this point, let us sample the values of π_{test} that yield 3156 hits with a program, Alg. 1.27 (`naive-bayes-pi`). Like

We note that

$$\sum_{j=1}^{N}\left(\xi_j - \frac{1}{N}\sum_{i=1}^{N}\xi_i\right)^2 =$$
$$\sum_{j=1}^{N}\xi_j^2 - \frac{2}{N}(\sum_{j=1}^{N}\xi_j)(\sum_{i=1}^{N}\xi_i) + \frac{1}{N}(\sum_{i=1}^{N}\xi_i)^2$$
$$= \sum_j \xi_j^2 - N\left(\frac{1}{N}\sum_j \xi_j\right)^2. \quad (1.69)$$

all other naive algorithms in this book, it is conceptually correct, but unconcerned with computational efficiency. This program picks a random value of π_{test} (from the a priori probability distribution), then samples this test value's binomial distribution. The "good" test values are kept in a table (see Table 1.13). They have a probability distribution (the a posteriori probability distribution), and it is reasonable to say that this distribution contains information on the mathematical constant π.

```
      procedure naive-bayes-pi
1     π_test ← ran(0,4)  (sampled with the a priori probability)
      N_hits ← 0
      for i = 1,...,4000 do
        { if (ran(0,1) < π_test/4) then
            { N_hits ← N_hits + 1
      if (N_hits ≠ 3156) goto 1 (reject π_test)
      output π_test (output with the a posteriori probability)
```

Algorithm 1.27 `naive-bayes-pi`. Generating a test value π_{test}, which leads to $N_{\text{hits}} = 3156$ for $N = 4000$ trials.

Table 1.13 Values of π_{test} that lead to 3156 hits for 4000 trials (from Alg. 1.27 (`naive-bayes-pi`))

Run	π_{test}
1	3.16816878
2	3.17387056
3	3.16035151
4	3.13338971
5	3.16499329
⋮	⋮

In the Bayesian approach, the choice of the a priori probability in Alg. 1.27 (`naive-bayes-pi`) influences the outcome in Table 1.13. We could use nonuniform a priori probability distributions, as for example $\pi^2_{\text{test}} \leftarrow \texttt{ran}(0,16)$, or $\sqrt{\pi_{\text{test}}} \leftarrow \texttt{ran}(0,2)$. Since we know Archimedes' result, it would be an even better idea to use $\pi_{\text{test}} \leftarrow \texttt{ran}\left(3\frac{10}{71}, 3\frac{1}{7}\right)$ (which does not contain the point 3.156...). Some choices are better than others. However, there is no best choice for the a priori probability, and the final outcome, the a posteriori probability distribution, will (for all finite N) carry the mark of this input distribution.

Let us rephrase the above discussion in terms of probability distributions, rather than in terms of samples, for random π_{test} between 0 and 4:

$$\underbrace{\left\{\begin{array}{c}\text{probability of having}\\ \pi_{\text{test}} \text{ with 3156 hits}\end{array}\right\}}_{\text{a posteriori probability for } \pi} = \underbrace{\int_0^4 \mathrm{d}\pi_{\text{test}}}_{\substack{\text{a priori}\\ \text{probability}}} \underbrace{\left\{\begin{array}{c}\text{probability that } \pi_{\text{test}}\\ \text{yields 3156 hits}\end{array}\right\}}_{\substack{\text{binomial probability of obtaining}\\ \text{3156 hits, given } \pi_{\text{test}}}} .$$

This integral is easily written down and evaluated analytically. For the binomial distribution, it leads to essentially equivalent expressions as the error analysis from the beginning of this subsection.

Other choices for the a priori probability are given by

$$\int_0^4 \mathrm{d}\pi_{\text{test}} \ , \quad \int_0^{16} \mathrm{d}(\pi^2_{\text{test}}) \ , \quad \int_0^2 \mathrm{d}\sqrt{\pi_{\text{test}}} \ , \ldots \underbrace{\int_{3\frac{10}{71}}^{3\frac{1}{7}} \mathrm{d}\pi_{\text{test}}}_{\substack{\text{Archimedes,}\\ \text{see eqn (1.1)}}} , \text{ etc.} \quad (1.70)$$

Some a priori probabilities are clearly preferable to others, and they all give different values a posteriori probability distributions, even though

the differences are rarely as striking as the ones shown in Fig. 1.37. In the limit $N \to \infty$, all choices become equivalent.

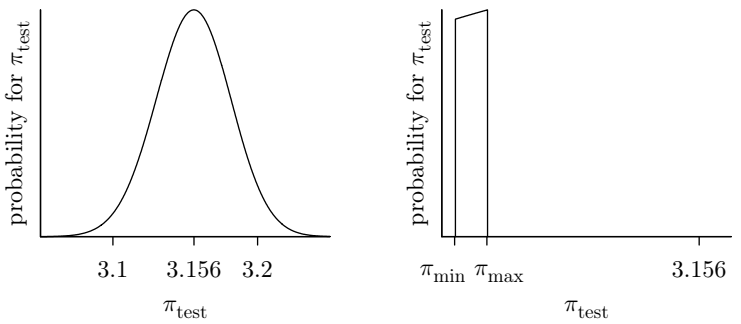

Fig. 1.37 A posteriori probability for a priori choices $\pi_{\text{test}} \leftarrow \texttt{ran}\,(0,4)$ (*left*) and $\pi_{\text{test}} \leftarrow \texttt{ran}\left(3\frac{10}{71}, 3\frac{1}{7}\right)$ (*right*).

Bayesian statistics is a field of great importance for complicated classification and recognition problems, which are all beyond the scope of this book. We saw how easy it is to incorporate Archimedes' bias $3\frac{10}{71} < \pi < 3\frac{1}{7}$. Within Bayesian statistics, it is nevertheless impossible to decide which choice would least influence data analysis (be an unbiased choice), for example among the first three a priori probabilities in eqn (1.70). We stress that no such ambiguity affected the derivation of the error bar of eqn (1.67) (using the shifted variables η_i). The concept of an unbiased choice will come up again in the discussion of entropy and equiprobability in statistical mechanics (see Section 2.2).

We have seen in the first part of the present subsection that error analysis stands on an extremely solid basis. The children's Monte Carlo calculation of π could have been done a long time before Archimedes obtained a good analytical bound for it. In the same way, we may obtain, using Monte Carlo methods, numerical results for a model in statistical mechanics or other sciences, years before the model is analytically solved. Our numerical results should agree closely, and we should claim credit for our solution. If, on the other hand, our numerical results turn out to be wrong, we were most probably sloppy in generating, or in analyzing, our results. Under no circumstances can we excuse ourselves by saying that it was "just a simulation".

1.3.5 Error estimates for Markov chains

We have so far discussed the statistical errors of independent data, as produced by Alg. 1.1 (`direct-pi`) and other direct-sampling algorithms. We must now develop an equally firm grip on the analysis of correlated data. Let us look again at Alg. 1.2 (`markov-pi`). This archetypal Markov-chain program has left 20 000 pebbles lying on the heliport (from five

runs with 4000 pebbles each), which we cannot treat as independent. The pebbles on the ground are distributed with a probability $\pi(x, y)$, but they tend to be grouped, and even lie in piles on top of each other. What we can learn about $\pi(x, y)$ by sampling is less detailed than what is contained in the same number of independent pebbles. As a simple consequence, the spread of the distribution of run averages is wider than before.

It is more sensible to treat not the 5×4000 pebbles but the five run averages for $4N_{\text{hits}}/N$ (that is, the values $\{3.123, \ldots, 3.263\}$) as independent, approximately Gaussian variables (see Table 1.14). We may then compute an error estimate from the means and mean square deviations of these five numbers. The result of this naive reanalysis of the heliport data is shown in Table 1.14.

Table 1.14 Naive reanalysis of the Markov-chain data from Table 1.2, treating N_{hits} for each run as independent

Run	N_{hits}	Estimate of π
1	3123	
2	3118	3.122
3	3040	$\pm$
4	3066	0.04
5	3263	

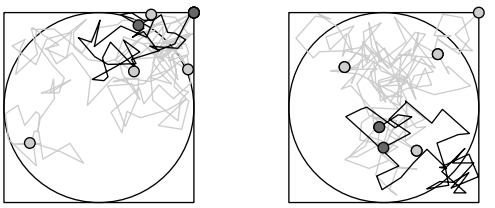

Fig. 1.38 Markov chains on the heliport. *Left*: all chains start at the clubhouse. *Right*: one chain starts where the previous one stops.

Analyzing a few very long runs is a surefooted, fundamentally sound strategy for obtaining a first error estimate, especially when the influence of initial conditions is negligible. In cases such as in the left frame of Fig. 1.38, however, the choice of starting point clearly biases the estimation of π, and we want each individual run to be as long as possible. On the other hand, we also need a large number of runs in order to minimize the uncertainty in the error estimate itself. This objective favors short runs. With naive data analysis, it is not evident how to find the best compromise between the length of each run and the number of runs, for a given budget of computer time.

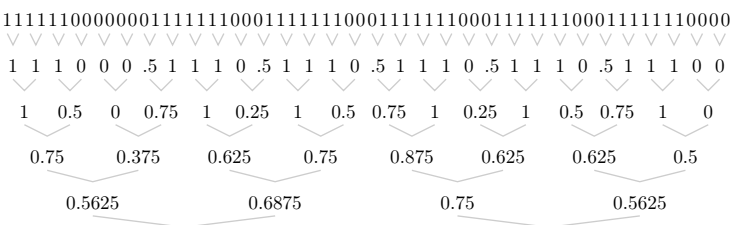

Fig. 1.39 Four iterations of Alg. 1.28 (`data-bunch`) applied to the correlated output of Alg. 1.2 (`markov-pi`).

In the case of the heliport, and also in general, there is an easy way out of this dilemma: rather than have all Markov chains start at the clubhouse, we should let one chain start where the last one left off (see the right frame of Fig. 1.38). This gives a single, much longer Markov chain. In this case, the cutting-up into five bunches is arbitrary. We could equally well produce bunches of size $\{2, 4, 8, \dots\}$, especially if the number of data points is a power of two. The bunching into sets of increasing length can be done iteratively, by repeatedly replacing two adjacent samples by their average value (see Fig. 1.39 and Alg. 1.28 (`data-bunch`)). At each iteration, we compute the apparent error, as if the data were independent. The average value remains unchanged.

Bunching makes the data become increasingly independent, and makes the apparent error approach the true error of the Markov chain. We want to understand why the bunched data are less correlated, even though this can already be seen from Fig. 1.38. In the kth iteration, bunches of size 2^k are generated: let us suppose that the samples are correlated on a length scale $\xi \lesssim 2^k$, but that original samples distant by more than 2^k are fully independent. It follows that, at level k, these correlations still affect neighboring bunches, but not next-nearest ones (see Fig. 1.39): the correlation length of the data decreases from length 2^k to a length $\lesssim 2$. In practice, we may do the error analysis on all bunches, rather than on every other one.

procedure `data-bunch`
input $\{x_1, \dots, x_{2N}\}$ (Markov-chain data)
$\Sigma \leftarrow 0$
$\Sigma' \leftarrow 0$
for $i = 1, \dots, N$ **do**
$$\begin{cases} \Sigma \leftarrow \Sigma + x_{2i-1} + x_{2i} \\ \Sigma' \leftarrow \Sigma' + x_{2i-1}^2 + x_{2i}^2 \\ x'_i \leftarrow (x_{2i-1} + x_{2i})/2 \end{cases}$$
$\text{error} \leftarrow \sqrt{\Sigma'/(2N) - (\Sigma/(2N))^2}/\sqrt{2N}$
output $\Sigma/(2N), \text{error}, \{x'_1, \dots, x'_N\}$

Algorithm 1.28 `data-bunch`. Computing the apparent error (treating data as independent) for $2N$ data points and bunching them into pairs.

It is interesting to apply Alg. 1.28 (`data-bunch`) repeatedly to the data generated by a long simulation of the heliport (see Fig. 1.40). In this figure, we can identify three regimes. For bunching intervals smaller than the correlation time (here $\simeq 64$), the error is underestimated. For larger bunching intervals, a characteristic plateau indicates the true error of our simulation. This is because bunching of uncorrelated data does not change the expected variance of the data. Finally, for a very small number of intervals, the data remain uncorrelated, but the error estimate itself becomes noisy.

Algorithm 1.28 (`data-bunch`), part of the folklore of Monte Carlo computation, provides an unbeaten analysis of Markov-chain data, and

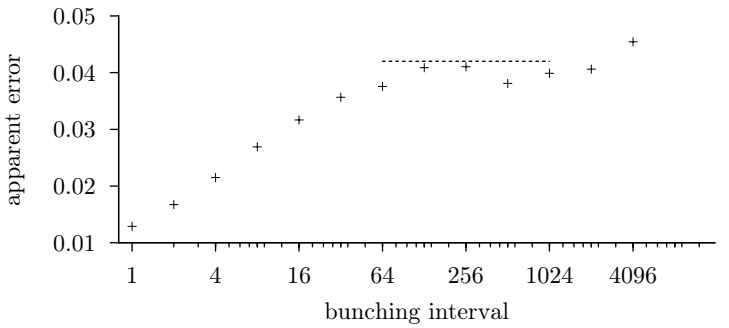

Fig. 1.40 Repeated bunching of Markov-chain data (from Alg. 1.2 (`markov-pi`) ($N = 2^{14}$, $\delta = 0.3$), analysis by Alg. 1.28 (`data-bunch`)).

is the only technique needed in this book. Data bunching is not fail-safe, as we shall discuss in Section 1.4, but it is the best we can do. What is missing to convince us of the pertinence of our numerical results must be made up for by critical judgment, rigorous programming, comparison with other methods, consideration of test cases, etc.

1.4 Computing

Since the beginning of this chapter, we have illustrated the theory of Monte Carlo calculation in simple, unproblematic settings. It is time to become more adventurous, and to advance into areas where things can go wrong. This will acquaint us with the limitations and pitfalls of the Monte Carlo method. Two distinct issues will be addressed. One is the violation of ergodicity—the possibility that a Markov chain never visits all possible configurations. The second limiting issue of the Monte Carlo method shows up when fluctuations become so important that averages of many random variables can no longer be understood as a dominant mean value with a small admixture of noise.

1.4.1 Ergodicity

The most characteristic limitation of the Monte Carlo method is the slow convergence of Markov chains, already partly discussed in Subsection 1.1.2: millions of chain links in a simulation most often correspond to only a handful of independent configurations. Such a simulation may resemble our random walk on the surface of a sphere (see Fig. 1.32): many samples were generated, but we have not even gone around the sphere once.

It routinely happens that a computer program has trouble decorrelating from the initial configuration and settling into the stationary probability distribution. In the worst case, independent samples are not even

created in the limit of infinite computer time. One then speaks of a nonergodic algorithm. However, we should stress that a practically nonergodic algorithm that runs for a month without decorrelating from the clubhouse is just as useless and just as common.[7]

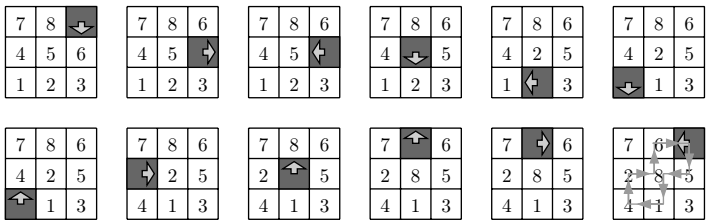

Fig. 1.41 Local Monte Carlo moves applied to the sliding puzzle, a variant of the pebble game.

In later chapters, we shall be closely concerned with slow (practically nonergodic) algorithms, and the concept of ergodicity will be much discussed. At the present stage, however, let us first look at a nonergodic Monte Carlo algorithm in a familiar setting that is easy to analyze: a variant of our pebble game, namely the sliding puzzle, a popular toy (see Fig. 1.41).

The configurations of the sliding puzzle correspond to permutations of the numbers $\{0, \ldots, 8\}$, the zero square being empty. Let us see why not all configurations can be reached by repeated sliding moves in the local Monte Carlo algorithm, whereby a successful move of the empty square corresponds to its transposition with one of its neighbors. We saw in Subsection 1.2.2 that all permutations can be constructed from transpositions of two elements. But taking the zero square around a closed loop makes us go left and right the same number of times, and we also perform equal numbers of successful up and down moves (a loop is shown in the last frame of Fig. 1.41). Therefore, on a square lattice, the zero square can return to its starting position only after an even number of moves, and on completion of an even number of transpositions. Any accessible configuration with this square in the upper right corner thus corresponds to an even permutation (compare with Subsection 1.2.2), and odd permutations with the zero square in the upper right corner, as in Fig. 1.42, can never be reached: the local Monte Carlo algorithm for the sliding puzzle is nonergodic.

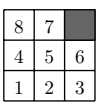

Fig. 1.42 A puzzle configuration that cannot be reached from the configurations in Fig. 1.41.

1.4.2 Importance sampling

Common sense tells us that nonergodic or practically nonergodic algorithms are of little help in solving computational problems. It is less evident that even ergodic sampling methods can leave us with a tricky job.

[7]This definition of ergodicity, for a random process, is slightly different from that for a physical system.

Difficulties appear whenever there are rare configurations—configurations with a low probability—which contribute in an important way to the value of our integral. If these rare yet important configurations are hardly ever seen, say in a week-long Monte Carlo simulation, we clearly face a problem. This problem, though, has nothing to do with Markov chains and shows up whether we use direct sampling as in the children's algorithm or walk around like adults on the heliport. It is linked only to the properties of the distribution that we are trying to sample.

In this context, there are two archetypal examples that we should familiarize ourselves with. The first is the Monte Carlo golf course. We imagine trying to win the Monaco golf trophy not by the usual playing technique but by randomly dropping golf balls (replacing pebbles) onto the greens (direct sampling) or by randomly driving balls across the golf course, with our eyes closed (Markov-chain sampling). Either way, random sampling would take a lot of trials to hit even a single hole out there on the course, let alone bring home a little prize money, or achieve championship fame. The golf course problem and its cousin, that of a needle in a haystack, are examples of genuinely hard sampling problems which cannot really be simplified.

The second archetypal example containing rare yet important configurations belongs to a class of models which have more structure than the golf course and the needle in a haystack. For these problems, a concept called importance sampling allows us to reweight probability densities and observables and to overcome the basic difficulty. This example is defined by the integral

$$I(\gamma) = \int_0^1 \mathrm{d}x \; x^\gamma = \frac{1}{\gamma + 1} \text{ for } \gamma > -1. \tag{1.71}$$

We refer to this as the γ-integral, and shall go through the computational and mathematical aspects of its evaluation by sampling methods in the remainder of this section. The γ-integral has rare yet important configurations because, for $\gamma < 0$, the integrand all of a sudden becomes very large close to $x = 0$. Sampling the γ-integral is simple only in appearance: what happens just below the surface of this problem was cleared up only recently (on the timescale of mathematics) by prominent mathematicians, most notably P. Lévy. His analysis from the 1930s will directly influence the understanding of our naive Monte Carlo programs.

To evaluate the γ-integral—at least initially—one generates uniform random numbers x_i between 0 and 1, and averages the observable $\mathcal{O}_i = x_i^\gamma$ (see Alg. 1.29 (`direct-gamma`)). The output of this program should approximate $I(\gamma)$.

To obtain the error, a few lines must be added to the program, in addition to the mean

$$\frac{\Sigma}{N} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}_i \simeq \langle \mathcal{O} \rangle \,,$$

procedure `direct-gamma`
$\Sigma \leftarrow 0$
for $i = 1, \ldots, N$ **do**
$\left\{ \begin{array}{l} x_i \leftarrow \texttt{ran}\,(0,1) \\ \Sigma \leftarrow \Sigma + x_i^\gamma \text{ (running average: } \Sigma/i) \end{array} \right.$
output Σ/N

Algorithm 1.29 `direct-gamma`. Computing the γ-integral in eqn (1.71) by direct sampling.

which is already programmed. We also need the average of the squared observables,

$$\frac{1}{N}\sum_{i=1}^{N} x_i^{2\gamma} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{O}_i^2 \simeq \langle \mathcal{O}^2 \rangle .$$

This allows us to estimate the variance (see Subsection 1.3.5):

$$\text{error} = \frac{\sqrt{\langle \mathcal{O}^2 \rangle - \langle \mathcal{O} \rangle^2}}{\sqrt{N}}. \tag{1.72}$$

With direct sampling, there are no correlation times to worry about. The output for various values of γ is shown in Table 1.15.

Table 1.15 Output of Alg. 1.29 (`direct-gamma`) for various values of γ ($N = 10\,000$). The computation for $\gamma = -0.8$ is in trouble.

γ	$\Sigma/N \pm$ Error	$1/(\gamma+1)$
2.0	0.334 ± 0.003	$0.333\ldots$
1.0	0.501 ± 0.003	0.5
0.0	1.000 ± 0.000	1
-0.2	1.249 ± 0.003	1.25
-0.4	1.682 ± 0.014	$1.666\ldots$
-0.8	3.959 ± 0.110	5.0

Most of the results in this table agree with the analytical results to within error bars, and we should congratulate ourselves on this nice success! In passing, we should look at the way the precision in our calculation increases with computer time, i.e. with the number N of iterations. The calculation recorded in Table 1.15 (with $N = 10\,000$) for $\gamma = 2$, on a year 2005 laptop computer, takes less than 1/100 s and, as we see, reaches a precision of 0.003 (two digits OK). Using 100 times more samples ($N = 10^6$, 0.26 s), we obtain the result 0.33298 (not shown, three significant digits). One hundred million samples are obtained in 25 seconds, and the result obtained is 0.3333049 (not shown, four digits correct). The precision increases as the square root of the number of samples, and gains one significant digit for each hundred-fold increase of computer time. This slow but sure convergence is a hallmark of Monte Carlo integration and can be found in any program that runs without bugs and irregularities and has a flawless random number generator.

However, the calculation for $\gamma = -0.8$ in Table 1.15 is in trouble: the average of the $\mathcal{O}_i$'s is much further off $I(\gamma)$ than the internally computed error indicates. What should we do about this? There is no rescue in sarcasm[8] as it is nonsense, because of Chebyshev's inequality, to think that one could be 10 standard deviations off the mean value. Furthermore, Alg. 1.29 (`direct-gamma`) is too simple to have bugs, and even our analytical calculation in eqn (1.71) is watertight: $I(-0.8)$ is indeed 5 and

[8] An example of sarcasm: there are three kinds of lies: lies, damned lies, and statistics.

not 4. . . . To make progress, we monitor our simulation, and output running averages, as indicated in Alg. 1.29 (`direct-gamma`). In Fig. 1.43, this average calmly approaches a wrong mean value. Then chance lets the program hit a very small value x_i, with an incredibly large $\mathcal{O}_i$ (we remember that $\gamma < 0$ so that small values of x give large observables). Thus one sample, in a simulation of a million trials, single-handedly hikes up the running average.

Figure 1.43 contains a nightmare scenario: a Monte Carlo simulation converges nicely (with an average of about 4.75, well established for a computational eternity), until a seemingly pathological sample changes the course of the simulation. In this situation, real insight and strength of human character are called for: *we must not doctor the data and suppress even a single sample* (which would let us continue with an average near 4.75). Sooner or later, someone would find out that our data were botched!

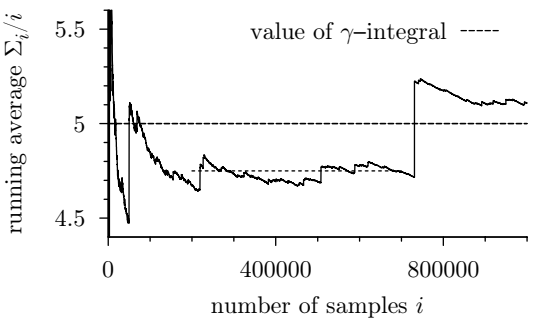

Fig. 1.43 Running average of Alg. 1.29 (`direct-gamma`) for $\gamma = -0.8$.

What happens in Table 1.15 is that the error formula in eqn (1.72) involves an average over the squared observable:

$$\underbrace{\frac{1}{N}\sum_{i=1}^{N}\mathcal{O}_i^2}_{\substack{\text{estimate}\\ \text{(always finite)}}} \simeq \underbrace{\int_0^1 \mathrm{d}x\; x^{2\gamma}}_{\substack{\text{variance}\\ \text{(infinite for } \gamma<-\frac{1}{2})}} \quad . \tag{1.73}$$

We see that the algorithm to evaluate the γ-integral, in the range $-1 < \gamma < -\frac{1}{2}$, where it is still finite, has a problem in estimating the error, because it uses a finite sum to approximate an integral with an infinite variance. Clearly, the situation is difficult to analyze in the absence of an analytic solution.

We can salvage the Monte Carlo calculation of the γ-integral by preferentially visiting regions of space where the expression $\mathcal{O}(x)\pi(x)$ is large. In our example, we split the integrand of the γ-integral appropriately into a new probability density $\pi(x) = x^\zeta$ and a new observable

$\mathcal{O}(x) = x^{\gamma-\zeta}$ (see Fig. 1.44). For negative ζ ($\gamma < \zeta < 0$), small values of x, with a large integrand, are visited more often and the variance of the observable is reduced, while the product of the probability density and the observable remains unchanged. This crucial technique is called importance sampling.

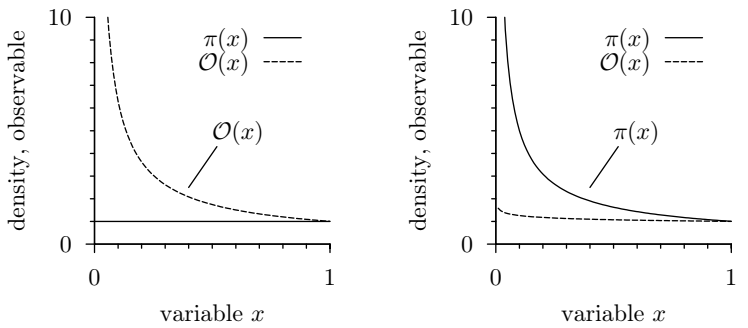

Fig. 1.44 Splits of the γ-integrand into a density and an observable ($\gamma = -0.8$). *Left*: $\pi(x) = 1$; $\mathcal{O}(x) = x^{-0.8}$. *Right*: $\pi(x) = x^{-0.7}$; $\mathcal{O}(x) = x^{-0.1}$.

procedure `direct-gamma-zeta`
$\Sigma \leftarrow 0$
for $i = 1, \ldots, N$ **do**
$\left\{ \begin{array}{l} x_i \leftarrow \texttt{ran}\,(0,1)^{1/(\zeta+1)} \quad (\pi(x_i) \propto x_i^{\zeta}, \text{ see eqn (1.29)}) \\ \Sigma \leftarrow \Sigma + x_i^{\gamma-\zeta} \end{array} \right.$
output Σ/N

Algorithm 1.30 `direct-gamma-zeta`. Using importance sampling to compute the γ-integral (see eqn (1.74)).

The idea is implemented in Alg. 1.30 (`direct-gamma-zeta`), our first application of importance sampling. The output of the program, Σ/N, corresponds to the ratio of two γ-integrals:

$$\Sigma/N = \frac{1}{N}\sum_{i=1}^{N} \mathcal{O}_i \simeq \langle \mathcal{O} \rangle = \frac{\int_0^1 \mathrm{d}x\ \pi(x)\mathcal{O}(x)}{\int_0^1 \mathrm{d}x\ \pi(x)} = \frac{\int_0^1 \mathrm{d}x\ x^{\zeta} x^{\gamma-\zeta}}{\int_0^1 \mathrm{d}x\ x^{\zeta}} = \frac{\int_0^1 \mathrm{d}x\ x^{\gamma}}{\int_0^1 \mathrm{d}x\ x^{\zeta}} = \frac{I(\gamma)}{I(\zeta)} = \frac{\zeta+1}{\gamma+1}. \quad (1.74)$$

This is because Monte Carlo simulations compute $\int \mathrm{d}x\ \pi(x)\mathcal{O}(x)$ only if the density function π is normalized. Otherwise, we have to normalize with $\int \mathrm{d}x\ \pi(x)$. A glimpse at Table 1.16 shows that the calculation comes out just right.

Table 1.16 Output of Alg. 1.30 (`direct-gamma-zeta`) with $N = 10\,000$. All pairs $\{\gamma, \zeta\}$ satisfy $2\gamma - \zeta > -1$ so that $\langle \mathcal{O}^2 \rangle < \infty$.

γ	ζ	Σ/N	$\frac{\zeta+1}{\gamma+1}$
-0.4	0.0	1.685 ± 0.017	1.66
-0.6	-0.4	1.495 ± 0.008	1.5
-0.7	-0.6	1.331 ± 0.004	1.33
-0.8	-0.7	1.508 ± 0.008	1.5

We must understand how to compute the error, and why reweighting is useful. The error formula eqn (1.72) remains valid for the observable $\mathcal{O}(x) = x^{\gamma-\zeta}$ (a few lines of code have to be added to the program). The variance of $\mathcal{O}$ is finite under the following condition:

$$\frac{1}{N}\sum_{i=1}^{N}\mathcal{O}_i^2 \simeq \langle \mathcal{O}^2 \rangle = \frac{\int_0^1 \mathrm{d}x\ \pi(x)\mathcal{O}^2(x)}{\int_0^1 \mathrm{d}x\ \pi(x)} = \int_0^1 \mathrm{d}x\ x^{\zeta} x^{2\gamma-2\zeta} < \infty \ \Leftrightarrow \gamma > -\frac{1}{2} + \zeta/2. \quad (1.75)$$

This gives a much larger range of possible γ values than does eqn (1.73) for negative values of ζ. All pairs $\{\gamma, \zeta\}$ in Table 1.16 satisfy the inequality (1.75). Together, the five different pairs in the table give a product of all the ratios equal to

$$\underbrace{\frac{\int_0^1 \mathrm{d}x\ x^{-0.8}}{\int_0^1 \mathrm{d}x\ x^{-0.7}}}_{\gamma=-0.8,\zeta=-0.7} \frac{\int_0^1 \mathrm{d}x\ x^{-0.7}}{\int_0^1 \mathrm{d}x\ x^{-0.6}} \frac{\int_0^1 \mathrm{d}x\ x^{-0.6}}{\int_0^1 \mathrm{d}x\ x^{-0.4}} \frac{\int_0^1 \mathrm{d}x\ x^{-0.4}}{\int_0^1 \mathrm{d}x\ x^{0.0}} = I(-0.8),$$

with the numerical result

$$\langle \mathcal{O} \rangle \simeq 1 \times 1.685 \times 1.495 \times 1.331 \times 1.508 = 5.057.$$

Using the rules of Gaussian error propagation, the variance is

$$\mathrm{Var}\,(\mathcal{O}) = \left[\left(\frac{0.017}{1.685}\right)^2 + \left(\frac{0.008}{1.495}\right)^2 + \left(\frac{0.004}{1.331}\right)^2 + \left(\frac{0.008}{1.508}\right)^2 \right] \times 5.057^2.$$

From the above expression, the final result for the γ-integral is

$$I(\gamma = -0.8) = 5.057 \pm 0.06,$$

obtained entirely by a controlled Monte Carlo calculation.

1.4.3 Monte Carlo quality control

In Subsection 1.4.2, the Monte Carlo evaluation of the γ-integral proceeded on quite shaky ground, because the output was sometimes correct and sometimes wrong. Distressingly, neither the run nor the data analysis produced any warning signals of coming trouble. We relied on an analytical quality control provided by the inequalities

$$\text{if } \gamma > \begin{cases} -1 & : \text{integral exists} \\ -\frac{1}{2} & : \text{variance exists} \\ -\frac{1}{2} + \zeta/2 & : \text{variance exists (importance sampling)} \end{cases}. \quad (1.76)$$

To have analytical control over a calculation is very nice, but this cannot always be achieved. Often, we simply do not completely understand the structure of high-dimensional integrals. We are then forced to

replace the above analytical "back office" with a numerical procedure that warns us, with a clear and intelligible voice, about a diverging integral or an infinite variance. Remarkably, the Markov-chain Monte Carlo algorithm can serve this purpose: we must simply reformulate the integration problem at hand such that any nonintegrable singularity shows up as an infinite statistical weight which attracts Markov chains without ever allowing them to get away again.

For concreteness, we discuss the γ-integral of Subsection 1.4.2, which we shall now try to evaluate using a Markov-chain Monte Carlo algorithm (see Alg. 1.31 (`markov-zeta`)). To see whether the γ-integral for $\gamma = -0.8$ exists and whether its variance is finite, we run Alg. 1.31 (`markov-zeta`) twice, once with statistical weights $\pi'(x) = |\mathcal{O}(x)\pi(x)| = x^{-0.8}$ and then with $\pi''(x) = |\mathcal{O}^2(x)\pi(x)| = x^{-1.6}$. The first case corresponds to putting $\zeta = -0.8$ in the algorithm, and the second to $\zeta = -1.6$.

procedure `markov-zeta`
input x
$\tilde{x} \leftarrow x + \texttt{ran}\,(-\delta, \delta)$
if $(0 < \tilde{x} < 1)$ **then**
  $p_{\text{accept}} \leftarrow (\tilde{x}/x)^{\zeta}$
  if $(\texttt{ran}\,(0,1) < p_{\text{accept}})$ $x \leftarrow \tilde{x}$
output x

Algorithm 1.31 `markov-zeta`. Markov-chain Monte Carlo algorithm for a point x on the interval $[0, 1]$ with $\pi(x) \propto x^{\zeta}$.

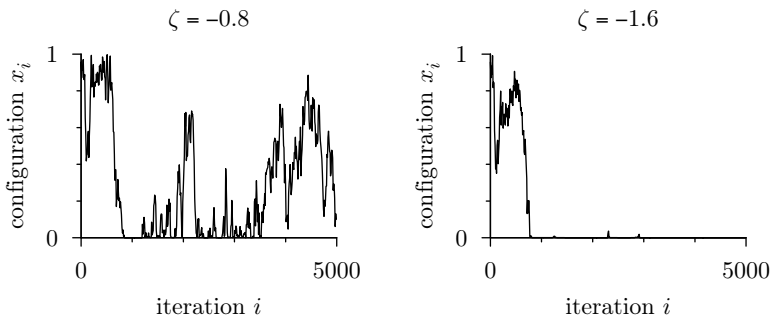

Fig. 1.45 Runs of Alg. 1.31 (`markov-zeta`). *Left*: exponent $\zeta = -0.8$: integrable singularity at $x = 0$. *Right*: $\zeta = -1.6$: nonintegrable singularity.

By simply monitoring the position x_i as a function of the iteration number i, we are able to decide whether the densities $\pi'(x)$ and $\pi''(x)$ are integrable. In the second case above, there will be a region around a point x' where $\int_{x-\epsilon}^{x+\epsilon} \mathrm{d}x'\ \pi(x') = \infty$. The simulation will be unable to

escape from the vicinity of this point. On the other hand, an integrable singularity at x ($\pi(x) \to \infty$ but $\int_{x-\epsilon}^{x+\epsilon} \mathrm{d}x'\ \pi(x')$ is finite) does not trap the simulation. Data for the two cases, $\pi'(x)$ and $\pi''(x)$, are shown in Fig. 1.45. We can correctly infer from the numerical evidence that the γ-integral exists for $\gamma = -0.8$ ($\langle \mathcal{O} \rangle$ is finite) but not for $\gamma = -1.6$ ($\langle \mathcal{O}^2 \rangle$ is infinite). In the present context, the Metropolis algorithm is purely qualitative, as no observables are computed. This qualitative method allows us to learn about the analytic properties of high-dimensional integrals when analytical information analogous to eqn (1.76) is unavailable (see Krauth and Staudacher (1999)).

1.4.4 Stable distributions

Many problems in the natural and social sciences, economics, engineering, etc. involve probability distributions which are governed by rare yet important events. For a geologist, the running-average plot in Fig. 1.43 might represent seismic activity over time. Likewise, a financial analyst might have to deal with similar data (with inverted y-axis) in a record of stock-exchange prices: much small-scale activity, an occasional Black Monday, and a cataclysmic crash on Wall Street. Neither the geologist nor the financial analyst can choose the easy way out provided by importance sampling, because earthquakes and stock crashes cannot be made to go away through a clever change of variables. It must be understood how often accidents like the ones shown in the running average in Fig. 1.43 happen. Both the geologist and the financial analyst must study the probability distribution of running averages outside the regime $\gamma > -\frac{1}{2}$, that is, when the variance is infinite and Monte Carlo calculations are impossible to do without importance sampling. The subject of such distributions with infinite variances was pioneered by Lévy in the 1930s. He showed that highly erratic sums of N random variables such as that in Fig. 1.43 tend towards universal distributions, analogously to the way that sums of random variables with a finite variance tend towards the Gaussian, as dictated by the central limit theorem. The limit distributions depend on the power (the parameter γ) and on the precise asymptotic behavior for $x \to \pm\infty$.

We shall first generate these limit distributions from rescaled outputs of Alg. 1.29 (`direct-gamma`), similarly to what was done for uniform bounded random numbers in Alg. 1.17 (`naive-gauss`). Secondly, we shall numerically convolute distributions in much the same way as we did for the bounded uniform distribution in Alg. 1.26 (`ran01-convolution`). Finally, we shall explicitly construct the limiting distributions using characteristic functions, i.e. the Fourier transforms of the distribution functions.

We run Alg. 1.29 (`direct-gamma`), not once, as for the initial running-average plot of Fig. 1.43, but a few million times, in order to produce histograms of the sample average Σ/N at fixed N (see Fig. 1.46, for $\gamma = -0.8$). The mean value of all the histograms is equal to 5, the value of the γ-integral, but this is not a very probable outcome of a single run: from

the histogram, a single run with $N = 1000$ samples is much more likely to give a sample average around 3.7, and for $N = 10\,000$, we are most likely to obtain around 4.2. This is consistent with our very first one-shot simulation recorded in Table 1.15, where we obtained $\Sigma/N = 3.95$. We note that the peak position of the distribution approaches $\langle x_i \rangle = 5$ very slowly as $N \to \infty$ (see Table 1.17).

The averages generated by Alg. 1.29 (`direct-gamma`) must be rescaled in order to fall onto a unique curve. The correct rescaling,

$$\Upsilon = \frac{\Sigma/N - \langle x_i \rangle}{N^{-1-\gamma}}, \tag{1.77}$$

will be taken for granted for the moment, and derived later, in eqn (1.80). Rescaling the output of Alg. 1.29 (`direct-gamma`), for our value $\gamma = -0.8$, consists in subtracting the mean value, 5, from each average of N terms and then dividing by $N^{-0.2}$ (see Fig. 1.46). These histograms of rescaled averages illustrate the fact that the distribution function of a sum of random variables, which we computed for the γ-integral, converges to a limit distribution for large values of N. We recall from Subsection 1.3.3 that for random variables with a finite variance, $\Sigma/\sqrt{N}$ gives a unique curve. This case is reached for $\gamma \to -\frac{1}{2}$.

Table 1.17 Peak positions of the histogram of sample averages Σ/N (from Alg. 1.29 (`direct-gamma`), with $\gamma = -0.8$)

N	Peak position of Σ/N
10	1.85
100	3.01
1000	3.74
10 000	4.21
100 000	4.50
1 000 000	4.68
10 000 000	4.80
100 000 000	4.87

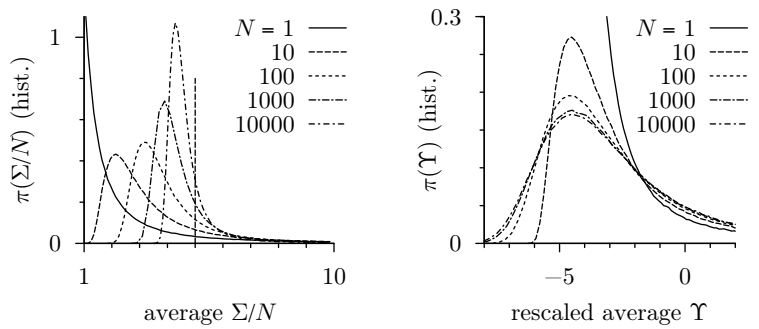

Fig. 1.46 Histograms of averages (*left*) and rescaled averages (*right*) (from Alg. 1.29 (`direct-gamma`), with $\gamma = -0.8$ and $\Upsilon = (\Sigma/N - 5)/N^{-0.2}$).

Up to now, we have formulated the γ-integral in terms of a uniform random variable $x = \texttt{ran}\,(0,1)$ and an observable $\mathcal{O}(x) = x^\gamma$. In what follows, it is better to incorporate the observable into the random variable $x = \texttt{ran}\,(0,1)^\gamma$. That is, we consider random variables ξ_i taking values x with probability

$$\pi(x) = \begin{cases} -\frac{1}{\gamma} \cdot 1/x^{1-1/\gamma} & \text{for } 1 < x < \infty \\ 0 & \text{otherwise} \end{cases} \tag{1.78}$$

(see eqn (1.29)). We shall also use $\alpha = -1/\gamma$, as is standard notation in this problem since the time of Lévy, who considered the sum of random

variables ξ_i taking values x with probability $\pi(x)$ characterized by a zero mean and by the asymptotic behavior

$$\pi(x) \simeq \begin{cases} A_+/x^{1+\alpha} & \text{for } x \to \infty \\ A_-/|x|^{1+\alpha} & \text{for } x \to -\infty \end{cases},$$

where $1 < \alpha < 2$. When rescaled as in eqn (1.77), the probability distribution keeps the same asymptotic behavior and eventually converges to a stable distribution which depends only on the three parameters $\{\alpha, A_+, A_-\}$. For the γ-integral with $\gamma = -0.8$, these parameters are $\alpha = -1/\gamma = 1.25$, $A_+ = 1.25$ and $A_- = 0$ (see eqn (1.78)). We shall later derive this behavior from an analysis of characteristic functions. As the first step, it is instructive to again assume this rescaling, and to empirically generate the universal function for the γ-integral (where $\alpha = 1.25, A_+ = 1.25$, and $A_- = 0$) from repeated convolutions of a starting distribution with itself. We may recycle Alg. 1.26 (`ran01-convolution`), but cannot cut off the distribution at large $|x|$. This would make the distribution acquire a finite variance, and drive the convolution towards a Gaussian. We must instead pad the function at large arguments, as shown in Fig. 1.47, and as implemented in Alg. 1.32 (`levy-convolution`) for $A_- = 0$. After a number of iterations, the grid of points x_k, between the fixed values $x_{\min}$ and $x_{\max}$, becomes very fine, and will eventually have to be decimated. Furthermore, during the iterations, the function $\pi(x)$ may lose its normalization and cease to be of zero mean. This is repaired by computing the norm of π—partly continuous, partly discrete—as follows:

$$\int \mathrm{d}x\ \pi(x) = \underbrace{\int_{-\infty}^{x_0} \mathrm{d}x\ \frac{A_-}{|x|^{1+\alpha}}}_{A_- \cdot \frac{1}{\alpha}\frac{1}{|x_0|^\alpha}} + \Delta\left(\frac{\pi_0+\pi_K}{2} + \sum_{k=1}^{K-1}\pi_k\right) + \underbrace{\int_{x_K}^{\infty} \mathrm{d}x\ \frac{A_+}{|x|^{1+\alpha}}}_{A_+ \cdot \frac{1}{\alpha}\frac{1}{x_K^\alpha}}.$$

The mean value can be kept at zero in the same way.

procedure `levy-convolution`
input $\{\{\pi_0, x_0\}, \ldots, \{\pi_K, x_K\}\}$ (see Fig. 1.47)
for $k = K+1, K+2\ldots$ **do** (padding)
$\quad\begin{cases} x_k \leftarrow x_0 + k\Delta \\ \pi_k \leftarrow A_+/x_k^{1+\alpha} \end{cases}$
for $k = 0, \ldots, 2K$ **do** (convolution)
$\quad\begin{cases} x'_k \leftarrow (x_0 + x_k)/2^{1/\alpha} \\ \pi'_k \leftarrow (\Delta \sum_{l=0}^{k} \pi_l \pi_{k-l}) \cdot 2^{1/\alpha} \end{cases}$
output $\{\{\pi'_m, x'_m\}, \ldots, \{\pi'_n, x'_n\}\}$ (all x' in interval $[x_{\min}, x_{\max}]$)

Algorithm 1.32 `levy-convolution`. Convolution of $\pi(x)$ with itself. $\pi(x)$ is padded as in Fig. 1.47, with $A_- = 0$.

We now discuss stable distributions in terms of their characteristic functions. Lévy first understood that a non-Gaussian stable law (of zero

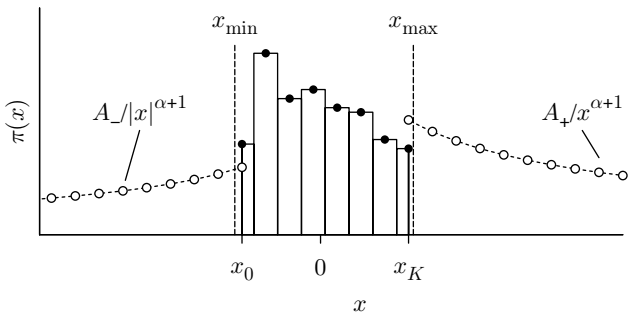

Fig. 1.47 Padding of a discrete function $\pi(x_i)$ with continuous functions $A_+/x^{1+\alpha}$ (for $x > x_{\max} \gg 0$) and $A_-/|x|^{1+\alpha}$ (for $x < x_{\min} \ll 0$).

mean) can only have a characteristic function

$$\phi(t) = \int_{-\infty}^{\infty} \mathrm{d}x\ \pi(x)\mathrm{e}^{\mathrm{i}tx} = \exp\left[-\left(c_0 + \mathrm{i}c_1\frac{t}{|t|}\right)|t|^{\alpha}\right], \tag{1.79}$$

where $1 \le \alpha < 2$ (we exclude the case $\alpha = 1$ from our discussion). This choice, with $c_0 > 0$, ensures that $\pi(x)$ is real, correctly normalized, and of zero mean (see Table 1.18). Furthermore, the square of $\phi(t)$, the characteristic function of the convolution of $\pi(t)$ with itself, is given by a rescaled $\phi(t)$:

$$\phi^2(t) = \exp\left[-\left(c_0 + \mathrm{i}c_1\frac{t}{|t|}\right)\cdot 2|t|^{\alpha}\right] = \phi(t\cdot 2^{1/\alpha}). \tag{1.80}$$

Table 1.18 Basic requirements on the stable distribution $\pi(x)$ and its characteristic function $\phi(t)$

$\pi(x)$	$\phi(t)$
Real	$\phi(t) = \phi(-t)^*$
Normalized	$\phi(0) = 1, \lvert\phi(t)\rvert \le 1$
Zero mean	$\phi'(0) = 1$
Positive	$\lvert c_1/c_0\rvert < \lvert\tan\frac{\pi\alpha}{2}\rvert$
	$c_0 \ge 0$

The characteristic function $\phi(t)$ is related to the probability density $\pi(x)$ by an inverse Fourier transformation:

$$\pi(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{d}t\ \phi(t)\mathrm{e}^{-\mathrm{i}xt}.$$

This means that the empirical limiting function for the rescaled output of Alg. 1.29 (`direct-gamma`) shown in Fig. 1.46 can be described by eqn (1.79) with $\alpha = 1.25$. We need only to clarify the relationship between the Fourier parameters $\{c_0, c_1\}$ and the real-space parameters $\{A_+, A_-\}$. This point will now be investigated by means of an asymptotic expansion of a Fourier integral.

Because $\alpha > 1$, we may multiply the integrand by a small subdominant term $\mathrm{e}^{-\epsilon|t|}$. This leaves us with a function

$$\begin{aligned}\pi_\epsilon(x) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{d}t\ \mathrm{e}^{-\mathrm{i}xt}\exp\left[-\left(c_0 + \mathrm{i}c_1\frac{t}{|t|}\right)|t|^{\alpha}\right]\mathrm{e}^{-\epsilon|t|} \\ &= \underbrace{\int_{-\infty}^{0} \mathrm{d}t\ \ldots}_{\pi_\epsilon^-(x)} + \underbrace{\int_{0}^{\infty} \mathrm{d}t\ \ldots.}_{\pi_\epsilon^+(x)}\end{aligned} \tag{1.81}$$

We now look in more detail at $\pi_\epsilon^+(x)$, where

$$\pi_\epsilon^+(x) = \frac{1}{2\pi}\int_0^\infty \mathrm{d}t\ \mathrm{e}^{-\mathrm{i}xt}\underbrace{\exp\left[-(c_0+\mathrm{i}c_1)t^\alpha\right]}_{1-(c_0+\mathrm{i}c_1)t^\alpha+\cdots}\mathrm{e}^{-\epsilon t}. \tag{1.82}$$

After the indicated expansion of the exponential, we may evaluate the integral[9] for nonzero ϵ:

$$\pi_\epsilon^+(x) \simeq \frac{1}{2\pi}\sum_{m=0}^\infty \frac{(-1)^m}{m!}\frac{\Gamma(m\alpha+1)}{(x^2+\epsilon^2)^{(m\alpha+1)/2}}(c_0+\mathrm{i}c_1)^m \times \exp\left[-\mathrm{i}\ (m\alpha+1)\arctan\frac{x}{\epsilon}\right]. \tag{1.83}$$

Table 1.19 The function $\pi_\epsilon^+(x=10)$ for $c_0=1$ and $c_1=0$ and its approximations in eqn (1.83). All function values are multiplied by 10^3.

		m-values included		
ϵ	π_ϵ^+	0	0, 1	0, 1, 2
0	0.99	0	0.94	0.99
0.1	1.16	0.16	1.10	1.16
0.2	1.32	0.32	1.27	1.33

For concreteness, we may evaluate this asymptotic expansion term by term for fixed x and ϵ and compare it with the numerical evaluation of eqn (1.82) (see Table 1.19). We can neglect the imaginary part of $\pi(x)$ (see Table 1.18) and, in the limit $\epsilon \to 0$, the $m=0$ term vanishes for $x \neq 0$, and the $m=1$ term dominates. This allows us to drop terms with $m=2$ and higher: the small-t behavior of the characteristic function $\phi(t)$ governs the large-$|x|$ behavior of $\pi(x)$.

We now collect the real parts in eqn (1.83), using the decomposition of the exponential function into sines and cosines ($\mathrm{e}^{-\mathrm{i}xt} = \cos(xt) - \mathrm{i}\ \sin(xt)$), the asymptotic behavior of $\arctan x$ ($\lim_{x\to\pm\infty}\arctan x = \pm\pi/2$), and the relations between sines and cosines ($\cos(x+\pi/2) = -\sin x$ and $\sin(x+\pi/2)=\cos x$). We then find

$$\pi^+(x) \simeq \frac{\Gamma(1+\alpha)}{2\pi x^{1+\alpha}}\left(c_0\sin\frac{\pi\alpha}{2} - c_1\cos\frac{\pi\alpha}{2}\right) \text{ for } x\to\infty.$$

The other term gives the same contribution, i.e. $\pi^-(x)=\pi^+(x)$ for large x. We find, analogously,

$$\pi^+(x) \simeq \frac{\Gamma(1+\alpha)}{2\pi|x|^{1+\alpha}}\left(c_0\sin\frac{\pi\alpha}{2} + c_1\cos\frac{\pi\alpha}{2}\right) \text{ for } x\to-\infty, \tag{1.84}$$

with again the same result for π^-.

The calculation from eqn (1.81) to eqn (1.84) shows that any characteristic function $\phi(t)$ whose expansion starts as in eqn (1.79) for small t belongs to a probability distribution $\pi(x)$ with an asymptotic behavior

$$\pi(x) \simeq \begin{cases} A_+/x^{1+\alpha} & \text{for } x\to\infty \\ A_-/|x|^{1+\alpha} & \text{for } x\to-\infty \end{cases}, \tag{1.85}$$

where

$$A_\pm = \frac{\Gamma(1+\alpha)}{\pi}\left(c_0\sin\frac{\pi\alpha}{2} \pm c_1\cos\frac{\pi\alpha}{2}\right). \tag{1.86}$$

[9] We use

$$\int_0^\infty \mathrm{d}t\ t^\alpha \mathrm{e}^{-\epsilon t}\begin{matrix}\sin\\ \cos\end{matrix} xt = \frac{\Gamma(\alpha+1)}{(\epsilon^2+x^2)^{(\alpha+1)/2}}\begin{matrix}\sin\\ \cos\end{matrix}\left[(\alpha+1)\arctan\frac{x}{\epsilon}\right].$$

This integral is closely related to the gamma function.

Equations (1.85) and (1.86) relate the asymptotic behavior of $\pi(x)$ to the characteristic function of the corresponding stable distribution. We can test the formulas for the characteristic distribution of the γ-integral, where we find, from eqn (1.86),

$$\alpha = 1.25\ (\gamma = -0.8): \quad \begin{bmatrix} A_+ = 1.25 \\ A_- = 0 \end{bmatrix} \Leftrightarrow \begin{bmatrix} c_0 = 1.8758 \\ c_1 = 4.5286 \end{bmatrix}. \tag{1.87}$$

This gives the following as a limit function for the rescaled sum of random variables obtained from Alg. 1.29 (`direct-gamma`), for $-1 < \gamma < -0.5$:

$$\pi(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathrm{d}t\ \exp\left[-\mathrm{i}xt - \left(c_0 + \mathrm{i}c_1 \frac{t}{|t|} \right) |t|^{\alpha} \right], \tag{1.88}$$

with parameters from eqn (1.87). Equation (1.88), an inverse Fourier transform, can be evaluated by straightforward Riemann integration, using suitable finite limits for the t-integration. The agreement between all of our three approaches to the Lévy distribution is excellent (see Fig. 1.48, and compare it with the rescaled averages in Fig. 1.46).

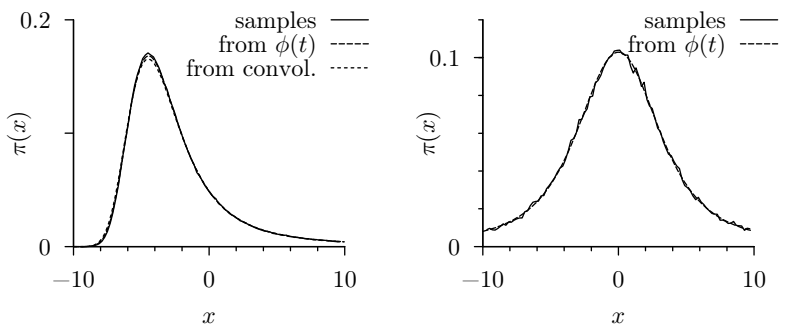

Fig. 1.48 Stable distributions for $\alpha = 1.25$. *Left*: one-sided case ($A_- = 0$, $A_+ = 1.25$). *Right*: symmetric case ($A_- = A_+ = 1.25$).

For a second illustration of Lévy statistics, we symmetrize output of Alg. 1.29 (`direct-gamma`) and take $\Upsilon \leftarrow \texttt{ran}\left(-\frac{1}{2}, \frac{1}{2}\right)$. Samples x_i are generated as follows:

$$x_i = \begin{cases} \Upsilon^{\gamma} & \text{if } \Upsilon \geq 0 \\ -|\Upsilon|^{\gamma} & \text{if } \Upsilon < 0 \end{cases}.$$

The symmetric distribution $\pi(x_i)$ has the same asymptotic behavior for $x \to +\infty$ as the γ-integral, so that $A_+ = A_- = 1.25$. Equation (1.86) once more allows us to compute the parameters $\{c_0, c_1\}$ from $\{A_+, A_-\}$:

$$\alpha = 1.25\ (\gamma = -0.8): \quad \begin{bmatrix} A_+ = 1.25 \\ A_- = 1.25 \end{bmatrix} \Leftrightarrow \begin{bmatrix} c_0 = 3.7516 \\ c_1 = 0 \end{bmatrix}.$$

The parameters $\{\alpha, c_0, c_1\}$ can again be entered into the inverse Fourier transform of eqn (1.88), and the result can be compared with the rescaled simulation data (see Fig. 1.48).

In conclusion, we have progressed in this subsection from a naive case study of Alg. 1.29 (`direct-gamma`) to a systematic analysis of distributions with a finite mean and infinite variance. The mathematical law and order in these very erratic random processes is expressed through the scaling in eqn (1.77), which fixes the speed of convergence towards the mean value, and through the parameters $\{\alpha, A_+, A_-\}$ or, equivalently, $\{\alpha, c_0, c_1\}$, which are related to each other by eqn (1.86).

1.4.5 Minimum number of samples

In this chapter, we have witnessed Monte Carlo calculations from all walks of life, and with vastly different convergence properties. With one exception, when we used Alg. 1.31 (`markov-zeta`) to sniff out singularities, all these programs attempted to estimate an observable mean $\langle \mathcal{O} \rangle$ from a sample average. The success could be outstanding, most pronouncedly in the simulation related to the crazy cobbler's needle, where a single sample gave complete information about the mean number of hits. The situation was least glamorous in the case of the γ-integral. There, typical sample averages remained different from the mean of the observable even after millions of iterations. Practical simulations are somewhere in between these extremes, even if the variance is finite: it all depends on how different the original distribution is from a Gaussian, how asymmetric it is, and how quickly it falls to zero for large absolute values of its arguments. In a quite difficult case, we saw how the situation could be saved through importance sampling.

In many problems in physics, importance sampling is incredibly efficient. This technique counterbalances the inherent slowness of Markov-chain methods, which are often the only ones available and which, in billions of iterations and months of computer time, generate only a handful of independent samples. In most cases, though, this relatively small amount of data allows one to make firm statements about the physical properties of the system studied. We can thus often get away with just a few independent samples: this is the main reason why the equilibrium Monte Carlo method is so firmly established in statistical physics.

Exercises

(Section 1.1)

(1.1) Implement Alg. 1.1 (`direct-pi`). Run it twenty times each for $N = 10, 100, \ldots, 1 \times 10^8$. Convince yourself that N_{hits}/N converges towards $\pi/4$. Estimate the mean square deviation $\left\langle (N_{\text{hits}}/N - \frac{\pi}{4})^2 \right\rangle$ from the runs and plot it as a function of N. How does the mean square deviation scale with N?

(1.2) Implement and run Alg. 1.2 (`markov-pi`), starting from the clubhouse. Convince yourself, choosing a throwing range $\delta = 0.3$, that N_{hits}/N again converges towards $\pi/4$. Next, study the precision obtained as a function of δ and check the validity of the one-half rule. Plot the mean square deviation $\left\langle (N_{\text{hits}}/N - \frac{\pi}{4})^2 \right\rangle$, for fixed large N, as a function of δ in the range $\delta \in [0, 3]$. Plot the rejection rate of this algorithm as a function of δ. Which value of the rejection rate yields the highest precision?

(1.3) Find out whether your programming language allows you to check for the presence of a file. If so, improve handling of initial conditions in Alg. 1.2 (`markov-pi`) by including the following code fragment:

```
...
if (∃ initfile) then
  { input {x, y} (from initfile)
else
  { {x, y} ← {1, 1} (legal initial condition)
...
```

Between runs of the modified program, the output should be transferred to *initfile*, to get new initial conditions. This method for using initial conditions can be adapted to many Markov-chain programs in this book.

(1.4) Implement Alg. 1.6 (`markov-discrete-pebble`), using a subroutine for the numbering scheme and neighbor table. The algorithm should run on arbitrary rectangular pads without modification of the main program. Check that, during long runs, all sites of the pebble game are visited equally often.

(1.5) For the 3×3 pebble game, find a rejection-free local Monte Carlo algorithm (only moving up, down, left or right). If you do not succeed for the 3×3 system, consider $n \times m$ pebble games.

(1.6) Implement Alg. 1.4 (`direct-needle`), and Alg. 1.5 (`direct-needle(patch)`). Modify both programs to allow you to simulate needles that are longer than the floorboards are wide ($a > b$). Check that the program indeed computes the number π. Which precision can be reached with this program?

(1.7) Check by an analytic calculation that the relationship between the mean number of hits and the length of the needle (eqn (1.8)) is valid for round (cobbler's) needles of any size. Analytically calculate the function $N_{\text{hits}}(x, \phi)$ for semicircular needles, that is, for cobblers' needles broken into two equal pieces.

(1.8) Determine all eigenvalues and eigenvectors of the transfer matrix of the 3×3 pebble game in eqn (1.14) (use a standard linear algebra routine). Analogously compute the eigenvalues of $n \times n$ pebble games. How does the correlation time Δ_i depend on the pad size n?

(Section 1.2)

(1.9) Sample permutations using Alg. 1.11 (`ran-perm`) and check that this algorithm generates all 120 permutations of five elements equally often. Determine the cycle representation of each permutation that is generated. For permutations of K elements, determine the histogram of the probability for being in a cycle of length l (see Subsection 4.2.2). Consider an alternative algorithm for generating random permutations of K elements. Sort random numbers $\{x_1, \ldots, x_K\} = \{\texttt{ran}\,(0,1), \ldots, \texttt{ran}\,(0,1)\}$ in ascending order $\{x_{P_1} < \cdots < x_{P_K}\}$. Show that $\{P_1, \ldots, P_K\}$ is a random permutation.

(1.10) Consider the following algorithm which combines transformation and rejection techniques:

```
1  x ← −log ran (0, 1)
   ϒ ← exp [−(x−1)²/2]
   if (ran (0, 1) ≥ ϒ) goto 1 (reject sample)
   output x
```

Analytically calculate the distribution function $\pi(x)$ sampled by this program, and its rejection rate. Implement the algorithm and generate a histogram to check your answers.

(1.11) Consider the following code fragment, which is part of Alg. 1.19 (`gauss(patch)`):

$x \leftarrow \texttt{ran}\,(-1,1)$
$y \leftarrow \texttt{ran}\,(-1,1)$
$\Upsilon \leftarrow x^2 + y^2$
output Υ

Compute the distribution function $\pi(\Upsilon)$ using elementary geometric considerations. Is it true that Υ is uniformly distributed in the interval $\Upsilon \in [0,1]$? Implement the algorithm and generate a histogram to check your answers.

(1.12) Implement both the naive Alg. 1.17 (`naive-gauss`) with arbitrary K and the Box–Muller algorithm, Alg. 1.18 (`gauss`). For which value of K can you still detect statistically significant differences between the two programs?

(1.13) Generate uniformly distributed vectors $\{x_1, \ldots, x_d\}$ inside a d-dimensional unit sphere. Next, incorporate the following code fragment:

```
...
x_{d+1} ← ran (−1, 1)
if ( Σ_{k=1}^{d+1} x_k^2 > 1) then
  { output "reject"
...
```

Show that the acceptance rate of the modified program yields the ratio of unit-sphere volumes in $(d+1)$ and in d dimensions. Determine $V_{252}(1)/V_{250}(1)$, and compare with eqn (1.39).

(1.14) Sample random vectors $\{x_1, \ldots, x_d\}$ on the surface of the d-dimensional unit sphere, using Alg. 1.22 (`direct-surface`). Compute histograms of the variable $I_{12} = x_1^2 + x_2^2$. Discuss the special case of four dimensions ($d = 4$). Determine the distribution $\pi(I_{12})$ analytically.

(1.15) Generate three-dimensional orthonormal coordinate systems with axes $\{\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z\}$ randomly oriented in space, using Alg. 1.22 (`direct-surface`). Test your program by computing the average scalar products $\langle(\hat{\mathbf{e}}_x \cdot \hat{\mathbf{e}}'_x)\rangle$, $\langle(\hat{\mathbf{e}}_y \cdot \hat{\mathbf{e}}'_y)\rangle$, and $\langle(\hat{\mathbf{e}}_z \cdot \hat{\mathbf{e}}'_z)\rangle$ for pairs of random coordinate systems.

(1.16) Implement Alg. 1.13 (`reject-finite`) for $K = 10\,000$ and probabilities $\pi_k = 1/k^\alpha$, where $1 < \alpha < 2$. Implement Alg. 1.14 (`tower-sample`) for the same problem. Compare the sampling efficiencies. NB: Do not recompute $\pi_{\max}$ for each sample in the rejection method; avoid recomputing $\{\Pi_0, \ldots, \Pi_K\}$ for each sample in the tower-sampling algorithm.

(1.17) Use a sample transformation to derive how to generate random numbers ϕ distributed as $\pi(\phi) = \frac{1}{2}\sin\phi$ for $\phi \in [0,\pi]$. Likewise, determine the distribution function $\pi(x)$ for $x = \cos[\texttt{ran}\,(0,\pi/2)]$. Test your answers with histograms.

(Section 1.3)

(1.18) Implement Alg. 1.25 (`binomial-convolution`). Compare the probability distribution $\pi(N_{\text{hits}})$ for $N = 1000$, with histograms generated from many runs of Alg. 1.1 (`direct-pi`). Plot the probability distributions for the rescaled variables $x = N_{\text{hits}}/N$ and $\tilde{x} = (x - \pi/4)/\sigma$, where $\sigma^2 = (\pi/4)(1-\pi/4)$.

(1.19) Modify Alg. 1.26 (`ran01-convolution`) to allow you to handle more general probability distributions, which are nonzero on an arbitrary interval $x \in [a,b]$. Follow the convergence of various distributions with zero mean their convergence towards a Gaussian.

(1.20) Implement Alg. 1.28 (`data-bunch`). Test it for a single, very long, simulation of Alg. 1.2 (`markov-pi`) with throwing ranges $\delta \in \{0.03, 0.1, 0.3\}$. Test it also for output of Alg. 1.6 (`markov-discrete-pebble`) (compute the probability to be at site 1). If possible, compare with the correlation times for the $n \times n$ pebble game obtained from the second largest eigenvalue of the transfer matrix (see Exerc. 1.8).

(Section 1.4)

(1.21) Determine the mean value of $\mathcal{O} = x^{\gamma-\zeta}$ in a simple implementation of Alg. 1.31 (`markov-zeta`) for $\zeta > -\frac{1}{2}$. Monitor the rejection rate of the algorithm as a function of the step size δ, and compute the mean square deviation of $\mathcal{O}$. Is the most precise value of $\langle\mathcal{O}\rangle$ obtained with a step size satisfying the one-half rule?

(1.22) Implement Alg. 1.29 (`direct-gamma`), subtract the mean value $1/(\gamma+1)$ for each sample, and generate histograms of the average of N samples, and also of the rescaled averages, as in Fig. 1.46.

(1.23) Implement a variant of Alg. 1.29 (`direct-gamma`), in order to sample the distribution

$$\pi(x) \propto \begin{cases} (x-a)^\gamma & \text{if } x > a \\ -c|x-a|^\gamma & \text{if } x < a \end{cases}.$$

For concreteness, determine the mean of the distribution analytically as a function of $\{a, c, \gamma\}$, and subtract it for each sample. Compute the histograms of the distribution function for the rescaled sum of random variables distributed as $\pi(x)$. Compute the parameters $\{A_\pm, c_{1,2}\}$ of the Lévy distribution as a function of $\{a, c, \gamma\}$, and compare the histograms of rescaled averages to the analytic limit distribution of eqn (1.86).

References

Aigner M., Ziegler G. M. (1992) *Proofs from THE BOOK (2nd edn)*, Springer, Berlin, Heidelberg, New York

Barbier E. (1860) Note sur le problème de l'aiguille et le jeu du joint couvert [in French], *Journal de Mathématiques Pures et Appliquées* **(2) 5**, 273–286

Gnedenko B. V., Kolmogorov A. N. (1954) *Limit Distributions for Sums of Independent Variables*, Addison-Wesley, Cambridge, MA

Krauth W., Staudacher M. (1999) Eigenvalue distributions in Yang–Mills integrals, *Physics Letters B* **453**, 253–257

Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., Teller E. (1953) Equation of state calculations by fast computing machines, *Journal of Chemical Physics* **21**, 1087–1092

Propp J. G., Wilson D. B. (1996) Exact sampling with coupled Markov chains and applications to statistical mechanics, *Random Structures & Algorithms* **9**, 223–252

Ulam S. M. (1991) *Adventures of a Mathematician*, University of California Press, Berkeley, Los Angeles, London

This page intentionally left blank