Masanori Hanada
So Matsuura

MCMC from Scratch

A Practical Introduction to Markov Chain Monte Carlo

Springer

MCMC from Scratch

Masanori Hanada · So Matsuura

MCMC from Scratch

A Practical Introduction to Markov Chain Monte Carlo

Springer

Masanori Hanada
Department of Mathematics
University of Surrey
Guildford, Surrey, UK

So Matsuura
Hiyoshi Departments of Physics, and
Research and Education Center
for Natural Sciences
Keio University
Yokohama, Kanagawa, Japan

ISBN 978-981-19-2714-0 ISBN 978-981-19-2715-7 (eBook)
https://doi.org/10.1007/978-981-19-2715-7

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Acknowledgements

The authors learned MCMC via trials and errors during various research projects, rather than from books. At the early stages of our research career, we could get instructions from many people including Hikaru Kawai and Jun Nishimura. We sincerely thank these people, without their help this book could not exist. We had a rare opportunity to have many experts around us. We hope this book will make MCMC accessible for wider audience.

A large fraction of this book is a translation of a Japanese book published from Kodansha Scientific in 2020. While we were writing the Japanese version, Hitoshi Hayami provided us with various comments that were useful to improve Sect. 6.1. Soichiro Isoyama, Kiwamu Kudoh, Hiromitsu Kobayashi, Shiro Sakai, Masaki Tezuka, Yoshimi Hanada, and Hiromasa Watanabe read the draft of the Japanese version very carefully and gave us feedback. Comments from Sinya Aoki, Guy Gur-Ari, Cammy Kramer, Enrico Rinaldi, and David Schaich were useful for deciding the contents of the book. Norio Otsuka, who was the Editor-in-Charge at Kodansha Scientific, helped us at various stages. Jack Holden read a draft of the English translation carefully and gave us valuable feedback.

M. H. would like to thank people at Brown University, especially Antal Jevicki, Cheng Peng, Mary Ann Rotondo, Marcus Spradlin, and Anastasia Volovich, for providing him with a comfortable environment at the early stage of the writing of the Japanese version. S. M. thanks his colleagues at Keio University for consistent support.

Contents

Chapter 1
Introduction

In this book, we will learn about the Markov Chain Monte Carlo (MCMC) methods. MCMC is a powerful framework that is useful when we want to integrate a complicated function or want to handle a complicated probability distribution. Historically, it has been a popular tool in physics, which is the authors' main research field. Recently, it became a common tool in statistics as well, and also in the fields where statistical methods are important, for example, machine learning and finance.

MCMC is not difficult at all, rather it is a simple technique based on a very natural idea. Of course, it is a huge mistake if you think a simple technique can be applied to only easy problems. Rather, a simple technique can have a variety of applications. Indeed, many problems in various fields such as quantum physics, Bayesian statistics, and combinatorial optimization reduce to the calculation of *probability* and *expectation values* that can be solved via MCMC. Therefore, if you have some basic knowledge of MCMC, you can write simulation code by yourself whenever you come up with a problem, regardless of the field of your interest. You can also understand a complicated simulation code somebody wrote for you.

Very unfortunately, however, it is not easy to find an introductory textbook that explains the MCMC methods from the very basic points to the practical level in plain language. Although there are a few good advanced textbooks, it is not easy for a student or engineer without strong mathematical background and programming skills to digest the materials and run simulation codes. For this reason, if you want to learn MCMC, probably you have to collect information from many sources, get advice from experts, and repeat trials and errors. Often, it is a hard task even for talented researchers in theoretical physics; believe us, we saw many examples. The entrance barrier is too high!

In this book, we will explain the basic ideas of MCMC without assuming advanced knowledge of mathematics and programming. To make the subject more accessible, we provide plenty of examples. We hope that readers will be able to write simulation code by themselves based on a proper understanding after reading this book. The best way to understand a subject is to get our hands dirty, and for that purpose, we provide

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7_1

a lot of sample code. You may think you can just use a ready-made software package, but a proper understanding of MCMC can help you understand what is going on in the black box. Furthermore, it would help you when you come up with a problem to which no ready-made package can be used. Note also that, if you learn MCMC, you will see that it is very simple and you can immediately calculate various things by writing very short simulation code. Often, it takes a longer time to understand how to use a software package than to write a program by yourself. In this book, we explain the MCMC methods step by step, without leaving a black box. The goal is to master practically useful techniques based on transparent logic.

This book is organized as follows.

- In the rest of this chapter, we will briefly explain what kinds of problems are efficiently solved via MCMC, and why it is better to use the MCMC algorithms.
- In Chap. 2, we will see a "naive" Monte Carlo algorithm that is not MCMC. We will learn the advantage of using random numbers and the Markov chain. We will also learn when and how a naive approach fails. The purpose of this chapter is to illuminate the advantages of MCMC.
- In Chap. 3, we will learn the general aspects of MCMC. There are various kinds of Markov Chain Monte Carlo algorithms, but they are called by this specific name because all of them satisfy certain conditions. By understanding such conditions, we can design a suitable algorithm depending on the actual problems we want to solve.
- Chapter 4 introduces the Metropolis algorithm. We start with the integral of a function of one variable. This very simple example contains almost all the essence of MCMC. The generalization to multivariate functions is discussed in Sect. 4.8.
- Chapter 5 contains a few other algorithms: the HMC algorithm, which has a wide variety of applications; the Gibbs-sampling algorithm, whose applications are limited compared to the HMC but extremely convenient when it does work; and the metropolis-Hastings algorithm, which is the foundation of those two algorithms. Each algorithm has its pros, cons, and pitfalls.
- In Chap. 6, several applications of the MCMC methods are shown. Among countlessly many applications, here we introduce Bayesian statistics, which is a popular topic in statistical analysis; the Ising model, whose importance in physics cannot be overstated; the traveling salesman problem, which is a typical combinatoric optimization problem; and high energy physics, which is a pursuit of the fundamental laws of nature. Each of these applications has its features that (sometimes) prevent us from using simple algorithms. In such situations, we have to design appropriate strategies flexibly, while keeping the basics of Markov Chain Monte Carlo.
- Some exercises are provided at the end of Chaps. 3, 4, and 5.
- In Appendix A, programs mentioned in the main text are explained one by one. In Appendix B, we summarized some mathematical tips used in this book, such as matrix calculations and Gaussian integral. In Appendix C, the Hamilton equation in classical mechanics is explained, which is used in the HMC algorithm. Appendix D contains supplementary comments on the Jackknife method intro-

duced in Chap. 4. In Appendix E, the conjugate-gradient method, which solves simultaneous equations iteratively, is explained.

Now let us start learning about MCMC: what is the target, and what is the strategy?

Probability and Expectation Value

Let us start with a simple example regarding probability and expectation values. Suppose you have a lottery ticket. You may hit the first prize or second prize, or you may miss it. In any event, the outcome will be unique and mutually exclusive. Let us denote such mutually exclusive events by $A_1, A_2, A_3, \cdots$, and the probability of an event A_i by $P(A_i)$. By definition, the sum of the probabilities is 1:

$$\sum_{i=1,2,\cdots} P(A_i) = P(A_1) + P(A_2) + \cdots = 1. \tag{1.1}$$

The *expectation value* of the prize money you get is obtained by multiplying the prize money (say $f(A_i)$, with $f(A_1)$=\$10,000,000 and so on) by the probability and summing over:

$$\begin{aligned}\text{The expectation value of prize money} &= \sum_{i=1,2,\cdots} f(A_i)P(A_i) \\ &= f(A_1)P(A_1) + f(A_2)P(A_2) + \cdots .\end{aligned} \tag{1.2}$$

Similar math appears when we consider the correlation between the income and characteristics or abilities of individuals. Suppose n parameters $\{x\} = x_1, x_2, \cdots, x_n$ describe factors such as math skill, networking skill, or geekiness. Let $P(M|x_1, x_2, \cdots, x_n)$ be the probability that the income is \$ M when the individual is characterized by the parameters $\{x\}$, and $P(x_1, x_2, \cdots, x_n)$ be the distribution of the abilities of the alumni from a university (e.g., more students are good at math, snobby or geeky). The latter can change depending on the education policy. Hence, to see how the education policy affects alumni's average income, one should calculate[1]

$$\begin{aligned}&\text{The expectation value of alumni's income} = \\ &\int dM \int dx_1 \cdots \int dx_n M \cdot P(M|x_1, \cdots, x_n) \cdot P(x_1, \cdots, x_n).\end{aligned} \tag{1.3}$$

[1] Money is not the most important thing in life, but we do not want to make the argument complicated here.

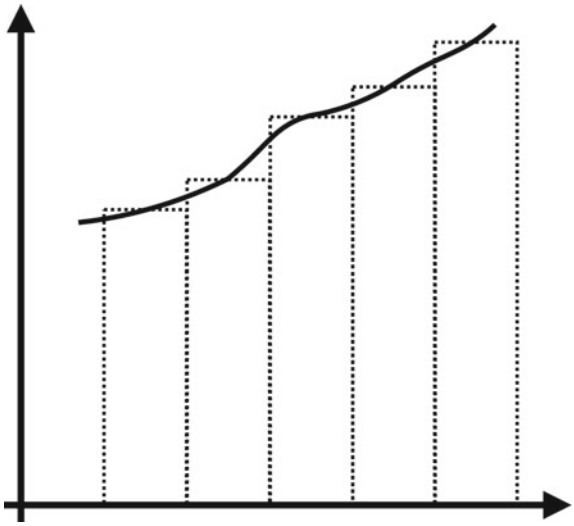

Fig. 1.1 The integral can be approximated by the sum of the areas of rectangles, by dividing the domain of integration into small pieces. This picture is for univariate integration. The generalization to multivariate integration is straightforward—modulo the curse of dimensionality!

Curse of Dimensionality and Markov Chain Monte Carlo Method

It is easy to perform integrals like Eq. (1.3) if n is 1 or 2. Just by dividing the domain of integration like in Fig. 1.1 and replacing the integral with a sum, a reasonably good approximation is obtained. By dividing the domain into smaller and smaller pieces, the approximation becomes better and better. In principle, such calculations are very easy. Even a beginner in programming would be able to code this algorithm immediately.

In practice, however, this approach does not really work. Suppose that there are n variables $x_1, \cdots, x_n$, and each of them is divided into 10 intervals. Then the domain of integration is divided into 10^n pieces, which means we need at least 10^n operations just to take the sum. 10 intervals for each may be too coarse, so you would want to have 100 intervals for each variable. Then there are $100^n = 10^{2n}$ pieces. The cost increases exponentially with n, and hence, it works only for small values of n. In 2012, the then-fastest supercomputer *Kei* could carry out 10 petaflops $= 10^{16}$ floating-point operations per second. If we tried the integration with $n = 10$ variables and neglected all the costs but the summation, it could take $100^n/10^{16} = 10^4$ seconds, which is about three hours. For $n = 12$, it could take $100^n/10^{16} = 10^8$ seconds, which is about three years. Note that this is a very optimistic estimate; in reality, we need more operations such as the evaluation of the integrand at each point, so it costs much more. As of the time of writing this book, good machines in the world are more than ten times faster than Kei, but the improvement over almost a decade is not enough to increase the value of n by 1. This problem—the exponential increase of the cost—is called *the curse of dimensionality*.

The Markov Chain Monte Carlo method enables us to do such calculations by using *random numbers*. The key idea that leads to the drastic reduction of the cost is that, *in many cases, most of the domain of integration does not contribute to the integral.*

In the example of alumni's salaries: have you ever seen anybody who is an Olympic-level athlete, extremely good at math and rumored to get the Fields Medal soon, with a perfect and likable personality loved by everybody? Probably not, such people are very rare. Also, nobody is bad at everything. Most people are more or less similar, they have good and bad parts, and most of n parameters are close to the average. Therefore, $P(x_1, x_2, \cdots, x_n)$ is negligibly small in most of the n-dimensional parameter space. Perhaps few very talented people get high income, but the expectation value does not change much even if such rare cases are ignored.[2]

Then, when the integral (1.3) is performed, a good approximation should be obtained just by summing up the contribution from the region where $P(x_1, x_2, \cdots, x_n)$ is not too small. This is the key idea of MCMC: by cutting out the "important parameter region", or "states with high probability", the computational cost can be reduced.[3]

How can we realize this idea? What is the role of random numbers? We will see the answer in the following sections.

[2] If such people founded a big company and got astronomically big income, the expectation value would be affected significantly. In MCMC, such rare cases can also be treated properly.

[3] In addition to the property that "states with high probability" are typically a small region in the domain of integration, another important property that contributes to the reduction of the computational cost is that, in many problems, such "states with high probability" give similar results regardless of the microscopic details of the states. If you know some statistical physics, you can understand it as: typical states ($\simeq$ states with high probability) cannot be distinguished by macroscopic quantities (e.g., income).

Chapter 2
What is the Monte Carlo Method?—A Simulation with Random Numbers

The Monte Carlo method is a general term for computational methods that utilize random numbers (see e.g., [1]). Usually, it specifically means a class of algorithms that are guaranteed to give the right answer if we continue the simulation long time. Let us first see a few simple examples of Monte Carlo algorithms that are different from the "Markov Chain" Monte Carlo [2, 3]. (More nontrivial applications of "naive" Monte Carlo algorithms than those mentioned in this chapter can be found, e.g., in Ref. [4].) These algorithms are very easy, but there are significant limitations. By knowing the limitations of such "naive" Monte Carlo algorithms, we can understand the advantage of the Markov Chain Monte Carlo algorithms.

2.1 Random Numbers

First of all, what are random numbers? Random numbers are a sequence of numbers that are randomly generated following a certain probability distribution $P(x)$. A typical example is numbers determined by throwing dice. In this case, integers from 1 to 6 appear with the same probability,[1] $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = \frac{1}{6}$. Suppose we threw dice n times and obtained a sequence $a_1, a_2, \ldots, a_n$. We throw dice one more time and get a_{n+1}. Can we predict a_{n+1} from $a_1, a_2, \ldots, a_n$? Not really. We cannot tell what the likely value of a_{n+1} is. We can only say any integer between 1 and 6 is equally likely. In this way, *x is determined without referring to the history*. The adjective "random" means this property. The random numbers associated with a generic probability distribution $P(x)$ can be defined in the same manner: the probability that $a_{n+1} = x$ is obtained is $P(x)$, regardless of the history $a_1, a_2, \ldots, a_n$.

[1] We assume that the dice are "fair".

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7_2

Below, we introduce two kinds of random numbers that are used very frequently. We will also explain that the "random numbers" used in numerical simulations are not literal random numbers in the mathematical sense and hence some care is needed to obtain the right answer. We will provide only minimal materials needed for understanding later sections in this book. For a more complete presentation, see e.g., Ref. [5].

2.1.1 Uniform Random Numbers

Suppose that any real number in an interval $[0, 1]$ can be obtained with the same probability. Random numbers obtained in this way are called uniform random numbers. Roughly speaking, we can imagine a gigantic die that returns integers between 1 and N with the equal probability $\frac{1}{N}$. Then, by dividing these integers by N, we can obtain discrete values $\frac{1}{N}, \frac{2}{N}, \frac{N-1}{N}, 1$ with uniform probability. By sending N to infinity, we can get uniform random numbers.

Because computers can handle only a finite number of digits, the uniform random numbers we can use on a computer are not literal real numbers. They are discrete random numbers, just like the ones obtained by throwing a gigantic die. For example, in C, there is a function called rand, which randomly returns integers from 0 to RAND_MAX. The value of RAND_MAX depends on the environment. On authors' machines, it is $2147483647 = 2^{31} - 1$.

2.1.2 Gaussian Random Numbers (Normal Distribution)

Random numbers following nontrivial probability distributions are useful for various purposes. Here, as an example, let us see the Gaussian random numbers whose distribution is proportional to the Gaussian function e^{-x^2}.

The Gaussian random numbers play important roles later in this book. Furthermore, because of the central limit theorem, which we will mention shortly, the Gaussian function appears in many places. This is a very basic example of the probability distribution and is a crucial piece when we introduce the Hybrid Monte Carlo (HMC) algorithm in Sect. 5.1.

In Fig. 2.1, the Gaussian function e^{-x^2} is plotted. Because of the shape, it is often called a "bell curve". We can see that this function decays very quickly as $|x|$ becomes large. For this reason, the integral of this function between $-\infty$ and $+\infty$ is finite, though the domain of integration is infinite. The precise answer is $\int_{-\infty}^{\infty} dx e^{-x^2} = \sqrt{\pi}$. Therefore, by normalizing it as

$$P(x) = \frac{e^{-x^2}}{\sqrt{\pi}} \tag{2.1}$$

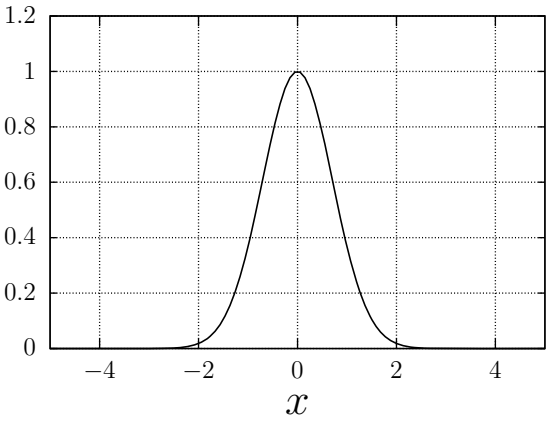

Fig. 2.1 Gaussian function e^{-x^2}

such that the integral is 1, we can interpret it as a probability distribution of x. (For a given function to be interpreted as the probability distribution, it has to be non-negative everywhere, and the integral has to be 1.)[2]

More generally, we can introduce parameters σ and μ that control the width and mean of the distribution, such that

$$P_{\sigma,\mu}(x) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}. \tag{2.2}$$

The example above (Eq. (2.1)) was $\sigma = \frac{1}{\sqrt{2}}$, $\mu = 0$. In Fig. 2.2, graphs for $\sigma = 1, 2, 4$, $\mu = 0$ are shown. Random numbers generated following this probability distribution (Gaussian distribution, or normal distribution) are called Gaussian random numbers.

We encounter the Gaussian distribution in our daily lives, often without realizing it. A common situation is associated with measurements. Suppose we did a measurement. It may be a complicated experiment, or it may be as simple as measuring the length of something by using a measuring tape. In any event, any measurement involves errors. There can be many sources of errors. For simplicity, let us assume that there are K sources, and each of them contributes randomly. Then the total error is the sum of K random numbers.

When K is sufficiently large, something very interesting happens: the "error" which arises as a sum of many random numbers follows the Gaussian distribution, regardless of the details of the sources. This can be shown mathematically and is called the central limit theorem. (See e.g., Ref. [6] for details including a proof.)

Let us see a concrete example. For simplicity, we assume that each source of the error gives a uniform random number between -0.5 and $+0.5$ and that the total

[2] Strictly speaking, this is probability density rather than probability. For infinitesimal dx, the probability that a value between x and $x + dx$ is obtained is given by $P(x)dx$.

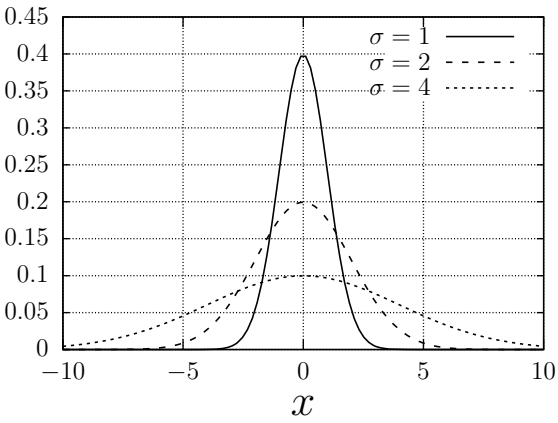

Fig. 2.2 The normalized Gaussian function (the Gaussian distribution, the normal distribution, or a bell curve) $P_{\sigma,\mu}(x) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}$ for $\sigma = 1, 2, 4$ and $\mu = 0$

error is the sum of K uniform random numbers. We generated uniform random numbers and calculated the sum. We repeated this procedure many times and plotted the distribution of the sums. (This example is essentially the same as the random walk that will be explained in Sect. 3.1.) In Fig. 2.3, we plot such distributions for $K = 1, 2, 3, 4, 5$, and 100. The sum is denoted by x, and the horizontal axis is taken to be $x/\sqrt{K}$.

The uniform distribution is obtained for $K = 1$, of course. The plot for $K = 2$ looks like a triangle. These two distributions do not look like the Gaussian distribution at all. However, for $K = 3$ and $K = 4$, the distributions look similar to a bell curve. To obtain a perfect agreement with the Gaussian distribution, we need to take the $K \to \infty$ limit. However, the distribution approaches Gaussian rather closely only with 3 or 4 sources of errors. Therefore, it is not unnatural to assume that the errors in actual measurements follow the Gaussian distribution. Hopefully, the reason that the Gaussian distribution is important in statistical analyses is clear to you by now.

Let us apply the same reasoning to other examples such as the distribution of the heights of humans. Some people are short, some people are tall. There can be many factors affecting the heights, but very roughly, let us assume that the height is determined by summing the positive and negative contributions from various factors (e.g., lifestyles such as early bird or night owl, eating habits such as meat-eater or vegetarian, genetic factors such as the heights of the parents). By identifying such positive or negative effects with the errors in the measurements, we can repeat the same argument as above. Therefore, we can expect that the distribution of the heights is Gaussian about the mean—and it is indeed the case. Our guess based on the central limit theorem does explain the reality. There are many other examples of this kind.

Note that several assumptions have to be satisfied for the central limit theorem to be applicable, and hence, we must not assume the Gaussian distribution uncritically for anything and everything. For example, it has been pointed out that the price volatility in the financial market follows a power law (something like x^{-p}), and if the log-normal distribution, which is a relative of Gaussian distribution, is used, the

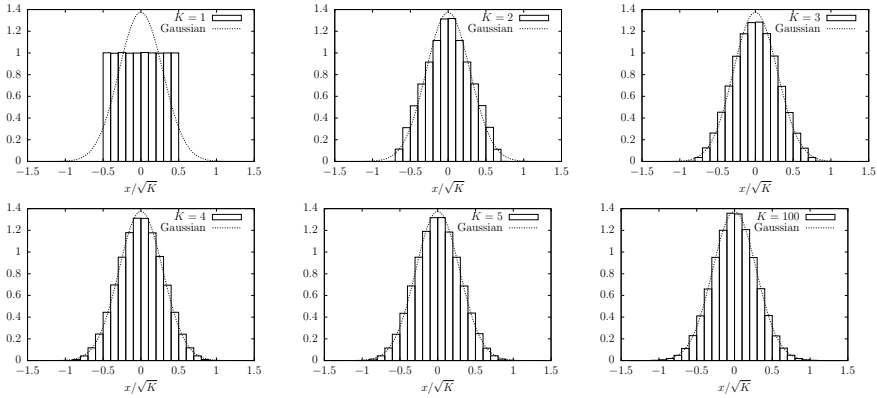

Fig. 2.3 x is defined as a sum of K uniform random numbers in $[-0.5, +0.5]$. The distribution of x is plotted by taking the horizontal axis to be $x/\sqrt{K}$. 1,000,000 samples are used for each K. The dashed line is the Gaussian distribution whose width is determined by fitting the distribution at $K = 100$. We can see the convergence to the Gaussian distribution as K becomes large. The width σ is proportional to $\sqrt{K}$

risk of large price fluctuation can be underestimated. The Gaussian distribution is a popular option because it is mathematically simple, but it is dangerous to use it when it is not justified empirically or theoretically.

2.1.3 Random Numbers Versus Pseudorandom Numbers

We cannot throw dice every time we need a random number, because many random numbers are needed for a numerical simulation. In practice, we have to generate pseudorandom numbers which look almost random and use them. It is important to understand the difference between pseudorandom numbers and actual random numbers, although it is unlikely that the difference leads to troubles at the level of simulations discussed in this book.

As a concrete example, let us see the linear congruential generator. Firstly, for natural numbers a, b, and M, we define a recurrence relation

$$x_{n+1} = ax_n + b \pmod{M}.$$

On the right-hand side, "mod M" means the remainder after the division by M. From any initial value x_0 between 0 and $M - 1$, we can make a sequence of numbers $\{x_n\}$ that looks random. This is the linear congruential generator.

This algorithm is very simple and convenient. However, the sequence $x_0, x_1, x_2, \ldots$ is not random in the mathematical sense, and hence, some care is needed when we use it for simulations. To understand that it is not random in the literal sense, note

that the same sequence is obtained from the same initial value x_0. Therefore, once $x_m = x_0$ holds for certain m, after that point the same sequence $x_0, x_1, x_2, \ldots, x_m$ is repeated. Namely, this sequence has a periodicity, and the period is at most M. It is also known that, depending on a choice of the recurrence relation and the initial value x_0, the distribution of x's can be biased.

In this sense, the sequence of numbers obtained from the linear congruential generator is only pseudorandom, which looks almost random but not exactly random. The same applies to any algorithm to generate random numbers (unless you use a quantum computer; see e.g., Ref. [7]). Indeed, there is a famous quote by John von Neumann [8]: "*Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number—there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.*"

To create pseudorandom numbers, something like the initial value x_0—the *seed* of the sequence—is specified, and the same "random numbers" are generated from the same seed. The sequence is periodic, and depending on the algorithm, there may be a bias in the distribution. This is the reason that they are called "pseudo"-random numbers.

If we use a pseudorandom number generator without understanding its features, we may get stuck in pitfalls. For example, for calculations that consume a lot of random numbers, we should not use a generator that has a short period. The linear congruential generator is often used as a default random number generator on personal computers. In the authors' personal computers, the period is at most $2^{31} - 1$. If we write a program that uses a lot of random numbers, the periodicity may cause trouble. Another, more common problem is to repeat the same sequence of pseudorandom numbers multiple times, due to the error in the seed setting. Then, as demonstrated in Sect. 4.7.3, a wrong answer may be obtained.

The Mersenne twister [9], which is regarded as a very good random number generator, has a very long period $2^{19937} - 1$. For any practical purposes, this value is large enough.

2.2 Integration Utilizing Uniform Random Numbers

Next, let us perform simple calculations by using random numbers. We use a rather naive kind of Monte Carlo method which is different from the Markov Chain Monte Carlo. We provide you with sample codes, so please actually run them on your computer. Then you can get a better sense of the meaning of "calculating by using random numbers", and you can fully comprehend the limitations of a naive method.

2.2.1 Calculation of π with Uniform Random Numbers

As a starter, let us consider a simple example for which a naive Monte Carlo method works efficiently: we calculate the area of the gray fan-shaped region in Fig. 2.4 by using random numbers. This region is a quarter of a unit disk, so the area has to be $\frac{\pi}{4}$. Therefore, this exercise is essentially a determination of the numerical value of π via Monte Carlo simulation.

A standard method is to divide the entire region into meshes like in the left panel of Fig. 2.4 and count the number of cells overlapping with the gray region. This method leads to a reasonable approximation that becomes exact when the mesh becomes infinitely fine.

A calculation via Monte Carlo is as follows. Firstly, we generate a pair of uniform random numbers (x, y) between 0 and 1. If $x^2 + y^2 < 1$, a point (x, y) is in the fan-shaped region. In Fig. 2.4, the area of the entire region $(0 \leq x \leq 1, 0 \leq y \leq 1)$ is 1, and the area of the fan-shaped region is $\frac{\pi}{4}$. Therefore, if we generate many pairs of random numbers (x, y), the probability that $x^2 + y^2 < 1$ holds (i.e., the point (x, y) falls into the fan-shaped region) should eventually converge to $\frac{\pi}{4}$.

Let us see a sample code written in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void){
  int niter=1000; //Specify the number of samples.
  srand((unsigned)time(NULL)); //Set the seed of random
  number generator.

  int n_in=0;   //Initialize the counter.
  /*************/
  /* Main loop */
  /*************/
  for(int iter=1;iter<niter+1;iter++){

    double x = (double)rand()/RAND_MAX;
    double y = (double)rand()/RAND_MAX;
    //Generate random numbers x,y in [0,1].

    if(x*x+y*y < 1e0)   //If x^2+y^2<1....
      n_in=n_in+1;   //Add 1 to n_in.
    printf(""%d   %.10f\n",iter,(double)n_in/iter);}
}
```

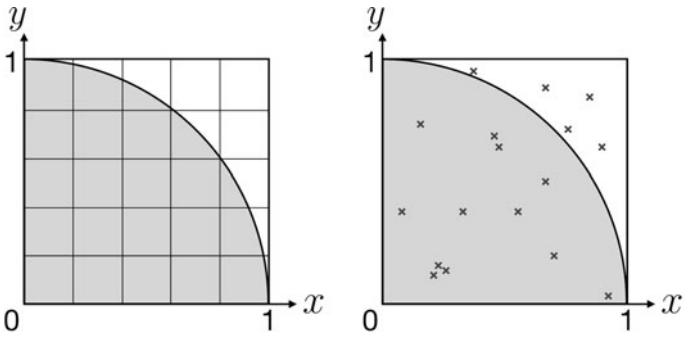

Fig. 2.4 We want to calculate the area of the fan-shaped region $x^2 + y^2 < 1$ (gray region). [Left] Divide the entire region into meshes and count the number of cells overlapping with the gray region. [Right] Generate random numbers (x, y) many times, and count how many times they fall into the gray region

Firstly, in order to use a random number generator `rand`, we included `stdlib.h`. Next, we set the seed of the random numbers by

```
srand((unsigned)time(NULL));
```

Here, to avoid using the same sequence of random numbers every time, we use the system time for the seed. The variable `n_in` is a counter for the number of points in the fan-shaped region.

The following part, denoted as "main loop", is the main part of the code that iterates the Monte Carlo steps. The variable `niter` means the number of iterations. By

```
double x = (double)rand()/RAND_MAX;
double y = (double)rand()/RAND_MAX;
```

the uniform random numbers x and y in the interval [0, 1] are generated. (Note that `rand()` returns an integer between 0 and `RAND_MAX`.) Then, in the `if` statement, 1 is added to `n_in` if $x^2 + y^2 < 1$ holds. The the probability that $x^2 + y^2 < 1$ was satisfied is `(double)n_in/iter`.

In Fig. 2.5, the probability that $x^2 + y^2 < 1$ was satisfied is shown. The horizontal axis is the number of trials K, which is `iter` in the code. We generated 100 different sequences of random numbers and plotted 5 typical results. The error bar is the standard deviation obtained from 100 sequences. We can see a gradual convergence to the analytic value $\frac{\pi}{4}$. From a theoretical consideration, we expect that the error decreases as $\frac{1}{\sqrt{K}}$.[3] To confirm such a K-dependence, in the right panel of Fig. 2.5, we plotted the average of 100 sequences and the standard deviation by taking $\frac{1}{\sqrt{K}}$ as the horizontal axis. As expected, the standard deviation shrinks, namely the deviation

[3] The reason can be understood from the property of the random walk, which is explained in Sect. 3.1. It is a good exercise, please try.

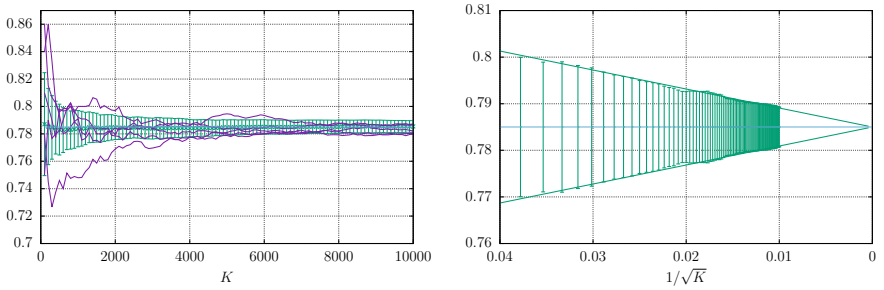

Fig. 2.5 [Left] The probability that $x^2 + y^2 < 1$ was satisfied. The horizontal axis is the number of pairs of random numbers (x, y) used for the calculation. The error bar shown in green is the variance obtained from 100 different sequences of random numbers, obtained by using different seeds. The purple lines are the plots of typical results. We can see the convergence to $\frac{\pi}{4} \simeq 0.785$. [Right] The behavior of the error, as a function of $1/\sqrt{K}$. We can see that the error is proportional to $1/\sqrt{K}$

from the analytic value $\frac{\pi}{4}$ decreases, as $\frac{1}{\sqrt{K}}$. In this way, we could see that a naive Monte Carlo method works well for this simple task.

2.2.2 *Integral with Uniform Random Numbers*

Next let us consider an integral $\int_a^b dx f(x)$. If we literally follow the definition of the integral, we would divide the domain of integration into small pieces as in Fig. 1.1 and approximate the integral by the sum of the areas of the rectangles. Instead, if we use random numbers, we can calculate the integral as follows.

Let x be a uniform random number between a and b.[4] If we randomly pick up only one x and calculate $f(x)$, we just get a random value. But what if we use a lot of random numbers? If K is sufficiently large, K random numbers $x^{(1)}, x^{(2)}, ..., x^{(K)}$ should be uniformly distributed between a and b. Then the average $\frac{1}{K}\sum_{k=1}^{K} f(x^{(k)})$ is a good approximation of $\frac{1}{b-a}\int_a^b dx f(x)$. In particular, in the limit $K \to \infty$, the exact value should be obtained. Namely,

$$\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} f(x^{(k)}) = \frac{1}{b-a}\int_a^b dx f(x). \tag{2.3}$$

As an example, let us consider $a = 0$, $b = 1$, $f(x) = \sqrt{1-x^2}$. This is the same as the previous example—the area of the fan-shaped region in Fig. 2.4—and hence, we should get $\frac{\pi}{4}$. The result is shown in Fig. 2.6. Just as in Fig. 2.5, the purple lines

[4] It can be obtained from the uniform random number between 0 and 1, which we denote by x', as $x = a + (b-a)x'$.

in the left panels are the results of 5 typical simulations, and the error bar (green) is the standard deviation obtained from 100 different sequences of random numbers. In the right panel, the average value of 100 sequences and the standard deviations are plotted by taking the horizontal axis to be $\frac{1}{\sqrt{K}}$, so that the large-K behavior of the error bars becomes clearer. We can see that the answer obtained from each sequence converges to the analytic value $\frac{\pi}{4}$, and the error is proportional to $\frac{1}{\sqrt{K}}$. In this case, again, a naive Monte Carlo is working well.

Let us see a sample code written in C. It is almost identical to the previous one:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int main(void){
  int niter=1000;  //Specify the number of samples.
  srand((unsigned)time(NULL)); //Set the seed of random
  number generator.

  double sum_y=0e0;
  /*************/
  /* Main loop */
  /*************/
  for(int iter=1;iter<niter+1;iter++){

    double x = (double)rand()/RAND_MAX;
    //Generate random numbers x in [0,1].
    double y=sqrt(1e0-x*x);
    sum_y=sum_y+y;
    printf(""%d   %.10f\n",iter,sum_y/iter);}  //Output
    the average of y.
}
```

Uniform random number x in the interval [0, 1] is generated, then $y = \sqrt{1 - x^2}$ is calculated, the sum of y's is saved as `sum_y`, and by dividing it by the number of trials `iter` the average is obtained.

2.2.3 *The Importance of the Importance Sampling*

So far, we have seen a few examples that can be solved efficiently via a naive Monte Carlo method. Next, let us see examples to which a naive Monte Carlo is not effective. The simplest example is the Gaussian function $\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$. This function is obtained

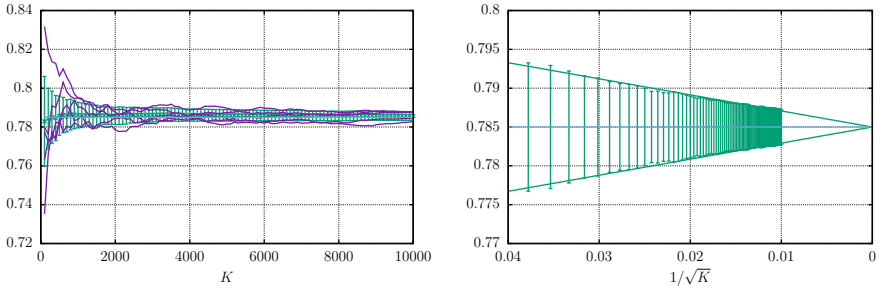

Fig. 2.6 The integral $\int_0^1 dx\sqrt{1-x^2}$ is estimated by generating the uniform random numbers x in [0, 1] and taking the average of $\sqrt{1-x^2}$. [Left] The same calculations are performed 100 times by using different sequences of random numbers, and the error bar (standard deviation) is estimated. Purple lines are 5 typical runs. The convergence to $\frac{\pi}{4} \simeq 0.785$ can be seen. [Right] The error bar versus $1/\sqrt{K}$. We can see that the error is proportional to $1/\sqrt{K}$

by setting $\sigma = 1$ and $\mu = 0$ in (2.2). It is normalized such that the integral from $-\infty$ to $+\infty$ is 1.

Naive Monte Carlo is Not Efficient

Let us integrate this function between $-a$ and a. By generating uniform random number in the interval $[-a, a]$, the average of $\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$ in this range can be estimated. By multiplying $2a$ to the average, the value of the integral $\int_{-a}^{a} \frac{dx}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$ is obtained.

In Fig. 2.7, the values obtained in this way are shown for $a = 2, 10, 100, 1000, 10000$. In the left panel, the horizontal axis K is the number of random numbers used for the calculation. In the right panel, the errors estimated from 100 different sequences of random numbers are shown. The horizontal axis is taken to be $\frac{1}{\sqrt{K}}$. What can we see from these plots?

Firstly, for any a, the average converges if we wait patiently. The value obtained in this way is guaranteed to be the right one. This is the same as the previous examples. However, as we can easily see, especially from the right panel, the convergence becomes slower as a becomes larger. Furthermore, for $a = 1000$ and 10000, the lines are not smooth and there are many jumps. Why?

To understand the reason, let us consider the case of $a = 10000$. From Fig. 2.2, we can see that the Gaussian function approaches zero very quickly as x becomes large. For this reason, unless x is small, the contribution to the integral is almost negligible. Indeed, $\int_{-2}^{2} \frac{dx}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \simeq 0.95$, $\int_{-3}^{3} \frac{dx}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \simeq 0.997$, $\int_{-4}^{4} \frac{dx}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \simeq 0.9999$, and $\int_{-5}^{5} \frac{dx}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \simeq 0.999999$. On the other hand, if we generate uniform random numbers between -10000 and $+10000$, the probability that $|x| > 2$ is obtained is 99.98%. Therefore, among K samples, more than 99.9% have almost no contribution

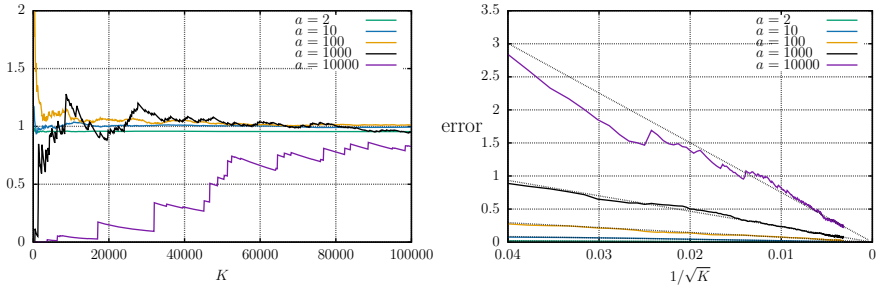

Fig. 2.7 The integral $\int_{-a}^{a} \frac{dx}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ is estimated by using uniform random number in the interval $[-a, a]$. [Left] The average values versus the number of random numbers used for the estimate are denoted by K. [Right] The error is estimated from 100 different sequences of random numbers and plotted against $1/\sqrt{K}$. The dotted lines are the fit results in the large-K region, with the ansatz error $\propto 1/\sqrt{K}$

to the integral. In other words, effectively we only have $\frac{K}{10000}$ samples or so. More than 99.9% of the resources are wasted on meaningless computation! The lines are not smooth because, only once in several thousand or tens of thousands of times, a sufficiently small x appears and gives a large contribution to the integral that looks like a sudden jump.

Even if we waste more than 99.9% of the resources, it will not take a long time for such a simple calculation, so we may not have to worry too much. However, when there are many integration variables, we do have to worry.

As a simple example, let us consider the integral of the multivariate Gaussian function $f(x_1, \ldots, x_n) = e^{-\frac{x_1^2+\cdots x_n^2}{2}}$ at $0 \le r \le 10000$ ($r = \sqrt{x_1^2 + \cdots + x_n^2}$). Even when the number of variables n is large, we can repeat the same procedure: generate $(x_1, \ldots, x_n)$ many times in a uniform and random manner, calculate $f(x_1, \ldots, x_n)$, take the average, and multiply the volume of the domain of integration. Essentially, this is a higher dimensional version of Fig. 1.1. Just as in the case of the univariate integration, the large-x region does not contribute much to the integral, and the resources used for the large-x region are wasted. As the dimension becomes higher, such an irrelevant region occupies a larger and larger fraction of the total domain of the integration. For $n = 1$ and $n = 10$, the probabilities that r is larger than 2 is 99.98% and 99.999996%, respectively. For $n = 100$, the probability that $9000 < r < 10000$ is 99.997%. Needless to say, this region does not contribute to the integral at all. If the waste is this much, the time needed for the convergence to the right value is very long, and hence, a naive Monte Carlo approach does not work.

Turning Difficulty to Advantage: Importance Sampling

Let us slightly change the viewpoint. That *most of the domain of the integration does not contribute to the integral* can be rephrased that *only a tiny fraction of the domain of*

integration contributes to the integral. Would it mean that, if we generate the random numbers only in this tiny region, we can calculate the integral very efficiently? As a simple implementation of this idea, suppose that the random numbers are generated only in a region that significantly contributes to the integral, and the rest of the domain of integration is neglected. This method works well to some extent; in the case of the multivariate Gaussian integral, by restricting the domain of integration at somewhere around $x_1^2 + \cdots x_n^2 < 3$, a reasonable approximation is achieved. In order to handle the error more precisely, we can prepare random numbers in $R_1 \leq x_1^2 + \cdots x_n^2 < R_2$; we first study $(R_1, R_2) = (0, 1)$, then $(R_1, R_2) = (1, 2)$, then $(R_1, R_2) = (2, 3)$ and so on, and we can stop the calculation when the addition of a new region does not change the answer any more.

There are two problems with this method. Firstly, when the integrand is a complicated function, it is hard to estimate the error precisely. Secondly and more seriously, when there are many variables, it is difficult to find the "important" region of the domain of integration. For one or two variables, we can plot a graph and identify the important regions by our eyeballs, but with three or more variables, it is difficult to find the important region unless the integrand is a simple function.

As we will see in later sections, the Markov Chain Monte Carlo methods resolve these problems [2]. Even if we do not know where is the important region contributing to the integral, most of the computational resources are automatically used for the calculation in the important region. This is called the *importance sampling*. (Of course, it is better if you know where the important region is.) The "irrelevant" regions are not completely ignored, their contributions are also picked up appropriately. Therefore, we are guaranteed to get a more and more accurate answer if we continue the computation longer. In the previous examples, it was not crucial to use random numbers because, whether we use random numbers or divide the integral domain into meshes, a difficult calculation was difficult and an easy calculation was easy. However, in Markov Chain Monte Carlo, the use of random numbers is crucial.

We wish we could move on to Markov Chain Monte Carlo immediately, but we need one more preparation: the notions of probability distribution and expectation value.

2.3 Expectation Value and Integral

To get straight to the point, Markov Chain Monte Carlo methods generate probability distributions. Therefore, the easiest thing to calculate via MCMC is the expectation value. To understand this point better, it is necessary to see the connection between probability distribution, expectation value, and integral. (In Sect. 4.5, we will explain how the integral itself, not the expectation value, can be calculated by using the MCMC methods.)

Suppose that the probability distribution $P(x)$ is defined at an interval $a \leq x \leq b$. Being "probability", two conditions have to be satisfied:

1. $P(x) \geq 0$ everywhere in $a \leq x \leq b$.
2. $\int_a^b dx\, P(x) = 1$.

The former means a probability is not negative and the latter means the sum of the probabilities for all possibilities is 1. The expectation value of a function $f(x)$, which is denoted by $\langle f(x) \rangle$, is defined by

$$\langle f(x) \rangle = \int_a^b dx f(x) P(x). \tag{2.4}$$

This is similar to the integral we discussed. Indeed, to calculate the expectation value by using random numbers, we just have to generate the uniform random number x between a and b, calculate $f(x)P(x)$ and take the average, and multiply $b - a$. (Alternatively, we can generate the probability distribution $P(x)$ via the "rejection sampling", by generating many uniform random numbers x and then accepting each x with probability $P(x)$. The average of $f(x)$ over such a distribution is $\langle f(x) \rangle$. This is essentially the same as taking the average of $f(x)P(x)$ over uniform random numbers. When there are many variables, however, such naive approaches are highly inefficient due to the curse of dimensionality. The MCMC method circumvents this problem and generates the probability distribution efficiently.)

The definite integral is a special case of the expectation values. To perform the integral, in Sect. 2.2.2 we used the property that the average of $f(x)$ in this interval is the definite integral divided by $b - a$. This average is the expectation value of $f(x)$ with the uniform probability distribution at $a \leq x \leq b$, that is $P(x) = \frac{1}{b-a}$:

$$\langle f(x) \rangle = \int_a^b dx f(x) P(x) = \frac{1}{b-a} \int_a^b dx f(x). \tag{2.5}$$

As a little bit more complicated example, let us integrate a bivariate function $f(x, y)$ in the fan-shaped region in Fig. 2.4 ($x > 0, y > 0, x^2 + y^2 < 1$). The only difference is that a two-dimensional domain (fan-shaped region) is used instead of the one-dimensional interval $a \leq x \leq b$. Therefore, the probability distribution $P(x, y)$ is taken to be uniform in this domain, while outside this region $P(x, y)$ is set to zero outside. Because the area of the fan-shaped region is $\frac{\pi}{4}$, in order for $\int_0^1 dx \int_0^1 dy\, P(x, y) = 1$ to be satisfied the probability distribution is taken as

$$P(x, y) = \begin{cases} \frac{4}{\pi} & x^2 + y^2 < 1 \\ 0 & x^2 + y^2 \geq 1 \end{cases} \tag{2.6}$$

By using it, the average is expressed as

$$\langle f(x, y) \rangle = \int_0^1 dx \int_0^1 dy f(x, y) P(x, y). \tag{2.7}$$

The simplest way to calculate this is to choose uniform random numbers x, y between 0 and 1, calculate $f(x, y)P(x, y)$, and take the average. The only difference from the integral $\int_{x^2+y^2<1} dxdyf(x, y)$ is whether it is divided by $\frac{\pi}{4}$ or not. As in the univariate integration $\int dxf(x)$, a naive Monte Carlo method is good enough for such a simple calculation.

Even when the shape of the domain of integration is complicated and the area cannot be calculated analytically, the numerical calculation is straightforward. As an example let us calculate the volume of a ball defined by $x^2 + y^2 + z^2 \leq 1$, which we denote by V_3 (it can analytically be obtained, though). We use a fact that $\frac{V_3}{8}$ is obtained by integrating the height $z = \sqrt{1 - x^2 - y^2}$ at $x > 0, y > 0, x^2 + y^2 < 1$. Hence,

$$\frac{V_3}{8} = \left\langle \sqrt{1 - x^2 - y^2} \right\rangle \times \frac{\pi}{4} \tag{2.8}$$

and

$$V_3 = 2\pi \times \left\langle \sqrt{1 - x^2 - y^2} \right\rangle. \tag{2.9}$$

Here is a sample code in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int main(void){
  int niter=100000;  //Specify the number of samples.
  srand((unsigned)time(NULL)); //Set the seed of random
  number generator.

  double pi=asin(1e0)*2e0;  //Calculate pi.
  double sum_z=0e0;
  int n_in=0;   //Initialize the counter.
  /*************/
  /* Main loop */
  /*************/
  for(int iter=1;iter<niter+1;iter++){
    double x = (double)rand()/RAND_MAX;
    double y = (double)rand()/RAND_MAX;
    //Generate random numbers x in [0,1].

    if(x*x+y*y < 1e0){  //If x^2+y^2<1.....
        n_in=n_in+1;  //Add 1 to the counter n_in, and
```

```
        double z=sqrt(1e0-x*x-y*y);  //calculate z.
        sum_z=sum_z+z;
      }
    printf(""%d   %.10f\n",iter,sum_z/n_in*2e0*pi);}
    //Output the expectation value.
}
```

There is almost nothing new in this code. We can define π explicitly as $\pi = 3.14159\ldots$, but just to avoid a typo we calculated it as

```
double pi=asin(1e0)*2e0;
```

A function `asin` is arcsin, which is the inverse of sin. Because $\sin\frac{\pi}{2} = 1$, we can easily obtain $\arcsin 1 = \frac{\pi}{2}$. In order to calculate the expectation value at $x^2 + y^2 < 1$, we count the number that $x^2 + y^2 < 1$ was recorded, by using a counter `n_in`. If you run this code, you can see a convergence to the exact analytic value $V_3 = \frac{4\pi}{3}$.[5]

When we try to calculate the expectation values, very often we have headache due to the curse of dimensionality. It is also common that the probability P is extremely small in most of the domain of integration. As an easy exercise to get a rough feeling regarding this problem: suppose the domain of integration is $0 \leq x_1, \ldots, x_n \leq 1$, then how big fraction of this domain satisfies $x_1^2 + \cdots + x_n^2 < 1$? The answers for $n = 2, 3, 4$, and 10 are about 79%, 52%, 31%, and 0.25%, respectively.[6]

The idea of the importance sampling can be used to reduce the computational cost also in this case. As we state repeatedly, the Markov Chain Monte Carlo methods enable us to do it.

2.4 Calculation of Expectation Value with Gaussian Random Numbers

Gaussian random numbers play important roles in various applications. Here, let us see the calculation of the expectation values by using Gaussian random numbers. As a byproduct, you can get a rough intuition into the Gaussian random numbers that will be useful later in this book. The starting point is: how can we generate the Gaussian random numbers?

---

[5] Strictly speaking, that we are dividing `sum_z` by `n_in` means we are using the rejection algorithm here. We could also multiply $\frac{4}{\pi}$ and then divide by `iter`, because `n_in`/`iter` converges to $\frac{\pi}{4}$.

[6] $x_1^2 + \cdots + x_n^2 < 1$ is satisfied with the probability $2^{-n}V_n$, where V_n is the volume of the n-dimensional disk. By using a formula $V_n = \frac{\pi^{n/2}}{\Gamma\left(n+\frac{1}{2}\right)}$, which uses the Gamma function Γ, we obtain $V_2 = \pi$, $V_3 = \frac{4\pi}{3}$, $V_4 = \frac{\pi^2}{2}$, ..., $V_{10} = \frac{\pi^5}{120}$.

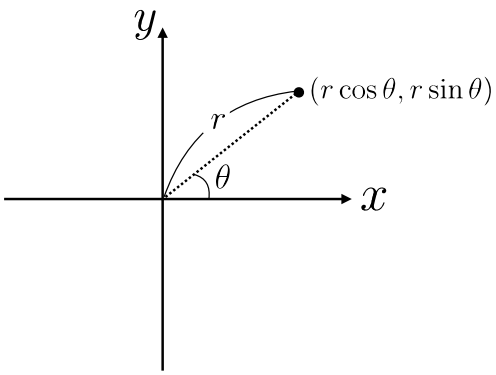

Fig. 2.8 The polar coordinates system, $(x, y) = (r\cos\theta, r\sin\theta)$

2.4.1 Box-Muller Method

Here, we introduce the Box-Muller method [10] that can be used to generate the Gaussian random numbers x and y with the probability distribution $\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$ and $\frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}$. This method is very easy: from the uniform random numbers p and q in $[0, 1]$, x and y are obtained as

$$x = \sqrt{-2\log p}\cos(2\pi q), \qquad y = \sqrt{-2\log p}\sin(2\pi q). \tag{2.10}$$

Let us confirm that x and y obtained in this way are indeed Gaussian random. For that, it is convenient to use the polar coordinates system (r, θ). The polar coordinates system (r, θ) and the Cartesian coordinates system (x, y) are related as (Fig. 2.8):

$$x = r\cos\theta, \qquad y = r\sin\theta. \tag{2.11}$$

The radial coordinate r runs from 0 to ∞, and the angular coordinate θ runs from 0 to 2π. In the polar coordinates system, the Gaussian distribution is decomposed as $\rho(r) = re^{-\frac{r^2}{2}}$ and $\rho(\theta) = \frac{1}{2\pi}$. From (2.10), we obtain $p = e^{-\frac{r^2}{2}}$.[7] Because the right-hand side runs from 1 ($r = 0$) to 0 ($r = \infty$), it agrees with the range of p ($0 \le p \le 1$). Furthermore, because $|dp| = |d(e^{-\frac{r^2}{2}})| = re^{-\frac{r^2}{2}}dr$, the relation $dr\rho(r) = |dp|$ holds, which is consistent with our choice $\rho(p) = 1$ (the uniform distribution). As for the angular coordinate, the correspondence is simply $\theta = 2\pi q$.

[7] Because $r^2 = x^2 + y^2 = -2\log p(\cos^2(2\pi q) + \sin^2(2\pi q)) = -2\log p$, we obtain $\log p = -\frac{r^2}{2}$, and hence, $p = e^{\log p} = e^{-\frac{r^2}{2}}$.

2.4.2 Expectation Values from the Gaussian Distribution

Let us calculate the expectation values from the Gaussian distribution $P(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. For generic function $f(x)$, the expectation value is defined as

$$\langle f(x) \rangle = \int_{-\infty}^{\infty} dx f(x) P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} dx f(x) e^{-\frac{x^2}{2}}. \tag{2.12}$$

The counterpart of Eq. (2.3) for the integration via the uniform random numbers is

$$\lim_{K \to \infty} \frac{1}{K} \sum_{k=1}^{K} f(x^{(k)}) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} dx f(x) e^{-\frac{x^2}{2}}. \tag{2.13}$$

Here, it is assumed that $x^{(1)}, x^{(2)}, \ldots$ are Gaussian random numbers (generated by the Box-Muller method, for example). By actually generating the Gaussian random numbers, we can easily check $\langle x \rangle = 0$ and $\langle x^2 \rangle = 1$. It is a good exercise, we recommend you to try it. (We will do the same calculation by using MCMC in Sect. 4.2.)

We can also calculate the usual integral, rather than the expectation value. If you want to calculate $\int_{-\infty}^{\infty} dx g(x)$, then you can use a simple relation

$$\int_{-\infty}^{\infty} dx g(x) = \int_{-\infty}^{\infty} dx \left(g(x) \cdot \sqrt{2\pi} e^{+\frac{x^2}{2}} \right) \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} = \left\langle g(x) \cdot \sqrt{2\pi} e^{+\frac{x^2}{2}} \right\rangle \tag{2.14}$$

to relate the integral and the expectation value. This is a special version of a powerful method explained in Sect. 4.5.

2.5 Example in Which Randomness is Essential

In the examples we have seen so far, the use of random numbers might have made our lives a little bit convenient, but it was not essential. We could solve the problems without using random numbers.

As an example in which the use of random numbers is essential, let us think about how we might be able to make money from the stock market. It is very hard to predict how the price p will change as a function of time t, but suppose we are confident about the trend shown in Fig. 2.9 by the arrow. Of course, there are uncertainties in any predictions, and the uncertainties become larger in further future. To make a profit based on the prediction, we can introduce thresholds: if the price becomes higher than the upper threshold, we sell the stock to realize the gain, and if the price

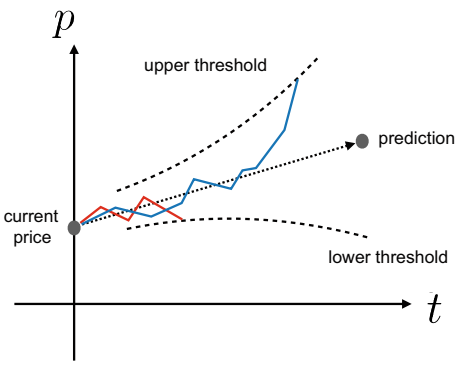

Fig. 2.9 A cartoon picture of the simulation of the stock price. The vertical and horizontal axes are price and time

becomes lower than the lower threshold, we again sell the stock to avoid further loss. In Fig. 2.9, the thresholds are shown by the dashed lines. To test this strategy, we should make a model of the stock price taking into account the randomness, and then simulate various time evolutions by using different random numbers. Perhaps we wait for a while and realize the gain, as shown by a blue line; or perhaps we have to sell the stock immediately to avoid a loss, as shown by a red line. By repeating numerical experiments many times, we can get a reliable result. Based on such simulation, we can determine good threshold functions that lead to a large profit avoiding unnecessary risk. Such modeling is essentially the same as the Brownian motion in physics. The random walk, which we will see later in this book, is a similar problem as well.

Note that the authors are not finance experts, we never made a profit in the financial market based on numerical simulations. If you want to make money, please ignore this section. You should prepare well by reading actual finance books or by learning from experts before diving into the market.

References

1. N. Metropolis, S. Ulam, The Monte Carlo method. J. Am. Stat. Assoc. **44**(247), 335–341 (1949)
2. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087–1092 (1953)
3. M. Rosenbluth, Proof of validity on Monte Carlo method for canonical averaging. Technical report, Los Alamos Scientific Lab., 1953; AIP Conference Proceedings (American Institute of Physics, 2003)
4. D. Landau, K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge University Press, 2000)
5. J.E. Gentle, *Random Number Generation and Monte Carlo Methods* (Springer, 2003)
6. H. Cramer, *Mathematical Methods of Statistics* (Princeton University Press, 1946)
7. M.A. Nielsen, I. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, 2010)
8. J. von Neumann, Various techniques used in connection with random digits, in *Monte Carlo Method*. National Bureau of Standards Applied Mathematics Series, Chap. 13, ed. by A.S.

Householder, G.E. Forsythe, H.H. Germond, vol. 12 (US Government Printing Office, Washington, DC, 1951), pp. 36–38
9. M. Matsumoto, T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 30 (1998)
10. G.E. Box, M.E. Muller, A note on the generation of random normal deviates. Ann. Math. Statist. **29**, 610–611 (1958)

Chapter 3
General Aspects of Markov Chain Monte Carlo

In this chapter, we explain general aspects of the Markov Chain Monte Carlo (MCMC) method. Concrete examples such as the Metropolis algorithm [1, 2] and the HMC algorithm [3] will be discussed in later chapters. (There are interesting and conflicting recollections regarding the birth of MCMC by the inventors [4–7].)

Suppose a probability $P(x_1, x_2, \ldots, x_n)$ is given as a function of n variables $x_1, x_2, \ldots, x_n$. Because it is tedious to write n variables every time, we use $\{x\}$ meaning $x_1, x_2, \ldots, x_n$, like

$$P(x_1, x_2, \ldots, x_n) = P(\{x\}). \tag{3.1}$$

By definition, "probability" P has to be non-negative:

$$P(\{x\}) \geq 0. \tag{3.2}$$

A set of variables $\{x\}$ is often called a "configuration" or a "sample". Sometimes, we want to consider a more generic function that can take negative values as well. We will explain useful techniques for such a case later.

In MCMC, the values of $x_1, x_2, \ldots, x_n$ are varied in such a way that the time spent at point $\{x\}$ is proportional to $P(\{x\})$. Then, the average over a sufficiently many configurations is a good approximation of the statistical average. More concretely, a sequence of configurations $\{x^{(0)}\} \to \{x^{(1)}\} \to \{x^{(2)}\} \to \cdots \to \{x^{(k)}\} \to \{x^{(k+1)}\} \to \cdots$ is constructed as follows [1, 2]:

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7_3

Basic conditions of Markov Chain Monte Carlo

- The sequence is a *Markov chain.* Namely, the probability that $\{x^{(k+1)}\}$ is obtained from $\{x^{(k)}\}$ depends only on $\{x^{(k)}\}$ and does not depend on the history $\{x^{(0)}\}, \{x^{(1)}\}, \ldots, \{x^{(k-1)}\}$. This probability is called the transition probability and denoted by $T(\{x^{(k)}\} \to \{x^{(k+1)}\})$.
- *Irreducibility*. A Markov chain is said to be irreducible if, for any pairs $\{x\}$ and $\{x'\}$, a transition between them is possible with a finite number of steps. In MCMC, an irreducible Markov chain is used.
- *Aperiodicity*. Suppose that it is possible to start from $\{x\}$ and come back to the same $\{x\}$ after n_s steps. There are many possible values of n_s, and the greatest common divisor of all of them is called the period. A given Markov chain is said to be aperiodic when the period is 1 for any $\{x\}$. In MCMC, an aperiodic Markov chain is used.
- *Detailed balance condition* is satisfied, i.e., the transition probability T satisfies

$$P(\{x\}) \cdot T(\{x\} \to \{x'\}) = P(\{x'\}) \cdot T(\{x'\} \to \{x\}). \tag{3.3}$$

for any $\{x\}$ and $\{x'\}$.

The initial configuration $\{x^{(0)}\}$ can be arbitrary.[1] If these four conditions are satisfied, the probability distribution of $\{x^{(k)}\}$ $(k = 1, 2, \ldots)$ converges to $P(\{x\})$. By using a sufficiently long chain, we can obtain the right expectation values:

$$\begin{aligned} \langle f \rangle &= \int dx_1 \cdots dx_n f(x_1, \ldots, x_n) P(x_1, x_2, \ldots, x_n) \\ &= \lim_{K \to \infty} \frac{1}{K} \sum_{k=1}^{K} f(x_1^{(k)}, \ldots, x_n^{(k)}). \end{aligned} \tag{3.4}$$

The most important feature of this method is that the statistically important configurations (i.e., the configurations with larger probability P) are picked up more frequently. This is the importance sampling, which was mentioned in Sect. 2.2.3. As we have seen there, for the integrals with many variables, it is often the case that most of the domain of integration does not contribute to the integral. In the importance sampling, configurations $\{x\}$ from such irrelevant regions rarely appear. Therefore, the computational cost can be reduced drastically, and hard calculations that are out of reach with other methods can be done.

The four conditions introduced above may look rather abstract. Below, by using several simple examples, we will explain the meaning of the conditions and why they are required.

[1] The efficiency of the simulation can depend on the initial configuration, as we will see later.

3.1 Markov Chain

A sequence of configurations $\{x^{(0)}\} \to \{x^{(1)}\} \to \{x^{(2)}\} \to \cdots \to \{x^{(k)}\} \to \{x^{(k+1)}\} \to \cdots$ is called Markov chain if the probability that $\{x^{(k+1)}\}$ is obtained after $\{x^{(k)}\}$ depends only on $\{x^{(k)}\}$ and not on $\{x^{(0)}\}, \{x^{(1)}\}, \ldots, \{x^{(k-1)}\}$ (Fig. 3.1).

A typical example of Markov chain is random walk: starting with $x^{(0)} = 0$, add or subtract 1 with probability $\frac{1}{2}$ for each, such that $x^{(k+1)} = x^{(k)} + 1$ or $x^{(k+1)} = x^{(k)} - 1$. If there is a certain pattern such as "after $+1$, it is more likely to get $+1$" or "-1 is more likely after $+1$", it is not Markov chain.

Let us see a few more examples:

- Suppose there are five red balls and five white balls in a box. If we do not look into the box and randomly take out one ball, the chance we get red or white is 50% for each. We put the ball back in the box immediately, and make a sequence as follows: we add 1 every time we get red ($x^{(k+1)} = x^{(k)} + 1$), and subtract 1 every time we get white ($x^{(k+1)} = x^{(k)} - 1$). If the initial value is taken to be $x^{(0)} = 0$, then $x^{(k)} =$ (number of red during k trials) $-$ (number of white during k trials). The probability of red or white is always 50% for each, regardless of the history. Hence, this is nothing but a random walk, which is a typical example of the Markov chain.
- Suppose we do not return the balls to the box. If we get red, red, and white, then three red balls and four white balls remain, so the probabilities that we get red or white at the fourth trial are $\frac{3}{7}$ and $\frac{4}{7}$, respectively. But if we get red, red, and red, then the probabilities for red and white are $\frac{2}{7}$ and $\frac{5}{7}$, respectively. Hence, you would say the probability at the k-th trial depends on the history, and it is not a Markov chain—wouldn't you? Well, that is wrong.
  The point is that, whether the results of the first three trials were red, red, white, or red, white, red, or white, red, red, the probabilities that we get red or white at the fourth trial are $\frac{3}{7}$ and $\frac{4}{7}$, respectively; the ordering does not matter. From the numbers of red and white balls after the k-th trial, which we denote by $n^{(k)}_{\text{red}}$ and $n^{(k)}_{\text{white}}$, respectively, the transition probability $T((n^{(k)}_{\text{red}}, n^{(k)}_{\text{white}}) \to (n^{(k+1)}_{\text{red}}, n^{(k+1)}_{\text{white}}))$

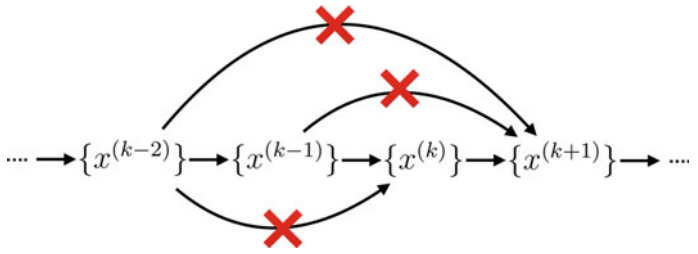

Fig. 3.1 Schematic picture of a Markov chain. The probability that $\{x^{(k+1)}\}$ is obtained from $\{x^{(k)}\}$ does not depend on $\{x^{(k-2)}\}$, $\{x^{(k-1)}\}$, etc

is uniquely determined, without relying on the history. Therefore, this is a Markov chain, too.

- Suppose we went to a restaurant in the US and split the bill evenly. One by one, each of us pays the bill and tip. Let $x^{(k)}$ be the amount of tip paid by the k-th person. How much the k-th person tips would depend not only on the total amount by then or the amount paid by the previous person but also on the history. After \$5, \$3, \$5, \$3, \$5, we would think \$3 is enough; but after \$3, \$3, \$5, \$5, \$5, we would be obliged to pay at least \$5, wouldn't we? Therefore, it is reasonable to think that $x^{(1)} \to x^{(2)} \to x^{(3)} \to \cdots$ is not a Markov chain in this case.

Because the random walk is a typical example of Markov chain, it is instructive to understand its property for later purposes. Let us consider the Japan Series. The Japan series is the annual baseball championship series in Japan. It is a best-of-seven series between the winners of the Central League and the Pacific League in a 2-3-2 format. We take $y = +1$ (resp. $y = -1$) when the winner of the Central League (resp. the Pacific League) wins the Japan Series. We also use C and P to mean $y = +1$ and $y = -1$. The index k attached to y like $y^{(k)}$ stands for the year. By choosing $x^{(k)}$ such that $x^{(0)} = 0$ and $x^{(k)} = x^{(k-1)} + y^{(k)}$, $x^{(k)}$ is the Central League's total winning record (if it is negative, the losing record). The Japan Series started in 1950, so let us take $y^{(1)}$ to the result of 1950. The actual historical results are $y^{(1)} = -1$, $y^{(2)} = +1$, $y^{(3)} = +1, \cdots$.

The Japan Series was competed 70 times by 2019, and both Leagues won 35 times. Therefore, naively we would guess that each League wins randomly with a 50% chance each year. In this simplified model, $x^{(k)}$ is a random walk.

Does such a simple model make sense? We would naturally think that, if one team is good in a given year, they are likely to be good in the following year, too. If so, we would not expect a random walk. Indeed, consecutive wins are observed very often. In the random walk model, we would expect that $+1$ and -1 appear alternately, wouldn't we?

Let us do a numerical experiment to test this intuition. We generate random numbers r between 0 and 1, and choose $y = +1$ (C) if $r \leq \frac{1}{2}$. Otherwise, we choose $y = -1$ (P). A random walk generated this way is shown in Table 3.1. By comparing it with the actual results of the Japan Series, we can find similar patterns. In actual history, the Central League won nine seasons in a row from 1965 to 1973. In the random walk model, the Central League won eight seasons in a row from 1954 to 1961. From 2010 to 2019, both in actual history and random walk, the Pacific League won nine times. In the random-walk model, the total result is the Central League's 27 wins and 43 losses, which deviates a lot from 35 to 35. This is because the number of samples is too small and the result fluctuates if we use a different sequence of random numbers. We tried a few other sequences, which led to 40–30, 37–33, 40–30, 33–37, and 31–39. After 1000 years, we obtained 485–515, which is roughly half-and-half.[2]

[2] In a random walk, the distance from the origin after K steps is typically $\sqrt{K}$. The sum of K uniform random numbers shown in Sect. 2.1.2 is similar to random walk, and as we can see from Fig. 2.3, the Gaussian distribution of the width of order $\sqrt{K}$ is obtained. Therefore, in the random-walk model, the deviation from the 50% win rate scales as $\frac{1}{\sqrt{K}}$.

Table 3.1 Random Walk (RW) and Japan Series (JS)

Year	RW	JS	Year	RW	JS	Year	RW	JS	Year	RW	JS
1950	P	P	1970	P	C	1990	P	P	2010	P	P
1951	P	C	1971	C	C	1991	P	P	2011	P	P
1952	P	C	1972	C	C	1992	P	P	2012	P	C
1953	P	C	1973	P	C	1993	P	C	2013	P	P
1954	C	C	1974	P	P	1994	P	C	2014	P	P
1955	C	C	1975	P	P	1995	P	C	2015	C	P
1956	C	P	1976	C	P	1996	C	P	2016	P	P
1957	C	P	1977	C	P	1997	P	C	2017	P	P
1958	C	P	1978	C	C	1998	C	C	2018	P	P
1959	C	P	1979	P	C	1999	P	P	2019	P	P
1960	C	C	1980	P	C	2000	P	C			
1961	C	C	1981	P	C	2001	C	C			
1962	P	P	1982	P	P	2002	C	C			
1963	C	C	1983	P	P	2003	P	P			
1964	C	P	1984	C	C	2004	P	P			
1965	P	C	1985	P	C	2005	P	P			
1966	C	C	1986	P	P	2006	P	P			
1967	P	C	1987	P	P	2007	C	C			
1968	P	C	1988	C	P	2008	C	P			
1969	C	C	1989	P	C	2009	C	C			

While further analyses can be performed, the honest answer would be that we cannot extract a definite conclusion from such limited data as of 2019. It would be nice if the readers several hundred years from now could add more data after 2020 and make better analyses.

Similar subtleties are there when we discuss whether we can estimate the true ability of batters from the batting records, or whether such a thing as a team's "momentum" really exists. We recommend Ref. [8] if you are interested.

From this experiment, we learned that the naive expectation ("$+1$ and -1 appear alternately in random walk") is wrong. We can understand this via simple calculations, too. Let us choose four consecutive years and estimate the probability that $+1$ and -1 appear alternately. There are $2^4 = 16$ possible outcomes, among which only two of them—$+1, -1, +1, -1$ and $-1, +1, -1, +1$—are like that. Therefore, the chance is only 12.5%. Still, if we do not specify the time frame and consider the probability that $+1$ and -1 appear alternately starting some year, then the probability is much higher. Indeed, we can see such a pattern a few times in Table 3.1. For the same reason, eight consecutive wins or nine consecutive wins are very rare if we specify the starting point, but over a long history, such rare events have to happen at some point. In this way, seemingly non-random events happen in random walk. This is a very important feature.

3.2 Irreducibility

A Markov chain is said to be irreducible when any pair of $\{x\}$ and $\{x'\}$ is connected by a finite number of steps. It should be easy to understand the importance of this property because if it is impossible to reach some points the information on those points will never be obtained. Below, let us see some examples which are *not* irreducible.

Suppose x runs through real numbers, and by using a uniform random number r between -1 and 1 let us define $x^{(k+1)} = x^{(k)} \times r$. It is not irreducible, because the absolute value of x never increases. In particular, once x becomes zero, it stays there forever.

As another typical example, suppose that the domain of integration is not connected. Let C_1 and C_2 be $x < 0$ and $x > 1$, respectively, and consider an integral on C, which is a union of C_1 and C_2. We can construct a Markov chain by choosing a uniform random number between $-c$ and $+c$, which we denote by Δx, and setting $x^{(k+1)}$ as $x^{(k)} \to x^{(k+1)} = x^{(k)} + \Delta x$ if $x^{(k)} + \Delta x \in C$ and $x^{(k)} \to x^{(k+1)} = x^{(k)}$ if $x^{(k)} + \Delta x \notin C$. Then, it is impossible to go back and forth between C_1 and C_2 when $c \leq 1$. If $x^{(0)} < 0$, then the chain is trapped in C_1; if $x^{(0)} > 1$, then the chain is trapped in C_2. Hence, such a Markov chain is not irreducible. However, when $c > 1$, it is possible to go back and forth between C_1 and C_2, and the Markov chain is now irreducible.

A similar issue arises when $P(x)$ becomes zero at $x = 0$, for example, $P(x) \propto e^{-1/x^2 - x^2}$ (Fig. 3.2). In this case, $P(x)$ is almost zero near $x = 0$, so if we use the Metropolis algorithm (which will be explained in Chap. 4) with a small value of c, it takes a very long time to pass through $x = 0$. In this case, the irreducibility is not broken in the strict mathematical sense—we just have to wait very very long—but it is a big problem in practice because we cannot live forever. In short:

Be careful when the probability distribution is split into multiple islands.

3.3 Aperiodicity

Let us consider a transition from a configuration $\{x\}$ to itself. Suppose such a transition is possible with n_s steps. There can be various n_s, and the greatest common divisor of all possible n_s's is called the period. If the period is 1 for all configurations, the Markov chain is called *aperiodic*.

Let us construct a Markov chain as $x^{(k)} \to x^{(k+1)} = x^{(k)} + \Delta x$, where Δx is uniformly random between -1 and 1. (This can be regarded as a random walk, whose step size is not fixed.) If $\Delta x = 0$, $x^{(k)} = x^{(k+1)}$ holds, and hence, it is possible to come back to the same configuration just after one step. It is also possible to come back to the same point after two steps, for example, as $x^{(k+1)} = x^{(k)} + 0.5$ and $x^{(k+2)} = x^{(k+1)} - 0.5 = x^{(k)}$. Of course, it is possible to come back after three steps, e.g., $x^{(k+1)} = x^{(k)} + 0.25$, $x^{(k+2)} = x^{(k+1)} + 0.25 = x^{(k)} + 0.5$ and $x^{(k+3)} =$

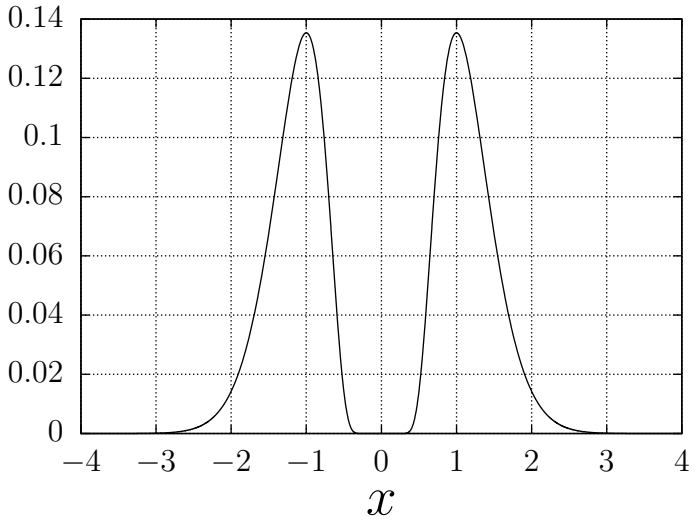

Fig. 3.2 Plot of e^{-1/x^2-x^2}

$x^{(k+2)} - 0.5 = x^{(k)}$. It should be clear that n_s can be any positive integer. Therefore, the period is 1. This holds for any $x^{(k)}$, and hence, this Markov chain is aperiodic.

Next, let us consider an example that is not aperiodic. Consider a random walk with a fixed step size ± 1. To come back to the same point, $+1$ and -1 have to appear the same number of times. Therefore, n_s has to be even. Hence, the period is 2, and this example is not aperiodic.

As another example, suppose x runs over real numbers, and take $x^{(k+1)} = x^{(k)} \times r$. We take r randomly from negative numbers, e.g., $P(r) = 0$ for $r \geq 0$ and $P(r) = \frac{2}{\sqrt{\pi}} e^{-r^2}$ for $r < 0$. If $x^{(0)} = 0$, then $x^{(k)} = 0$ for any k, and hence the period of $x = 0$ is 1. However, if $x \neq 0$, the sign flips at each step, and hence it takes an even number of steps to come back to the same value. It is possible to come back with 2 steps, for example, $x \to -x \to x$. Therefore, the period is 2 for any $x \neq 0$. Hence, this example is not aperiodic either.

What is the problem if the Markov chain is not aperiodic? As a simple example, let us consider a Markov chain with period 2, in which $x > 0$ at even steps and $x < 0$ at odd steps, like the one discussed above. Then, unless the probability that $x > 0$ and $x < 0$ are both $\frac{1}{2}$ (in equations, $\int_{-\infty}^{0} dx P(x) = \int_{0}^{\infty} dx P(x) = \frac{1}{2}$)—which is apparently a very special case—the right distribution cannot be obtained.

3.4 Detailed Balance Condition

The detailed balance condition means that the transition probability T satisfies $P(\{x\}) \cdot T(\{x\} \to \{x'\}) = P(\{x'\}) \cdot T(\{x'\} \to \{x\})$ for any configurations $\{x\}$ and $\{x'\}$. To get intuition into this condition, first let us consider the balance condition, or equivalently, the equilibrium condition.

Consider, as a slightly unrealistic setup, a village consisting of 100 residents, totally isolated from the outer world. The total assets of all the residents are \$10,000,000, which never changes. There is economic activity inside the village, and residents pay money to each other. We use $x = 1, 2, \ldots, 100$ to label the people in the village and k to label the year. On January 1 of year k, resident x has $P_k(x)$ dollars. By definition,

$$\sum_{x=1}^{100} P_k(x) = 10,000,000 \tag{3.5}$$

Let $Q_k(x \to x')$ be the amount of money paid from resident x to resident x' during the year. The carryover can be regarded as payment from resident x to himself/herself, so let us denote it by $Q_k(x \to x)$. Then, because all assets of resident x are paid to somebody including himself/herself,

$$P_k(x) = \sum_{x'=1}^{100} Q_k(x \to x'). \tag{3.6}$$

Also, resident x's assets in the next year are obtained from the residents including himself/herself, so

$$P_{k+1}(x) = \sum_{x'=1}^{100} Q_k(x' \to x). \tag{3.7}$$

That the system is in the equilibrium means the assets of each resident do not change over the years,

$$P_k(x) = P_{k+1}(x) = P_{k+2}(x) = \cdots. \tag{3.8}$$

We can also say that the incomes and payments are balanced, namely

$$\sum_{x'=1}^{100} Q_k(x \to x') = \sum_{x'=1}^{100} Q_k(x' \to x). \tag{3.9}$$

The condition (3.9) is called balance condition or equilibrium condition.

The detailed balance condition is stronger than the balance condition that requires the payment from resident x to resident x' and the payment from resident x' to resident x are the same for all combinations of (x, x'):

$$Q_k(x \to x') = Q_k(x' \to x). \tag{3.10}$$

Note that the summation $\sum$ in the balance condition (3.9) has disappeared in the detailed balance condition (3.10). The detailed balance condition (3.10) is sufficient for the balance condition (3.9), but not necessary. As an extreme example, by taking

$$\begin{aligned} Q_k(1 \to 2) &= 100,000, \\ Q_k(2 \to 3) &= 100,000, \\ &\cdots \\ Q_k(99 \to 100) &= 100,000, \\ Q_k(100 \to 1) &= 100,000 \end{aligned} \tag{3.11}$$

and setting all other $Q_k(x \to x')$ to be zero, the balance condition (3.9) can be satisfied. In this case, you may like the person paying \$100,000 to you, while you may have a complicated feeling toward the person to whom you have to pay \$100,000. On the other hand, if the detailed balance condition (3.10) is satisfied, the same amount of money is paid between any two people, so it is less likely to have a sense of unfairness toward anybody. In Fig. 3.3, the difference between the detailed balance condition and a mere balance condition is visually explained.

The detailed balance condition (3.3) used in Markov Chain Monte Carlo may look slightly different from the example above, but they are essentially the same. We want to interpret $P(\{x\})$ as the probability that the state is in $\{x\}$, so it is the same as "what percentage of \$10,000,000 is owned by resident x". That the total assets in

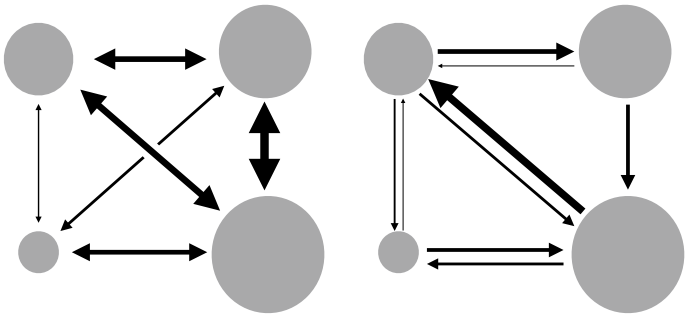

Fig. 3.3 [Left] The detailed balance condition. The payment is the same for any pair of residents. [Right] A mere balance condition. The income and payment of each person are balanced. The size of the circles indicates the "net worth" $P(\{x\})$ (the probability that the state is in $\{x\}$), and the thickness of the arrows stands for the "amount of payment" $Q(\{x\} \to \{x'\}) = P(\{x\}) \cdot T(\{x\} \to \{x'\})$

the village do not change corresponds to the fact that the probabilities always sum up to 1. The transition function $T(\{x\} \to \{x'\})$ can be interpreted as the percentage of resident x's assets that is paid to resident x', and then $P(\{x\}) \cdot T(\{x\} \to \{x'\})$ and $Q(x \to x')$ are actually the same thing. Therefore, the balance condition (3.9) can be rewritten as

$$\sum_{\{x'\}} P(\{x\}) \cdot T(\{x\} \to \{x'\}) = \sum_{\{x'\}} P(\{x'\}) \cdot T(\{x'\} \to \{x\}). \tag{3.12}$$

The detailed balance condition (3.10) is, as we have already seen,

$$P(\{x\}) \cdot T(\{x\} \to \{x'\}) = P(\{x'\}) \cdot T(\{x'\} \to \{x\}). \tag{3.13}$$

As we already mentioned, if the detailed balance condition (3.13) is imposed, the balance condition (3.12), which is weaker than the detailed balance condition, follows automatically.

If the balance condition is satisfied, then the probability distribution $P(\{x\})$ does not change, as shown in (3.8). The derivation of this from the detailed balance condition (3.13) is as follows:

$$\begin{aligned} \sum_{\{x\}} P(\{x\}) \cdot T(\{x\} \to \{x'\}) &= \sum_{\{x\}} P(\{x'\}) \cdot T(\{x'\} \to \{x\}) \\ &= P(\{x'\}) \cdot \sum_{\{x\}} T(\{x'\} \to \{x\}) \\ &= P(\{x'\}). \end{aligned} \tag{3.14}$$

The detailed balance condition is used in the first line. To go from the second line to the third line, $\sum_{\{x\}} T(\{x'\} \to \{x\}) = 1$ (i.e., the sum of the transition probabilities is 1) is used.

In MCMC, as the chain $\{x^{(0)}\} \to \{x^{(1)}\} \to \{x^{(2)}\} \to \cdots \to \{x^{(k)}\} \to \{x^{(k+1)}\} \to \cdots$ becomes longer the probability distribution converges to a certain stationary distribution $P'(\{x\})$. That the distribution is stationary means it is invariant when it is shifted by one step, namely $\sum_{\{x\}} P'(\{x\}) T(\{x\} \to \{x'\}) = P'(\{x'\})$. We want such P' to be the target distribution P. For this, the condition (3.14) is required, and hence, we impose the detailed balance condition on the transition probability.

Strictly speaking, only the balance condition is required for (3.14) to hold, and we do not necessarily need the detailed balance condition. However, while it is not so hard to find algorithms satisfying the detailed balance condition, it is not easy to find an example satisfying only the balance condition and not the detailed balance condition. Therefore, the detailed balance condition is usually imposed. (There are algorithms that satisfy the balance condition but not the detailed balance condition, e.g., the Look-Ahead HMC method [9]. See also Exercise 2 of Chap. 4.)

3.5 Exercises

1. Let us throw dice, and add the numbers one after another. Is it a Markov chain? Is it irreducible, and/or aperiodic?
2. Alice and Bob play rock-paper-scissors. Alice repeats the same throw after a win or draw, and changes after loss so that she can win if Bob repeats the same throw. Bob always chooses rock, paper, or scissors randomly with $\frac{1}{3}$ probability for each. Is it a Markov chain? Is it irreducible, and/or aperiodic?
3. Suppose Bob used the same strategy as Alice. Is it a Markov chain? Is it irreducible, and/or aperiodic?

Solutions

1. Let the sum of the first k rolls be $x^{(k)}$. Then $x^{(k+1)}$ can take $x^{(k)}+1, \ldots, x^{(k)}+6$ with a probability $\frac{1}{6}$ for each. The past history $x^{(1)}, \ldots, x^{(k-1)}$ does not matter, so it is a Markov chain. However, the sum only increases, i.e., $x^{(1)} < x^{(2)} < \cdots < x^{(k)} < x^{(k+1)} < \cdots$, hence this Markov chain is not irreducible. Because it never comes back to the same value, we cannot define a period. Therefore "whether it is periodic or not" is not even a well-posed question.
2. Because Alice makes a decision based only on the prior throw and Bod does not care about the prior throw, it is a Markov chain. Intuitively, the irreducibility is obvious because Bob's choice is completely random, but let us confirm it explicitly.

   - **When the first throw is draw**: If the first throw is (R, R), then the second throw can be (R, R), (R, S), or (R, P). After (R, R) and (R, S) only the same options (R, R), (R, S), or (R, P) can follow, but (S, R), (S, S), or (S, P) can follow after (R, P), and (P, R), (P, S), or (P, P) can follow after (S, R). In this way, any combinations can be obtained starting with (R, R). The same holds when the first throw is (S, S) or (P, P).
   - **When Alice wins the first throw**: If the first throw is (R, S), the second throw can be (R, R) with a probability $\frac{1}{3}$. We already saw any combinations can be obtained starting from (R, R), so the same is true starting from (R, S). The same holds when the first throw is (P, R) or (S, P).
   - **When Bob wins the first throw**: If the first throw is (R, P), the second throw can be (S, S) with a probability $\frac{1}{3}$. We already saw any combinations can be obtained starting from (S, S), so the same is true starting from (R, P). The same holds when the first throw is (S, R) or (P, S).

   Therefore, this Markov chain is irreducible. Next, let us confirm the aperiodicity.

   - **When the first throw is draw**: If the first throw is (R, R), the second throw can be (R, R) again, with a probability $\frac{1}{3}$. Therefore the period is 1. The same holds when the first throw is (S, S) or (P, P).

- **When Alice wins the first throw**: If the first throw is (R, S), the second throw can be (R, S) again, with a probability $\frac{1}{3}$. Therefore the period is 1. The same holds when the first throw is (P, R) or (S, P).
- **When Bob wins the first throw**: If the first throw is (R, P), it is possible to come back to (R, P) as (R, P) $\rightarrow$ (S, P) $\rightarrow$ (S, P) $\rightarrow \cdots \rightarrow$ (S, P) $\rightarrow$ (S,R) $\rightarrow$ (P, S) $\rightarrow$ (R, P). Here, (S,P) can be repeated arbitrary times. Hence the period is 1. The same holds when the first throw is (S, R) or (P, S).

Hence, the period is 1 for any combinations, and this Markov chain is aperiodic.

3. Because both Alice and Bob make decisions based only on the prior throw, it is a Markov chain. However, it is neither irreducible nor aperiodic. If the first throw is draw, then the same combination continues forever. If the first throw is not draw, then the following pattern with period 6 repeats forever: (R, S) $\rightarrow$ (R, P) $\rightarrow$ (S, P) $\rightarrow$ (S, R) $\rightarrow$ (P, R) $\rightarrow$ (P, S) $\rightarrow$ (R, S).

References

1. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087–1092 (1953)
2. M. Rosenbluth, Proof of validity on Monte Carlo method for canonical averaging. Technical report, Los Alamos Scientific Lab., 1953; AIP Conference Proceedings (American Institute of Physics, 2003)
3. S. Duane, A.D. Kennedy, B.J. Pendleton, D. Roweth, Hybrid Monte Carlo. Phys. Lett. B **195**(2), 216–222 (1987)
4. M.N. Rosenbluth, Genesis of the monte carlo algorithm for statistical mechanics, in *AIP Conference Proceedings*, vol. 690 (American Institute of Physics, 2003), pp. 22–30
5. J.E. Gubernatis, Marshall Rosenbluth and the metropolis algorithm. Phys. Plasm. **12**(5), 057303 (2005)
6. E. Teller, J. Shoolery, *Memoirs: A Twentieth-century Journey in Science and Politics* (Perseus Publishing, 2001)
7. N. Metropolis, The beginning of the Monte Carlo method. Los Alamos Sci. **15**(584), 125–130 (1987)
8. J. Albert, J. Bennett, *Curve Ball: Baseball Statistics and the Role of Chance in the Game* (Springer Science & Business Media, 2001)
9. J. Sohl-Dickstein, M. Mudigonda, M. DeWeese, Hamiltonian Monte Carlo without detailed balance, in *International Conference on Machine Learning* (PMLR, 2014), pp. 719–726

Chapter 4
Metropolis Algorithm

General aspects of the Markov Chain Monte Carlo algorithms, which we learned in the previous chapter, may have left you confused. To really understand the subject, we need concrete examples and implementations. In this chapter, we introduce the Metropolis algorithm [1, 2], which is the most famous version of MCMC, and use it to demonstrate how actual simulations are done.

Though MCMC is used for complicated calculations, when we learn how to use them there is absolutely no need for using complicated examples. In this chapter, we start with the easiest example: univariate integration. This example tells you all the essence of the Markov Chain Monte Carlo algorithms. Toward the end of the chapter, we discuss the multivariate version as well. Hopefully, you will be surprised in a good sense: the generalization to multivariate integral is very straightforward.

4.1 Metropolis Algorithm

Suppose that a probability distribution $P(x)$ is written as

$$P(x) = \frac{e^{-S(x)}}{Z}. \tag{4.1}$$

In physics, the function $S(x)$ is called the action and the normalization factor Z is called the partition function. If you are reading this book having the application to statistics in mind, you can regard S as log-likelihood up to a sign. Below, we assume that $S(x)$ is a continuous function of a real variable x.[1] In the case of the Gaussian

[1] An example with discrete variables is the Ising model which will be explained in Sect. 6.2.

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7_4

distribution, the action is $S(x) = \frac{x^2}{2}$ and the partition function is $Z = \sqrt{2\pi}$. Usually, in practical applications, only $S(x)$ is known and Z is unknown.

In the Metropolis algorithm, starting with an initial value $x^{(0)}$, a sequence $x^{(1)}$, $x^{(2)}, \ldots, x^{(k)}, x^{(k+1)}, \ldots$ is generated as follows:

Metropolis algorithm

1. Choose a real number Δx randomly, and propose $x' = x^{(k)} + \Delta x$ as a candidate for $x^{(k+1)}$. To satisfy the detailed balance condition, we choose the probability distribution of Δx such that Δx and $-\Delta x$ appear with the same probability. (Here, we choose an appropriate $c > 0$ and use the uniform random numbers between $-c$ and $+c$.)
2. Metropolis test: the candidate x' is accepted and the value of x is updated as $x^{(k+1)} = x'$ with a probability $\min(1, e^{S(x^{(k)})-S(x')})$. (Here $\min(1, e^{S(x^{(k)})-S(x')})$ means the smaller one of 1 and $e^{S(x^{(k)})-S(x')}$.) Otherwise x' is rejected and the value of x remains unchanged, as $x^{(k+1)} = x^{(k)}$.

Note that Δx is chosen randomly at each k. For the Metropolis test, a uniform random number r between 0 and 1 is generated, and the proposal x' is accepted if $r < e^{S(x^{(k)})-S(x')}$.

Among the four conditions listed in Chap. 3, we can immediately check three of them:

- We choose Δx randomly without referring to the history, so it is trivially a Markov chain.
- Because we are considering a connected domain of integration, any pair of x and x' can be connected with a finite number of steps. Hence, this Markov chain is irreducible.
- For any $n_s = 1, 2, \ldots$ and x, there is a path connecting x and itself with n_s steps. ($n_s = 1$ is realized when $\Delta x = 0$. Except for the maxima of $S(x)$, $n_s = 1$ can be realized also when the proposed candidate x' is rejected at the Metropolis test.) Hence, the period is 1 for any x, and this Markov chain is aperiodic.

The detailed balance condition is also satisfied. This is a little bit nontrivial, so let us follow the logic carefully:

- First of all, because we assume $-c < \Delta x < c$, the transition probability is zero if $|x - x'| \geq c$:

$$T(x \to x') = T(x' \to x) = 0 \quad \text{if} \quad |x - x'| \geq c. \tag{4.2}$$

  In this case, the detailed balance condition is trivially satisfied as

$$P(x) \cdot T(x \to x') = P(x') \cdot T(x' \to x) = 0. \tag{4.3}$$

- If $|x - x'| < c$, $\Delta x = x' - x$ and $-\Delta x = x - x'$ appear with the same probability $\frac{1}{2c}$. (More precisely, this is the probability density. The probability that $x' - x <$

$\Delta x < x' - x + \epsilon$ is $\epsilon/2c$.) By multiplying this and the probability of passing the Metropolis test, we obtain

$$T(x \to x') = \frac{1}{2c} \times \min(1, e^{S(x)-S(x')}), \tag{4.4}$$

$$T(x' \to x) = \frac{1}{2c} \times \min(1, e^{S(x')-S(x)}). \tag{4.5}$$

Suppose $S(x) \geq S(x')$. Then $e^{S(x)-S(x')} \geq 1$, and hence, a proposal $x \to x'$ passes the Metropolis test with 100% probability, and hence, $T(x \to x') = \frac{1}{2c}$. Therefore,

$$P(x) \cdot T(x \to x') = \frac{e^{-S(x)}}{Z} \times \frac{1}{2c}. \tag{4.6}$$

On the other hand, $e^{S(x')-S(x)} \leq 1$, and hence $T(x' \to x) = \frac{e^{S(x')-S(x)}}{2c}$. Therefore,

$$P(x') \cdot T(x' \to x) = \frac{e^{-S(x')}}{Z} \cdot \frac{e^{S(x')-S(x)}}{2c} = \frac{e^{-S(x)}}{Z} \times \frac{1}{2c}. \tag{4.7}$$

In this way, we could confirm that the detailed balance condition $P(x) \cdot T(x \to x') = P(x') \cdot T(x' \to x)$ is satisfied.
When $S(x) < S(x')$, we can easily check the detailed balance condition by repeating the same argument exchanging the roles of x and x'.

4.2 Calculation of Expectation Value

Let us see a concrete example of the calculation based on the Metropolis algorithm. As before, we use $S(x) = \frac{x^2}{2}$. Here is a sample code in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int main(void){
  int niter=100;  //Collect 100 samples.
  double step_size=0.5e0;  //Set step size to be 0.5.

  srand((unsigned)time(NULL));
  //Set the seeds of random numbers by using the system clock.

  /*******************************/
```

```
  /* Set the initial configuration */
  /*********************************/
  double x=0e0;
  int naccept=0;  //Counter for the number of acceptance.
  /*************/
  /* Main loop */
  /*************/
  for(int iter=1;iter<niter+1;iter++){
    double backup_x=x;
    double action_init=0.5e0*x*x;

    double dx = (double)rand()/RAND_MAX;
    dx=(dx-0.5e0)*step_size*2e0;
    x=x+dx;

    double action_fin=0.5e0*x*x;
    /*******************/
    /* Metropolis test */
    /*******************/
    double metropolis = (double)rand()/RAND_MAX;
    if(exp(action_init-action_fin) > metropolis)
      /* accept */
      naccept=naccept+1;
    else
      /* reject */
      x=backup_x;
    /***************/
    /* data output */
    /***************/
    printf("%.10f   %f\n",x,(double)naccept/iter);}
}
```

Let us decipher this sample code line by line. Firstly, we set the seed of the random number generator:

```
srand((unsigned)time(NULL));
```

Here, we are using the default random number generator in the system. It is not a good habit to use the same sequence of random numbers every time, so we used the system time as a seed. For more serious, large-scale simulations, we recommend you use a more sophisticated random number generator such as the Mersenne Twister [3].

Next, we set the initial condition. Here we chose $x = 0$:

```
double x=0e0;
int naccept=0;
```

`naccept` is a counter for the number of acceptances (i.e., how many times proposals of the update $x \to x'$ are accepted).

The following "main loop" is the main part. The variable `iter` corresponds to k. (`iter` means iteration.) An integer `niter` is the number of iterations, or equivalently, the number of configurations generated during the simulation. Before

the Metropolis test, we do not know whether the proposed value will be accepted or not, so we save the value of $x = x^{(k)}$ in backup_x,

```
double backup_x=x;
```

then we calculate the action action_init $= S(x^{(k)})$ as

```
double action_init=0.5e0*x*x;
```

Here "init" means initial, namely, this is the 'initial' action before the candidate x' is proposed. Next, dx $= \Delta x$ is randomly generated and $x' = x^{(k)} + \Delta x$ is calculated:

```
double dx = (double)rand()/RAND_MAX;
dx=(dx-0.5e0)*step_size*2e0;
x=x+dx;
```

Note that a random number between 0 and 1 is generated as rand()/RAND_MAX, then it is shifted and rescaled such that a uniform random number between $-c$ and $+c$ is obtained. By using x' obtained in this way, action_fin $= S(x')$ is calculated ("fin" means final). Finally, the Metropolis test is performed:

```
/*******************/
/* Metropolis test */
/*******************/
double metropolis = (double)rand()/RAND_MAX;
if(exp(action_init-action_fin) > metropolis)
  /* accept */
  naccept=naccept+1;
else
  /* reject */
  x=backup_x;
```

metropolis is a uniform random number between 0 and 1, which corresponds to r. Depending on the outcome of the Metropolis test, the candidate x' is accepted or rejected.

All programs based on Markov Chain Monte Carlo have essentially the same structure. Depending on the details of the problems, more sophisticated algorithms may be used, but essentially, any algorithm amounts to the improvement of $x \to x' = x + \Delta x$. Therefore, if you could understand how this program works, you can understand any complicated programs for MCMC except for technical details.

Let us see a result of an actual simulation. We take the initial configuration to be $x^{(0)} = 0$, and use the step size $c = 0.5$. (As we will see later, this choice of step size c is not optimal.) In Fig. 4.1, the distribution of $x^{(1)}, x^{(2)}, \ldots, x^{(K)}$ is shown for $K = 10^3, 10^5, 10^7$. We can clearly see the convergence to the target distribution $P(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$ as K becomes larger. In Fig. 4.2, the expectation values $\langle x \rangle = \frac{1}{K}\sum_{k=1}^{K} x^{(k)}$ and $\langle x^2 \rangle = \frac{1}{K}\sum_{k=1}^{K} \left(x^{(k)}\right)^2$ are plotted. As K becomes larger, they converge to the correct values (i.e., the expectation values under the target distribution) 0 and 1.

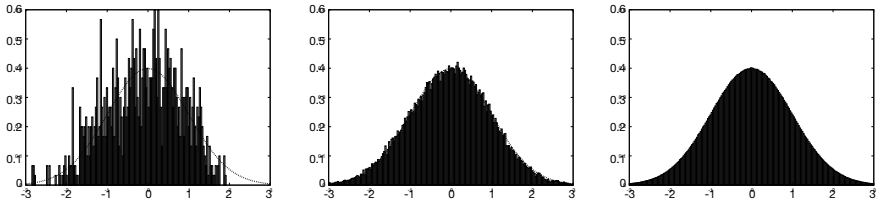

Fig. 4.1 The histogram of $x^{(1)}, x^{(2)}, \ldots, x^{(K)}$ for $K = 10^3, 10^5$ and 10^7. It converges to the target distribution $P(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$ as K becomes large

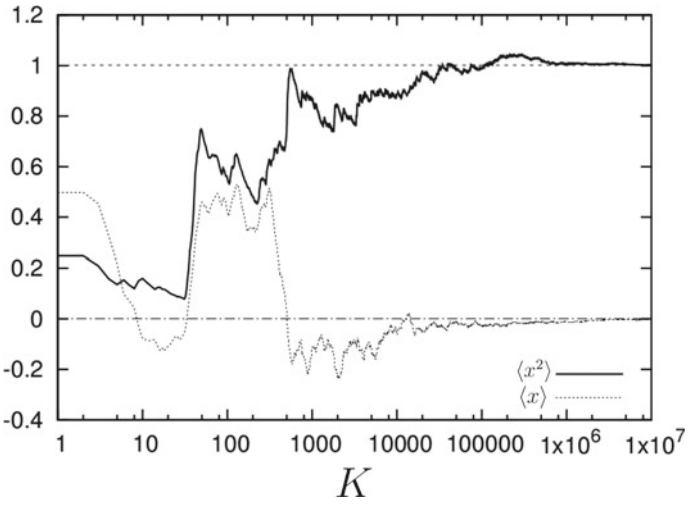

Fig. 4.2 $\langle x \rangle = \frac{1}{K} \sum_{k=1}^{K} x^{(k)}$ and $\langle x^2 \rangle = \frac{1}{K} \sum_{k=1}^{K} \left(x^{(k)}\right)^2$. They approach the correct values 0 and 1 as K increases

The step size c is taken such that the acceptance rate (the probability that the proposal $x \to x' = x + \Delta x$ is accepted) is not too large and not too small. If c is too large, the acceptance rate is very small, and the value of x rarely changes. If c is too small, the acceptance rate becomes almost 100%, but the change at each step is too small and hence x stays more or less the same value for a long time. Either way, compared to the optimal value of c, a lot more steps are needed in order to approximate the correct statistical distribution. This point is explained in detail in Sect. 4.3.

Typically, 30%–80% acceptance rate is the sweet spot. However, it depends on the algorithm and/or the kind of integral under consideration.

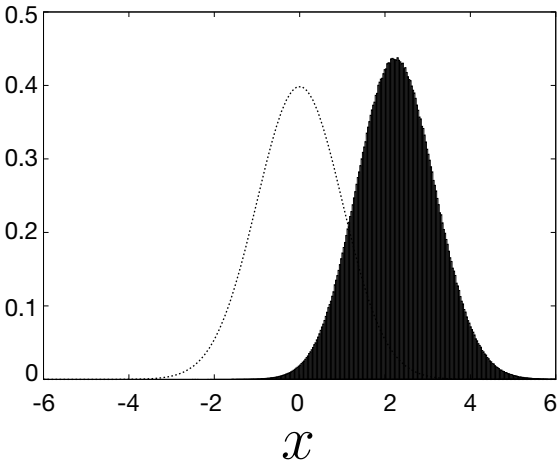

Fig. 4.3 The histogram of $K = 10^7$ configurations obtained by using the Metropolis algorithm, with a *wrong* choice $\Delta x \in \left[-\frac{1}{2}, 1\right]$. The dashed line is the target distribution $P(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$. Because the detailed balance condition is not satisfied, the target distribution is not correctly reproduced

Example of Incorrect Implementation

People learn from their mistakes. To use the Metropolis algorithm correctly, let us see an example of a wrong way of using the algorithm. Suppose we chose Δx randomly between $-\frac{1}{2}$ and 1. Then the detailed balance condition is not satisfied; it can easily be seen by noticing that a transition $0 \to 1$ can happen with a nonzero probability while $1 \to 0$ cannot. The distribution of x calculated this way is shown in Fig. 4.3. Obviously, the target distribution is not correctly reproduced.

4.3 Autocorrelation

In principle, just by following the rules we have seen so far, we can always generate the target distribution. However, that it is correct in principle does not mean it is practically useful. Because we cannot live forever, we need to reach the target distribution as quickly as possible. Furthermore, we have to make sure that the samples we got are "good" ones. For example, even if we got 10,000 samples, if the acceptance rate is too low and we got only 10 different values, each of them appearing 1000 times, then the value of such "bad" samples is just the same as the value of 10 "good" samples.

The correlation is an important keyword to understand this issue. In Markov Chain Monte Carlo, because $x^{(k+1)}$ is obtained by slightly changing $x^{(k)}$, they are correlated to each other. This correlation is called the autocorrelation. If the step size is too small or the acceptance rate is too low, the autocorrelation can be very large. If the autocorrelation is larger, it takes a longer time for the convergence to

the target probability distribution, and the quality of the samples becomes poorer. In this section, we will see how the autocorrelation can be estimated, and how it can be reduced.

4.3.1 Correlation with the Initial Value and Thermalization (Burn-in)

When we studied the Gaussian distribution $P(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$, we took the initial configuration to be $x^{(0)} = 0$. We chose this value because we knew the center of the target distribution is $x = 0$. What could happen if we took the initial configuration far away from the center of the target distribution, say $x^{(0)} = 100$?

The result of the simulation with $x^{(0)} = 100$ is shown in Fig. 4.4. We can see that the value of x stays large for a while, because of a strong correlation with the initial value $x^{(0)} = 100$. After some time, the correlation with the initial configuration disappears, and the fluctuation about the center of the target distribution ($x = 0$) sets in. That the simulation reached the center of the distribution in this way is sometimes expressed as "the simulation (or the Markov chain) thermalized" or "the simulation reached thermalization". It is also said that "the Markov chain has burned in".

As the word "thermalization" suggests, intuition from physics is useful to understand this phenomenon. Imagine we put a small piece of ice into a cup of water. Until the ice completely melts, heat moves from water to ice. This is not a thermalized state, rather the effect from a particular choice of the initial condition ("put a small piece of ice into the water") is still there. However, after some time, the ice completely melts, temperature becomes uniform everywhere in the cup, and macroscopically we will not see a change anymore. This is the thermalized state. Each molecule is moving fast even in the thermalized state, but macroscopically it is just a "typical state". The analogous situation in the Markov Chain Monte Carlo simulation is that the corre-

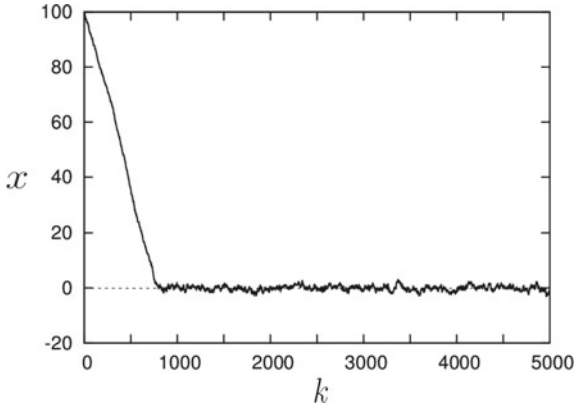

Fig. 4.4 The history of the simulation of the Gaussian distribution via the Metropolis algorithm. The step size is $c = 0.5$. Because we chose the initial value to be $x = 100$, which is very far from the typical values, the strong correlation with the initial value remains for a while, and it took a lot of steps to reach the typical values $|x| \sim 1$

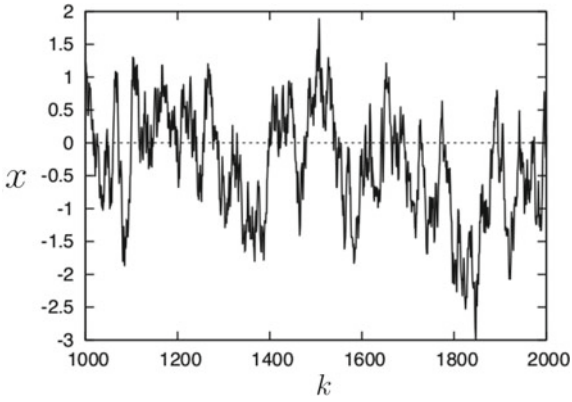

Fig. 4.5 The zoom-in of Fig. 4.4, from $k = 1000$ to $k = 2000$. Strong correlations survive at least for 20 or 30 steps

lation with the initial configuration became sufficiently small and the configurations are moving in the important regions dominating the integral. From this analogy, the meaning of the word "thermalization" should be clear. (As we will explain shortly, "thermalization" is used for another meaning as well; be cautious!)

When we calculate the expectation value, (unless the number of samples is extremely large) a large error arises if we use the non-thermalized configurations, because of the strong influence from the initial condition. Practically, we have to discard the samples before the thermalization. In Fig. 4.4, the thermalization is achieved by $k \sim 800$, so if we discard $k \leq 1000$ it should be more than enough. To make a more quantitative estimate, we should plot the expectation value calculated by using $k \geq K_{\text{cut}}$ as a function of K_{cut}. At sufficiently large K_{cut}, the dependence on K_{cut} disappears, which means that the effect of the initial condition disappeared.

When we study more complicated probability distributions, often we do not even know the rough shape of the target distribution. In such cases, we should plot several quantities, e.g., the energy or pressure if we are solving a physics problem. If they change monotonically, it is likely that the simulation has not thermalized yet. If the simulation reached thermalization, they would oscillate around the expectation values. In Fig. 4.5, at first x monotonically decreases, then it fluctuates about zero.

4.3.2 Autocorrelation

Some care is needed even after the thermalization. In Fig. 4.5, we zoomed in on the interval $1000 \leq k < 2000$ of Fig. 4.4. We can see the correlation between configurations, at least for 20 or 30 steps. This length (number of steps) that is needed for the autocorrelation to disappear is called the autocorrelation length. (A more quantitative estimate is given in Sect. 4.3.3.) If we treat correlated configurations as if they are independent, the statistical error is under-estimated. We cannot call two configurations independent unless they are separated by at least the autocorrelation length.

In order to estimate the expectation values precisely, sufficiently many independent configurations are needed. Otherwise, the expectation values fluctuate as the number of samples grows. The expression "the simulation thermalized" is used also to mean that sufficiently many independent configurations are collected, and the expectation values cease to fluctuate too much.

4.3.3 Jackknife Method

The Jackknife method [4–6] is an easy and effective way of estimating the autocorrelation. Here, for simplicity, we assume that the quantities we want to calculate can be obtained at each configuration.[2] Namely, we consider the quantities that can be expressed as a function of x as $f(x)$. For more generic cases, see Appendix D.

First we divide the samples into groups consisting of w configurations; the first group is $\{x^{(1)}, x^{(2)},\ldots,x^{(w)}\}$, the second group is $\{x^{(w+1)},x^{(w+2)},\ldots,x^{(2w)}\}$, and so on. Let the number of groups obtained this way be n. The average of $f(x)$ in the l-th group is

$$\tilde{f}^{(l,w)} \equiv \frac{1}{w} \sum_{j=(l-1)w+1}^{lw} f(x^{(j)}). \tag{4.8}$$

By using $\tilde{f}^{(l,w)}$, the Jackknife error is defined as

$$\Delta_w \equiv \sqrt{\frac{1}{n(n-1)} \sum_{l=1}^{n} \left(\tilde{f}^{(l,w)} - \overline{f}\right)^2}. \tag{4.9}$$

Here $\bar{f}$ is the average of $f(x)$ obtained by using all configurations. Namely, the Jackknife error is the standard error obtained by treating each of the n groups as an independent sample and regarding $\tilde{f}^{(l,w)}$ as the value obtained from those independent samples.

If there are sufficiently many samples, Δ_w grows gradually with w, and beyond some point (say at $w \geq w_c$) Δ_w becomes almost constant. The values of w_c and Δ_{w_c} obtained in this way give a reasonable estimate of the autocorrelation length and the statistical error, respectively.

In Fig. 4.6, the expectation value of x^2 and the Jackknife error Δ_w are shown. The error bar spreads quickly up to $w = 20$ or so, but after $w = 40$ there is almost no change. Hence $w = 50$ should be a safe choice. In Fig. 4.7, the values of $\tilde{f}^{(l,w)}$ calculated with $w = 50$ are plotted. Almost no autocorrelation is visible, and hence, we can safely regard each group as an independent sample. The result obtained this

[2] Quantities not in this class include the variance of the probability distribution. Another example is the mass of a composite particle in a physics simulation.

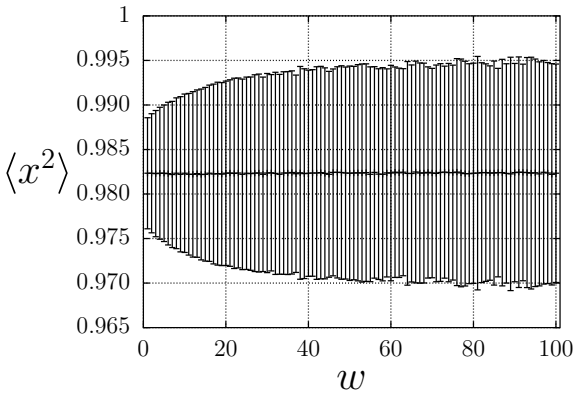

Fig. 4.6 The expectation value of x^2 and the Jackknife error

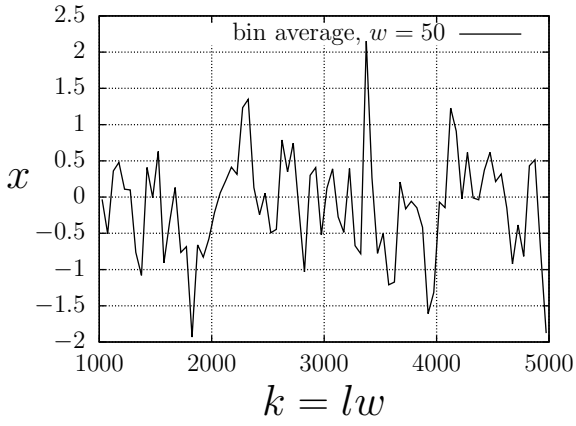

Fig. 4.7 The average of each group $\tilde{f}^{(l,w)}$, with the width $w = 50$

way was $\langle x^2 \rangle = 0.982 \pm 0.012$, which is in a reasonably good agreement with the analytic value $\langle x^2 \rangle = 1$.

Let us calculate a little bit by hand, to understand why the autocorrelation length can be estimated by using the Jackknife method. We consider two different widths w and $2w$ for the grouping. Then, by construction,

$$\tilde{f}^{(l,2w)} = \frac{\tilde{f}^{(2l-1,w)} + \tilde{f}^{(2l,w)}}{2}. \tag{4.10}$$

If n groups are obtained when the width is w, then $\frac{n}{2}$ groups are obtained when the width is $2w$. Hence the Jackknife error obtained by using the width $2w$ is

$$\Delta_{2w} = \sqrt{\frac{1}{\frac{n}{2}\left(\frac{n}{2}-1\right)} \sum_{l=1}^{n/2} \left(\tilde{f}^{(l,2w)} - \overline{f}\right)^2}$$
$$= \sqrt{\frac{4}{n(n-2)} \sum_{l=1}^{n/2} \left(\frac{\left(\tilde{f}^{(2l-1,w)} - \overline{f}\right)}{2} + \frac{\left(\tilde{f}^{(2l,w)} - \overline{f}\right)}{2} \right)^2}. \tag{4.11}$$

If the w is sufficiently large such that $\tilde{f}^{(2l-1,w)} - \overline{f}$ and $\tilde{f}^{(2l,w)} - \overline{f}$ can be interpreted as independent samples fluctuating about zero, then the products of independent quantities, $\left(\tilde{f}^{(2l-1,w)} - \overline{f}\right) \cdot \left(\tilde{f}^{(2l,w)} - \overline{f}\right)$, should be averaged to zero when summed over l. Therefore, approximately,

$$\Delta_{2w} \sim \sqrt{\frac{1}{n^2} \sum_{l=1}^{n} \left(\tilde{f}^{(l,w)} - \overline{f}\right)^2} \sim \Delta_w \tag{4.12}$$

has to hold. (Note that we assumed that n is sufficiently large.) Therefore, when w is larger than the autocorrelation length, Δ_w is almost constant.

4.3.4 Adjustment of the Step Size

To perform the simulation efficiently, we have to tune the parameters appropriately such that *more independent samples are generated with less cost*.[3] In the current case, the step size is the parameter to be tuned.

In the Metropolis algorithm, the candidate of the new configuration is accepted with the probability $\min(1, e^{-\Delta S})$, where ΔS is the increment of the action S. Let us consider the case of the Gaussian integral $S(x) = \frac{x^2}{2}$ again, and suppose that the transition from $x \sim 0$ to $x + \Delta x$ was proposed. Then, if $\Delta x \gg 1$, the acceptance probability is $\min(1, e^{-\Delta S}) = e^{-\Delta S} \ll 1$, namely, the candidate is rejected almost with 100% probability. In other words, only $\Delta x \lesssim 1$ has a reasonable chance of being accepted. If the step size c is too large, $\Delta x \lesssim 1$ is obtained only with probability $\frac{1}{c}$. Hence, the acceptance rate becomes very small. Furthermore, even when the value of x is updated, it just means the change Δx was of order 1, regardless of the value of c. Therefore, the acceptance rate is sacrificed for nothing, and the autocorrelation length increases proportionally to c. In such a parameter region, the product of the step size c and the acceptance rate becomes constant. On the other hand, if c becomes too small, the acceptance rate is almost 100%, but the amount of the change at each step decreases proportionally to c. Such a process can be regarded as the random

[3] Here, we assume that the "cost" is the amount of computation, or equivalently the time or the electricity bills for the computation. When we use parallel computers, we can save time by paying more electricity bills, so the notion of "cost" is not trivial.

Table 4.1 The relation between the step size c and the acceptance rate, measured from 10,000 samples

Step size c	Acceptance rate	$c \times$ acceptance rate
0.5	0.9077	0.454
1.0	0.8098	0.810
2.0	0.6281	1.256
3.0	0.4864	1.459
4.0	0.3911	1.564
6.0	0.2643	1.586
8.0	0.1993	1.594

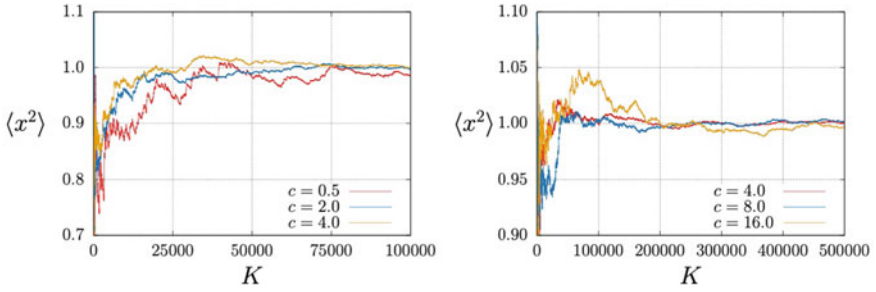

Fig. 4.8 $\langle x^2 \rangle = \frac{1}{K} \sum_{k=1}^{K} \left(x^{(k)}\right)^2$ with several different choices of the step size c. The simulation is not efficient when the step size is too big or too small

walk with the step size c. Typically, in the random walk, the change of the value of x after n steps is $c\sqrt{n}$. For this reason, the autocorrelation length is proportional to $\frac{1}{c^2}$. Therefore, the autocorrelation length will be minimized when c is not too large and not too small.

In Table 4.1, the acceptance rates at several values of c are summarized. The product of c and the acceptance rate is almost constant at $c > 4$, which suggests that c is too large there. At $c = 0.5$ and $c = 1.0$ the acceptance rate is high, which suggests c is too small. Hence, $c = 2.0 \sim 4.0$ appears to be the optimal choice.

In Fig. 4.8, we showed how $\langle x^2 \rangle$ converges to 1, for several values of c. With $c = 2.0$ or $c = 4.0$, faster convergence can be seen compared to smaller or larger c.

4.3.5 Box-Muller Method Revisited

The Box-Muller method introduced in Sect. 2.4.1 can be regarded as a special kind of Markov Chain Monte Carlo. Let us call the Gaussian random numbers generated by the Box-Muller method as $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$. We can confirm that they satisfy the conditions for MCMC:

- It is a Markov chain, namely, the probability that $x^{(k+1)}$ is obtained does not depend on $x^{(0)}, x^{(1)}, \ldots, x^{(k-1)}$. This condition is satisfied trivially—in fact $x^{(k+1)}$ does not even depend on $x^{(k)}$. Expressed as an equation, the transition probability is $T(x \to x') = P(x')$.
- The irreducibility is also trivially satisfied. Any transition can happen just by one step.
- It is easy to confirm the aperiodicity as well. Starting from x, there is a chance to come back to x after any number of steps.
- The detailed balance condition can be confirmed as follows:

$$P(x) \cdot T(x \to x') = P(x') \cdot T(x' \to x) = P(x) \cdot P(x'). \tag{4.13}$$

In the usual Markov Chain Monte Carlo methods such as the Metropolis algorithm, $x^{(k+1)}$ is obtained by slightly changing $x^{(k)}$, which inevitably leads to autocorrelation. In the Box-Muller method, $x^{(k+1)}$ is created without referring to $x^{(k)}$ and hence there is no autocorrelation. This is the reason that it is called a "random number". In this sense, the Box-Muller method is much better than the Metropolis algorithm. On the other hand, such efficient algorithms are known only for simple probability distributions. The Metropolis algorithm is very powerful because it can be applied to any probability distribution.

The Metropolis-Hastings algorithm (Sect. 5.3) and the Gibbs sampling algorithm (Sect. 5.2) are based on a very simple idea: less autocorrelation leads to higher efficiency. The Box-Muller method can also be regarded as a special case of these algorithms.

4.4 Examples Other Than the Gaussian Distribution

So far we have studied only the Gaussian distribution $P(x) = \frac{e^{-S(x)}}{Z} = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$. Some readers may wonder if such a simple method can always work, so let us see a few other examples.

First, we consider the superposition of two Gaussian distributions,

$$P(x) = \frac{e^{-\frac{(x-3)^2}{2}} + e^{-\frac{(x+3)^2}{2}}}{2\sqrt{2\pi}}. \tag{4.14}$$

The action $S(x)$ can be taken as

$$S(x) = -\log\left(e^{-\frac{(x-3)^2}{2}} + e^{-\frac{(x+3)^2}{2}}\right). \tag{4.15}$$

Hence we only have to rewrite two lines in the sample code, namely, we change

```
action_init=0.5e0*x*x;
```

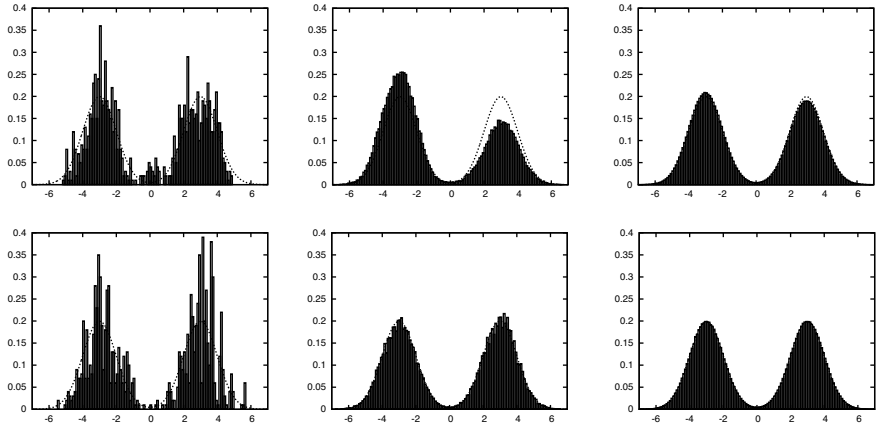

Fig. 4.9 The histograms of $x^{(1)}, x^{(2)}, \ldots, x^{(K)}$ with $K = 10^3, 10^5, 10^7$. The dashed lines are the target distribution $P(x) = \frac{e^{-\frac{(x-3)^2}{2}} + e^{-\frac{(x+3)^2}{2}}}{2\sqrt{2\pi}}$. The step size is 0.5 (top) or 5.0 (bottom). When the step size is 0.5, the deviation from the target distribution is visible even with 10^7 configurations. When the step size is 5.0, the convergence to the target distribution is much faster

and

```
action_fin=0.5e0*x*x;
```

to

```
action_init=-log(exp(-0.5e0*(x-3e0)*(x-3e0))+exp(-0.5e0*
(x+3e0)*(x+3e0)));
```

and

```
action_fin=-log(exp(-0.5e0*(x-3e0)*(x-3e0))+exp(-0.5e0*
(x+3e0)*(x+3e0)));
```

We performed simulations with step sizes 0.5 and 5.0, and have shown the histograms of x in Fig. 4.9. The top and bottom rows are for step size 0.5 and 5.0, respectively. The target distribution $P(x) = \frac{e^{-\frac{(x-3)^2}{2}} + e^{-\frac{(x+3)^2}{2}}}{2\sqrt{2\pi}}$ is shown with the dashed lines. We can see the convergence to the target distribution for both cases. When the step size is 5.0, convergence is achieved quickly. However, when the step size is 0.5, convergence is slower; even with 10^7 samples, the heights of the two peaks do not completely agree. What is the reason?

In the importance sampling, the configurations with smaller weights are avoided. In the current setup with two peaks, a bottleneck near $x = 0$ consists of low-weight configurations that are not sampled frequently. Therefore, if the step size is small, it is difficult to go through this bottleneck and reach the other peak. If the transitions between the two peaks happen frequently, the heights quickly become the same. Otherwise, more time is spent at one of the peaks and the heights differ for a long time.

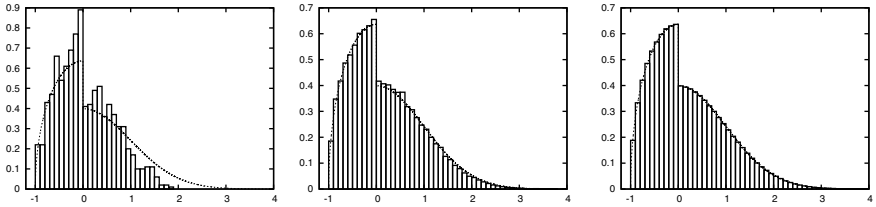

Fig. 4.10 The probability distribution (4.16) reproduced by using the Metropolis algorithm with the step size 0.5. The histograms of $x^{(1)}, x^{(2)}, \ldots, x^{(K)}$ with $K = 10^3, 10^5, 10^7$ are shown

As explained in Sect. 3.2, such a bottleneck can effectively break the irreducibility. (Of course, the irreducibility is not completely broken, so if we wait very long we will get the right answer. However a "very long time" can often be more than the human lifespan.) When the step size is 5.0, configurations can jump over the bottleneck and go directly from one peak to the other, so a quick convergence can be achieved. In Sect. 4.7.1, we consider a more extreme example, $P(x) = \frac{e^{-\frac{x^2}{2}} + e^{-\frac{(x-100)^2}{2}}}{2\sqrt{2\pi}}$. Please think about the optimum step size for that case. (In Sect. 6.3.3, we will introduce the replica-exchange method, which can be used for more complicated problems.)

As yet another example, let us consider a combination of the semi-circle distribution at $x < 0$ and the Gaussian distribution at $x \geq 0$:

$$P(x) = \begin{cases} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} & (x \geq 0) \\ \frac{2}{\pi}\sqrt{1-x^2} & (-1 \leq x < 0) \\ 0 & (x < -1) \end{cases} \tag{4.16}$$

The probability is zero at $x < -1$, hence $x' < -1$ is always rejected. (Equivalently, we take $S(x) = \infty$ at $x < -1$.) The distribution obtained in this way is shown in Fig. 4.10. As expected, $x < 0$ is a semi-circle, and $x > 0$ is Gaussian. In this case, the choice of the step size is not as important as in the last example because there is no bottleneck.

4.5 Application to Complicated Integrals

In Markov Chain Monte Carlo, the expectation values can be obtained, but the partition function Z cannot be calculated directly. In many cases, the partition function is merely a normalization factor that is not particularly useful. However, sometimes we happen to be interested in the value of Z itself. In such a case, how can we calculate it?

When there is only one variable, we can plot the probability distribution $P(x) = \frac{e^{-S(x)}}{Z}$ obtained via MCMC and take the ratio with $e^{-S(x)}$. However, this method does

not work when there are many variables. Below, we show a method that can easily be generalized to multivariate distributions.

The action $S(x)$ can be any complicated real-valued function, as long as the partition function $Z = \int dx e^{-S(x)}$ is finite. If the action is a simple function, for example $S_0(x) = \frac{x^2}{2}$, the partition function can be calculated analytically, as $Z_0 = \int dx e^{-S_0(x)} = \sqrt{2\pi}$. By using such an S_0, we can calculate the ratio of Z and Z_0 via MCMC:

$$\frac{Z}{Z_0} = \frac{1}{Z_0} \int dx e^{-S_0} \cdot e^{S_0 - S} = \left\langle e^{S_0 - S} \right\rangle_0 . \tag{4.17}$$

Here $\langle \cdot \rangle_0$ is the expectation value calculated by using e^{-S_0} as the weight. If we know Z_0, we can get the value of Z as well.

As an example, let us consider

$$S(x) = \begin{cases} -\frac{1}{2}\log(1-x^2) & (-1 < x < 1) \\ \infty & (x < -1, x > 1) \end{cases} \tag{4.18}$$

Then the partition function Z has to be the area of the semi-circle, $\frac{\pi}{2}$. If we calculate the expectation value of $e^{S_0(x) - S(x)}$ which is expressed as

$$e^{S_0(x) - S(x)} = \begin{cases} e^{\frac{1}{2}x^2}\sqrt{1-x^2} & (-1 < x < 1) \\ 0 & (x < -1, x > 1) \end{cases} \tag{4.19}$$

we should obtain

$$\frac{Z}{Z_0} = \frac{\pi}{2} \cdot \frac{1}{\sqrt{2\pi}} = 0.6266.... \tag{4.20}$$

In the left panel of Fig. 4.11, we can see the convergence to this value.

In principle, this method always works. In practice, however, it works only when the probability distributions $P(x) = \frac{e^{-S(x)}}{Z}$ and $P_0(x) = \frac{e^{-S_0(x)}}{Z_0}$ have a sufficiently large overlap. To illuminate this point, let us consider $S(x) = \frac{(x-\alpha)^2}{2}$. In this case, $P(x)$ and $P_0(x)$ are peaked around $x = \alpha$ and $x = 0$, respectively. The weight factor to be calculated via MCMC is

$$\frac{P(x)}{P_0(x)} = e^{S_0(x) - S(x)} = e^{\frac{x^2}{2} - \frac{(x-\alpha)^2}{2}} . \tag{4.21}$$

If α is very large, say $\alpha = 100$, the values of $e^{S_0 - S}$ appearing in the simulation are almost always extremely small, e.g., e^{-5000}, because the value of x in the probability distribution P_0 is typically 1. However, once in e^{+5000} configurations or so, x can become as large as 100, and then $e^{S_0 - S}$ takes a large value like e^{+5000}. After taking the average over infinitely many configurations, we obtain $\frac{Z}{Z_0} = 1$. However, we will not see the configurations dominating this average, $x \sim 100$, during a realistic

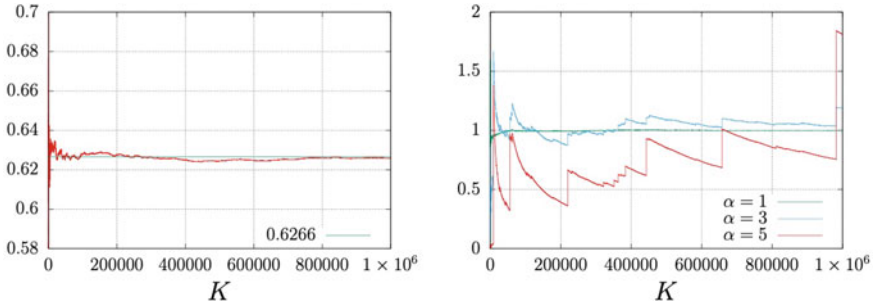

Fig. 4.11 [Left] The expectation value of (4.19) calculated by using the Metropolis algorithm with step size 4. [Right] The expectation value of (4.21) calculated via the Metropolis algorithm with step size 4. The parameter α is 1, 3, or 5. The horizontal axis is the number of configurations K used for the calculation of the expectation value

simulation time. Obviously, we cannot get the right answer in any practical sense. In the right panel of Fig. 4.11, we can see that the convergence is getting significantly slow already at $\alpha = 3$. At $\alpha = 5$, we see a large deviation from the right answer $\langle e^{\frac{x^2}{2} - \frac{(x-\alpha)^2}{2}} \rangle_0 = 1$ even with 1,000,000 configurations. This issue is called the overlap problem because it happens due to the lack of overlap between $P(x)$ and $P_0(x)$.[4]

In this example, the overlap problem can be resolved rather easily. Let us be less ambitious and split the problem into M tasks. We choose a chain of actions S_0, S_1, S_2, ..., $S_M = S$, in such a way that S_n and S_{n+1} are sufficiently close. For example, we can use $S_n = \frac{1}{2}\left(x - \frac{\alpha n}{M}\right)^2$, with $\frac{\alpha}{M} \sim 1$. Then $\frac{Z_{n+1}}{Z_n}$ (where $Z_n \equiv \int dx e^{-S_n(x)}$) can be calculated without suffering from a serious overlap problem. By calculating $\frac{Z_1}{Z_0}$, $\frac{Z_2}{Z_1}$, ..., $\frac{Z_M}{Z_{M-1}}$, we can determine $Z = Z_M$. Similar methods can be used for more complicated $S(x)$ and Z as well. One of the authors applied this method to a complicated integral in a physics problem [7].

4.6 Sign Problem

All the arguments above assumed $e^{-S(x)} \geq 0$ so that it can be regarded as a probability. This assumption is not valid in many important applications in physics, i.e., the weight $e^{-S(x)}$ can be negative or complex at certain values of x. Then the Markov Chain Monte Carlo methods are not directly applicable. This is the infamous sign problem.[5] For reviews, see e.g., Refs. [8, 9]. The sign problem in physics is one of the biggest motivations for the quantum computer [10, 11].

[4] The negative sign problem, which will be discussed in Sect. 4.6, can be regarded as a version of the overlap problem.

[5] When $e^{-S(x)}$ is complex, it is also called the "phase problem", but even in that case "sign problem" is more commonly used.

There is no generic solution to the sign problem known to date. The sign problem is a very difficult problem that is related to one of the biggest issues in computer science and mathematics: the $P \neq NP$ conjecture. If there were a generic solution, it would mean $P = NP$ [12], and hence it is widely believed that a generic solution cannot exist. However, several case-by-case solutions specific to concrete problems are known. Also, sometimes we can beat the sign problem by brute force, simply by investing a lot of computational resources. In this section, we introduce a typical example of such a brute-force method: the reweighting method.

Suppose $e^{-S(x)}$ is complex. We write it as a product of the absolute value $|e^{-S(x)}| = e^{-S_0(x)}$ and the complex phase $e^{i\theta(x)}$:

$$e^{-S(x)} = e^{-S_0(x)} \times e^{i\theta(x)}. \tag{4.22}$$

It is straightforward to perform MCMC by adopting e^{-S_0} as the weight. (This is called the phase-quenched simulation.) We use $\langle \cdot \rangle_0$ to denote the expectation value with this weight. Then

$$\int dx e^{-S(x)} = \langle e^{i\theta(x)} \rangle_0 \times \int dx e^{-S_0(x)}, \tag{4.23}$$

and hence, by calculating $\langle e^{i\theta(x)} \rangle_0$ and $\int dx e^{-S_0(x)}$ we can determine $\int dx e^{-S(x)}$. Note that we can use the method explained in Sect. 4.5 to calculate $\int dx e^{-S_0(x)}$.

This method is simple in principle, but often $e^{i\theta}$ fluctuates very violently and $\langle e^{i\theta} \rangle_0$ becomes very close to zero. This is often the case when there are many variables in the integral. In case such large fluctuations appear, we need to determine $\langle e^{i\theta} \rangle_0$ very precisely, which makes the simulation harder.

The calculation of the expectation value $\langle f(x) \rangle$ is also simple, by using

$$\langle f(x) \rangle = \frac{\langle f(x) e^{i\theta} \rangle_0}{\langle e^{i\theta} \rangle_0}. \tag{4.24}$$

In this case, again, both the numerator and denominator can become very small if $e^{i\theta}$ fluctuates a lot.

The sign problem becomes particularly severe when the presence of the phase factor affects the configurations dominating the integral.[6] In such cases, the sign problem can be regarded as a version of the overlap problem.

[6] In such cases, contributions from dominant configurations in the phase-quenched simulation are canceled by particularly violent oscillations of $e^{i\theta}$ and a new peak emerges from the tail of the phase-quenched probability distribution.

4.7 Common Mistakes

In this section, we show a few common mistakes among beginners. These mistakes are sometimes made by experts, too.

4.7.1 Changing Step Size in the Middle of the Simulation

Imagine there is a bottleneck in the probability distribution, like in Fig. 4.9. As an extreme example, we can consider $S(x) = -\log\left(e^{-\frac{x^2}{2}} + e^{-\frac{(x-100)^2}{2}}\right)$. Then $e^{-S(x)}$ has two peaks around $x = 0$ and $x = 100$, in between $e^{-S(x)}$ is almost zero. In such cases, it is not easy to sample the entire distribution because it is difficult to go through the bottleneck. Also, it often happens that the simulation is trapped in a special configuration surrounded by bottlenecks, gets stuck there for a long time, and the acceptance rate becomes very low. Then, you might be tempted to change the step size, say make it smaller so that the acceptance rate goes up. However, if you do that, you get a wrong result. *You must not change the step size in the middle of the simulation*.

Still, it is totally fine to combine multiple step sizes. *As long as the conditions explained in Chap. 3 are not broken, we can do whatever we like*. For example, you can take step size $c = 1$ for the even steps and $c = 100$ for the odd steps. With this choice, at the odd steps, the transition between the peaks around $x = 0$ and $x = 100$ can take place, and hence the entire distribution can be sampled (Fig. 4.12). Alternatively, we can set the step size by throwing a dice. Namely, at each step, you can choose the step size to be $c = 1, 2, 3, 4, 5$ or 6 with probability $\frac{1}{6}$ for each value, without breaking the conditions listed in Chap. 3. We can do whatever, so we recommend you try various options.

When the configurations are far from thermalization, the simulation often gets stuck at a special configuration. To avoid this, we can choose a good initial configuration (if we know about basic features of the probability distribution), or we can take the step size to be small at the beginning and then switch to a larger step size after thermalization. There is no problem as long as we use the same step size when we calculate the expectation values. Because the configurations before thermalization are simply discarded, we can change the step size during the thermalization process.[7]

[7] Another common technique is to skip the Metropolis test at the early stage of the thermalization process. This technique is particularly powerful when it is combined with the HMC algorithm introduced in Sect. 5.1. Because those configurations are not used for the calculation of the expectation value, there is no need for the detailed balance condition there.

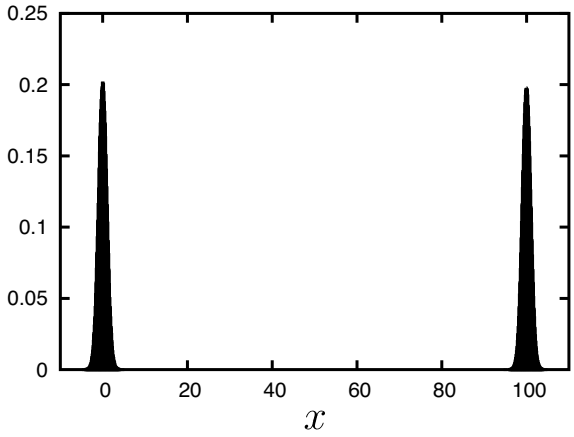

Fig. 4.12 The probability distribution $P(x) = \frac{e^{-\frac{x^2}{2}} + e^{-\frac{(x-100)^2}{2}}}{2\sqrt{2}}$ reproduced by using the Metropolis algorithm with the step size $c = 1$ for the even steps and $c = 100$ for the odd steps. The histogram of 10,000,000 samples is shown. The target distribution $P(x)$ is also drawn, but it is invisible to the naked eyes because it agrees well with the histogram

4.7.2 Mixing the Configurations Obtained by Using Different Step Sizes

This example is similar to the one in Sect. 4.7.1. If several sequences of the configurations are obtained by using different step sizes, the convergence to the right answer is guaranteed for each step size. However, if the simulations are terminated at finite numbers of steps and the configurations obtained by using different step sizes are mixed, the statistical error might become uncontrollable, and unreasonable results may be obtained. Still, if the autocorrelation length is properly estimated for each step size, it is possible to take the average by mixing the independent samples.

If several sequences of the configurations are obtained by using the same step size, there is no problem in using all the configurations for the analyses, as long as each run is sufficiently well thermalized.

4.7.3 "Random Numbers" Were Not Really Random

As we have mentioned, the "random numbers" used in numerical simulations are actually pseudorandom. Therefore, we can make unexpected mistakes if we are not careful enough. For example, what happens if we repeat the same sequence of pseudorandom numbers every 1000 steps? The outcome is shown in Fig. 4.13. Obviously, the answer is wrong.

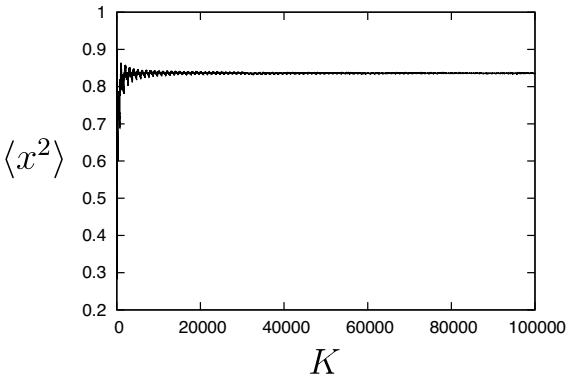

Fig. 4.13 *Wrong* example of the Gaussian integral with Metropolis algorithm, step size $c = 1$. The same sequence of pseudorandom numbers was repeated every 1000 steps (i.e., "random numbers" were not really random). In the correct simulation, $\langle x^2 \rangle = \frac{1}{K} \sum_{k=1}^{K} \left(x^{(k)} \right)^2$ has to converge to 1. However, in this example, convergence to a wrong value is observed

Such mistakes happen very easily. Suppose we work on a very large-scale simulation that takes several months. Then we have to split the task into small jobs which can be finished at a short time scale, say one day or an hour. For example, 10 steps are processed in each job, and we repeat 1000 jobs so that 10000 steps are obtained in total. Then if we make an error in the setting of the seeds, the same sequence of pseudorandom numbers would be repeated every 10 steps.

In order to avoid such mistakes, it is better to save both the configuration and the information regarding the pseudorandom numbers at the end of each job. It is kind of tedious at first, but once you write a code, you can copy-and-paste the same one.

4.8 Multivariate Metropolis Algorithm

So far, we have only dealt with the univariate distributions. Although all the essence is there, it is instructive to learn about the cases with many variables that suffer from the curse of dimensionality. In this section, we generalize the Metropolis algorithm to multivariate distributions.

This generalization is very simple. Let the variables be $(x_1, x_2, \ldots, x_n)$. Because there are multiple variables, there are roughly two possible ways we can update them. The first one is ⟨Update simultaneously⟩:

Multivariate Metropolis ⟨Update simultaneously⟩

1. For all $i = 1, 2, \ldots, n$, choose Δx_i randomly from $[-c_i, +c_i]$, and propose $x_i' \equiv x_i^{(k)} + \Delta x_i$ as a candidate of $x_i^{(k+1)}$. The step sizes $c_1, c_2, \ldots, c_n$ can be different from each other.
2. Metropolis test: the candidate $\{x'\}$ is accepted and the value of $\{x\}$ is updated as $\{x^{(k+1)}\} = \{x'\}$ with a probability $\min(1, e^{S(\{x^{(k)}\}) - S(\{x'\})})$. Otherwise $\{x'\}$ is rejected and the value of $\{x\}$ remains unchanged, as $\{x^{(k+1)}\} = \{x^{(k)}\}$.

The other one is ⟨Update one by one⟩:

Multivariate Metropolis ⟨Update one by one⟩

1. Choose Δx_1 randomly from $[-c_1, +c_1]$, and take $x_1' \equiv x_1^{(k)} + \Delta x_1$. Other variables are left untouched, $x_i' \equiv x_i^{(k)}$ $(i = 2, 3, \ldots, n)$.
2. Metropolis test: the candidate $\{x'\}$ is accepted and the value of $\{x\}$ is updated as $\{x^{(k+1)}\} = \{x'\}$ with a probability $\min(1, e^{S(\{x^{(k)}\}) - S(\{x'\})})$. Otherwise $\{x'\}$ is rejected and the value of x remains unchanged, as $\{x^{(k+1)}\} = \{x^{(k)}\}$. (Whether the proposal is accepted or not, $x_2, x_3, \ldots, x_n$ are left untouched.)
3. Update x_2 in the same manner. ($x_1, x_3, \ldots, x_n$ are left untouched.)
4. In the same manner, update $x_3, \ldots, x_n$ one by one.

For the Metropolis test, a uniform random number r between 0 and 1 is generated and the candidate $\{x'\}$ is accepted if $r < e^{S(\{x^{(k)}\}) - S(\{x'\})}$.

We recommend that the reader check that, either way, the four conditions of Markov Chain Monte Carlo explained in Chap. 3 are satisfied. (Strictly speaking, the situation is a little bit subtle in ⟨Update one by one⟩; see the exercise at the end of this chapter.)

When there are many variables, ⟨Update simultaneously⟩ usually forces us to take the step size c_i small, because otherwise the acceptance rate becomes low. On the contrary, ⟨Update one by one⟩ allows us to take the step size relatively large. Furthermore, in case the variable x_i interacts with only a few other variables (e.g., the nearest-neighbor interaction which can be written as $S = f(x_1, x_2) + f(x_2, x_3) + \ldots f(x_{n-1}, x_n)$) x_i can be updated by calculating only the terms containing x_i (in the example above, $f(x_{i-1}, x_i) + f(x_i, x_{i+1})$), and hence, the computational cost can be reduced.

Either way, a different value of step size c_i can be used for each variable x_i. Please check that the detailed balance condition is still satisfied, as an instructive exercise. In case the widths of the probability distribution heavily depend on the variables, (there can be exceptions, but in many cases) the simulation becomes more efficient if c_i is proportional to the width. When we do not have a good guess regarding the width, we can try several different step sizes and see how the acceptance rate changes.

4.8.1 Multivariate Gaussian Distribution

Let us consider the multivariate Gaussian distribution[8]

$$S(x_1, \ldots, x_n) = \frac{1}{2} \sum_{i,j=1}^{n} A_{ij} x_i x_j \qquad (A_{ij} = A_{ji}). \tag{4.25}$$

As an example, we take $n = 2$. We write $x_1 = x$ and $x_2 = y$, and choose the coefficients to be $A_{11} = 1$, $A_{22} = 1$, $A_{12} = \frac{1}{2}$. Then, the action $S(x, y)$ becomes[9]

$$S(x, y) = \frac{x^2 + y^2 + xy}{2}. \tag{4.26}$$

The term $\frac{1}{2}xy$ in $S(x, y)$ introduces the correlation between x and y. For example, we can imagine that x and y parametrize a person's mathematical skills and baseball skills, respectively. Larger x means better mathematical skills, and larger y means better baseball skills. Zero is average, and a large negative value means bad skill. If $S(x, y) = \frac{x^2+y^2}{2}$ and $P(x, y) \propto e^{-\frac{x^2+y^2}{2}}$, the values of x and y are not correlated at all, which means whether one is good or bad at mathematics is not related to baseball skills. If $S(x, y) = \frac{x^2+y^2+xy}{2}$ and $P(x, y) \propto e^{-\frac{x^2+y^2+xy}{2}}$, it is unlikely that one is good both at math and baseball or bad both at math and baseball, rather if one is good at math or baseball he/she is likely to be bad at the other. It would be a reasonable assumption because they have to split a finite amount of time to the study of mathematics and practice of baseball, and also because God usually does not bless a person twice.

We use the Metropolis algorithm to generate this probability distribution. If the number of variables is as small as 2, there is no big difference between ⟨Update simultaneously⟩ and ⟨Update one by one⟩. Here we use the former. Below we show some sample code written in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int main(void){
  int niter=10000;
  double step_size_x=0.5e0;
  double step_size_y=0.5e0;
```

[8] See Appendix B.2 for basic properties.

[9] If we consider a slightly more generic version $S(x, y) = \frac{x^2+y^2+2Axy}{2}$, the same kind of calculation applies to $-1 < A < 1$. If $A \geq 1$ or $A \leq -1$, then $P(x, y) \propto e^{-S(x,y)}$ does not make sense as a probability distribution. Why? (Hint: See Sect. 6.1.2.)

```
  srand((unsigned)time(NULL));
  /*********************************/
  /* Set the initial configuration */
  /*********************************/
  double x=0e0;
  double y=0e0;
  int naccept=0;
  /*************/
  /* Main loop */
  /*************/
  for(int iter=1;iter<niter+1;iter++){
    double backup_x=x;
    double backup_y=y;
    double action_init=0.5e0*(x*x+y*y+x*y);

    double dx = (double)rand()/RAND_MAX;
    double dy = (double)rand()/RAND_MAX;
    dx=(dx-0.5e0)*step_size_x*2e0;
    dy=(dy-0.5e0)*step_size_y*2e0;
    x=x+dx;
    y=y+dy;
    double action_fin=0.5e0*(x*x+y*y+x*y);
    /*******************/
    /* Metropolis test */
    /*******************/
    double metropolis = (double)rand()/RAND_MAX;
    if(exp(action_init-action_fin) > metropolis){
      /* accept */
      naccept=naccept+1;
    }else{
      /* reject */
      x=backup_x;
      y=backup_y;}
    /***************/
    /* data output */
    /***************/
    // output the results every ten steps.
    if(iter%10==0){
      printf("%.10f   %.10f   %f\n",x,y,(double)naccept/
      iter);}
  }
}
```

The code is almost identical to the one for $S(x) = \frac{x^2}{2}$ shown in Sect. 4.2. The only differences are that there is one more variable y and $S(x) = \frac{x^2}{2}$ is replaced with $S(x, y) = \frac{x^2+y^2+xy}{2}$. The same applies even if there are thousands or millions of variables, and even if the probability distribution is a very complicated function.

As we have seen, different step sizes can be used for x and y. However, in this specific example, it is natural to use the same step size because x and y appear in $S(x, y)$ symmetrically. We used the Metropolis algorithm with the step size $c = 0.5$,

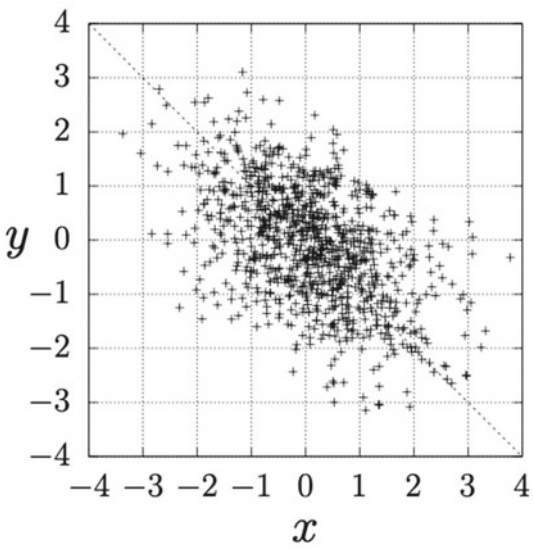

Fig. 4.14 A scatter plot of (x, y) that follows the bivariate Gaussian distribution $P(x, y) \propto e^{-\frac{x^2+y^2+xy}{2}}$. We used the Metropolis algorithm, with the step size 0.5 for both x and y. One configuration is sampled every 10 steps, and 10,000 configurations are collected in total

and output the configuration (x, y) every ten steps. We collected 1,000 configurations (10,000 steps) and plotted them in Fig. 4.14. (In this case, we know the probability distribution has the largest weight at $x = y = 0$. Therefore we set the initial configuration to be $x = y = 0$, to save the time for thermalization.) The dashed diagonal line is $y = -x$. The points (x, y) are distributed roughly along this line, and hence, we can confirm the tendency that "if x is a large positive value (good at mathematics) then y is a large negative value (bad at baseball)" and "if x is a large negative value (bad at mathematics) then y is a large positive value (good at baseball)". As a point (x, y) goes further from the center of the distribution ($(x, y) = (0, 0)$), the density becomes lower.

Now we have generated the probability distribution. As an application, let us design a life plan. Suppose some boys and girls do not know what they are good at, and they are wondering whether they should become a physicist or a baseball player. They would want to know the expectation value of their income as one of the factors for their career choice. For such an estimate, we need "salary functions" s_{physics} and s_{baseball} that relate the math and baseball skills to the salaries as a physicist or baseball player.

To become a physicist, it is better to be good at mathematics. However, even if one is not good at mathematics, somehow it is possible to survive and write papers. In this sense, some income is expected even if x is small. Another crucial fact is that, even if one is extremely good at mathematics, a physicist's salary cannot be huge. It is reasonable to assume that baseball skills do not affect a salary as a physicist, so we assume the salary function of physicists is a function of math skill x only and takes the following form:

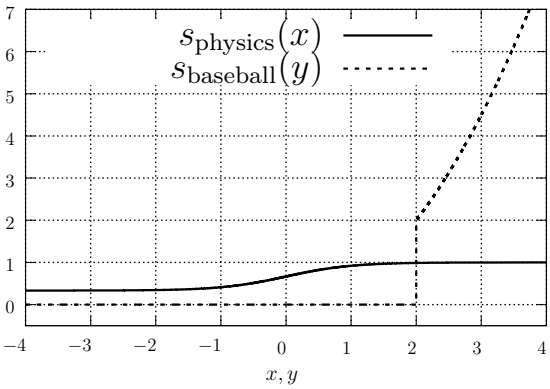

Fig. 4.15 Math skills x vs salary as physicist $s_{\text{physics}}(x)$ (solid line) and baseball skills y vs salary as baseball player $s_{\text{baseball}}(y)$ (dashed line)

$$s_{\text{physics}}(x) = \frac{2 + \tanh x}{3}. \tag{4.27}$$

The graph is shown in Fig. 4.15. Overall, this function is very smooth.

We expect that the salary function of baseball players s_{baseball} is very different. Unless one is exceptionally good at baseball, namely, unless y is sufficiently large, one cannot become a professional baseball player. Hence, the salary is zero below a certain threshold. However, if one actually becomes a professional player, their salary can be astronomical. Therefore, salary increases quickly beyond the threshold value of y. Probably, math skills do not affect the salary as a baseball player, so let us assume that s_{baseball} is a function of y only, and takes the following form:

$$s_{\text{baseball}}(y) = \begin{cases} 0 & (y \le 2) \\ \frac{y^2}{2} & (y > 2) \end{cases} \tag{4.28}$$

The graph is shown in Fig. 4.15, together with s_{physics}. A discontinuity at $y = 2$ is an important feature.

Let us calculate the expectation values $\langle s_{\text{physics}}(x) \rangle$ and $\langle s_{\text{baseball}}(y) \rangle$ by using the configurations generated via the Metropolis algorithm.

Let us see $\langle s_{\text{physics}}(x) \rangle$ first. Analytically, we can show that $\langle s_{\text{physics}}(x) \rangle = 2/3 = 0.66\ldots$. Can we reproduce this value via MCMC? We repeated the simulation 100 times with different sequences of the pseudorandom numbers, and calculated the average and standard deviation of 100 streams. The result is shown in Fig. 4.16. The horizontal axis is the number of samples K used to calculate the expectation value. We picked up four typical streams and showed them by purple lines. As K becomes large, the expectation value quickly converges to the exact analytic value. The convergence is quick because $s_{\text{physics}}(x)$ does not change violently with x (the difference is at most factor three) and all configurations give similar contributions. As a result, all configurations are effectively used and values close to the right answer can be obtained without using too many configurations.

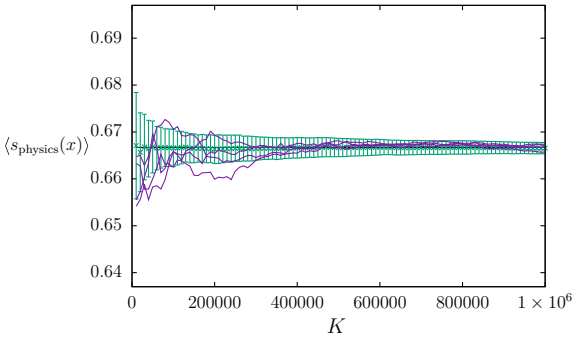

Fig. 4.16 The expectation value of $s_{\text{physics}}(x)$ is calculated by using the Gaussian distribution $P(x, y) \propto e^{-\frac{x^2+y^2+xy}{2}}$. The horizontal axis is the number of configurations K used for the calculation. 100 independent simulations are performed by using different sequences of random numbers. The average and the standard deviation of the 100 streams are shown in green. The purple lines are four typical streams. Quick convergence to the right answer is observed

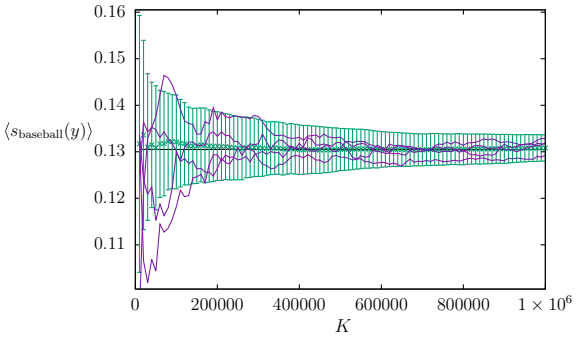

Fig. 4.17 The expectation value of $s_{\text{baseball}}(y)$ is calculated by using the Gaussian distribution $P(x, y) \propto e^{-\frac{x^2+y^2+xy}{2}}$. As in Fig. 4.16, the horizontal axis is the number of configurations K used for the calculation. 100 independent simulations are performed by using different sequences of the random numbers. The average and the standard deviation of the 100 streams are shown in green. The purple lines are four typical streams. Because $s_{\text{baseball}}(y)$ is an extreme function that has a big jump, the convergence to the right answer is rather slow and many configurations are needed for a precise estimate

Next, let us calculate $\langle s_{\text{baseball}}(y)\rangle$. The answer is $\langle s_{\text{baseball}}(y)\rangle = 0.1305\ldots$. Can we reproduce this value, as we did for $\langle s_{\text{physics}}(x)\rangle$? The result is shown in Fig. 4.17, by using the same scale as Fig. 4.16. This plot looks very different from that for $\langle s_{\text{physics}}(x)\rangle$; even with a fairly large-K, the expectation values from different streams differ significantly and the convergence to the right value is very slow. This is because rare configurations ($y > 2$) have large contributions while the majority of the configurations ($y \leq 2$) do not contribute at all. In other words, the peak of the probability distribution $P(x, y)$ and the configurations important for the expectation value do

not overlap. If we calculate the expectation value in such a situation, wrong answers may be obtained unless we collect a very large number of configurations. Although, in the current case, we can increase the number of configurations by brute force, if the integrand is even more extreme (say, if only $y > 10$ contribute), or if the simulation is costly and it is hard to collect many configurations, some improvement is needed. Below, we will explain a method to solve this problem.

Solving the Overlap Problem

As some of you might have noticed, this is essentially the overlap problem explained in Sect. 4.5, and hence, can be resolved in the same manner. For example, let us define a new action by shifting the value of y by α, as

$$S(x, y; \alpha) = \frac{x^2 + (y-\alpha)^2 + x(y-\alpha)}{2}. \tag{4.29}$$

The partition function associated with $S(x, y; \alpha)$ is defined as

$$Z_\alpha = \int dx \int dy e^{-S(x,y;\alpha)}. \tag{4.30}$$

Then we can express $\langle s_{\text{baseball}}(y)\rangle$ in the following manner, by using $\{\alpha_i\}$ $(i = 1, \ldots, M)$:

$$\begin{aligned}\langle s_{\text{baseball}}(y)\rangle &= \frac{1}{Z_0}\int dx \int dy s_{\text{baseball}}(y) e^{-S(x,y;0)} \\ &= \frac{Z_{\alpha_1}}{Z_0}\cdot\frac{Z_{\alpha_2}}{Z_{\alpha_1}}\cdot\frac{Z_{\alpha_3}}{Z_{\alpha_2}}\cdots\frac{Z_{\alpha_M}}{Z_{\alpha_{M-1}}}\cdot\frac{1}{Z_{\alpha_M}}\int dx\int dy s_{\text{baseball}}(y) e^{-S(x,y;0)}.\end{aligned} \tag{4.31}$$

Let us denote the expectation value of a function $f(x, y)$ with the weight $e^{-S(x,y;\alpha)}$ as

$$\langle f(x, y)\rangle_\alpha = \frac{1}{Z_\alpha}\int dx \int dy f(x, y) e^{-S(x,y;\alpha)}. \tag{4.32}$$

Then, the partition functions can be written as

$$Z_{\alpha_{i+1}} = \int dx \int dy e^{-S(x,y;\alpha_{i+1})} = \int dx \int dy e^{-S(x,y;\alpha_{i+1})+S(x,y;\alpha_i)} e^{-S(x,y;\alpha_i)} \tag{4.33}$$

and hence the ratio is

$$\frac{Z_{\alpha_{i+1}}}{Z_{\alpha_i}} = \left\langle e^{-S(x,y;\alpha_{i+1})+S(x,y;\alpha_i)}\right\rangle_{\alpha_i} \equiv \left\langle e^{-\Delta_{i+1,i}}\right\rangle_{\alpha_i}. \tag{4.34}$$

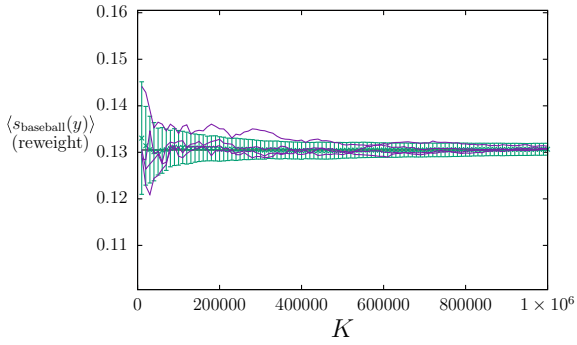

Fig. 4.18 The expectation value of $s_{\text{baseball}}(y)$ is calculated by using the reweighting method (4.35). We took $M = 2$, $\alpha_1 = 1.5$ and $\alpha_2 = 3$. As expected, the convergence to the right answer is fast

Here we used a shorthand notation $\Delta_{i+1,i} \equiv S(x, y; \alpha_{i+1}) - S(x, y; \alpha_i)$. Therefore, $\langle s_{\text{baseball}}(y)\rangle$ written in the form of (4.31) can be expressed as a product of the expectation values with the probability weights $e^{-S(x,y;\alpha_i)}$ $(i = 0, \ldots, M)$,

$$\langle s_{\text{baseball}}(y)\rangle = \left\langle e^{-\Delta_{1,0}}\right\rangle_{\alpha_0} \left\langle e^{-\Delta_{2,1}}\right\rangle_{\alpha_1} \cdots \left\langle e^{-\Delta_{M,M-1}}\right\rangle_{\alpha_{M-1}} \left\langle e^{-\Delta_{0,M}} s_{\text{baseball}}(y)\right\rangle_{\alpha_M} . \quad (4.35)$$

Here we set $\alpha_0 = 0$. The key point of this deformation is that, by choosing $\{\alpha_1, \ldots, \alpha_M\}$ appropriately, all expectation values $\left\langle e^{-\Delta_{i+1,i}}\right\rangle_{\alpha_i}$ and $\left\langle e^{-\Delta_{0,M}} s_{\text{baseball}}(y)\right\rangle_{\alpha_M}$ appearing in (4.35) can be calculated without encountering the overlap problem. To calculate $\left\langle e^{-\Delta_{i+1,i}}\right\rangle_{\alpha_i}$ efficiently, we just have to take α_i and α_{i+1} sufficiently close. To calculate the last term $\left\langle e^{-\Delta_{0,M}} s_{\text{baseball}}(y)\right\rangle_{\alpha_M}$, we should take α_M to be 2 or 3.

We applied this method with $M = 2$, $\alpha_1 = 1.5$, and $\alpha_2 = 3$. The result is shown in Fig. 4.18. Although we divided the task into only three pieces $(i = 0, 1, 2)$, the convergence to the correct value became much quicker than the naive approach (Fig. 4.17). This is evidence that the overlap problem was resolved. It may not be a very sophisticated method, but that we use MCMC would mean we focus on the practical utility rather than the beauty, so we should not mind!

Now we could design a life plan. The expectation value of the salary as a physicist is about 0.667, while as a baseball player about 0.131 is expected. If the talent is unknown, the probability distribution used here and the salary functions are reasonable, and the salary is an important factor in life, then one should become a physicist rather than a baseball player. Note however that, for people who seriously care about salary, there are better jobs than a physicist.

Distributions with Many Variables

Even if there are a lot more variables, say $n = 100$, the same method can be used. It is better to update the variables one by one because the acceptance rate can become very small otherwise, unless the step size is very small.

When x_a is varied to $x_a + \Delta x_a$, the change of $S(x_1, \ldots, x_n) = \frac{1}{2}\sum_{i,j=1}^{n} A_{ij}x_i x_j$ can be calculated just by looking at the terms containing x_a:

$$\frac{1}{2}A_{aa}x_a x_a + \sum_{i \neq a} A_{ia}x_i x_a \tag{4.36}$$

Among $\frac{n(n+1)}{2}$ terms in $S(x_1, \ldots, x_n)$ (n terms for $i = j$ and $\frac{n(n-1)}{2}$ terms for $i < j$; no need for considering $i > j$ because $A_{ij} = A_{ji}$), there are only n terms that contain x_a, so the computational cost can be reduced drastically.

4.9 Exercises

1. In the main text, when $x' = x^{(k)} + \Delta x$ is proposed as a candidate of $x^{(k+1)}$, Δx was taken from the uniform random number. Actually, other kinds of random numbers can also be fine. Show that Δx can be chosen from the Gaussian random number. Show that, more generally, the detailed balance condition can be preserved as long as Δx and $-\Delta x$ appear with the same probability.
2. Show that Δx can be chosen from the uniform distributions with the step size $c = 1$ and $c = 100$ at the even and odd steps, respectively, as mentioned in Sect. 4.7.1. Is the detailed balance condition preserved?
3. Show that the step size can be chosen randomly from $c = 1, 2, 3, 4, 5$ or 6, with the probability $\frac{1}{6}$ at each step. (see Sect. 4.7.1.)
4. What if the variable is discrete? For example, if x takes only integer values, how should we choose Δx?
5. How can we estimate the error bar of the histogram of the probability distribution $P(x)$ obtained via the Markov Chain Monte Carlo simulation?
6. If the integral of $e^{-S(x)}$ is not finite, what happens in MCMC?
7. When we showed the detailed balance condition, we implicitly used the fact that the Jacobian is 1. (If a transformation $x \to x'$ maps $[x, x + \epsilon]$ to $[x', x', +\epsilon']$, the Jacobian is the ratio of the width, $\frac{\epsilon'}{\epsilon}$.) In the examples considered in this book, we can easily see that Jacobian is 1, unless otherwise stated. (For a little bit nontrivial case, see the HMC algorithm explained in Sect. 5.1.) What could be a problem if the Jacobian were not 1?
8. Show that both ⟨Update simultaneously⟩ and ⟨Update one by one⟩ are legitimate procedures.
9. Show that a different step size c_i can be used for each variable x_i.

Solutions

1. We generate Δx with the Gaussian weight $\frac{e^{-\frac{(\Delta x)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}$. Then the transition probability $T(x \to x')$ is

$$T(x \to x') = \frac{e^{-\frac{(x-x')^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \times \min\left(1, e^{S(x)-S(x')}\right). \tag{4.37}$$

This is obtained by replacing $\frac{1}{2c}$ in (4.4) with $\frac{e^{-\frac{(x-x')^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}$. The Eqs. (4.6) and (4.7) change in a similar manner. When $S(x) \geq S(x')$, we have

$$P(x) \cdot T(x \to x') = \frac{e^{-S(x)}}{Z} \times \frac{e^{-\frac{(x-x')^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \tag{4.38}$$

$$P(x') \cdot T(x' \to x) = \frac{e^{-S(x')}}{Z} \times \frac{e^{-\frac{(x'-x)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \times e^{S(x')-S(x)} = \frac{e^{-S(x)}}{Z} \times \frac{e^{-\frac{(x-x')^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}. \tag{4.39}$$

In this way, we can show the detailed balance condition $P(x) \cdot T(x \to x') = P(x') \cdot T(x' \to x)$.
More generally, let $f(\Delta x)$ be the probability distribution of $f(\Delta x)$. Then, if $S(x) \geq S(x')$,

$$P(x) \cdot T(x \to x') = \frac{e^{-S(x)}}{Z} \times f(x' - x) \tag{4.40}$$

$$P(x') \cdot T(x' \to x) = \frac{e^{-S(x')}}{Z} \times f(x - x') \times e^{S(x')-S(x)} = \frac{e^{-S(x)}}{Z} \times f(x - x'). \tag{4.41}$$

Therefore, the detailed balance condition is satisfied as long as $f(x' - x) = f(x - x')$.
2. Because the step size alternates between $c = 1$ and $c = 100$, two different transition probabilities appear at even and odd steps. Therefore, to define the transition probability, we need a label to distinguish even and odd steps. Hence, let us introduce a variable y that takes two values, $y = 0$ (even) or $y = 1$ (odd), and specify the state by a pair (x, y). Then,

$$T((x, 0) \to (x', 1)) = T_{c=1}(x \to x'), \qquad T((x, 1) \to (x', 0)) = T_{c=100}(x \to x'). \tag{4.42}$$

It is easy to check that it is an irreducible Markov chain. However the value of y alternates as 0, 1, 0, 1, ..., and hence the period is 2. Furthermore, the detailed balance does not hold; if we forget about y, then the following equalities hold:

$$\begin{aligned} P(x)T_{c=1}(x \to x') &= P(x')T_{c=1}(x' \to x), \\ P(x)T_{c=100}(x \to x') &= P(x')T_{c=100}(x' \to x). \end{aligned} \tag{4.43}$$

However, if we take y into account then the actual relations are

$$\begin{aligned} P(x)T((x,0) \to (x',1)) &= P(x')((x',0) \to (x,1)), \\ P(x)T((x,1) \to (x',0)) &= P(x')((x',1) \to (x,0)), \end{aligned} \tag{4.44}$$

which are slightly different from the detailed balance condition.
If we combine two steps $y = 0 \to 1 \to 0$ and regard it as one step, it is an irreducible, aperiodic Markov chain. Still, the combined transition function

$$T(x \to x'') = \int dx' T_{c=1}(x \to x')T_{c=100}(x' \to x'') \tag{4.45}$$

does not satisfy the detailed balance condition, i.e.,

$$P(x)T(x \to x') \neq P(x')T(x' \to x). \tag{4.46}$$

If we use a different transition probability

$$\tilde{T}(x \to x'') = \int dx' T_{c=100}(x \to x')T_{c=1}(x' \to x''), \tag{4.47}$$

then

$$P(x)T(x \to x') = P(x')\tilde{T}(x' \to x) \tag{4.48}$$

holds, but it is slightly different from the detailed balance condition.
That the detailed balance condition is not satisfied is not necessarily a bad news, because the detailed balance condition is just a sufficient condition for MCMC to work. What we really need is the equilibrium condition

$$P(x) = \int dx' P(x')T(x' \to x), \tag{4.49}$$

namely, there is no problem as long as $P(x)$ is stationary. This relation can be confirmed as follows. From (4.43), we can show that $P(x)$ is a stationary under the transition $T_c(x' \to x)$:

$$P(x) = \int dx' P(x') T_c(x' \to x). \tag{4.50}$$

By using it twice, we can show that $P(x)$ is a stationary under the transition $T(x' \to x)$:

$$\begin{aligned}
\int dx' P(x') T(x' \to x) &= \int dx' P(x') \int dx'' T_{c=1}(x' \to x'') T_{c=100}(x'' \to x) \\
&= \int dx'' \left(\int dx' P(x') T_{c=1}(x' \to x'') \right) T_{c=100}(x'' \to x) \\
&= \int dx'' P(x'') T_{c=100}(x'' \to x) \\
&= P(x).
\end{aligned} \tag{4.51}$$

Therefore, the distribution of $(x, y = 0)$ converges to $P(x)$. The same holds for $(x, y = 1)$ as well.

This is just a small technical issue that depends on the details of the setup. For example, if the step size changes as $c = 1, 1, 100, 1, 1, 100, \ldots$, then by combining three steps $c = 1$, $c = 100$, $c = 1$ and regarding it as one step, all four conditions including the detailed balance can be satisfied.

3. We can easily show that Δx and $-\Delta x$ appear with the same probability; e.g., the probability of $\Delta x = \pm 0.5$ is $\frac{1}{6} \sum_{c=1}^{6} \frac{1}{2c}$, that of $\Delta x = \pm 1.41$ is $\frac{1}{6} \sum_{c=2}^{6} \frac{1}{2c}$, and so on. Hence we can use the result of Exercise 1 without modification. Unlike Exercise 2, there is no subtlety in this case, and the detailed balance condition is satisfied almost trivially.
4. We only have to make Δx discrete.
5. We can use the Jackknife method. To make a histogram from K samples $x^{(1)}, x^{(2)}, \ldots, x^{(K)}$, we divide x to bins with width dx, count the number of samples in each bin, and normalize such that the integral becomes 1. If the number of samples in the i-th bin is n_i, the height of the histogram is $\rho_i = \frac{n_i}{K \cdot dx}$.

   Let us divide the samples into n groups consisting of w samples. Let $\tilde{\rho}_i^{(l,w)}$ be the histogram calculated from the l-th group of samples. Then the Jackknife error at each bin is $\Delta_{w,i} = \sqrt{\frac{1}{n(n-1)} \sum_{l=1}^{n} \left(\tilde{\rho}_i^{(l,w)} - \rho_i \right)^2}$.
6. Because the "probability" and "expectation value" cannot be defined, MCMC is not applicable. Let us consider $S(x) = -x^2$ as an example. Because $S(x)$ is smaller when $|x|$ is larger, x diverges to $+\infty$ or $-\infty$.
7. To obtain a probability from a probability density, we have to multiply the width of an infinitesimal interval. Therefore, strictly speaking, we needed to multiply ϵ, ϵ', etc, in the expressions appeared in a proof of the detailed balance condition. We did not include them because, if Jacobian is 1, they are the same overall factor and could be ignored. When the Jacobian is not 1, we cannot ignore them. Depending on the choice of Δx, the detailed balance condition could be broken.

8. For the ⟨Update simultaneously⟩, we can easily show that it is a Markov chain that satisfies the irreducibility (unless the domain of integration splits into multiple islands), aperiodicity, and the detailed balance condition. The proof is almost identical to the one for the case of one variable.
   ⟨Update one by one⟩ is similar to the setup in Exercise 2. There were two kinds of transitions in Exercise 2, but now there are n of them. It is straightforward to see it is an irreducible Markov chain. If we update x_j when k is j modulo n ($j = 1, 2, \ldots, n$), then by regarding $j = 1 \to j = 2 \to \cdots \to j = n \to j = 1$ as one step, such a Markov chain is aperiodic. The detailed balance does not hold, but the equilibrium condition is still satisfied, which is enough for our purpose.
   If we change the method slightly such that one of n variables $x_1, \ldots, x_n$ is chosen randomly with a probability $\frac{1}{n}$ and updated, then all conditions hold including the aperiodicity and detailed balance.
9. There is no problem as long as Δx and $-\Delta x$ appear with the same probability. This condition is kept trivially even if we take different step sizes c_i for different variables.

References

1. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087–1092 (1953)
2. M. Rosenbluth, Proof of validity on Monte Carlo method for canonical averaging. Technical report, Los Alamos Scientific Lab., 1953; AIP Conference Proceedings (American Institute of Physics, 2003)
3. M. Matsumoto, T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 30 (1998)
4. M.H. Quenouille, Approximate tests of correlation in time-series. J. R. Statist. Soc. Ser. B (Methodol.) **11**(1), 68–84 (1949)
5. M.H. Quenouille, Notes on bias in estimation. Biometrika **43**(3–4), 353–360 (1956)
6. J. Tukey, Bias and confidence in not-quite large sample. Ann. Math. Statist. **29**, 614 (1958)
7. M. Hanada, M. Honda, Y. Honma, J. Nishimura, S. Shiba, Y. Yoshida, Numerical studies of the Abjm theory for arbitrary n at arbitrary coupling constant. J. High Energy Phys. **2012**(5), 1–34 (2012)
8. S. Lawrence, Sign problems in quantum field theory: classical and quantum approaches. Ph.D. thesis (Maryland, 2020)
9. K. Nagata, Finite-density lattice QCD and sign problem: current status and open problems, arXiv:2108.12423 [hep-lat]
10. R. Feynman, Simulating physics with computers. Int. J. Theor. Phys. **21**, 467–488 (1982)
11. J. Preskill, Simulating quantum field theory with a quantum computer, in *PoS, LATTICE2018* (2018), p. 024
12. M. Troyer, U.-J. Wiese, Computational complexity and fundamental limitations to fermionic quantum Monte Carlo simulations. Phys. Rev. Lett. **94**(17), 170201 (2005)

Chapter 5
Other Useful Algorithms

In Chap. 4, we learned about the Metropolis algorithm. In this chapter, we will introduce a few other useful algorithms. They might look complicated, but all of them are simply dealing with the same problem,

How can we reduce the autocorrelation and generate configurations more efficiently?,

by taking various approaches. *Except for technical details, there is no difference from the Metropolis algorithm.* Of course, technical details are important because they enable us to solve hard problems that cannot be solved via the Metropolis algorithm. However, with a proper understanding of the basic points of the Metropolis algorithm, it is not difficult to understand the essence of other algorithms.

5.1 The HMC Algorithm

In this section, we explain the HMC algorithm (Hybrid Monte Carlo algorithm, or Hamiltonian Monte Carlo algorithm). The original reference is Ref. [1]. A good review can be found in Ref. [2].

A large contribution to the integral comes from the region where the action $S(x)$ is small. If we identify $S(x)$ with elevation, we can say

$$\text{Important configurations} = \text{Bottom of the valley.}$$

Roughly speaking, Markov Chain Monte Carlo is like rolling a ball by hitting it in random directions. If we hit the ball toward the top of the mountain it does not move much, so the ball moves along the bottom of the valley most of the time. To perform the integral efficiently, we should avoid hitting the ball upward; instead, we should push the ball along the bottom of the valley.

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7_5

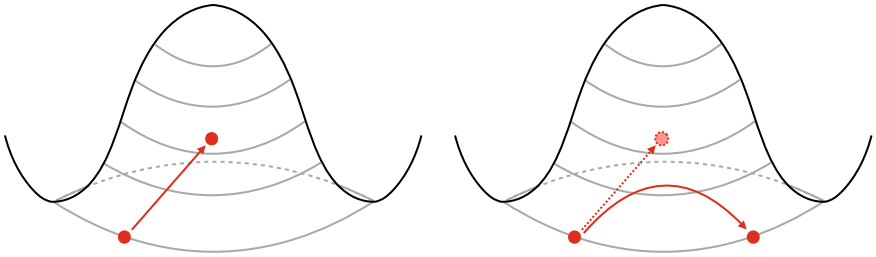

Fig. 5.1 The difference between the Metropolis algorithm and the HMC algorithm. The elevation corresponds to $S(\{x\})$. [Left] In the Metropolis algorithm, if the configuration is pushed uphill the acceptance probability becomes very low. [Right] In the HMC algorithm, even if the configuration is pushed uphill, it comes back down

One might imagine a three-dimensional map, but that is true only for two variables. When there are more variables, we have to imagine a higher dimensional space parametrized by $x_1, x_2, x_3, \ldots$. In such a higher dimensional space, the bottom of the valley is very limited, and if the ball is hit completely randomly as in the Metropolis algorithm, it is almost always pushed upward (the left panel of Fig. 5.1). Therefore, the acceptance rate is very small unless the step size is taken to be very small, and even if the acceptance rate becomes large by sacrificing the step size, the ball almost rarely moves along the bottom of the valley. As a result, the autocorrelation becomes very large.

The HMC algorithm is designed such that this problem is resolved cleverly and the configurations move efficiently near the bottom of the valley. The origin of the name HMC is "Hybrid Monte Carlo", which means the hybrid of two different methods (Metropolis and Molecular dynamics simulation). It is also called the "Hamiltonian Monte Carlo" because Hamiltonian dynamics in physics plays an important role. Either way, the abbreviation is the "HMC".

In the real world, even if you throw a ball upward such that it rolls up the slope, after some time it comes back due to gravity. The HMC utilizes this property. In a literal sense, the identification "small S = low elevation" is made, and a mechanism is realized such that the ball is pushed back toward the small-S region, by closely mimicking a basic equation in physics (the right panel of Fig. 5.1).[1] To propose a candidate x' that results from mimicking classical mechanics, we introduce a fictitious time τ and momentum p. We choose the momentum randomly and calculate the "time evolution" with respect to τ.

In the HMC algorithm, the configurations $\{x^{(k)}\}$ $(k = 0, 1, 2, \ldots)$ are generated in the following manner. As usual, $\{x^{(0)}\}$ can be arbitrary. When $\{x^{(k)}\}$ is given, $\{x^{(k+1)}\}$ is generated as follows:

[1] The HMC algorithm was invented by physicists. When physicists work on abstract math problems, they often think about concrete examples in physics and rely on the intuition from physics to find good approaches.

The HMC algorithm

1. Generate the momentum p_i conjugate to x_i randomly with the Gaussian weight $\frac{1}{\sqrt{2\pi}} e^{-p_i^2/2}$. (To generate the Gaussian random number, the Box-Muller method (Sect. 2.4.1) can be used.)
2. Molecular dynamics evolution: we calculate the "time evolution" and determine $x_i(\tau)$ and $p_i(\tau)$ by using the leapfrog method (Sect. 5.1.2). The fictitious time τ runs from 0 to τ_{fin}. The initial condition is taken to be $x_i(\tau = 0) = x_i^{(k)}$ and $p_i(\tau = 0) = p_i$. The Hamiltonian used in the leapfrog methods is defined by

$$H(\{x\}, \{p\}) = S(\{x\}) + \frac{1}{2} \sum_i p_i^2. \tag{5.1}$$

   The final configuration at $\tau = \tau_{\text{fin}}$ is the candidate for the update.
3. Calculate the initial and final values of the Hamiltonian, which we denote by H_{init} and H_{fin}, at $\tau = 0$ and τ_{fin}.
4. Metropolis test: the proposal is accepted with a probability $\min(1, e^{H_{\text{init}} - H_{\text{fin}}})$ and the value of x is updated as $x_i^{(k+1)} = x_i(\tau_{\text{fin}})$. Otherwise, the proposal is rejected and the value of x remains unchanged, as $x_i^{(k+1)} = x_i^{(k)}$.

For the Metropolis test, a uniform random number r between 0 and 1 is generated and the proposal is accepted if $r < e^{H_{\text{init}} - H_{\text{fin}}}$.

5.1.1 *Intuition from Physics*

Before looking into the details of the HMC algorithm, let us understand the essence without using mathematics. At the first glance, HMC would look very complicated compared to the Metropolis algorithm. Why are we making things (seemingly) complicated? The keys to understanding the advantage are the equation of motion (the Hamilton equation) and the energy conservation.

Just like in the Metropolis algorithm, there is a randomness in the generation of new configurations, because the momentum p_i is chosen randomly. As far as the change in the action is concerned, it may be as large as in the Metropolis algorithm. However, in the HMC algorithm, the change in the Hamiltonian is used for the Metropolis test. (The reason will be explained later. At this moment please accept this fact and proceed.) Hence, if the change of the Hamiltonian can be made small, the acceptance rate becomes large.

As is explained shortly, time evolution via the leapfrog method is a discretization of the continuum time evolution in classical mechanics. The Hamiltonian is the same thing as the energy, whose value does not change with time. Therefore, by making the discretization error smaller, i.e., by sending the step size $\Delta\tau$ and the number of steps N_τ to zero and infinity, respectively, while keeping $\tau_{\text{fin}} = N_\tau \Delta\tau$ fixed, the

energy conservation becomes better and better, and the change of the Hamiltonian $\Delta H = H_{\text{fin}} - H_{\text{init}}$ vanishes. Then the acceptance rate becomes 100%.

In the HMC algorithm, the momentum p_i ($\sim$ the direction and the speed of the ball) at $\tau = 0$ is chosen randomly, and the candidate $\{x'\}$ is obtained by solving the equation of motion. After a long time, the ball can go far, so the difference between $\{x^{(k)}\}$ and $\{x'\}$ can be large. Because we want to collect many independent samples, we should take $\tau_{\text{fin}} = N_\tau \Delta\tau$ large such that the configuration can change a lot during the leapfrog time evolution. As we have seen, the acceptance rate for fixed τ_{fin} can be made large by taking N_τ sufficiently large and $\Delta\tau$ sufficiently small. However, the computational cost increases linearly with N_τ, so if N_τ is taken too large the simulation will not end by the end of our lives. Ideally, we should choose appropriate values of N_τ and $\Delta\tau$ by checking the efficiency of the simulations in turn by measuring the autocorrelation length.

By the way, the Hamiltonian contains $\frac{1}{2}\sum_i p_i^2$ in addition to the action S. The Hamiltonian is conserved during the time evolution, but if $\frac{1}{2}\sum_i p_i^2$ changes significantly, then the change of the action is large as well. Hence one might be worried that the basic idea—"move around the bottom of the valley"—is not realized. However, a very nice trick is hidden in the HMC algorithm so that the action does not become too large even when $\tau_{\text{fin}} = N_\tau \Delta\tau$ is large and the configuration changes drastically. People familiar with physics would notice that $\frac{1}{2}\sum_i p_i^2$ and S correspond to the kinetic energy and potential energy, respectively. If you imagine gravity, you can read off the following correspondence:

$$S \text{ is large} = \text{Potential energy is large} = \text{High elevation}$$

The Hamilton equation describes the time evolution of such a classical system. Even if you roll the ball upward, it cannot go up forever; it comes back eventually and moves around by oscillating between the bottom of the valley and some limited elevation. You do not have to know the equation of motion in order to understand this; this is what we see in our daily lives. The same holds for the HMC algorithm. Because the same rule is adopted for the time evolution, even if the initial momentum is upward, the ball is pushed back and moves near the bottom of the valley. Just by taking $\tau_{\text{fin}} = N_\tau \Delta\tau$ to be large, we can move the configuration significantly along the valley, while avoiding going uphill. Therefore, as we had hoped, the important configurations are collected efficiently.

5.1.2 Leapfrog Method

Before the discretization, the Hamilton equation is given by

$$\frac{dp_i}{d\tau} = -\frac{\partial H}{\partial x_i} = -\frac{\partial S}{\partial x_i}, \qquad \frac{dx_i}{d\tau} = \frac{\partial H}{\partial p_i} = p_i. \tag{5.2}$$

As we mentioned already a few times, the Hamilton equation describes the time evolution of the system with the potential energy $S(\{x\})$ with respect to time τ. In Appendix. C, we summarized a few important points regarding the Hamilton equation. You do not have to know the connection to physics in order to use the HMC algorithm though; practically, the only thing you need is the conservation of energy, i.e., the value of H does not change with time τ. This follows from the property that the energy during the infinitesimal time evolution $\tau \to \tau + \Delta\tau$ vanishes at the linear order in $\Delta\tau$:

$$\begin{aligned}\Delta H &= \sum_i \left(\frac{\partial H}{\partial x_i} \Delta x_i + \frac{\partial H}{\partial p_i} \Delta p_i \right) \\ &= \sum_i \left(\frac{\partial H}{\partial x_i} \frac{dx_i}{d\tau} \Delta\tau + \frac{\partial H}{\partial p_i} \frac{dp_i}{d\tau} \Delta\tau \right) \\ &= \sum_i \left(\frac{\partial H}{\partial x_i} \frac{\partial H}{\partial p_i} - \frac{\partial H}{\partial p_i} \frac{\partial H}{\partial x_i} \right) \Delta\tau \\ &= 0. \end{aligned} \tag{5.3}$$

See Exercise 3 at the end of this chapter for more details of the discretization error.

The leapfrog discretization is defined as follows. *Please be careful about the factors* $\frac{1}{2}$ *at the beginning and the end.* They are crucial for the reversibility of the leapfrog time evolution and the detailed balance condition.

The HMC algorithm

1. First step
   $x_i(\Delta\tau/2) = x_i(0) + p_i(0) \cdot \frac{\Delta\tau}{2}$
2. Repeat the following for $n = 1, 2, N_\tau - 1$:
   $p_i(n\Delta\tau) = p_i((n-1)\Delta\tau) - \frac{\partial S}{\partial x_i}((n-1/2)\Delta\tau) \cdot \Delta\tau$
   $x_i((n+1/2)\Delta\tau) = x_i((n-1/2)\Delta\tau) + p_i(n\Delta\tau) \cdot \Delta\tau$
3. Final steps
   $p_i(N_\tau\Delta\tau) = p_i((N_\tau - 1)\Delta\tau) - \frac{\partial S}{\partial x_i}((N_\tau - 1/2)\Delta\tau) \cdot \Delta\tau$
   $x_i(N_\tau\Delta\tau) = x_i((N_\tau - 1/2)\Delta\tau) + p_i(N_\tau\Delta\tau) \cdot \frac{\Delta\tau}{2}$

Figure 5.2 shows the case of $N_\tau = 5$. There are $2N_\tau + 1 = 11$ steps of the discretized time evolution. It is just like two kids—x and p—are playing the leapfrog. If N_τ is sent to larger values fixing $N_\tau \cdot \Delta\tau$, the change of the Hamiltonian $H_{\text{fin}} - H_{\text{init}}$ decreases proportionally to N_τ^{-2}. (See an exercise at the end of this chapter.) By using a more complicated improved method, $H_{\text{fin}} - H_{\text{init}}$ can be made even smaller.

Reversibility of the Leapfrog Time Evolution

In the leapfrog method, a half-a-step shift between x and p is of crucial importance. This is because the reversibility of the time evolution is guaranteed due to this half-

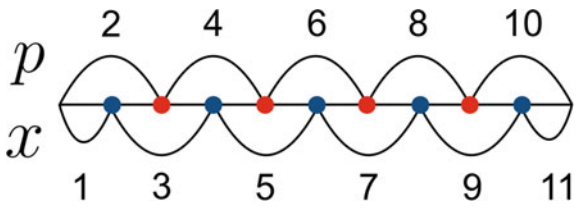

Fig. 5.2 Leapfrog method with $N_\tau = 5$. The discrete time evolutions of x and p are half-a-step shifted

a-step shift. As we will see in Sect. 5.1.3, the reversibility of the time evolution is necessary for the detailed balance condition to hold in the HMC algorithm. To understand the relationship between reversibility and half-a-step shift, let us see what happens if we do not perform the half-a-step shift.

Suppose that we did not use the half-a-step shift prescription and defined the n-th step as

$$\begin{aligned} x_i((n-1)\Delta\tau) \to x_i(n\Delta\tau) &= x_i((n-1)\Delta\tau) + p_i((n-1)\Delta\tau)\cdot\Delta\tau, \\ p_i((n-1)\Delta\tau) \to p_i(n\Delta\tau) &= p_i((n-1)\Delta\tau) - \frac{\partial S}{\partial x_i}((n-1)\Delta\tau)\cdot\Delta\tau. \end{aligned} \tag{5.4}$$

Then, by repeating it we could define the time evolution as

$$\{x(0),\, p(0)\} \to \{x(\Delta t),\, p(\Delta t)\} \to \cdots \to \{x(N_\tau\Delta\tau),\, p(N_\tau\Delta\tau)\}. \tag{5.5}$$

The final step would be

$$\begin{aligned} x_i((N_\tau-1)\Delta\tau) \to x_i(N_\tau\Delta\tau) &= x_i((N_\tau-1)\Delta\tau) + p_i((N_\tau-1)\Delta\tau)\cdot\Delta\tau, \\ p_i((N_\tau-1)\Delta\tau) \to p_i(N_\tau\Delta\tau) &= p_i((N_\tau-1)\Delta\tau) - \frac{\partial S}{\partial x_i}((N_\tau-1)\Delta\tau)\cdot\Delta\tau, \end{aligned} \tag{5.6}$$

and hence,

$$\begin{aligned} x_i((N_\tau-1)\Delta\tau) &= x_i(N_\tau\Delta\tau) - p_i((N_\tau-1)\Delta\tau)\cdot\Delta\tau, \\ p_i((N_\tau-1)\Delta\tau) &= p_i(N_\tau\Delta\tau) + \frac{\partial S}{\partial x_i}((N_\tau-1)\Delta\tau)\cdot\Delta\tau. \end{aligned} \tag{5.7}$$

Let us reverse the time. We start with $\{x'(0),\, p'(0)\} \equiv \{x(N_\tau\Delta\tau), -p(N_\tau\Delta\tau)\}$. With the same prescription without the half-a-step shift, the first step would be

$$\begin{aligned} x_i'(0) \to x_i'(\Delta\tau) &= x_i'(0) + p_i'(0)\cdot\Delta\tau = x_i(N_\tau\Delta\tau) - p_i(N_\tau\Delta\tau)\cdot\Delta\tau, \\ p_i'(0) \to p_i'(\Delta\tau) &= p_i'(0) - \frac{\partial S}{\partial x_i'}(0)\cdot\Delta\tau = -p_i(N_\tau\Delta\tau) - \frac{\partial S}{\partial x_i}(N_\tau\Delta\tau)\cdot\Delta\tau. \end{aligned} \tag{5.8}$$

By comparing it with (5.7), we obtain

$$x_i'(\Delta\tau) \neq x_i((N_\tau - 1)\Delta\tau), \qquad p_i'(\Delta\tau) \neq -p_i((N_\tau - 1)\Delta\tau). \tag{5.9}$$

This is because, in general,

$$p_i(N_\tau\Delta\tau) \neq p_i((N_\tau - 1)\Delta\tau), \qquad \frac{\partial S}{\partial x_i}(N_\tau\Delta\tau) \neq \frac{\partial S}{\partial x_i}((N_\tau - 1)\Delta\tau). \tag{5.10}$$

Therefore, a naive discretization (5.4) breaks the reversibility of the time evolution. If such a non-reversible discretization is used, the detailed balance condition is not satisfied.

The reversibility of the leapfrog time evolution can be confirmed by following the steps one by one. Let us write the final few steps explicitly:

$$\begin{aligned} x_i\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right) &= x_i\left(\left(N_\tau - \frac{3}{2}\right)\Delta\tau\right) + p_i((N_\tau - 1)\Delta\tau)\cdot\Delta\tau, \\ p_i(N_\tau\Delta\tau) &= p_i((N_\tau - 1)\Delta\tau) - \frac{\partial S}{\partial x_i}\left((N_\tau - 1/2)\,\Delta\tau\right)\cdot\Delta\tau, \\ x_i(N_\tau\Delta\tau) &= x_i\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right) + p_i(N_\tau\Delta\tau)\cdot\frac{\Delta\tau}{2}. \end{aligned} \tag{5.11}$$

By rewriting them a little bit, we obtain

$$x_i\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right) = x_i(N_\tau\Delta\tau) - p_i(N_\tau\Delta\tau)\cdot\frac{\Delta\tau}{2}, \tag{5.12}$$

$$p_i((N_\tau - 1)\Delta\tau) = p_i(N_\tau\Delta\tau) + \frac{\partial S}{\partial x_i}\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right)\cdot\Delta\tau, \tag{5.13}$$

$$x_i\left(\left(N_\tau - \frac{3}{2}\right)\Delta\tau\right) = x_i\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right) - p_i((N_\tau - 1)\Delta\tau)\cdot\Delta\tau. \tag{5.14}$$

We want to confirm that this is the same as the time-reversed process starting with $\{x'(0), p'(0)\} \equiv \{x(N_\tau\Delta\tau), -p(N_\tau\Delta\tau)\}$. The first step of the time-reversed process is

$$\begin{aligned} x_i'\left(\frac{1}{2}\Delta\tau\right) &= x'(0) + p_i'(0)\cdot\frac{\Delta\tau}{2} \\ &= x_i(N_\tau\Delta\tau) - p_i(N_\tau\Delta\tau)\cdot\frac{\Delta\tau}{2} \\ &= x_i\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right). \end{aligned} \tag{5.15}$$

To get the last line, we used (5.12). So far so good! The second step is

$$\begin{aligned} p_i'(\Delta\tau) &= p_i'(0) - \frac{\partial S}{\partial x_i'}\left(\frac{1}{2}\Delta\tau\right)\cdot\Delta\tau \\ &= -p_i(N_\tau\Delta\tau) - \frac{\partial S}{\partial x_i}\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right)\cdot\Delta\tau \\ &= -p_i((N_\tau - 1)\Delta\tau). \end{aligned} \tag{5.16}$$

In the second step, we used $\frac{\partial S}{\partial x_i'}\left(\frac{1}{2}\Delta\tau\right) = \frac{\partial S}{\partial x_i}\left(\left(N_\tau - \frac{1}{2}\right)\Delta\tau\right)$ which holds because of (5.15). At the last step, we used (5.13). We can still see the agreement! Just repeating the same kind of calculations, we can see that the agreement can be seen all the way to the final step and that $\{x'(N_\tau\Delta\tau),\, p'(N_\tau\Delta\tau)\} = \{x(0), -p(0)\}$ is obtained. Hence, the leapfrog time evolution is reversible indeed.

Above, our prescription was that x goes half a step forward at the beginning. We could also let p go half a step forward at the beginning; the leapfrog time evolution could be reversible anyways. In large-scale simulations in physics, most of the computational effort is spent on the calculation of $\frac{\partial S}{\partial x_i}$, and hence, by using our prescription we could save the calculation of $\frac{\partial S}{\partial x_i}$ once, and hence, it is slightly more economical.

5.1.3 Checking the Conditions for MCMC

Several steps are needed to confirm the detailed balance condition $e^{-S(\{x\})}T(\{x\}\to\{x'\}) = e^{-S(\{x'\})}T(\{x'\}\to\{x\})$. Each step is very easy.

- First of all, note that the leapfrog time evolution is reversible. Namely, by taking the initial configuration and momentum to be $x(\tau_{\text{fin}})$ and $-p(\tau_{\text{fin}})$, we obtain $x(0)$ and $-p(0)$ as the final configuration and momentum.
- Let the time evolution including the momentum be $\{x,\, p\}\to\{x',\, p'\}$. The probability that p is chosen is $\prod_i\left(\frac{e^{-p_i^2/2}}{\sqrt{2\pi}}\right)$. By multiplying this factor to the acceptance probability $\min(1, e^{H-H'})$, the transition probability $T(\{x\}\to\{x'\})$ is obtained. Here, H and H' are the values of the Hamiltonian calculated from $\{x,\, p\}$ and $\{x',\, p'\}$, respectively.
- Let us consider the time-reversed process $\{x', -p'\}\to\{x, -p\}$. The momentum $-p'$ is chosen with the probability $\prod_i\left(\frac{e^{-p_i'^2/2}}{\sqrt{2\pi}}\right)$. By multiplying the acceptance probability (which is $\min(1, e^{H'-H})$ this time), the transition probability $T(\{x'\}\to\{x\})$ is obtained.
- Suppose that $H \geq H'$. Then the acceptance probabilities for $\{x,\, p\}\to\{x',\, p'\}$ and $\{x', -p'\}\to\{x, -p\}$ are $\min(1, e^{H-H'}) = 1$ and $\min(1, e^{H'-H}) = e^{H'-H}$, respectively. The transition probabilities are

$$T(\{x\} \to \{x'\}) = \prod_i \left(\frac{e^{-p_i^2/2}}{\sqrt{2\pi}} \right) \times 1 = \prod_i \left(\frac{e^{-p_i^2/2}}{\sqrt{2\pi}} \right) \tag{5.17}$$

and

$$\begin{aligned} T(\{x'\} \to \{x\}) &= \prod_i \left(\frac{e^{-p_i'^2/2}}{\sqrt{2\pi}} \right) \times e^{S(\{x'\}) + \sum_i p_i'^2/2 - S(\{x\}) - \sum_i p_i^2/2} \\ &= e^{-S(\{x\}) + S(\{x'\})} \prod_i \left(\frac{e^{-p_i^2/2}}{\sqrt{2\pi}} \right). \end{aligned} \tag{5.18}$$

Therefore,

$$e^{-S(\{x\})} T(\{x\} \to \{x'\}) = e^{-S(\{x'\})} T(\{x'\} \to \{x\}) = e^{-S(\{x\})} \prod_i \left(\frac{e^{-p_i^2/2}}{\sqrt{2\pi}} \right). \tag{5.19}$$

In this way, we could confirm that the detailed balance condition is satisfied.

- In the case that $H < H'$, essentially the same argument can be repeated, and the detailed balance condition can be confirmed.

In the above, we implicitly used the fact that the Jacobian associated with the leapfrog time evolution is 1. (see Exercise 7 in Chap. 4 and Exercise 5 at the end of this chapter.) In addition, we assumed that only one set of $\{p\}$ gives transition from $\{x\}$ to $\{x'\}$. It is an easy exercise to remove this assumption.

Let us check the other conditions as well.

- The sequence of $\{x^{(k)}\}$'s is a Markov chain because the momentum $\{p\}$ is chosen randomly at each step.
- Whether this Markov chain is irreducible or not depends on the detail of the target probability distribution and the range of the variables. This is the same as in the Metropolis algorithm. When the probability distribution splits into multiple islands, we have to be careful.
  The HMC algorithm is like playing catch. The momentum $\{p\}$ specifies the direction and speed of the ball. The force acting on the ball (e.g., gravity) is determined by the action $S(\{x\})$.
  Suppose we want to send a ball from point A to point D in the left panel of Fig. 5.3. To make the argument as simple as possible, suppose that the force coming from $S(\{x\})$ is not so strong and the ball goes more or less straight. Then, it is impossible to send a ball directly from A to D. However, because the domain of the integration (grey region) is connected, it is possible to send a ball from A to B, then from B to C, and then from C to D.

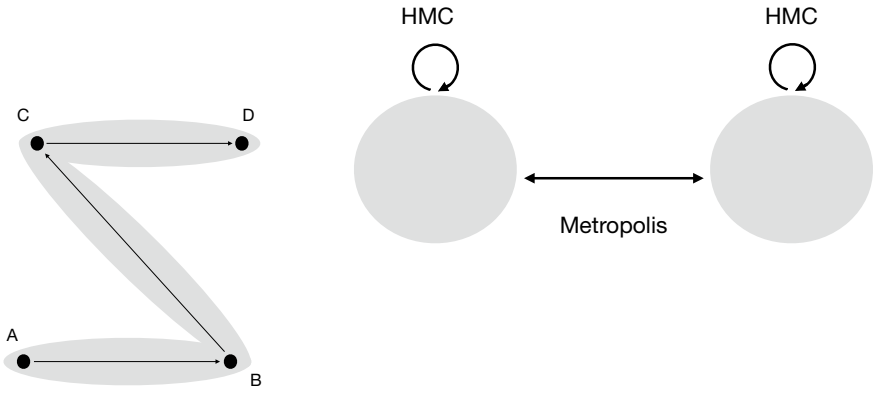

Fig. 5.3 Irreducibility in HMC: [Left] If the domain of integration is not split into multiple islands, two points can be connected by a finite number of steps, and hence the chain is irreducible. [Right] If there are multiple islands, it is impossible to go to other islands, and hence the chain is not irreducible. Some sort of care, e.g., the combination with the Metropolis algorithm with a large step size, is needed

However, because x changes continuously in the HMC algorithm,[2] it does not function properly if the domain of the integration splits into two islands as shown in the right panel of Fig. 5.3. In such cases, we have to use our brains a little bit more. For example, the combination of the Metropolis algorithm with a large step size and the HMC algorithm might work. See the discussion around Fig. 4.12 and Sect. 5.4.

- Regarding the aperiodicity: if a ball goes straight, it is possible to come back to the same point with various numbers of steps, as in the left panel of Fig. 5.4, e.g., $A \to B \to A$ and $A \to B \to C \to A$. Depending on the way the force acts on the ball, it is possible to throw a ball upward such that the ball comes back to the original point after one step, as in the right panel of Fig. 5.4. If we play catch literally, the ball comes back after two steps. Except for pathological situations, the greatest common divisor of the number of steps to come back to the same point should be 1.[3]

[2] Strictly speaking, the change is discrete because of the leapfrog discretization. Note however that the crucial point of the HMC algorithm is to make the leapfrog time evolution almost continuous so that the Hamiltonian is conserved with good precision.

[3] $S(x) = x^2$, $\Delta\tau = 1$, $x(0) = 0$ is a pathological example. Regardless of the value of $p(0)$, the leapfrog time evolution is a regular oscillation $x(\frac{1}{2}\Delta\tau) = \frac{1}{2}p(0)$, $p(\Delta\tau) = 0$, $x(\frac{3}{2}\Delta\tau) = \frac{1}{2}p(0)$, $p(2\Delta\tau) = -p(0)$, $x(\frac{5}{2}\Delta\tau) = -\frac{1}{2}p(0)$, $p(3\Delta\tau) = 0, \cdots$.

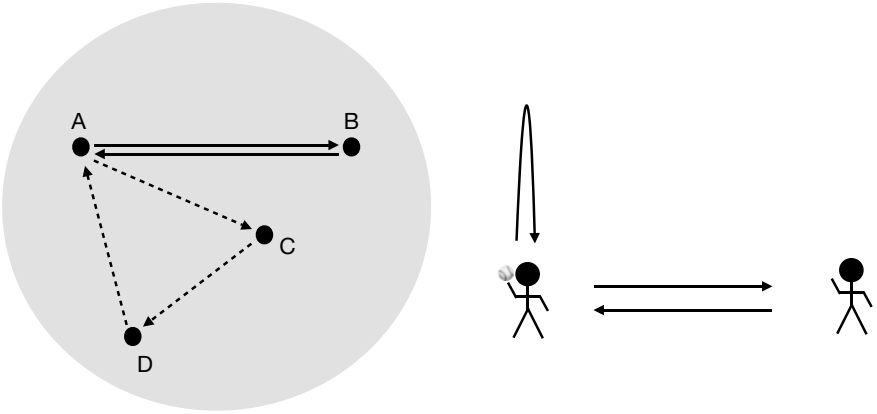

Fig. 5.4 Aperiodicity in HMC: except for pathological cases, the greatest common divisor of the periods is 1

5.1.4 Univariate HMC

Because we are learning a new algorithm, let us consider the simplest case of the Gaussian integral again. (This is a stupid example because we need to use the Gaussian random number in the HMC algorithm. Still, it is an instructive example to understand the essence of the algorithm.) Let us see some sample code written in C++:

```
#include <iostream>
#include <cmath>
#include<fstream>
const int niter=10000;       // Collect 10000 samples.
const int ntau=40;           // number of steps for leapfrog
const double dtau=1e0;       // step size for leapfrog
/*****************************************************************/
/*** Gaussian Random Number Generator with Box Muller Algorithm ***/
/*****************************************************************/
int BoxMuller(double& p, double& q){

  double pi=2e0*asin(1e0);
  //uniform random numbers between 0 and 1
  double r = (double)rand()/RAND_MAX;
  double s = (double)rand()/RAND_MAX;
  //Gaussian random numbers,
  //with weights proportional to e^{-p^2/2} and e^{-q^2/2}
  p=sqrt(-2e0*log(r))*sin(2e0*pi*s);
  q=sqrt(-2e0*log(r))*cos(2e0*pi*s);

  return 0;
}
/*********************************/
/*** Calculation of the action ***/
```

```
/*********************************/
// When you change this part, you have to change "calc_delh" as well.
double calc_action(const double x){

  double action=0.5e0*x*x;

  return action;
}
/*************************************/
/*** Calculation of the Hamiltonian ***/
/*************************************/
double calc_hamiltonian(const double x,const double p){

  double ham=calc_action(x);    // Calculate the action (potential energy)

  ham=ham+0.5e0*p*p;     // add kinetic energy

  return ham;
}
/****************************/
/*** Calculation of dH/dx ***/
/****************************/
// Derivative of the Hamiltonian with respect to x,
// which is equivalent to the derivative of the action.
// When you change "calc_action", you have to change this part as well.
double calc_delh(const double x){

  double delh=x;

  return delh;
}
/**************************/
/*** Molecular evolution ***/
/**************************/
// Leapfrog time evolution
int Molecular_Dynamics(double& x,double& ham_init,double& ham_fin){

  double r1,r2;
  BoxMuller(r1,r2);
  double p=r1;   //generate momentum p as the Gaussian random number.

  //*** calculate Hamiltonian ***
  ham_init=calc_hamiltonian(x,p);
  //*** first step of leapfrog; be careful about 0.5. ***
  x=x+p*0.5e0*dtau;
  //*** 2nd, ..., Ntau-th steps ***
  for(int step=1; step!=ntau; step++){
    double delh=calc_delh(x);
    p=p-delh*dtau;
    x=x+p*dtau;
  }
  //*** last step of leapfrog; be careful about 0.5.  ***
  double delh=calc_delh(x);
  p=p-delh*dtau;
  x=x+p*0.5e0*dtau;
  //*** calculate Hamiltonian again ***
  ham_fin=calc_hamiltonian(x,p);
```

```
  return 0;
}

int main()
{
  srand((unsigned)time(NULL));
  /*********************************/
  /* Set the initial configuration */
  /*********************************/
  double x=0e0;
  /*****************/
  /*** Main part ***/
  /*****************/
  std::ofstream outputfile("output.txt");   // prepare the output file.
  int naccept=0;           // counter for the number of acceptance.
  // sum of x^2, which is used for the calculation of <x^2>.
  double sum_xx=0e0;

  for(int iter=0; iter!=niter; iter++){

    double backup_x=x;
    double ham_init,ham_fin;
    Molecular_Dynamics(x,ham_init,ham_fin);     // Leapfrog time evolution
    double metropolis = (double)rand()/RAND_MAX;
    if(exp(ham_init-ham_fin) > metropolis){      // Metropolis test
      naccept=naccept+1;      // accept, or
    }else{
    //reject
      x=backup_x;       // reject
    }
    /*******************/
    /*** data output ***/
    /*******************/
    sum_xx=sum_xx+x*x;

    // output x, <x^2>, acceptance

    std::cout << std::fixed << std::setprecision(6)
      << x << "   "
      << sum_xx/((double)(iter+1)) <<  "   "
      << ((double)naccept)/((double)iter+1)
      << std::endl;

    outputfile << std::fixed << std::setprecision(6)
      << x << "   "
      << sum_xx/((double)(iter+1)) <<  "   "
      << ((double)naccept)/((double)iter+1)
      << std::endl;


  }      // The end of the simulation

  outputfile.close();      // close the output file

  return 0;
}
```

The values of several parameters are set at the beginning of the code. `niter` is the number of configurations generated in the simulation. `ntau` and `dtau` mean N_τ and $\Delta\tau$, respectively.

Next, several routines and functions are defined:

- `BoxMuller` generates the Gaussian random numbers via the Box-Muller method. When you use the Gaussian random numbers, be careful about the width σ. Depending on the type of variables (real or complex) and the normalization of the p^2-term, the appropriate value of σ should be used.
- `calc_action` calculates the value of the action $S(x)$. In the current example of the Gaussian integral, we use $S(x) = \frac{x^2}{2}$. This routine is used in `calc_hamiltonian`.
- `calc_hamiltonian` calculates the value of the Hamiltonian by adding $\frac{p^2}{2}$ to the value of the action. This routine is called in `Molecular_Dynamics`.
- `calc_delh` is used to calculate the derivative of the Hamiltonian with respect to x, which is $\frac{dH}{dx} = \frac{dS}{dx} = x$ in this case. This routine is called in `Molecular_Dynamics` as well.
- `Molecular_Dynamics` performs the leapfrog time evolution. It returns the value of x after the time evolution and the values of the Hamiltonian before and after the evolution. (For a historical reason, it is called molecular dynamics evolution.)

In `main`, the only differences from the Metropolis algorithm are that the value of x is varied by using `Molecular_Dynamics` rather than just adding a random number, and ΔH is used for the Metropolis test instead of ΔS. Even when $S(x)$ is a more complicated function, we only have to rewrite `calc_action` and `calc_delh`.

5.1.5 *Multivariate HMC*

Let us generate multivariate probability distributions by using the HMC algorithm.

Multivariate Gaussian Distribution

Let us consider the multivariate Gaussian distribution,

$$S(x_1, \ldots, x_n) = \frac{1}{2} \sum_{i,j=1}^{n} A_{ij} x_i x_j \qquad (A_{ij} = A_{ji}). \tag{5.20}$$

The Hamilton equation (5.2) becomes

$$\frac{dp_i}{d\tau} = -\sum_{j=1}^{n} A_{ij} x_j, \qquad \frac{dx_i}{d\tau} = p_i. \tag{5.21}$$

Here is a sample code:

```
#include <iostream>
#include <cmath>
#include<fstream>
const int niter=1000;
const int ntau=20;
const double dtau=0.5e0;
const int ndim=3;   // Number of variables
/*****************************************************************/
/*** Gaussian Random Number Generator with Box Muller Algorithm ***/
/*****************************************************************/
int BoxMuller(double& p, double& q){

  double pi=2e0*asin(1e0);
  //uniform random numbers between 0 and 1
  double r = (double)rand()/RAND_MAX;
  double s = (double)rand()/RAND_MAX;
  //Gaussian random numbers,
  //with weights proportional to e^{-p^2/2} and e^{-q^2/2}
  p=sqrt(-2e0*log(r))*sin(2e0*pi*s);
  q=sqrt(-2e0*log(r))*cos(2e0*pi*s);

  return 0;
}
/*********************************/
/*** Calculation of the action ***/
/*********************************/
// When you change this part, you have to change "calc_delh" as well.
double calc_action(const double x[ndim],const double A[ndim][ndim]){

  double action=0e0;

  for(int idim=0; idim!=ndim; idim++){
      for(int jdim=0; jdim!=idim; jdim++){
    action=action+x[idim]*A[idim][jdim]*x[jdim];
      }
      action=action+0.5e0*x[idim]*A[idim][idim]*x[idim];
  }

  return action;
}
/*************************************/
/*** Calculation of the Hamiltonian ***/
/*************************************/
double calc_hamiltonian(const double x[ndim], const double p[ndim],
                        const double A[ndim][ndim]){

  double ham=calc_action(x,A);

  for(int idim=0; idim!=ndim; idim++){
    ham=ham+0.5e0*p[idim]*p[idim];
  }

  return ham;
}
/*****************************/
/*** Calculation of dH/dx ***/
/*****************************/
// Derivative of the Hamiltonian with respect to x,
// which is equivalent to the derivative of the action.
// When you change "calc_action", you have to change this part as well.
int calc_delh(const double x[ndim],const double A[ndim][ndim],
                double (&delh)[ndim]){
```

```
  for(int idim=0; idim!=ndim; idim++){
    delh[idim]=0e0;
  }

  for(int idim=0; idim!=ndim; idim++){
    for(int jdim=0; jdim!=ndim; jdim++){
      delh[idim]=delh[idim]+A[idim][jdim]*x[jdim];
    }
  }

  return 0;
}
/*************************/
/*** Molecular evolution ***/
/*************************/
// Leapfrog time evolution
int Molecular_Dynamics(double (&x)[ndim],const double A[ndim][ndim],
                       double& ham_init,double& ham_fin){

  double p[ndim];
  double delh[ndim];
  double r1,r2;

  for(int idim=0; idim!=ndim; idim++){
    BoxMuller(r1,r2);
    p[idim]=r1;  //generate momentum p as the Gaussian random number.
  }
  //*** calculate Hamiltonian ***
  ham_init=calc_hamiltonian(x,p,A);
  //*** first step of leapfrog; be careful about 0.5. ***
  for(int idim=0; idim!=ndim; idim++){
    x[idim]=x[idim]+p[idim]*0.5e0*dtau;
  }
  //*** 2nd, ..., Ntau-th steps ***
  for(int step=1; step!=ntau; step++){
    calc_delh(x,A,delh);
    for(int idim=0; idim!=ndim; idim++){
      p[idim]=p[idim]-delh[idim]*dtau;
    }
    for(int idim=0; idim!=ndim; idim++){
    x[idim]=x[idim]+p[idim]*dtau;
    }
  }
  //*** last step of leapfrog; be careful about 0.5.  ***
  calc_delh(x,A,delh);
  for(int idim=0; idim!=ndim; idim++){
    p[idim]=p[idim]-delh[idim]*dtau;
  }
  for(int idim=0; idim!=ndim; idim++){
    x[idim]=x[idim]+p[idim]*0.5e0*dtau;
  }
  //*** calculate Hamiltonian again ***
  ham_fin=calc_hamiltonian(x,p,A);

  return 0;
}

int main()
{
  double x[ndim];
  double A[ndim][ndim];

  A[0][0]=1e0;A[1][1]=2e0;A[2][2]=2e0;
  A[0][1]=1e0;A[0][2]=1e0;A[1][2]=1e0;
  for(int idim=1; idim!=ndim; idim++){
```

```
    for(int jdim=0; jdim!=idim; jdim++){
      A[idim][jdim]=A[jdim][idim];
    }
  }
  srand((unsigned)time(NULL));
  /*********************************/
  /* Set the initial configuration */
  /*********************************/
  for(int idim=0; idim!=ndim; idim++){
    x[idim]=0e0;
  }
  /*****************/
  /*** Main part ***/
  /*****************/
  std::ofstream outputfile("output.txt");   // prepare the output file.
  int naccept=0;          // counter for the number of acceptance.

  for(int iter=0; iter!=niter; iter++){
    double backup_x[ndim];
    for(int idim=0; idim!=ndim; idim++){
      backup_x[idim]=x[idim];
    }
    double ham_init,ham_fin;
    Molecular_Dynamics(x,A,ham_init,ham_fin);   // Leapfrog time evolution
    double metropolis = (double)rand()/RAND_MAX;
    if(exp(ham_init-ham_fin) > metropolis){         // Metropolis test
      naccept=naccept+1;        // accept, or
    }else{
      for(int idim=0; idim!=ndim; idim++){
      x[idim]=backup_x[idim];   // reject
      }
    }
    /*******************/
    /*** data output ***/
    /*******************/
    if((iter+1)%10 == 0){
      std::cout << std::fixed << std::setprecision(6)
        << x[0] << "   "
        << x[1] << "   "
        << x[2] << "   "
        << ((double)naccept)/((double)iter+1)
        << std::endl;

      outputfile << std::fixed << std::setprecision(6)
        << x[0] << "   "
        << x[1] << "   "
        << x[2] << "   "
        << ((double)naccept)/((double)iter+1)
        << std::endl;
    }
  }
  outputfile.close();
  return 0;
}
```

At the beginning of the code, the parameters are defined. The only difference compared to the single-parameter case is that the number of variables $n = \texttt{ndim}$ is added. The routines and functions are also very similar to those for the univariate version:

- `calc_action` calculates the value of the action $S(x_1, \ldots, x_n)$. This routine is used in `calc_hamiltonian`.
- `calc_hamiltonian` calculates the value of the Hamiltonian, by adding $\sum_{i=1}^{n} \frac{p_i^2}{2}$ to the action. This routine is called in `Molecular_Dynamics`.

- `calc_delh` calculates the derivative of the Hamiltonian with respect to x, which is $\frac{\partial H}{\partial x_i} = \frac{\partial S}{\partial x_i} = \sum_{j=1}^{n} A_{ij} x_j$. This routine is used in `Molecular_Dynamics`.
- `Molecular_Dynamics` performs the leapfrog time evolution. The values of $x_1, \ldots, x_n$ after the time evolution and the values of the Hamiltonian before and after the evolution are obtained.
- The values of A_{ij} are defined in `main`.

In Sect. 5.2, we generate the same probability distribution by using another algorithm (the Gibbs sampling algorithm). The distribution obtained by using the HMC algorithm will be shown there, together with the distribution obtained by using the Gibbs sampling algorithm (Fig. 5.8).

Matrix Integrals

The next example is a kind of calculation common in particle physics and superstring theory. Let ϕ be an $N \times N$ Hermitian matrix. We consider the action of the form

$$S(\phi) = N\mathrm{Tr}\left(\frac{1}{2}\phi^2 + \frac{1}{4}\phi^4\right). \tag{5.22}$$

Similar integrals appear very often in interesting problems such as the internal structure of nuclei and the quantum mechanical properties of a black hole. In most cases, they are not analytically solvable. The great thing about Markov Chain Monte Carlo is that if normal guys like us continue the simulation patiently, we can do very complicated calculations which even genius people cannot do by hand. The only necessary talent is patience.

It is not straightforward to apply the Gibbs sampling algorithm, which will be introduced later, to such a probability distribution. The Metropolis algorithm is not suitable for this problem either. If we vary the variable as $\phi_{ij} \to \phi'_{ij} = \phi_{ij} + \Delta\phi_{ij}$, the action changes as

$$\Delta S = N\mathrm{Tr}\left(\left(\phi + \phi^3\right)\Delta\phi\right) + O((\Delta\phi)^2), \tag{5.23}$$

but this ΔS becomes larger and larger as N goes up. Hence, if we vary all matrix entries simultaneously, the acceptance probability becomes too small unless a very small step size is used. You may think this problem can be avoided by changing one matrix entry at each time, but still, a large autocorrelation is inevitable.

When we use the HMC algorithm, the Hamilton equation is

$$\frac{dp_{ij}}{d\tau} = -\frac{\partial S}{\partial \phi_{ji}} = -N\phi_{ij} - N\left(\phi^3\right)_{ij}, \qquad \frac{d\phi_{ij}}{d\tau} = p_{ij}. \tag{5.24}$$

(Here, the conjugate of ϕ_{ij} is taken to be p_{ji}.) The simulation code is very similar to the one used for the multivariate Gaussian distribution; we only have to change

Table 5.1 The HMC simulation was performed with the action (5.22), and the acceptance rate of the Metropolis test was measured for various values of N_τ. The matrix size is $N = 100$, and $N_\tau \Delta\tau$ was fixed to 0.1. We collected 10,000 configurations starting with a well-thermalized configuration

N_τ	Acceptance rate	(Acceptance rate)$/N_\tau$
4	0.0633	0.01583
6	0.3418	0.05697
8	0.6023	0.07529
10	0.7393	0.07393
12	0.8169	0.06808
14	0.8645	0.06175
16	0.8963	0.05602

the action to (5.22) and the derivative of the action to (5.24). This is always the case, whatever complicated distribution we study.

Originally, the HMC algorithm was invented for large-scale simulations in particle physics. By now, the HMC algorithm and its improved version, the RHMC algorithm (Rational Hybrid Monte Carlo algorithm) [3, 4], are indispensable tools. As we will explain in Sect. 6.4, the mass of the proton and neutron can be calculated by solving the complicated dynamics of the standard model of particle physics [5, 6], or the properties of the black hole predicted by Hawking can be checked based on superstring theory [7, 8]. It is impossible to carry out such computations with other methods.

5.1.6 Tuning the Simulation Parameters

From here on, we will consider the practical issues necessary for actual simulations via the HMC algorithm. The first issue is the choice of N_τ and $\Delta\tau$.

As an example, we consider the matrix integral with the action (5.22). For the matrix size $N = 100$, we fixed $N_\tau \Delta\tau$ to be 0.1, and calculated the acceptance rate for various choices of N_τ. The result is shown in Table 5.1. As we mentioned in Sect. 5.1.1, as N_τ becomes large and $\Delta\tau$ becomes small, the time evolution becomes closer to the continuum limit, the conservation of the Hamiltonian becomes better (i.e., ΔH becomes closer to zero), and hence the acceptance rate goes up. We can confirm this trend in Table 5.1.

We can read off the optimal N_τ from this table. Firstly note that the computational cost is approximately proportional to N_τ. Note also that, because we are fixing $N_\tau \Delta\tau$ now, the change of the matrix ϕ before and after the time evolution does not depend much on N_τ. Then, because the configuration is not updated unless the proposed configuration passes the Metropolis test, the amount of the change of ϕ should be proportional to the acceptance rate, roughly speaking. Therefore,

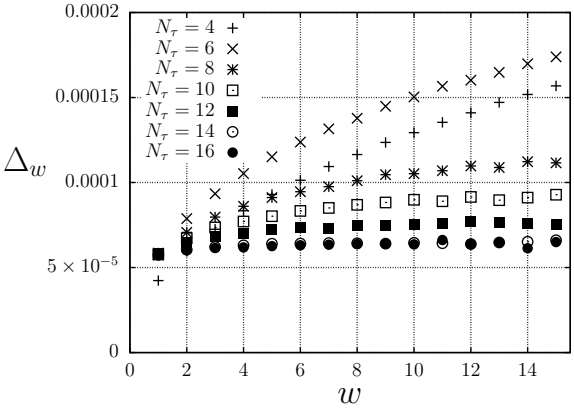

Fig. 5.5 By using the samples used for Table 5.1, we calculated the expectation value of the action (5.22), $\langle S/N^2 \rangle = \frac{1}{KN^2} \sum_{k=1}^{K} S(\phi^{(k)})$, and estimated the Jackknife error. The horizontal and vertical axes are the bin size w and Jackknife error Δ_w, respectively. With $N_\tau \geq 12$, the autocorrelation length is almost 1, and hence there is no point in taking N_τ larger

the amount of the change per computational cost should be approximately proportional to (acceptance rate)$/N_\tau$. Therefore, we should tune N_τ and $\Delta\tau$ such that (acceptance rate)$/N_\tau$ is maximized. From Table 5.1, we can conclude that somewhere around $N_\tau = 8$ or $N_\tau = 10$ is optimal.

Just in case, let us estimate the autocorrelation length, which is the number of steps necessary to obtain independent configurations. As explained in Sect. 4.3.3, the autocorrelation length can be read off from the behavior of the Jackknife error. From the configurations used to make Table 5.1, we calculated the expectation value of the action (5.22), $\langle S/N^2 \rangle = \frac{1}{KN^2} \sum_{k=1}^{K} S(\phi^{(k)})$, and estimated the Jackknife error; see Fig. 5.5. At $N_\tau \geq 12$, the Jackknife error is saturated quickly, and hence, the autocorrelation length is almost 1. This means independent samples are obtained at each step, and hence, there is no point in increasing N_τ further adding more computational cost. From this, we can see that $N_\tau = 8$ or $N_\tau = 10$ is optimal.

Note that the high acceptance rate does not always mean small autocorrelation. For complicated integrals, very long autocorrelation can appear. In such a case, it is necessary to estimate the autocorrelation and the cost for each step precisely. Note also that, ideally, we should vary $N_\tau \Delta\tau$ as well. Clearly, it is not good if $N_\tau \Delta\tau$ is too small. Too large $N_\tau \Delta\tau$ can also be bad, in that the change can be unnecessarily large such that the cost increases while the number of independent samples does not change. As always, the important point is to obtain as many independent samples as possible, with a smaller computational cost.

5.1.7 Using Different Step Sizes for Different Variables

Let us consider the bivariate distribution, with the action

$$S(x, y) = f(x) + g(y) + xy. \tag{5.25}$$

Suppose that the distribution of x is relatively wide, say $f(x) = x^2$, while the distribution of y is extremely narrow, say $g(y) = 1000000000000y^2$.[4] In this case, it is not efficient if we use the same step size for x and y. If there were only x rather high acceptance rate could be achieved with a large step size, but because the distribution of y is sharply peaked the step size has to be taken very small.

As we have seen in Sect. 4.8, in the Metropolis algorithm, it is standard to use different step sizes for different variables. The same is true for the HMC algorithm: different step sizes $\Delta\tau_x$ and $\Delta\tau_y$ can be used for x and y, respectively. As long as the conditions listed in Chap. 3 are satisfied, any step size is fine. Furthermore, it is straightforward to check that the conservation of the Hamiltonian is not affected. In large-scale simulations in physics, it is of crucial importance to adjust the step size for each variable and improve efficiency.

5.1.8 Useful Tips for Debugging

Utilizing the Conservation of the Hamiltonian

The conservation of the Hamiltonian does not just make the HMC very efficient. It provides us with a big bonus: the debugging becomes easier. In HMC, both the action $S(x)$ and its derivative $\frac{\partial S}{\partial x}$ are needed. If both of them are calculated correctly, then the Hamiltonian is conserved in the continuum limit ($\Delta\tau \to 0$, $N_\tau \to \infty$, $N_\tau \Delta\tau$ fixed). Human beings never stop making bugs in their code, but it would be very rare that the errors in the calculations of $S(x)$ and $\frac{\partial S}{\partial x}$ cancel precisely. Practically, we can assume that the Hamiltonian is not conserved unless both $S(x)$ and $\frac{\partial S}{\partial x}$ are calculated correctly. This feature is convenient for debugging; see Fig. 5.6.

As we have seen in Sect. 5.1.7, it is common to introduce different step sizes for different variables. Imagine there are two variables x and y, and the program is written such that the step sizes $\Delta\tau_x$ and $\Delta\tau_y$ can be chosen independently. Then, by setting $\Delta\tau_y = 0$, we can check if the time evolution of x is calculated correctly. By extracting a part of the program in this way, we can debug the code more efficiently.

[4] In physics, it would mean the particle x is light and can be moved easily, while the particle y is extremely heavy and it is almost impossible to move it.

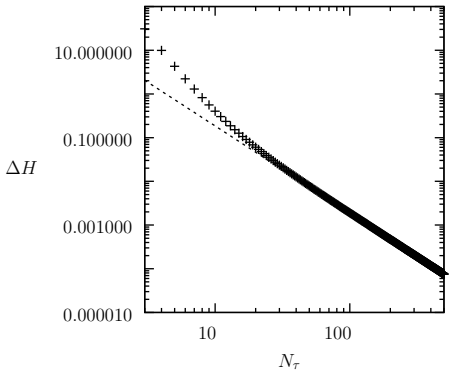

Fig. 5.6 This plot shows how $\Delta H = H_{\rm fin} - H_{\rm init}$ approaches zero when $N_\tau \Delta\tau$ is fixed to 0.1 and N_τ is increased. Both axes are on the logarithmic scale. The action is (5.22) with $N = 100$. We prepared a well-thermalized configuration $\{\phi\}$ and randomly generated momentum $\{p_\phi\}$, and used the same $\{\phi, p_\phi\}$ for all N_τ. We observed $\Delta H \simeq 18.5 N_\tau^{-2}$ for sufficiently large N_τ. When the conservation of the Hamiltonian is confirmed, the debugging is more or less done. Time for beer

Normalization of the Momentum

When we generate the momentum p for the HMC simulation, we have to be careful about normalization. In this book, we choose p with the probability distribution is $\frac{e^{-\frac{p^2}{2}}}{\sqrt{2\pi}}$ because we used a normalization such that the momentum appears in the Hamiltonian as $\frac{p^2}{2}$. If this part is modified, say to p^2 or $100p^2$, then the equation of motion and the width of the Gaussian distribution have to be modified. Such a modification is nothing but a redefinition of momentum variables. The expressions become messy but the final result does not change at all.

When the variable x is a complex number or a matrix, the conjugate momentum p becomes complex or a matrix as well. There are various kinds of matrices—real matrix, complex matrix, symmetric matrix, Hermitian matrix, ...—which sometimes makes the normalization a little bit complicated. We do not go into the details here, please have a look at references such as Ref. [9] when necessary.

When the normalization of the momentum is wrong, the result of the simulation is wrong. Whether the normalization of the momentum is correct cannot be checked by using the conservation of the Hamiltonian, so one has to be cautious regarding this point.

The Factor $\frac{1}{2}$ at the Beginning and the End of the Leapfrog

In the leapfrog time evolution, it is important to change x and p alternately. Furthermore, the factor $\frac{1}{2}$ at the beginning and the end is mandatory. The Hamiltonian is conserved even if we forgot this factor $\frac{1}{2}$, so we have to pay extra attention here.

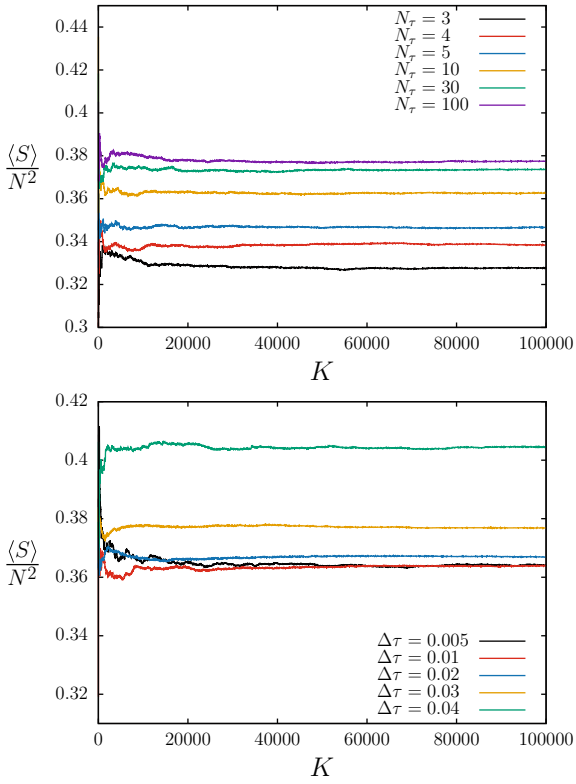

Fig. 5.7 An example of *wrong* simulation. The factor $\frac{1}{2}$ at the end of the leapfrog time evolution is dropped. The expectation value $\frac{\langle S \rangle}{N^2} = \frac{1}{K}\sum_{k=1}^{K} \frac{S(\phi^{(k)})}{N^2}$ at $N = 10$ is calculated for various different pairs of N_τ and $\Delta\tau$. [Top] $N_\tau \Delta\tau$ is fixed to 0.1; [Bottom] N_τ is fixed to 10. The right answer is obtained only when $N_\tau \Delta\tau$ is fixed and N_τ is sent to infinity

Let us drop the factor $\frac{1}{2}$ at the end of the leapfrog[5] and see the outcomes. We consider the matrix integral defined by the action (5.22) as an example. The outcome is a disaster; as shown in Fig. 5.7, different N_τ and $\Delta\tau$ lead to completely different answers. The right answer is obtained only when $N_\tau \Delta\tau$ is fixed and N_τ is sent to infinity.

5.2 Gibbs Sampling Algorithm (Heat Bath Algorithm)

The Metropolis algorithm and the HMC algorithm can be applied to practically any problem we can imagine. The range of applicability of the Gibbs sampling algorithm (which is also called the heat bath algorithm in the physics community; see Refs. [10, 11]) is smaller, but when it is applicable, it is often efficient. It is usually applicable to relatively simple probability distributions such as the ones commonly used in Bayesian statistics. Later in this book, we will apply it to the Ising model in Sect. 6.2.2.

[5] This is the bug that one of the authors actually made in his first HMC code.

In this section, we explain this algorithm by taking the Gaussian distribution as an example.

When the probability distribution of $P(x, y)$ is constructed via the Metropolis algorithm, two procedures, "*fix y and vary x a little bit*" and "*fix x and vary y a little bit*", can be combined, as we have already seen. Although the Metropolis algorithm can be applied to any problem, it often suffers from strong autocorrelation because x and y are varied little by little. In order to reduce the autocorrelation, we should combine "*fix y and vary x a lot*" and "*fix x and vary y a lot*", in case we can. The Gibbs sampling algorithm (see Refs. [12, 13] for earliest applications for Bayesian inference) is based on this almost trivial idea.

5.2.1 Bivariate Gaussian Distribution

Let us consider an action

$$S(x, y) = \frac{x^2 + y^2}{2}. \tag{5.26}$$

In this case, x and y are independent, and the probability distribution is the product of two Gaussian distributions:

$$P(x, y) = P(x) \cdot P(y) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} \cdot \frac{e^{-\frac{y^2}{2}}}{\sqrt{2\pi}}. \tag{5.27}$$

Therefore, we only have to prepare the distributions for x and y, which are denoted by $P(x)$ and $P(y)$, independently. We can use the Box-Muller algorithm for this purpose.

Next, let us change the action slightly:

$$S(x, y) = \frac{x^2 + y^2 + xy}{2}. \tag{5.28}$$

We studied the same example in Sect. 4.8. We will generate this distribution by using the Gibbs sampling algorithm.[6]

The conditional probability $P(x|y)$ is the probability distribution of x when the value of y is fixed. In the same manner, $P(y|x)$ is the probability distribution of y when the value of x is fixed. More concretely, they are

[6] In fact, by changing the variables, we can rewrite this distribution in such a way that the Box-Muller algorithm can be applied. Here, we use this example merely to explain the Gibbs sampling algorithm.

$$P(x|y) = \frac{e^{-\frac{1}{2}\left(x+\frac{y}{2}\right)^2}}{\sqrt{2\pi}} \tag{5.29}$$

and

$$P(y|x) = \frac{e^{-\frac{1}{2}\left(y+\frac{x}{2}\right)^2}}{\sqrt{2\pi}}. \tag{5.30}$$

As we will see in Sect. 5.2.3, these distributions can easily be obtained by shifting the Gaussian distribution generated by the Box-Muller method by $-\frac{y}{2}$ or $-\frac{x}{2}$. These conditional probabilities are utilized in the Gibbs sampling algorithm. The concrete steps are as follows:

Gibbs sampling for two variables x, y

1. Suppose that $(x^{(k)}, y^{(k)})$ are already obtained. Then, $x^{(k+1)}$ is generated following the probability distribution $P(x^{(k+1)}|y^{(k)})$.
2. Next $y^{(k+1)}$ is generated following the probability distribution $P(y^{(k+1)}|x^{(k+1)})$.
3. Repeat the same procedure.

The important point is that, instead of changing $x^{(k)}$ a little bit to obtain $x^{(k+1)}$, (under the condition that $y^{(k)}$ is fixed) $x^{(k+1)}$ is generated without referring to $x^{(k)}$ at all. Such update is possible only when the conditional probability is simple, and hence, the Gibbs sampling algorithm is not applicable to complicated distributions unlike the Metropolis algorithm or the HMC algorithm. However, when it is applicable, the autocorrelation can be small, and as a bonus, there is no need for tuning the parameters such as the step size.

Let us confirm that the conditions for Markov Chain Monte Carlo are satisfied. Let us regard the combination of "fix y and update x" and "fix x and update y" as one step. That the sequence is the Markov chain, the irreducibility and the aperiodicity are almost trivial, so let us focus on the detailed balance condition. Suppose that y is fixed and x is updated to x'. Because of

$$T((x, y) \to (x', y)) = P(x'|y), \qquad T((x', y) \to (x, y)) = P(x|y), \tag{5.31}$$

we obtain

$$\begin{aligned} P(x, y) \cdot T((x, y) \to (x', y)) &= P(x, y) \cdot P(x'|y), \\ P(x', y) \cdot T((x', y) \to (x, y)) &= P(x', y) \cdot P(x|y). \end{aligned} \tag{5.32}$$

By using the definition of the conditional probability

$$P(x, y) = P(x|y) \cdot P(y), \qquad P(x', y) = P(x'|y) \cdot P(y), \tag{5.33}$$

we can see the detailed balance at fixed y:

$$\begin{aligned} P(x, y) \cdot T((x, y) \to (x', y)) &= P(x|y) \cdot P(x'|y)P(y) \\ &= P(x', y) \cdot T((x', y) \to (x, y)). \end{aligned} \tag{5.34}$$

In the same manner, when x is fixed and y is updated, we obtain

$$P(x, y) \cdot T((x, y) \to (x, y')) = P(x, y') \cdot T((x, y') \to (x, y)). \tag{5.35}$$

When $(x, y) \to (x', y) \to (x', y')$ is regarded as one step, the detailed balance does not hold,[7] but it is not a problem in the following sense. If a Markov chain is irreducible and aperiodic, it converges to a stationary distribution (if such a thing exists). We imposed the detailed balance condition because we wanted this stationary distribution to be the target distribution $P(x, y)$. In the construction here, both $(x, y) \to (x', y)$ and $(x', y) \to (x', y')$ separately satisfy the detailed balance condition, and hence, $P(x, y)$ is stationary under both. When $(x, y) \to (x', y) \to (x', y')$ is regarded as one step, the detailed balance condition does not hold but $P(x, y)$ is still the stationary distribution. The situation is almost the same as the one considered in Exercise 2 in Chap. 4. If you are interested in the details, see the solution to Exercise 2 in Chap. 4.

5.2.2 *Multivariate Gibbs Sampling Algorithm*

For n variables, the Gibbs sampling algorithm works in the following manner.

Gibbs sampling algorithm for n variables

1. Suppose $(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})$ is given. Then, $x_1^{(k+1)}$ is generated following the probability distribution $P(x_1^{(k+1)}|x_2^{(k)}, \ldots, x_n^{(k)})$.
2. Next, $x_2^{(k+1)}$ is generated following the probability distribution $P(x_2^{(k+1)}|x_1^{(k+1)}, x_3^{(k)}, \ldots, x_n^{(k)})$.
3. For $i = 3, 4, \ldots, n-1$, $x_i^{(k+1)}$ is generated following the probability distribution $P(x_i^{(k+1)}|x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)}, x_{i+1}^{(k)}, \ldots, x_n^{(k)})$.
4. $x_n^{(k+1)}$ is generated following the probability distribution $P(x_n^{(k+1)}|x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_{n-1}^{(k+1)})$.
   Now $(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})$ has been updated to $(x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_n^{(k+1)})$.
5. Repeat the same procedure many many times.

[7] Note that, if the convention is such that x is updated first, then $(x', y') \to (x', y) \to (x, y)$ is not allowed as an inverse of $(x, y) \to (x', y) \to (x', y')$; we are forced to update in the order of $(x', y') \to (x, y') \to (x, y)$.

Just in case you find the notation a bit complicated, let us show the case of three variables. We use the notation $x_1 = x, x_2 = y, x_3 = z$. Then the Gibbs sampling goes as follows:

Trivariate Gibbs sampling algorithm

1. Suppose $(x^{(k)}, y^{(k)}, z^{(k)})$ are given.
   Then, $x^{(k+1)}$ is generated following the probability distribution $P(x^{(k+1)}|y^{(k)}, z^{(k)})$.
2. Next, $y^{(k+1)}$ is generated following the probability distribution $P(y^{(k+1)}|x^{(k+1)}, z^{(k)})$.
3. Finally, $z^{(k+1)}$ is generated following the probability distribution $P(z^{(k+1)}|x^{(k+1)}, y^{(k+1)})$.
   Now $(x^{(k)}, y^{(k)}, z^{(k)})$ has been updated to $(x^{(k+1)}, y^{(k+1)}, z^{(k+1)})$.
4. Repeat the same procedure many many times.

For the action of the form[8]

$$S(x_1, \ldots, x_n) = \frac{1}{2} \sum_{i,j=1}^{n} A_{ij} x_i x_j \qquad (A_{ij} = A_{ji}), \tag{5.36}$$

the conditional probability distributions are as follows:

$$\begin{aligned} P(x|y,z) &= \frac{e^{-\frac{A_{11}}{2}\left(x+\frac{A_{12}}{A_{11}}y+\frac{A_{13}}{A_{11}}z\right)^2}}{\sqrt{2\pi/A_{11}}}, \\ P(y|z,x) &= \frac{e^{-\frac{A_{22}}{2}\left(y+\frac{A_{21}}{A_{22}}x+\frac{A_{23}}{A_{22}}z\right)^2}}{\sqrt{2\pi/A_{22}}}, \\ P(z|x,y) &= \frac{e^{-\frac{A_{33}}{2}\left(z+\frac{A_{31}}{A_{33}}x+\frac{A_{32}}{A_{33}}y\right)^2}}{\sqrt{2\pi/A_{33}}}. \end{aligned} \tag{5.37}$$

5.2.3 Gibbs Sampling Simulation

We already learned how to obtain the Gaussian distribution with the mean 0 and width 1, i.e., $P(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, by using the Box-Muller algorithm. Let us multiply σ to x, and then add μ:

$$x' = \sigma x + \mu. \tag{5.38}$$

[8] What would be the condition which A_{ij} should satisfy for a legitimate probability distribution to be obtained?

Then the Gaussian distribution with the width σ and mean μ is obtained, i.e.,

$$P(x';\sigma,\mu)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x'-\mu)^2}{2\sigma^2}}. \tag{5.39}$$

In the case of three variables that we have just seen, we can obtain the conditional probability $P(x|y,z)$ by setting $\sigma=\frac{1}{\sqrt{A_{11}}}$, $\mu=-\frac{A_{12}}{A_{11}}y-\frac{A_{13}}{A_{11}}z$. The other distributions $P(y|x,z)$ and $P(z|x,y)$ are obtained in the same manner.

Below, we show a sample program in C++, for $S(x,y,z)=\frac{x^2+2y^2+2z^2+2xy+2yz+2zx}{2}$. We show only `main`. Other than this part, we only need `BoxMuller` which was used also for the HMC algorithm.

```
int main()
{
  double A[3][3];
  A[0][0]=1e0;A[1][1]=2e0;A[2][2]=2e0;
  A[0][1]=1e0;A[0][2]=1e0;A[1][2]=1e0;
  A[1][0]=A[0][1];A[2][0]=A[0][2];A[2][1]=A[1][2];

  srand((unsigned)time(NULL));  // set the seed of the random number
  double x=0e0;double y=0e0;double z=0e0; // set the initial configuration
  // Main part
  for(int iter=0; iter!=niter; iter++){
    double sigma,mu;
    double r1,r2;
    // update x
    sigma=1e0/sqrt(A[0][0]);
    mu=-A[0][1]/A[0][0]*y-A[0][2]/A[0][0]*z;
    BoxMuller(r1,r2);
    x=sigma*r1+mu;
    // update y
    sigma=1e0/sqrt(A[1][1]);
    mu=-A[1][0]/A[1][1]*x-A[1][2]/A[1][1]*z;
    BoxMuller(r1,r2);
    y=sigma*r1+mu;
    // update z
    sigma=1e0/sqrt(A[2][2]);
    mu=-A[2][0]/A[2][2]*x-A[2][1]/A[2][2]*y;
    BoxMuller(r1,r2);
    z=sigma*r1+mu;
    // output the values of x,y,z (every 10 steps)
    if((iter+1)%10==0){
    std::cout
    << x <<  "   "
    << y <<  "   "
    << z << std::endl;
    }
  }
  return 0;
}
```

Note that the Metropolis test is not needed because the configuration is always updated with 100% possibility.

In Sect. 5.1.5, the same probability distribution was studied by using the HMC algorithm. As a sanity check, let us compare the configurations obtained via HMC and Gibbs sampling. Both in HMC and Gibbs sampling, we collected 1 configuration every 10 steps. We had 100,000 steps, hence 10,000 configurations in total. For the HMC simulation, we used $N_\tau = 20$, $\Delta\tau = 0.5$. Figure 5.8 shows two-dimensional scatter plots for x-vs-y, x-vs-z, and y-vs-z. The left and right panels are results obtained via HMC and Gibbs sampling, respectively. We can see almost the same distributions.

In Fig. 5.9, the distributions of the product xy from HMC and Gibbs samplings are compared, by increasing the number of configurations to 1,000,000. A very good agreement can be seen.

5.3 Metropolis-Hastings Algorithm (MH Algorithm)

Next, we introduce the Metropolis-Hastings algorithm (MH algorithm) [14], which is a conceptual foundation of convenient algorithms such as Gibbs sampling and HMC.

In the Metropolis algorithm, the transition probability $T(\{x\} \to \{x'\})$ is

$$T(x \to x') = \left(\text{probability that } \Delta x = x' - x\right) \times \min\left(1, \frac{e^{-S(x')}}{e^{-S(x)}}\right). \tag{5.40}$$

We also impose

$$\left(\text{probability that } \Delta x = x' - x\right) = \left(\text{probability that } \Delta x = x - x'\right). \tag{5.41}$$

Then the detailed balance condition is satisfied.

What if the second condition is not satisfied? For example, if

$$(\text{probability that } \Delta x > 0) = (\text{probability that } \Delta x < 0) \times 2, \tag{5.42}$$

what happens? Of course, if we do not do anything, then the detailed balance condition does not hold anymore. However, if the acceptance probability of the proposed configuration, which is $\min\left(1, \frac{e^{-S(x')}}{e^{-S(x)}}\right)$ in the Metropolis algorithm, is properly modified, then the detailed balance condition can be restored. In the current example, if we set the acceptance probability to be half of the one for the Metropolis algorithm when $\Delta x > 0$, the detailed balance condition can be satisfied. In the Metropolis-Hastings algorithm, we use a cleverer prescription.

More generally, let us assume that the probability that $\Delta x = x' - x$ is obtained can depend on x and x' in a complicated manner, as $f(x \to x')$. We assume that if

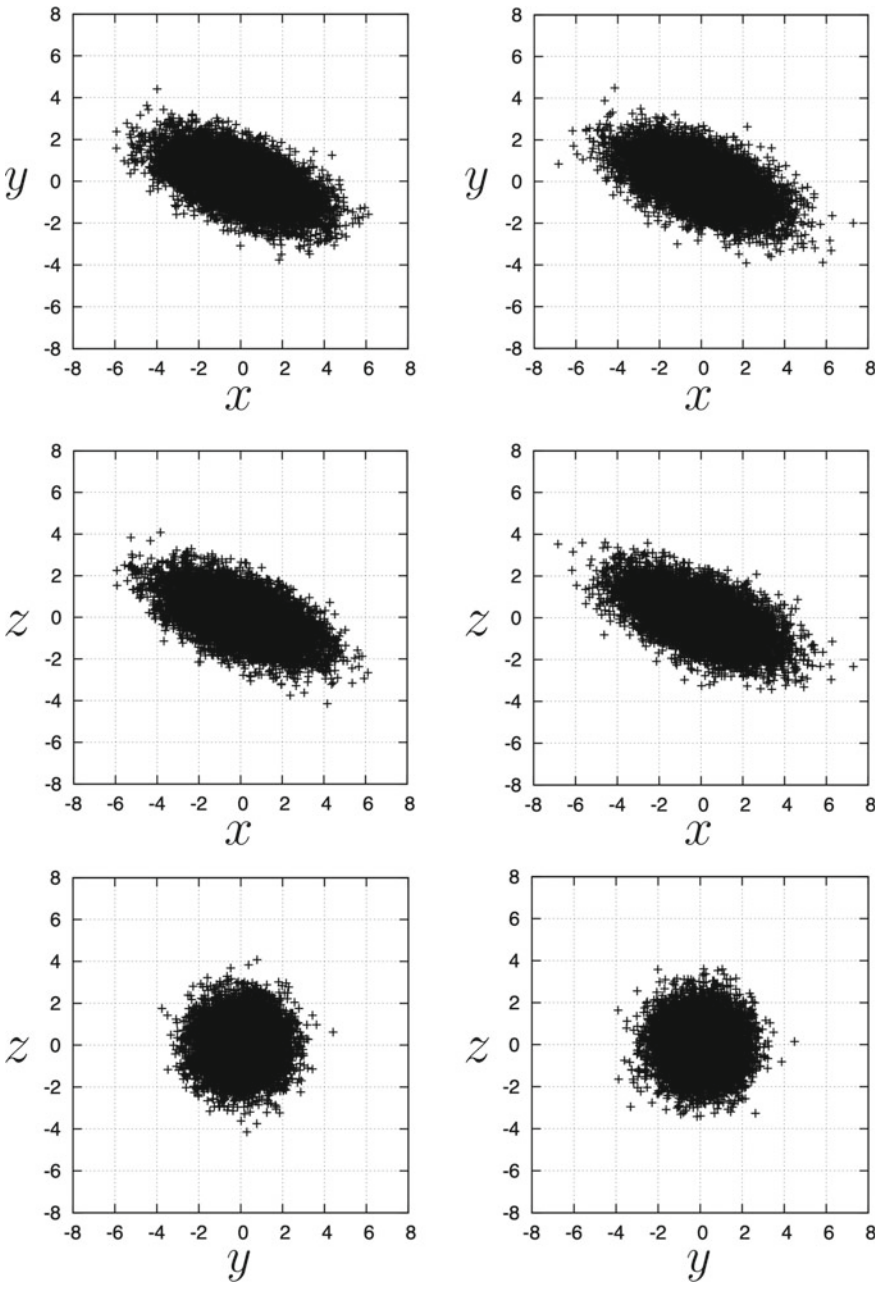

Fig. 5.8 For $S(x, y, z) = \frac{x^2+2y^2+2z^2+2xy+2yz+2zx}{2}$, the two-dimensional scatter plots of the pairs (x, y), (x, z) and (y, z) are shown. The left and right panels are obtained via the HMC algorithm and the Gibbs sampling algorithm, respectively

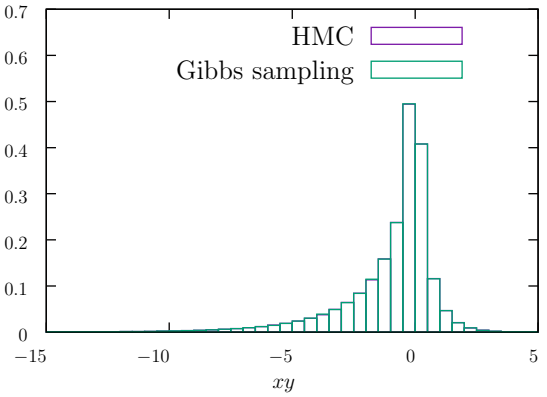

Fig. 5.9 For $S(x, y, z) = \frac{x^2+2y^2+2z^2+2xy+2yz+2zx}{2}$, the histogram of the product xy was generated by using the Gibbs sampling algorithm and the HMC algorithm

$f(x \to x') > 0$ then $f(x' \to x) > 0$ holds as well. (Also, if $f(x \to x') = 0$, then $f(x' \to x) = 0$ as well.) Then, even if $f(x \to x') \neq f(x' \to x)$, by modifying the acceptance probability for $x \to x'$ as

$$\min\left(1, \frac{e^{-S(x')} f(x' \to x)}{e^{-S(x)} f(x \to x')}\right) \tag{5.43}$$

we can keep the detailed balance condition $e^{-S(x)} \cdot T(x \to x') = e^{-S(x')} \cdot T(x' \to x)$ to be valid.

We can confirm the detailed balance condition as follows:

- If $f(x \to x') = f(x' \to x) = 0$, then the transition probability is $T(x \to x') = T(x' \to x) = 0$, and hence $e^{-S(x)} \cdot T(x \to x') = e^{-S(x')} \cdot T(x' \to x)$ holds trivially.
- Below, we focus on the case of $f(x \to x') > 0$ and $f(x' \to x) > 0$. The transition probability is

$$T(x \to x') = f(x \to x') \times \min\left(1, \frac{e^{-S(x')} f(x' \to x)}{e^{-S(x)} f(x \to x')}\right), \tag{5.44}$$

and

$$T(x' \to x) = f(x' \to x) \times \min\left(1, \frac{e^{-S(x)} f(x \to x')}{e^{-S(x')} f(x' \to x)}\right). \tag{5.45}$$

- If $e^{-S(x')} f(x' \to x) \geq e^{-S(x)} f(x \to x')$, we have

$$e^{-S(x)} T(x \to x') = e^{-S(x)} f(x \to x') \times 1, \tag{5.46}$$

and

$$e^{-S(x')}T(x' \to x) = e^{-S(x')}f(x' \to x) \times \frac{e^{-S(x)}f(x \to x')}{e^{-S(x')}f(x' \to x)}. \tag{5.47}$$

Therefore,

$$e^{-S(x)}T(x \to x') = e^{-S(x')}T(x' \to x) = e^{-S(x)}f(x \to x'), \tag{5.48}$$

and the detailed balance condition is satisfied.

- We can repeat the same argument for the case of $e^{-S(x')}f(x' \to x) < e^{-S(x)}f(x \to x')$ as well, just exchanging x and x'.

The expressions above are written as if there is only one variable, but the same argument applies to the multivariate distributions.

This method is called the Metropolis-Hastings algorithm (MH algorithm). If $f(\{x\} \to \{x'\}) = f(\{x'\} \to \{x\})$, the Metropolis-Hastings algorithm is the same as the Metropolis algorithm.

Metropolis-Hastings algorithm (MH algorithm)

1. Suppose a transition $\{x\} \to \{x'\}$ is proposed with a probability $f(\{x\} \to \{x'\})$. We assume that if $f(\{x\} \to \{x'\}) > 0$ then $f(\{x'\} \to \{x\}) > 0$, and if $f(\{x\} \to \{x'\}) = 0$ then $f(\{x'\} \to \{x\}) = 0$. By using this probability $f(\{x\} \to \{x'\})$, from $\{x\} = \{x^{(k)}\}$, the candidate of $\{x^{(k+1)}\}$ denoted by $\{x'\}$ is proposed.
2. Metropolis test: the proposal is accepted, i.e., $\{x^{(k+1)}\} = \{x'\}$, with probability $\min\left(1, \frac{e^{-S(\{x'\})}f(\{x'\} \to \{x\})}{e^{-S(\{x\})}f(\{x\} \to \{x'\})}\right)$. Otherwise the proposal is rejected, i.e., $\{x^{(k+1)}\} = \{x^{(k)}\} = \{x\}$.

5.3.1 Advantage over the Metropolis Algorithm

The acceptance probability in the Metropolis algorithm is

$$\min\left(1, \frac{e^{-S(\{x'\})}}{e^{-S(\{x\})}}\right), \tag{5.49}$$

while in the MH algorithm it is

$$\min\left(1, \frac{e^{-S(\{x'\})}f(\{x'\} \to \{x\})}{e^{-S(\{x\})}f(\{x\} \to \{x'\})}\right). \tag{5.50}$$

While $S(\{x\})$ is given and we cannot change it, we can choose whatever $f(\{x\} \to \{x'\})$ we like. Therefore, *by choosing* $f(\{x\} \to \{x'\})$ *appropriately, the acceptance*

rate can be made closer to 1. Of course, it is not easy to find such $f(\{x\} \to \{x'\})$ in general. As we will see shortly, the Gibbs sampling algorithm and the HMC algorithm can be regarded as the MH algorithm with very clever choices of $f(\{x\} \to \{x'\})$. The Wolff algorithm, which will be introduced in Sect. 6.2.4, is also a version of the MH algorithm.

If we just wanted to recover the detailed balance condition, then other acceptance probabilities such as $\min(1, e^{S(\{x\})-S(\{x'\})}) \times \min\left(1, \frac{f(\{x'\}\to\{x\})}{f(\{x\}\to\{x'\})}\right)$ could be fine. However, the acceptance rate does not go up with such a choice.

5.3.2 Gibbs Sampling is Metropolis-Hastings

In the Gibbs sampling algorithm, $y, z, \ldots$ are fixed and x is updated with the relative weight $e^{-S(x';y,z,\ldots)}$. Therefore,

$$\frac{f(x' \to x)}{f(x \to x')} = \frac{e^{-S(x;y,z,\ldots)}}{e^{-S(x';y,z,\ldots)}}, \tag{5.51}$$

and hence,

$$\frac{e^{-S(x';y,z,\ldots)} f(x' \to x)}{e^{-S(x;y,z,\ldots)} f(x \to x')} = 1. \tag{5.52}$$

Therefore, the Gibbs sampling is a version of the MH algorithm in which $f(x \to x')$ is chosen in a clever manner such that the acceptance rate becomes 100%.

5.3.3 HMC is Metropolis-Hastings

In the HMC algorithm, the auxiliary momentum p is introduced and the time evolution $(x, p) \to (x', p')$ is considered. Hence, the probability that the transition $x \to x'$ is proposed is $f(x \to x') \propto e^{-\frac{p^2}{2}}$. The probability that inverse process $x' \to x$ is proposed is $f(x' \to x) \propto e^{-\frac{p'^2}{2}}$. Therefore,

$$\frac{e^{-S(x')} f(x' \to x)}{e^{-S(x)} f(x \to x')} = \frac{e^{-H(x',p')}}{e^{-H(x,p)}}. \tag{5.53}$$

This is the same as $e^{H_{\text{init}}-H_{\text{fin}}}$ that is used for the Metropolis test in the HMC algorithm. Therefore, the HMC algorithm is a version of the MH algorithm in which $f(x \to x')$ is chosen cleverly.

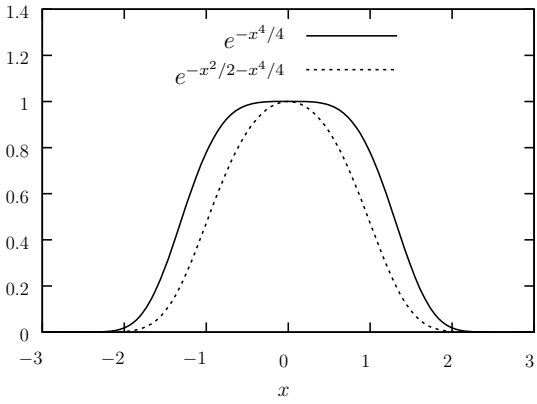

Fig. 5.10 $e^{-\frac{1}{2}x^2-\frac{1}{4}x^4}$ (dashed line) and $e^{-\frac{1}{4}x^4}$ (solid line)

5.3.4 More Elementary Example

As a more elementary example, let us consider a slight deformation of the Gaussian distribution,

$$S(x) = \frac{1}{2}x^2 + \frac{1}{4}x^4. \tag{5.54}$$

By mimicking the Gibbs sampling algorithm, let us generate x' with the probability $\frac{e^{-\frac{1}{2}x'^2}}{\sqrt{2\pi}}$, regardless of x. Then $f(x \to x') = \frac{e^{-\frac{1}{2}x'^2}}{\sqrt{2\pi}}$ and hence

$$\frac{e^{-S(x')}f(x' \to x)}{e^{-S(x)}f(x \to x')} = \frac{e^{-\frac{1}{4}x'^4}}{e^{-\frac{1}{4}x^4}}. \tag{5.55}$$

As we can see from Fig. 5.10, the function $e^{-\frac{1}{4}x^4}$ is rather flat, and hence, the acceptance rate is higher.

5.4 Combination of Different Algorithms

We can use different algorithms for different variables. This is a common and important technique in large-scale simulations in high-energy physics.

As an example, we consider

$$S(x, y) = \frac{y^2}{2}f(x) + g(x), \tag{5.56}$$

where $f(x)$ and $g(x)$ are some complicated functions. In this case, we can divide the simulation into two steps:

- Fix y and update x.
- Fix x and update y.

When x is fixed, y follows the Gaussian distribution. Therefore, we can use the Gibbs sampling algorithm in the following manner:

- Fix y and update x by using the Metropolis algorithm or the HMC algorithm.
- Fix x and update y by using the Gibbs sampling algorithm. Specifically, we generate the Gaussian random number z with variance 1 and take y to be $y = \frac{z}{\sqrt{f(x)}}$.

5.5 Exercises

1. In the HMC algorithm, neither the detailed balance condition nor the (approximate) conservation of the Hamiltonian is violated even if different step sizes are used for different variables. Why?
2. In the HMC algorithm, the detailed balance condition is not broken even if we use different actions $S(x)$ for the leapfrog time evolution and the Metropolis test. (The distribution obtained in this way is determined by the action $S(x)$ used in the Metropolis test.) Why? What is the advantage and disadvantage of using such a trick?
3. In the HMC algorithm, when the number of leapfrog steps N_τ becomes large, while $N_\tau \Delta\tau$ is fixed, the difference of the Hamiltonian before and after the leapfrog time evolution scales as $\Delta H \propto N_\tau^{-2}$ at sufficiently large N_τ (Fig. 5.6). Why? This property is useful for the debugging and the adjustment of the simulation parameters.
4. For the HMC algorithm, we learned how to optimize N_τ and $\Delta\tau$ when $\tau_{\text{fin}} = N_\tau \Delta\tau$ is fixed. How can we estimate the optimal value of τ_{fin} as well?
5. Show that the Jacobian is 1 for the leapfrog time evolution.
6. The Gibbs sampling algorithm is useful for the Gaussian distribution because the Gaussian random numbers can be generated directly without autocorrelation by using the Box-Muller algorithm. The Gibbs sampling algorithm can be powerful for other kinds of distributions if the associated random numbers can be generated without autocorrelation. Having this in mind, how can we generate the probability distribution $\rho(x) = e^{-x}$ $(0 \leq x < \infty)$ from the uniform random number?

Solutions

1. Let the step size for x_i, p_i be $\Delta\tau \times c_i$. When we showed the detailed balance in the HMC algorithm in Sect. 5.1.3, the reversibility of the leapfrog time evolution was the key. The reversibility is preserved even if we use different c_i for different i. It can be confirmed straightforwardly; we recommend checking it. Once the reversibility is established, the argument in Sect. 5.1.3 can be repeated without change.
   In the HMC algorithm, the conservation of the Hamiltonian in the limit of $\Delta\tau \to 0$ is very important. This property is also preserved, as we can see in the following manner. Firstly, the Hamilton equation is modified to

$$\frac{dp_i}{d\tau} = -c_i \frac{\partial H}{\partial x_i}, \qquad \frac{dx_i}{d\tau} = c_i \frac{\partial H}{\partial p_i}. \tag{5.57}$$

   By using it, we can immediately see that

$$\begin{aligned} \frac{dH}{d\tau} &= \sum_i \left(\frac{dx_i}{d\tau}\frac{\partial H}{\partial x_i} + \frac{dp_i}{d\tau}\frac{\partial H}{\partial p_i} \right) \\ &= \sum_i c_i \left(\frac{\partial H}{\partial p_i}\frac{\partial H}{\partial x_i} - \frac{\partial H}{\partial x_i}\frac{\partial H}{\partial p_i} \right) = 0. \end{aligned} \tag{5.58}$$

   Therefore, the Hamiltonian is still conserved.
2. As we mentioned above, we only need the reversibility of the leapfrog time evolution for the detailed balance condition to be satisfied. Therefore, we can use different $S(x)$ for the time evolution and the Metropolis test.
   If we use different $S(x)$, the Hamiltonian is not conserved, and hence, the acceptance rate goes down. This is an obvious disadvantage.
   To see a potential advantage, let us consider a probability distribution $P(x) \propto e^{-1/x^2 - x^2}$, $S(x) = \frac{1}{x^2} + x^2$. We considered this distribution in Sect. 3.2. If we apply the HMC algorithm to this target distribution naively, the simulation is trapped in $x > 0$ or $x < 0$, because $P(0) = 0$ at $x = 0$. However, if we use $S(x) = \frac{1}{x^2+\epsilon} + x^2$, where ϵ is a small positive number, for the leapfrog time evolution, then $P(0)$ becomes nonzero, and hence both $x > 0$ and $x < 0$ can be sampled. In this way, if the action $S(x)$ is singular at some points ($S(x = 0) = \infty$ in the example we have just seen), we might be able to improve the efficiency of the simulation by slightly modifying the action for leapfrog and removing the singularity.
3. Because $N_\tau \Delta\tau$ is fixed, $\Delta\tau$ is proportional to N_τ^{-1}. Therefore, if the error at each step is of order $(\Delta\tau)^3 \propto N_\tau^{-3}$, the contribution from all steps sum up to N_τ^{-2}.
   To confirm that the error at each step is of order $(\Delta\tau)^3$, let us expand the discretized Hamiltonian time evolution in powers of $\Delta\tau$. Then we obtain

$$
\begin{aligned}
x(\tau+\Delta\tau) &= x(\tau) + \Delta\tau \cdot \frac{dx}{d\tau}(\tau) + \frac{(\Delta\tau)^2}{2} \cdot \frac{d^2x}{d\tau^2}(\tau) + O((\Delta\tau)^3) \\
&= x(\tau) + \Delta\tau \cdot p(\tau) + \frac{(\Delta\tau)^2}{2} \cdot \frac{dp}{d\tau}(\tau) + O((\Delta\tau)^3) \\
&= x(\tau) + \Delta\tau \cdot \left(p(\tau) + \frac{\Delta\tau}{2} \cdot \frac{dp}{d\tau}(\tau) \right) + O((\Delta\tau)^3) \\
&= x(\tau) + \Delta\tau \cdot \left(p\left(\tau + \frac{\Delta\tau}{2}\right) + O((\Delta\tau)^2) \right) + O((\Delta\tau)^3) \\
&= x(\tau) + \Delta\tau \cdot p\left(\tau + \frac{\Delta\tau}{2}\right) + O((\Delta\tau)^3). \qquad (5.59)
\end{aligned}
$$

Therefore, the leapfrog time evolution $x(\tau) + \Delta\tau \cdot p\left(\tau + \frac{\Delta\tau}{2}\right)$ approximates the exact time evolution $x(\tau + \Delta\tau)$ up to the error of order $O((\Delta\tau)^3)$. We can repeat a similar calculation for the time evolution of $p(\tau)$ and easily check that the error is $O((\Delta\tau)^3)$.

4. For each N_τ and τ_{fin}, we can calculate the autocorrelation $w(N_\tau, \tau_{\text{fin}})$. The cost for one independent configuration is proportional to $N_\tau \times w(N_\tau, \tau_{\text{fin}})$. Hence we should find the values of N_τ and τ_{fin} that minimize $N_\tau \times w(N_\tau, \tau_{\text{fin}})$.
5. For simplicity, let us consider the case of one variable. The generalization to the multivariate version is straightforward.
   The Jacobian J associated with a transformation $(x, p) \to (x', p')$ is the following determinant:

$$
J = \det \begin{pmatrix} \frac{\partial x'}{\partial x} & \frac{\partial p'}{\partial x} \\ \frac{\partial x'}{\partial p} & \frac{\partial p'}{\partial p} \end{pmatrix} = \frac{\partial x'}{\partial x}\frac{\partial p'}{\partial p} - \frac{\partial p'}{\partial x}\frac{\partial x'}{\partial p}. \qquad (5.60)
$$

   We can easily see that such a determinant is 1 at each step of leapfrog. Indeed, it is $J = 1 \cdot 1 - 0 \cdot \Delta\tau = 1$ for $(x, p) \to (x', p') = (x + p \cdot \Delta\tau, p)$ and $J = 1 \cdot 1 + \frac{\partial^2 S}{\partial x^2}\Delta\tau \cdot 0 = 1$ for $(x, p) \to (x', p') = (x, p - \frac{\partial S}{\partial x} \cdot \Delta\tau)$. The Jacobian associated with the entire leapfrog evolution is the product of these determinants, and hence, it is also 1.
6. We can generate the uniform random number $0 \le y \le 1$, and then change the variable as $x = -\log y$.

References

1. S. Duane, A.D. Kennedy, B.J. Pendleton, D. Roweth, Hybrid Monte Carlo. Phys. Lett. B **195**(2), 216–222 (1987)
2. S. Brooks, A. Gelman, G. Jones, X.-L. Meng, *Handbook of Markov Chain Monte Carlo* (CRC Press, 2011)
3. M.A. Clark, A.D. Kennedy, The RHMC algorithm for two flavors of dynamical staggered fermions. Nucl. Phys. B, Proc. Suppl. **129**, 850–852 (2004)

4. M.A. Clark, The rational hybrid Monte Carlo algorithm, in *PoS, LAT2006* (2006), p. 004
5. S. Aoki, K.-I. Ishikawa, N. Ishizuka, T. Izubuchi, D. Kadoh, K. Kanaya, Y. Kuramashi, Y. Namekawa, M. Okawa, Y. Taniguchi et al., (2009) 2+ 1 flavor lattice QCD toward the physical point. Phys. Rev. D **79**(3), 034503 (2009)
6. S. Dürr, Z. Fodor, J. Frison, C. Hoelbling, R. Hoffmann, S. Katz, S. Krieg, T. Kurth, L. Lellouch, T. Lippert et al., Ab initio determination of light hadron masses. Science **322**(5905), 1224–1227 (2008)
7. M. Hanada, Y. Hyakutake, G. Ishiki, J. Nishimura, Holographic description of a quantum black hole on a computer. Science **344**(6186), 882–885 (2014)
8. E. Berkowitz, E. Rinaldi, M. Hanada, G. Ishiki, S. Shimasaki, P. Vranas, Precision lattice test of the gauge/gravity duality at large n. Phys. Rev. D **94**(9), 094501 (2016)
9. M. Hanada, Markov chain Monte Carlo for dummies, arXiv preprint arXiv:1808.08490 (2018)
10. M. Newman, G. Barkema, *Monte Carlo Methods in Statistical Physics* (Oxford University Press, 1999)
11. D. Landau, K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge University Press, 2000)
12. S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 721–741 (1984)
13. M.A. Tanner, W.H. Wong, The calculation of posterior distributions by data augmentation. J. Am. Stat. Assoc. **82**(398), 528–540 (1987)
14. W.K. Hastings, Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57**(1), 97–109 (1970)

Chapter 6
Applications of Markov Chain Monte Carlo

The Markov Chain Monte Carlo methods have a wide range of applicability. Because each application has its own characteristics, we have to use a suitable algorithm depending on the problem under consideration. For that purpose, it is important to understand what are the basic features common to all applications of MCMC and what are the problem-specific details. In this section, we will show a few concrete examples of the applications of the MCMC methods.

In Sect. 6.1, we will see the applications to statistics including the Bayesian analyses. We focus on basic materials that illuminate the relationship between statistics and the kind of calculations we have already done in previous chapters. We also explain the basic ideas of Bayesian statistics.

In Sect. 6.2, we will study a classic material in university physics courses: the Ising model. Hopefully, it is a familiar example for readers with a science or engineering background. Examples from physics lead us to better intuitive understandings because they are directly connected to natural phenomena. In systems that exhibit the second-order phase transition such as the Ising model, the autocorrelation becomes stronger near the transition point. Such a phenomenon is called the critical slowing down. We will introduce the cluster algorithm, with which we can avoid the critical slowing down in the Ising model.

In Sect. 6.3, we apply the MCMC methods to the combinatorial optimization problems, taking the traveling salesman problem as an example. Naive approaches often fail due to the existence of the local optimal solutions. To circumvent the problem associated with the local optimal solutions, we will introduce the simulated annealing algorithm that is (sometimes) effective, and its improved version, the replica exchange algorithm that is also called the parallel tempering algorithm.

The applications to high-energy physics are reviewed in Sect. 6.4. Various fancy techniques are used and large-scale simulations on supercomputers are routinely performed, but the basic points are just the same as much simpler cases discussed in previous sections. In this section, we will explain the RHMC (rational hybrid Monte Carlo) algorithm that was introduced to deal with the inverse of gigantic matrices.

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7_6

6.1 Likelihood and Bayesian Statistics

In this section, we apply MCMC to statistics. For standard textbooks on statistics, see e.g., Refs. [1–3].

6.1.1 Defining the "Likelihood" Quantitatively

When the Markov Chain Monte Carlo methods are applied to statistics, a confusing point for beginners is the distinction between variables and parameters. Let us get down to the basics and build a firm footing.

Let us consider the coin toss. Usually, every time the coin is tossed, the most natural assumption is that the head and tail appear with the same probability. Suppose, however, that head appeared 9 times out of 10 tosses. Then you would think somebody manipulated the coin. There can be $2^{10} = 1024$ outcomes when the coin is tossed ten times, among them only 10 outcomes are head 9—tail 1. If the head and tail appear with the same probability, there is only about a 1% chance that such an event can happen. It may be a bit too early to conclude that it is cheating, but it is worth considering such a possibility. So let us estimate the probability of heads p at each toss from this outcome. Intuitively, $p = \frac{9}{10}$ sounds likely. It is indeed "likely", but can we justify our intuition by defining the "likelihood" quantitatively? Let us construct a theoretical framework to answer this question.

Now we are assuming that the probability of a head is p at each coin toss. Hence, the probability that the head appears 9 times out of 10 tosses is

$$10p^9(1-p). \tag{6.1}$$

By plotting this as a function of p, we obtain Fig. 6.1. This function takes the maximum value (about 0.39) at $p = \frac{9}{10}$. Hence, it seems to be reasonable, at least to some extent, to conclude that $p = \frac{9}{10}$. On the other hand, if $p = \frac{1}{2}$, such an event (head 9—tail 1) can happen only with the probability 0.01, that is, not really "likely". But we cannot immediately conclude $p = 0.9$, because $10p^9(1-p)$ is large in a finite range, say between $p = 0.8$ and $p = 0.95$, and hence the values of p around that range are reasonably "likely".

Therefore, we adopt the value $10p^9(1-p)$, which is "the probability that the observed outcome (head 9—tail 1) can happen when the probability of a head at each toss is p", as a quantitative index characterizing the likelihood that the probability of a head is actually p. We call this quantity *likelihood*.

More generally, the probability that the head appears k times among 10 tosses is[1]

[1] The number of ways that the head appears k times among n tosses is $\binom{n}{k} = \frac{n!}{(n-k)!k!}$.

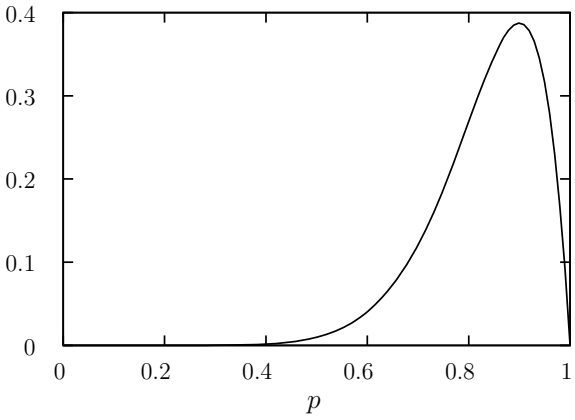

Fig. 6.1 $10p^9(1-p)$, which is the probability that the head appears 9 times out of 10 tosses when the probability of head at each toss is p

$$P(k|p) = \binom{10}{k} \cdot p^k(1-p)^{10-k} = \frac{10!}{k!(10-k)!} \cdot p^k(1-p)^{10-k} \tag{6.2}$$

when the ordering is not specified, and

$$P(k|p) = p^k(1-p)^{10-k} \tag{6.3}$$

when the ordering is specified. The factor $\binom{10}{k}$ does not depend on p and hence does not play any important roles later in this book. We thus use (6.3) in the following. To emphasize the assumption ("the probability of a head is p"), we used the notation $P(k|p)$. This quantity $P(k|p)$ is the likelihood, given the observed outcome that the head appeared k times out of 10 tosses. In the maximum likelihood method, we infer that the value of p that maximizes the likelihood is most natural and hence the right value.

In this way, the likelihood is defined as the probability that a certain event happens, when such an event is observed. The difference from usual probability is that the quantity characterizing the event (in this case, p) is regarded as the variable. "Probability" and "likelihood" are the same function, but the interpretations are different:

- If the probability of head p is given, then $P(k|p)$ is the "probability" that the head appears k times.
- If the outcome (the head appeared k times) is given and we try to estimate the value of p from the given outcome, $P(k|p)$ is the "likelihood" that the value is actually p.

If you find it confusing, it would be better to use different notations, say $P(k|p)$ for the probability and $L(p|k)$ for the likelihood.[2] In this book, we use only $P(k|p)$.

The same idea can be applied to other probability distributions. Consider the Gaussian distribution

[2] $P(p|k)$ means something different from $P(k|p)$; see Eq. (6.23).

$$\rho(x|\mu,\sigma) = \frac{e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}}{\sqrt{2\pi\sigma^2}}. \tag{6.4}$$

For later convenience, we use the notation $\rho(x|\mu,\sigma)$, emphasizing that the mean μ and the width σ are specified. If we generate n random numbers following this distribution, the probability that $x_1, x_2, \ldots, x_n$ are obtained is

$$\begin{aligned} P(x_1,\ldots,x_n|\mu,\sigma) &= \rho(x_1|\mu,\sigma)\times\rho(x_2|\mu,\sigma)\times\cdots\times\rho(x_n|\mu,\sigma) \\ &= \prod_{i=1}^{n}\rho(x_i|\mu,\sigma). \end{aligned} \tag{6.5}$$

Suppose you have a black-box routine that creates the Gaussian random number, but you do not know the values of σ and μ. You used this routine n times and obtained $x_1, \ldots, x_n$. From such an observed outcome, you want to make a good guess on the values of σ and μ. Then, by interpreting $P(x_1, \ldots, x_n|\mu, \sigma)$ as a function of μ and σ, we can think such (μ, σ) that makes $P(x_1, \ldots, x_n|\mu, \sigma)$ larger is more likely. Therefore, $P(x_1, \ldots, x_n|\mu, \sigma)$ can be interpreted as the likelihood as well.

In traditional statistics, it is assumed that parameters such as p for the coin toss and μ, σ for the Gaussian distribution are (even if you do not know the actual values, in principle) already uniquely determined. The values of those parameters are determined by using the maximum likelihood method. As we will see in Sect. 6.1.3, in Bayesian statistics, the notion of "probability distribution of p" is admitted. Then, by combining the likelihood and Bayes' theorem, the probability distribution of p is estimated.

So far we have assumed that the functional forms of the likelihood function $P(k|p)$ or $P(x_1, \ldots, x_n|\mu, \sigma)$ are known. In actual applications, even if the data like k or $\{x_i\}$ are given, usually we do not know the form of the likelihood functions, and we have to find reasonable likelihood functions. The methods we will explain below (especially the Metropolis algorithm and the HMC algorithm) can be applied to complicated likelihood functions, so we should try various functions and find a reasonable one that returns a sufficiently large value of the likelihood.

The Least Squares Method

The least squares method, which is used for the fit of the experimental data very often, is a version of the maximum likelihood method.

Imagine a theory parametrized by $\vec{x}$. In the case of the Ising model (Sect. 6.2), the size of the lattice n, temperature T, the coupling constant J, and the external magnetic field h correspond to $\vec{x}$. Suppose that a physical quantity (say the energy E) is measured at different points in the parameter space, $\vec{x}_1, \ldots, \vec{x}_K$, and as the results $y_1, \ldots, y_K$ are obtained. We fit the data by using a function $y = f(\vec{x}; \mu_1, \mu_2, \ldots)$. For the Ising model, you can imagine something like $E = \mu_1 T^{\mu_2}$. The fitting parameters

are $\mu_1, \mu_2, \ldots$. The least squares method gives us a guideline for the fit. In the least squares method, $\mu_1, \mu_2, \ldots$ are chosen such that

$$\sum_{k=1}^{K} (y_k - f(\vec{x}_k; \mu_1, \mu_2, \ldots))^2 \tag{6.6}$$

is minimized.

As we have seen in Sect. 2.1.2, the errors in the experiments often follow the Gaussian distribution. So, let us assume it here. It is natural to identify the width of the Gaussian distribution at each point $\vec{x}_k$ with the error bar associated with the measured value y_k, which we denote by σ_k. We can estimate them by using the Jackknife method. Then, when the fitting parameters $\mu_1, \mu_2, \ldots$ are fixed, the outcome would be $y_1, \ldots, y_K$ with the following probability:

$$P(y_1, \ldots, y_K | \mu_1, \mu_2, \ldots) = \prod_{k=1}^{K} \rho_{\sigma_k, \vec{x}_k}(y_k | \mu_1, \mu_2, \ldots), \tag{6.7}$$

$$\rho_{\sigma_k, \vec{x}_k}(y_k | \mu_1, \mu_2, \ldots) \equiv \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{(y_k - f(\vec{x}_k; \mu_1, \mu_2, \ldots))^2}{2\sigma_k^2}}. \tag{6.8}$$

We interpret this P as the likelihood function. In the maximum likelihood method, $\mu_1, \mu_2, \ldots$ are chosen such that the likelihood is maximized or, equivalently, such that

$$\sum_{k=1}^{K} \frac{(y_k - f(\vec{x}_k; \mu_1, \mu_2, \ldots))^2}{\sigma_k^2} \tag{6.9}$$

is minimized.

If we do not know the variance σ_k^2, let us just assume that $\sigma_1 = \cdots = \sigma_K = \sigma$. Then, regardless of the value of σ, the minimization of (6.9) reduces to the minimization of (6.6). In this sense, the least squares method is a version of the maximum likelihood method. The minimization of (6.9) is called the weighted least squares method.

6.1.2 Calculation of Likelihood

Now we calculate the likelihood via MCMC. As usual, we consider the Gaussian distribution as a concrete example.

One Variable

We start with the simplest case with just one variable. Suppose random numbers $x_1, x_2, \ldots, x_n$ are given and we know or we assume that these random numbers were obtained from the Gaussian distribution $\rho(x|\mu, \sigma)$ but we do not know the parameters of the distribution μ and σ. Our mission is to estimate the values of μ and σ from $x_1, x_2, \ldots, x_n$. The probability that these n numbers are obtained from the Gaussian distribution with the mean μ and width σ is given by (6.5). We regard it as a function of μ and σ, and interpret it as the likelihood of the assumption that the parameters are those values.

In order to use MCMC, we define the action $S(x_1, \ldots, x_n|\mu, \sigma)$ by $P(x_1, \ldots, x_n|\mu, \sigma) \propto e^{-S(x_1,\ldots,x_n|\mu,\sigma)}$. In the current example,

$$\begin{aligned} S(x_1, \cdots, x_n|\mu, \sigma) &= \sum_{i=1}^{n} \frac{(x_i - \mu)^2}{2\sigma^2} + n \log \sigma \\ &= \frac{n}{2\sigma^2}\mu^2 - \frac{\sum_{i=1}^n x_i}{\sigma^2}\mu + \frac{\sum_{i=1}^n x_i^2}{2\sigma^2} + n \log \sigma \\ &= n\left(\frac{1}{2\sigma^2}(\mu - \overline{x})^2 + \frac{1}{2\sigma^2}\left(\overline{x^2} - \overline{x}^2\right) + \log \sigma\right). \end{aligned} \tag{6.10}$$

Here $\overline{x} \equiv \frac{1}{n}\sum_{i=1}^n x_i$ and $\overline{x^2} \equiv \frac{1}{n}\sum_{i=1}^n x_i^2$ are the averages of x_i and x_i^2, respectively. The term $n \log \sigma$ comes from the normalization factor $(2\pi\sigma^2)^{-n/2}$.

We can get some amount of information already from this expression. Because of the overall factor n in the final line, only the minimum contributes to the limit of infinite n. Therefore, we obtain $\mu = \overline{x}$ and $\sigma = \sqrt{\overline{x^2} - \overline{x}^2}$ when $n = \infty$.[3] It is a natural result: when the number of samples n is infinite, the set of the samples describes the probability distribution precisely, and hence the mean and the variance of the samples are exactly the same as those of the probability distribution we want to find. When n is finite, the distributions of μ and σ have widths of order $\frac{1}{\sqrt{n}}$. Roughly speaking, these widths are the statistical errors.

Once the action is given, we can immediately calculate the likelihood via MCMC. For the Gaussian distribution, we can apply the Metropolis algorithm, the HMC algorithm, or the Gibbs sampling algorithm straightforwardly.

The Metropolis and HMC algorithms can be applied to more complicated distributions as well, and furthermore, we can write a code without using our brain too much. The Metropolis algorithm is especially easy, we can just use the algorithm explained in Sect. 4.1 without any modifications. To use the HMC algorithm, we have to calculate the derivative of $S(x_1, \cdots, x_n|\mu, \sigma)$ with respect to μ and σ:

[3] When S is minimum, $\frac{\partial S}{\partial \mu} = \frac{\partial S}{\partial \sigma} = 0$ holds. Therefore, we should find such μ and σ that make the right-hand side of (6.11) zero.

$$\frac{\partial S}{\partial \mu} = \frac{n(\mu - \overline{x})}{\sigma^2},$$
$$\frac{\partial S}{\partial \sigma} = n\left(-\frac{1}{\sigma^3}(\mu - \overline{x})^2 - \frac{1}{\sigma^3}\left(\overline{x^2} - \overline{x}^2\right) + \frac{1}{\sigma}\right). \tag{6.11}$$

Then, we can just repeat the procedures explained in Sect. 5.1. A disadvantage of the Metropolis and HMC algorithms is that we need to adjust the parameters for the simulations. This is not difficult but tedious.

If we use the Gibbs sampling, the coding is more involved, but there is no need for parameter tuning. When you have to solve similar problems many times, or if somebody already wrote a program for you, it would be better to use the Gibbs sampling algorithm. In the current example, μ follows the Gaussian distribution. Therefore, as explained in Sect. 5.2, the Box-Muller method can be used when σ is fixed and μ is updated. The update of σ is a bit complicated, but we can use the Metropolis or HMC algorithm; note that, as we have learned in Sect. 5.4, we can combine different algorithms.

Multiple Variables

The same argument goes through for the case of multiple variables. Let us consider the d-variate version of the Gaussian distribution,

$$\rho(x_1, \cdots, x_d | A, \mu) = \sqrt{\frac{\det A}{(2\pi)^d}} \cdot \exp\left(-\frac{1}{2}\sum_{i,j=1}^{d} A_{ij}(x_i - \mu_i)(x_j - \mu_j)\right). \tag{6.12}$$

When n sets of data $\{x^{(k)}\} = (x_1^{(k)}, \ldots, x_d^{(k)})$ $(k = 1, 2, \ldots, n)$ are given, the likelihood function is

$$\begin{aligned} &P(\{x^{(1)}\}, \ldots, \{x^{(n)}\} | A, \mu) \\ &= \prod_{k=1}^{n} \rho(x_1^{(k)}, \ldots, x_d^{(k)} | A, \mu) \\ &= \left(\frac{\det A}{(2\pi)^d}\right)^{n/2} \exp\left(-\frac{1}{2}\sum_{k=1}^{n}\sum_{i,j=1}^{d} A_{ij}(x_i^{(k)} - \mu_i)(x_j^{(k)} - \mu_j)\right). \end{aligned} \tag{6.13}$$

The action defined by $P(\{x^{(1)}\}, \ldots, \{x^{(n)}\} | A, \mu) \propto e^{-S(\{x^{(1)}\}, \ldots, \{x^{(n)}\} | A, \mu)}$ can be simplified in the following manner:

$$
\begin{aligned}
&S(\{x^{(1)}\},\ldots,\{x^{(n)}\}|A,\mu) \\
&= \frac{1}{2}\sum_{k=1}^{n}\sum_{i,j=1}^{d} A_{ij}(x_i^{(k)}-\mu_i)(x_j^{(k)}-\mu_j) - \frac{n}{2}\log\det A \\
&= \frac{n}{2}\sum_{i,j=1}^{d} A_{ij}\mu_i\mu_j - \sum_{k=1}^{n}\sum_{i,j=1}^{d} A_{ij}x_i^{(k)}\mu_j + \frac{1}{2}\sum_{k=1}^{n}\sum_{i,j=1}^{d} A_{ij}x_i^{(k)}x_j^{(k)} - \frac{n}{2}\log\det A \\
&= \frac{n}{2}\sum_{i,j=1}^{d} A_{ij}\left(\mu_i\mu_j - 2\overline{x_i}\mu_j + \overline{x_ix_j}\right) - \frac{n}{2}\log\det A \\
&= \frac{n}{2}\left\{\sum_{i,j=1}^{d} A_{ij}\left(\mu_i-\overline{x_i}\right)\left(\mu_j-\overline{x_j}\right) + \sum_{i,j=1}^{d} A_{ij}\left(\overline{x_ix_j}-\overline{x_i}\,\overline{x_j}\right) - \log\det A\right\}.
\end{aligned}
\tag{6.14}
$$

By throwing this expression into one of the algorithms we have already seen, we can calculate the likelihood.

Here we adopt the HMC algorithm. We use

$$
\log\det A = \mathrm{Tr}\log A \tag{6.15}
$$

and

$$
\frac{\partial \mathrm{Tr}\log A}{\partial A_{ij}} = \left(A^{-1}\right)_{ji}. \tag{6.16}
$$

(see Appendix B for derivations.) By using these formulas, we obtain[4]

$$
\frac{\partial S}{\partial A_{ij}} = \frac{n}{2}\left\{\left(\mu_i\mu_j - \overline{x_i}\mu_j - \overline{x_j}\mu_i + \overline{x_ix_j}\right) - (A^{-1})_{ij}\right\} \tag{6.17}
$$

and

$$
\frac{\partial S}{\partial \mu_i} = n\sum_{j=1}^{d} A_{ij}\left(\mu_j - \overline{x_j}\right). \tag{6.18}
$$

We calculate the right-hand side of these equations numerically for the leapfrog time evolution. The hardest part is the calculation of the inverse matrix A^{-1}. If d is not so large, the inverse matrix can be calculated easily. For example, for $d = 2$, we can use the formula

[4] We showed the expressions by treating A_{ij} and A_{ji} as independent variables. Strictly speaking, because we are assuming $A_{ij} = A_{ji}$, the factor 2 should be multiplied when $i \neq j$. The same factor 2 appears also for $p^{(A)}$, and hence this factor does not affect the Hamilton equation used in the HMC simulation.

$$A^{-1} = \frac{1}{A_{11}A_{22} - A_{12}A_{21}} \begin{pmatrix} A_{22} & -A_{12} \\ -A_{21} & A_{11} \end{pmatrix}. \tag{6.19}$$

Algorithms for calculating the inverse are known for $d \geq 3$ as well, so you can implement one of them by yourself, or you can use a package for linear-algebra calculations such as LAPACK.

The remaining steps are essentially the same as in Sect. 5.1. We introduce the "momenta" $p_{ij}^{(A)}$ and $p_i^{(\mu)}$ that correspond to the variables A_{ij} and μ_i, respectively. Because A_{ij} is a real symmetric matrix, $p_{ij}^{(A)}$ is also a real symmetric matrix. The Hamiltonian is defined as

$$H(p^{(A)}, p^{(\mu)}, A, \mu) = \frac{1}{2}\sum_{i,j=1}^{d}(p_{ij}^{(A)})^2 + \frac{1}{2}\sum_{i=1}^{d}(p_i^{(\mu)})^2 + S(A, \mu). \tag{6.20}$$

Note that $\frac{1}{2}\sum_{i,j}(p_{ij}^{(A)})^2 = \frac{1}{2}\sum_i (p_{ii}^{(A)})^2 + \sum_{i<j}(p_{ij}^{(A)})^2$, because $p_{ij}^{(A)} = p_{ji}^{(A)}$. From this it follows that, for $i \neq j$, the momentum $p_{ij}^{(A)} = p_{ji}^{(A)}$ has to be the Gaussian random number with the width $\frac{1}{\sqrt{2}}$. The width for other components $p_{ii}^{(A)}$ and $p_i^{(\mu)}$ is 1 as before. Except that we have to be careful about the normalization of the momenta, there is no difference from the computations that appeared in Sect. 5.1. The Hamilton equation is

$$\frac{dp_{ij}^{(A)}}{d\tau} = -\frac{n}{2}\left\{\left(\mu_i\mu_j - \overline{x_i}\mu_j - \overline{x_j}\mu_i + \overline{x_i x_j}\right) - (A^{-1})_{ij}\right\}, \qquad \frac{dA_{ij}}{d\tau} = p_{ij}^{(A)}, \tag{6.21}$$

$$\frac{dp_i^{(\mu)}}{d\tau} = -n\sum_{j=1}^{d} A_{ij}\left(\mu_j - \overline{x_j}\right), \qquad \frac{d\mu_i}{d\tau} = p_i^{(\mu)}. \tag{6.22}$$

For the Gaussian distribution to make sense, the matrix A_{ij} has to be positive definite (i.e., all eigenvalues are positive). But we did not impose such a condition in the HMC algorithm; we just let the value of A_{ij} change following the Hamilton equation. Then the meaningless configurations with negative eigenvalues would also be sampled, wouldn't they?

It is a good point, but, in fact, such meaningless configurations rarely appear. If we set the matrix A_{ij} in the initial configuration to be positive definite, meaningless configurations can appear only when the simulation goes through a region where $\det A = 0$. However, when $\det A = 0$, the action is infinite and hence such a region does not appear in the simulation. It is like an infinitely high wall is blocking us. Because the HMC algorithm mimics the time evolution in classical mechanics, there is almost no chance that the configuration moves toward $\det A = 0$; even if it happened it would not go beyond the infinitely high wall at $\det A = 0$. Hence,

the positive definiteness is rarely violated. Here we said "rarely" because there is a negligibly small but mathematically nonzero probability of a violation: because the time evolution is discretized, the configuration can go beyond $\det A = 0$ due to an unexpected combination of coincidences if the step size is large. If you want to use a large step size, it would be better to check the positive definiteness of the matrix A_{ij} after every update.

When the size of the matrix A_{ij} (denoted by d here) is large, the calculations of the inverse and determinant are costly, and hence some sort of improvement is needed. If the matrix size is a few hundred, we would only have to use linear-algebra packages such as LAPACK. Even if you have to use even bigger matrices, if the matrices can be made sparse (i.e., many components are zero) by using problem-specific properties or by performing appropriate preprocessing to the data, the computational cost can be reduced substantially. In high-energy physics (Sect. 6.4), it is necessary to deal with the inverse of a gigantic matrix, and various techniques are invented in order to reduce the simulation cost. When such techniques can be useful, you should not hesitate to adopt them.

6.1.3 Bayesian Statistics

So far, we argued that our assumption is more likely to be true if the likelihood function takes a larger value. But such an argument may be too naive. In the example considered in Sect. 6.1.1 (tossed a coin ten times and observed the head nine times), the likelihood function is maximum when $p = \frac{9}{10}$, where p is the probability of head at each coin toss. But honestly, $p = \frac{9}{10}$ is an unrealistic value. It is too big, isn't it? If the authors manipulate the coin, we would set p a bit closer to $\frac{1}{2}$, because somebody would find out the cheating otherwise. Usually, before tossing a coin, people assume the head and tail are equally likely. Even if the head appeared 9 times, it would not be reasonable to reject the original assumption immediately and jump to an extreme conclusion, $p = \frac{9}{10}$. It would be more reasonable to think like "I assumed that the head and tail are equally likely, but given that the head appeared so many times, it would be better to change my assumption a little bit". To put such an idea into a mathematical setting, let us introduce the probability distribution of p or, equivalently, the probability that the probability of a head is p, which we denote by $P(p)$. In Bayesian statistics, such "probability of probability" is inferred by using Bayes' theorem.

Suppose we have a reasonable guess about $P(p)$ before we observe the head 9 times out of 10. This is the distribution of p before tossing the coin, so we call it the prior probability distribution or simply the prior. We can imagine various prior probability distributions, for example:

- $P(p) \propto e^{-M\left(p-\frac{1}{2}\right)^2}$ (Fig. 6.2, top-left). The distribution peaks around $p = 1/2$, and the peak becomes sharper as M becomes larger. In the top-left panel of Fig. 6.2, $M = 100$ is used. This prior describes a rather common assumption that probably

nobody is cheating, and even if somebody is cheating they will not set p too far away from $\frac{1}{2}$. The value of M corresponds to the level of trust. When $M = \infty$ (i.e., 100% trust), the distribution completely localizes to $\frac{1}{2}$ and becomes Dirac's delta function $\delta\left(p - \frac{1}{2}\right)$.

- $P(p) = 1$ (Fig. 6.2, top-middle). This means any value of p is equally likely; we can see some distrust.
- $P(p) \propto \left(p - \frac{1}{2}\right)^2$ (Fig. 6.2, top-right). Because $P(p) = 0$ at $p = \frac{1}{2}$, some degree of cheating is assumed with 100% possibility. Hence, this prior distribution means a strong distrust.

Suppose we tossed a coin 10 times, and the head appeared k times (not necessarily 9 times). Based on this outcome, we would like to improve our guess. For that purpose, let us recall the definition of the likelihood function $P(k|p)$. It is the same as the conditional probability that the head appears k times given that the probability of head at each toss is p. Before we tossed a coin, we assumed that "the probability of the probability" is the prior $P(p)$. Therefore, it appears to be natural to think that, given the condition that the outcome was k heads and $10 - k$ tails, "the probability of the probability" is the product of the likelihood function and the prior probability distribution.[5] Hence, we denote the normalization factor by $P(k)$ and define the posterior probability distribution $P(p|k)$ or simply the posterior as

$$P(p|k) = \frac{P(k|p) \cdot P(p)}{P(k)}. \tag{6.23}$$

This is the improved guess after knowing the outcome. The normalization factor $P(k)$ is defined by $P(k) = \int dp P(k|p) \cdot P(p)$. (A "mathematical" justification of Eq. (6.23) will be given in Sect. 6.1.4.) In this way, by using the outcome of the trials (or experiments) we can improve our guess and get a more plausible probability distribution. Such a procedure is called the Bayesian updating.

For the previous example regarding $P(p)$, let us calculate the posterior distributions $P(p|k)$ for the cases of $k = 9$ and $k = 5$.

- The first example is $P(p) \propto e^{-M\left(p-\frac{1}{2}\right)^2}$ ($M = 100$). For $k = 9$, the center of the distribution moves to the right slightly (Fig. 6.2, middle-left). However, it is not very far from $\frac{1}{2}$, which means the trust has not been affected much. For $k = 5$ (Fig. 6.2, bottom-left), it may be difficult to see the difference between $P(p)$ and $P(p|k)$; the center does not move, but the width becomes slightly narrower, which would mean a slightly stronger trust. If M is infinity (i.e., 100% trust), no outcome can affect the trust; the posterior is also Dirac's delta function, $P(p|k) = P(p) = \delta\left(p - \frac{1}{2}\right)$.
- If $P(p) = 1$, the posterior probability distribution is $P(p|k) = \frac{P(k|p)}{P(k)}$. Because $P(k)$ is merely a normalization constant, $P(p|k) \propto P(k|p)$ (Fig. 6.2, the middle of the second third columns). As a prior, we assumed that any value of p is

[5] As we will see later, we can interpret it as an immediate consequence of the definition of the conditional probability, $P(A|B) = \frac{P(A,B)}{P(B)}$.

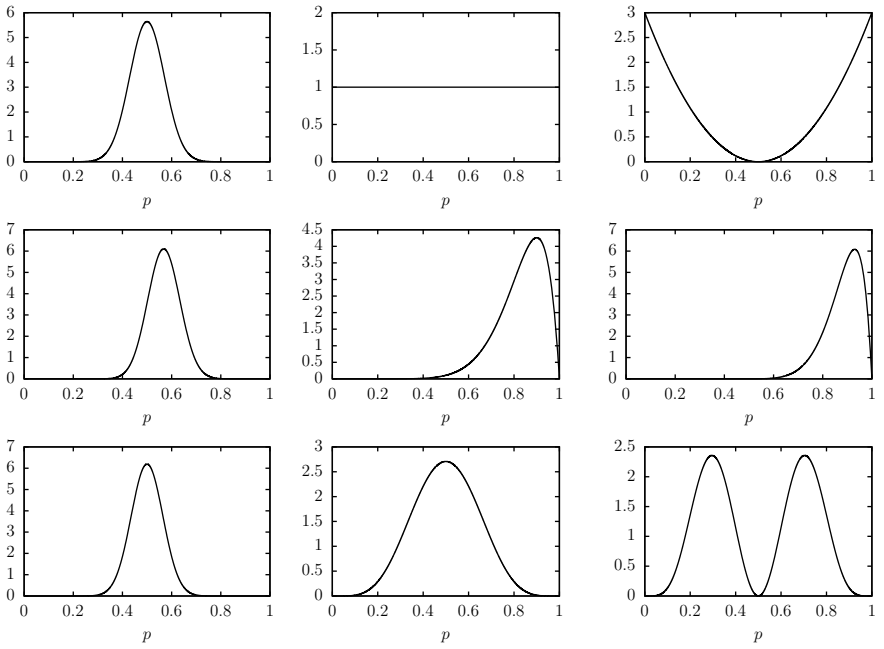

Fig. 6.2 Examples of the prior probability distribution (top row) and the posterior probability distribution (middle row ($k = 9$) and bottom row ($k = 5$)). From the left to right, the prior probability distributions $P(p) \propto e^{-100\left(x-\frac{1}{2}\right)^2}$, $P(p) = 1$, and $P(p) \propto \left(p - \frac{1}{2}\right)^2$

equally likely. Hence, the posterior is the same as the likelihood function up to normalization.

- The final example is $P(p) \propto \left(p - \frac{1}{2}\right)^2$. The posterior for $k = 9$ resembles the one with another choice of the prior $P(p) = 1$ at first glance, but the distribution is slightly tilted toward the right reflecting the stronger distrust in the prior (Fig. 6.2, middle-right).
  Note that, regardless of the value of k, the posterior is always zero at $p = \frac{1}{2}$, i.e., $P(p = \frac{1}{2}|k) = 0$. If one assumed the manipulation without a doubt ($P(p = \frac{1}{2}) = 0$), then even if a reasonable result suggesting the absence of manipulation ($k = 5$) is obtained, one would think "they are trying to hide the cheating, so they are pretending a fair result" or something. Still, the posterior is more centered, which would mean a decrease in the level of distrust.

We could obtain the posterior distribution $P(p|k)$ by improving the prior distribution based on the outcome of the experiment. By using the posterior distribution, we can do various calculations. For example, the probability that the head appears twice in a row was estimated as

$$\int_0^1 dp P(p) \times p^2 \tag{6.24}$$

before the experiment, but after the experiment it is estimated as

$$\int_0^1 dp P(p|k) \times p^2. \tag{6.25}$$

By repeating the experiments, we can improve the precision of the inference further. If we toss the coin 10 more times and get the head k' times, then we can re-interpret $P(p|k)$ as the prior distribution and derive a new posterior distribution $P(p|k', k)$ as

$$P(p|k', k) = P(k'|p) \cdot P(p|k)/(\text{normalization factor}). \tag{6.26}$$

We can repeat the same procedure to obtain $P(p|k'', k', k)$, $P(p|k''', k'', k', k)$, and so on.

As almost the same example, let us repeat the comparison of the Japan series and random walk (Sect. 3.1). In this case, "the coin is manipulated" should be rephrased as "either the Central League or the Pacific League is stronger". The outcome was as follows:

	Random Walk	Japan Series
1950 – 1959	C6 – P4	C5 – P5
1960 – 1969	C6 – P4	C8 – P2
1970 – 1979	C5 – P5	C6 – P4
1980 – 1989	C2 – P8	C5 – P5
1990 – 1999	C2 – P8	C5 – P5
2000 – 2009	C5 – P5	C5 – P5
2010 – 2019	C1 – P9	C1 – P9
1950 – 2019	C27 – P43	C35 – P35

(6.27)

We take the prior to be $P(p) = 1$. If the Central League wins k times within n years, the likelihood function is $P(p|n, k) = p^k(1-p)^{n-k}$. Here p is the probability that the Central League wins each year, i.e., their strength. Based on the actual historical data from 1950 to 1959, we can obtain the posterior as of 1959 as

$$P_{1959}(p) \propto p^5(1-p)^5. \tag{6.28}$$

The historical data from 1960 to 1969 (C8−P2) gives the likelihood function $P(p|10, 8) = p^8(1-p)^2$, and hence

$$P_{1969}(p) \propto p^5(1-p)^5 \times p^8(1-p)^2 = p^{13}(1-p)^7. \tag{6.29}$$

Above we treat 10 years as one batch, but we can use other units, say 20 years. Then the result from 1950 to 1969 (C13−P7) gives the likelihood function $P(p|20, 13) = p^{13}(1-p)^7$. By multiplying it to the prior distribution $P(p) = 1$, we obtain the same result as above. In a simple example considered here

$$\begin{aligned} P(p|n,k) \cdot P(p|n',k') &\propto p^k(1-p)^{n-k} \cdot p^{k'}(1-p)^{n'-k'} \\ &= p^{k+k'}(1-p)^{(n+n')-(k+k')} \\ &\propto P(p|n+n', k+k'), \end{aligned} \tag{6.30}$$

and hence the result does not change whether we update at once or with multiple steps. It is a desirable feature because we do not want the result to depend on artificial factors, e.g., whether one batch consists of 10 or 20 years. As a result, the posterior distribution obtained from all the data for 70 years is $P(p|k) \propto p^{35}(1-p)^{35}$. The center of the distribution is $p = 1/2$.

Can we conclude that the Central League and the Pacific League are equally competitive, based on this result? In other words, how precisely can we estimate the value of p? To find the answer to this question, we used the results of the random-walk experiment (C27−P43) and the actual results of the Japan series (C35−P35) to calculate the posterior distributions, assuming the prior distribution $P(p) = 1$; see Fig. 6.3. The peaks of the posterior distributions obtained from the actual result and random-walk experiment are at $p = 35/70 = 1/2$ and $p = 27/70 \simeq 0.39$, respectively. From this result, one may have the impression that the estimated value of p from the experimental data ($p \simeq 0.39$) is far from the true value used for the experiment ($p = 1/2$), but it is not a bad estimate because the distribution is rather wide. This is a typical example that a sharp estimate cannot be made due to the lack of statistics. For the same reason, we cannot immediately conclude that the Central League and the Pacific League are equally competitive, even though the peak of the posterior obtained from the actual result is at $p = 1/2$. Needless to say, if we collect more statistics in the random-walk experiment, the location of the peak goes close to $p = \frac{1}{2}$ and the width of the distribution becomes narrower. We repeated the random-walk experiment 1,000 times and obtained C485−P515. The posterior obtained by using this result is also shown in Fig. 6.3. We can clearly see that p is at least very close to $\frac{1}{2}$. It is easy to perform the same numerical experiment with different values of p. If you try, you can get intuition into the relationship between statistics and error.

The same idea can be applied to more generic probability distributions. For the Gaussian distribution,

$$P(\mu, \sigma|x_1, \ldots, x_n) = \frac{P(x_1, \ldots, x_n|\mu, \sigma) \cdot P(\mu, \sigma)}{P(x_1, \ldots, x_n)}. \tag{6.31}$$

Again, in this case, due to a relation $P(x_1, \ldots, x_n|\mu, \sigma) = \prod_{i=1}^n \rho(x_i|\mu, \sigma)$, whether we perform the Bayesian updating at once or with multiple steps does not affect the final result.

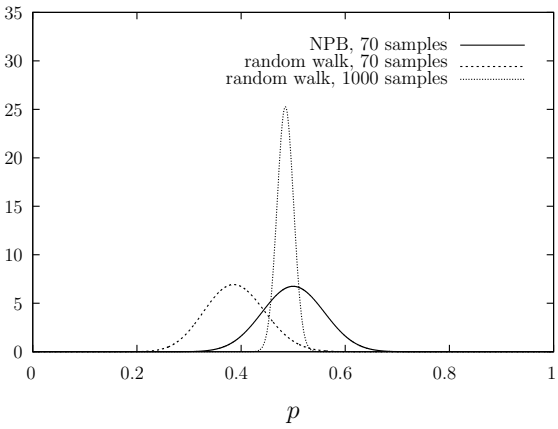

Fig. 6.3 The posterior probability distributions obtained from the actual result of the Japan series (C35−P35; Nippon Professional Baseball, NPB) and random-walk experiments (C27−P43, C485−P515). As the prior probability distribution, $P(p) = 1$ was used

6.1.4 Bayes' Theorem

Equation (6.23), which is crucial for the calculation of the posterior probability distribution, can be derived "mathematically". This relation is called Bayes' theorem. Here we put the quotation mark on "mathematically" because we have to specify the meaning of probability properly. To determine the probability of a head p, we can repeat the coin toss many times and count how many times the head is obtained, and then take the limit of the infinite number of trials.[6] Because such a probability is based on the frequency of the events, it is often called frequentist probability or frequentist inference. Bayes' theorem (6.23) can be proven naturally in the context of the frequentist inference.

With such a definition of probability, the value of p is uniquely determined. Therefore, for each given coin "the probability that p is $\frac{1}{2}$" (probability of probability) is 0 or 1, and hence, generic probability distribution $P(p)$ cannot be defined. On the other hand, in the Bayesian approach, the subjective probabilities are used and we think like "because we got the head nine times out of ten, the value of p should be around here". Other examples of the subjective probabilities include "the probability that Alice is the criminal", "the probability that Bob gets accepted to Cambridge", "the probability that Democratic party wins the majority", and "the probability that it is snowing the day after tomorrow at noon". Bayes' theorem (6.23) is applied to such subjective probabilities as well. This procedure may appear unnatural to some people, but it is usually possible to relate the subjective probability to the frequentist probability, at least at a conceptual level, as "collect many cases with similar sets of evidence, and check how many times the person like Alice was the criminal", "check how many students with comparable grades to Bob got accepted to

[6] Such a "definition" can drive mathematicians mad because it implicitly assumes the convergence to a certain well-defined limit. A more precise statement is "if there is such a thing as probability, it can be determined by repeating infinitely many trials".

Cambridge", or "given the meteorological conditions that are indistinguishable with current observation accuracy, check how often it is snowing two days later at noon". Many people around the authors (at least many physicists) understand the subjective probability in this way. Therefore, even if the theorem proven for the frequentist probability is applied to the subjective probability, a natural result (or at least a result that appears to be natural for many people) is obtained. We adopt such a viewpoint, and we use the frequentist approach to show Bayes' theorem.

Suppose there are many coins. We can toss each coin many times and determine the probability of a head, p. We write the value of p obtained in this way to the coin. By performing such an experiment on all coins, we can determine the probability distribution $P(p)$. Note that p is uniquely determined for each coin.

Next, we choose one coin randomly from the bunch of coins, toss it 10 times, and the number of times the head appeared, k. Then we keep (k, p) in a memo. Then we return the coin to the bunch, take another coin randomly, and repeat the same experiment. We repeat this procedure many times. Let $n_{k,p}$ be the number of times a particular set of (k, p) is observed.[7] From $n_{k,p}$, we can determine the probability that "a coin with a letter p is chosen, and then the head appeared k times", which is denoted as $P(k, p)$. Because both k and p are specified, it is called the joint probability. The explicit formula is

$$P(k, p) = \lim_{N\to\infty} \frac{n_{k,p}}{N}, \qquad N = \sum_{k,p} n_{k,p}. \tag{6.32}$$

We can also define the conditional probability $P(k|p)$ that "when a coin with the letter p is chosen, the head appeared k times", by focusing on the coins with a specific value of p:

$$P(k|p) = \lim_{N\to\infty} \frac{n_{k,p}}{N_p}, \qquad N_p = \sum_{k} n_{k,p}. \tag{6.33}$$

The normalization factor N_p is the number of times that the coins with the letter p were chosen. In the same manner, we can focus on the cases where the head appeared k times and define the conditional probability $P(p|k)$ that "when the head appeared k times, the letter written on the coin was p":

$$P(p|k) = \lim_{N\to\infty} \frac{n_{k,p}}{N_k}, \qquad N_k = \sum_{p} n_{k,p}. \tag{6.34}$$

The normalization factor N_k is the number of times that the head appeared k times. Furthermore, if we look only at k and ignore p, we can get the probability $P(k)$ that "head appeared k times", and if we look only at p and ignore k we can get $P(p)$ defined previously:

[7] Here we assumed p is discrete. Essentially the same argument goes through when p is continuous, just by replacing the sum with the integral.

$$P(k) = \lim_{N\to\infty} \frac{\sum_p n_{k,p}}{N} = \lim_{N\to\infty} \frac{N_k}{N}, \qquad P(p) = \lim_{N\to\infty} \frac{\sum_k n_{k,p}}{N} = \lim_{N\to\infty} \frac{N_p}{N}. \tag{6.35}$$

By using these relations, we can write the joint probability $P(k, p)$ as

$$P(k, p) = P(p|k)P(k), \tag{6.36}$$

and also as

$$P(k, p) = P(k|p)P(p). \tag{6.37}$$

By combining these two expressions, we obtain

$$P(k|p)P(p) = P(p|k)P(k), \tag{6.38}$$

which is equivalent to Eq. (6.23).

Although Bayes' theorem can be proven mathematically, the opinions regarding the interpretation are divided among the statisticians. In the Bayesian approach, the concept of the subjective probability is admitted, and Bayes' theorem is used as a tool to improve the inference on the subjective probability. The validity of such an approach is not something that can be proven or which should be proven. Such an approach appears natural to many people, and it is very useful when used properly. However, unless we choose good functional forms for the likelihood function and the prior distribution, and use reliable data for the Bayesian updating, unreasonable results may be obtained.

6.1.5 Bayesian Updating via MCMC

The estimate of the posterior distribution $P(p|k)$ or $P(\mu, \sigma|x_1, \ldots, x_n)$ via (6.23) or (6.31) is a typical task to which Markov Chain Monte Carlo is applicable. We only have to add the effect from the prior $P(p)$ or $P(\mu, \sigma)$ to the calculation of the likelihood function explained in Sect. 6.1.2.

Coin Toss via Metropolis

Suppose there is a coin with the prior distribution $P(p)$, where p is the probability of head. We tossed it n times and got the head k times. Then, to obtain the posterior distribution, we only have to generate p with the probability proportional to

$$e^{-S(p|k)} \equiv P(k|p)P(p) = p^k(1-p)^{n-k}P(p). \tag{6.39}$$

Note that this expression makes sense only at $0 \le p \le 1$, because p is a probability. At $p < 0$ and $p > 1$, we should set $e^{-S(p|k)} = 0$ (equivalently, $S(p|k) = \infty$) or $P(p) = 0$.

We can use the Metropolis algorithm to obtain a sequence $\cdots \to p^{(i)} \to p^{(i+1)} \to \cdots$. A concrete procedure is as follows:

Bayesian updating for coin toss via Metropolis

1. Choose a random number Δp between $-c$ and $+c$ and propose $p' = p^{(i)} + \Delta p$ as a candidate of $p^{(i+1)}$.
2. Reject the proposal if $p' < 0$ or $p' > 1$. (Then $p^{(i+1)} = p^{(i)}$.)
3. If $0 \le p' \le 1$, perform the Metropolis test, i.e., generate a random number r between 0 and 1 and accept the proposal if $r < e^{S(p|k)-S(p'|k)}$ ($p^{(i+1)} = p'$), otherwise reject the proposal ($p^{(i+1)} = p^{(i)}$).

As a prior distribution, let us use $P(p) \propto e^{-100\left(p-\frac{9}{10}\right)^2}$. Such a prior would mean we are assuming very bad cheating. We also assume $n = 1000$, $k = 515$. To obtain the posterior distribution via Metropolis, we use the action $S(p|k)$ defined by

$$\begin{aligned} S(p|k) &= -k \log p - (n-k)\log(1-p) - \log P(p) \\ &= -515 \log p - 485 \log(1-p) + 100\left(p - \frac{9}{10}\right)^2 . \end{aligned} \tag{6.40}$$

In Fig. 6.4, the history of the simulation is shown for two choices of the initial value, $p = 0.9$ and $p = 0.5$. The step size c was chosen to be 0.1. Regardless of the initial value, the oscillation around $p = 0.5 \sim p = 0.55$ can be seen after some time. With $p = 0.9$ it takes some time for thermalization, but still, we can obtain a reasonable result by discarding the first 100 steps or so.

In Fig. 6.5, the distribution of p after thermalization is shown. We can see the convergence to the right answer as statistics increases.

If we want to know the probability of the head appearing twice in a row, $\int_0^p dp\, P(p|k) \times p^2$, we calculate the expectation value $\langle p^2 \rangle$ by using thermalized configurations.

Multivariate Gaussian Distribution via Metropolis

The next example is the Gaussian distribution,

$$S(x_1, \ldots, x_d) = \frac{1}{2} \sum_{i,j=1}^{d} A_{ij}(x_i - \mu_i)(x_j - \mu_j) \qquad (A_{ij} = A_{ji}). \tag{6.41}$$

We consider the same setup as in Sect. 4.8.1: we choose $d = 2$, $A_{11} = 1$, $A_{22} = 1$, $A_{12} = \frac{1}{2}$, $\mu_1 = \mu_2 = 0$ and generate n sets of random numbers (x, y). We assume

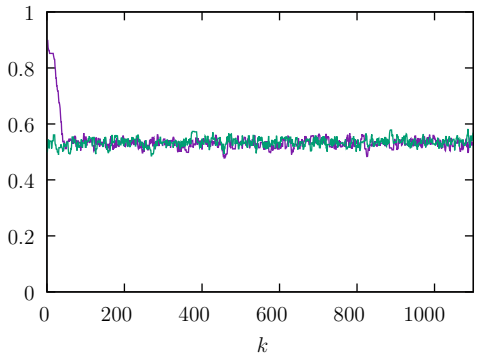

Fig. 6.4 Coin toss with the weight factor (6.40), simulated by using the Metropolis algorithm with step size $c = 0.1$. The initial values are $p = 0.5$ (green) and $p = 0.9$ (purple). The vertical axis is the value of p and the horizontal axis is the number of steps. To remove the un-thermalized configurations, it is enough to discard the first 100 steps or so

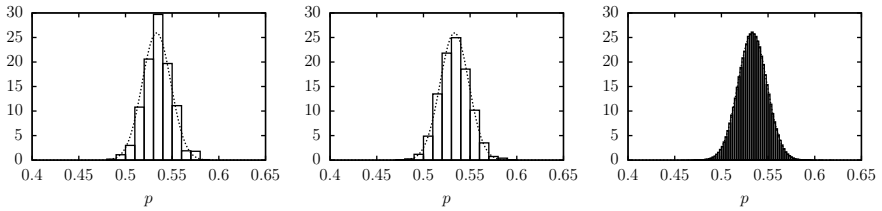

Fig. 6.5 Coin toss with the weight factor (6.40) via the Metropolis algorithm with step size 0.1. The initial value is $p = 0.5$. The first 100 steps are removed to ensure thermalization. From left to right, 1,000 configurations, 10,000 configurations, and 1,000,000 configurations. The dashed line is the target distribution

these numbers are described by the Gaussian distribution and infer the likely values of A_{ij} and μ_i. As the prior distribution, let us use $P(\{A_{ij}, \mu_i\}) \propto e^{-\frac{1}{2}\sum_{i,j}|A_{ij}|^2 - \frac{1}{2}\sum_i |\mu_i|^2}$. (If you like, you can use more complicated function.) Then, we only have to use the product of the likelihood function $P(\{x^{(1)}\}, \ldots, \{x^{(n)}\}|A, \mu)$ defined by (6.13) and the prior $P(\{A_{ij}, \mu_i\})$ as the probability distribution for the MCMC simulation. The action is the sum of (6.14) and the contribution from the prior $\Delta S = \frac{1}{2}\sum_{i,j}|A_{ij}|^2 + \frac{1}{2}\sum_i |\mu_i|^2$. The determinant $\det A$ in (6.14) is $\det A = A_{11}A_{22} - A_{12}^2$ in the case of $d = 2$.

We show the result of the simulation for $n = 100$, $\overline{x_1} = -0.0930181$, $\overline{x_2} = 0.0475899$, $\overline{x_1 x_1} = 1.06614$, $\overline{x_2 x_2} = 1.28152$, and $\overline{x_1 x_2} = -0.504944$ in Fig. 6.6. By regarding the widths of the distribution of A_{ij} and μ_i as the error bars, we can see that the correct values are obtained within errors.

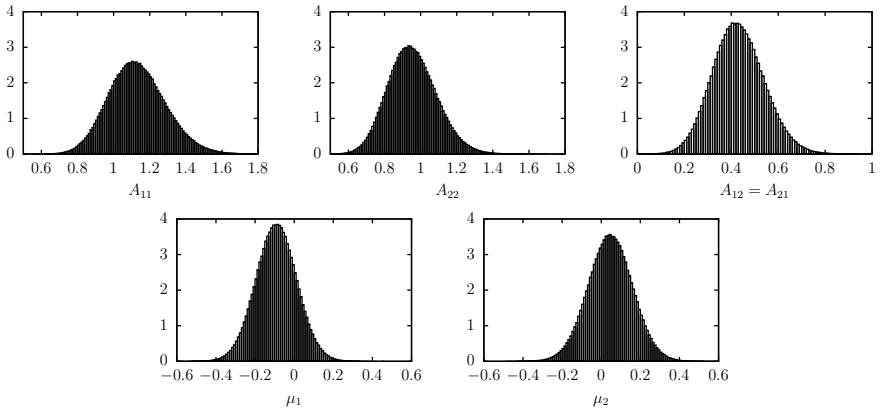

Fig. 6.6 The inferred values of A_{ij} and μ_i for $n = 100$, $\overline{x_1} = -0.0930181$, $\overline{x_2} = 0.0475899$, $\overline{x_1 x_1} = 1.06614$, $\overline{x_2 x_2} = 1.28152$, and $\overline{x_1 x_2} = -0.504944$. The top row, from left to right: A_{11}, A_{22} and $A_{12} = A_{21}$. The bottom row, from left to right: μ_1, μ_2

Multivariate Gaussian Distribution via HMC

The HMC algorithm is a convenient method that can be used whenever we can write down the Hamilton equation. In this example, we add the contribution from the prior distribution to (6.21) and (6.22):

$$\frac{dp_{ij}^{(A)}}{d\tau} = -\frac{n}{2}\left\{\left(\mu_i\mu_j - \overline{x_i}\mu_j - \overline{x_j}\mu_i + \overline{x_i x_j}\right) - (A^{-1})_{ij}\right\} - A_{ij}, \qquad \frac{dA_{ij}}{d\tau} = p_{ij}^{(A)}, \tag{6.42}$$

$$\frac{dp_i^{(\mu)}}{d\tau} = -n\sum_{j=1}^{d} A_{ij}\left(\mu_j - \overline{x_j}\right) - \mu_i, \qquad \frac{d\mu_i}{d\tau} = p_i^{(\mu)}. \tag{6.43}$$

As in the example in Sect. 6.1.2, the matrices A_{ij} and p_{ij} are symmetric. For each A_{ij}, $p_{ij}^{(A)}$ and μ_i, $p_i^{(\mu)}$, different step size can be used.

Multivariate Gaussian Distribution via a Combination of Gibbs Sampling and Metropolis

For simple priors such as the Gaussian distribution, μ can be treated via Gibbs sampling. The action $S(\{x^{(1)}\}, \ldots, \{x^{(n)}\}|A, \mu)$ is given by (6.14), and terms containing μ_i are written as $\frac{n}{2}\sum_{i,j=1}^{d} A_{ij}\left(\mu_i - \overline{x_i}\right)\left(\mu_j - \overline{x_j}\right)$. To this, we add the contribution

from the prior distribution, $\frac{1}{2}\sum_{i=1}^{d}\mu_i^2$. Then, for each i, the weight factor can be written in the Gaussian form,

$$\begin{aligned}
&\frac{n}{2}\sum_{j,k=1}^{d} A_{jk}\left(\mu_j-\overline{x_j}\right)\left(\mu_k-\overline{x_k}\right)+\frac{1}{2}\sum_{j=1}^{d}\mu_j^2 \\
&=\left\{\frac{1+nA_{ii}}{2}\mu_i^2+n\left(\sum_{j\neq i}A_{ij}(\mu_j-\overline{x_j})-A_{ii}\overline{x_i}\right)\mu_i\right\}+\text{terms not containing } \mu_i \\
&=\frac{1+nA_{ii}}{2}\left(\mu_i+\frac{n}{1+nA_{ii}}\left(\sum_{j\neq i}A_{ij}(\mu_j-\overline{x_j})-A_{ii}\overline{x_i}\right)\right)^2+\text{terms not containing } \mu_i .
\end{aligned} \tag{6.44}$$

Therefore, μ_i can be updated by using the Gibbs sampling method. To handle A_{ij}, it is convenient to use the Metropolis algorithm or the HMC algorithm.

6.2 Ising Model

In this section, we study the Ising model [4, 5]. This model is the classic among classics taught in university physics courses. Good textbooks on the Monte Carlo study of the Ising model include Refs. [6, 7].

We consider the spins (small magnets) sitting on lattice points. The dimension of the lattice can be arbitrary. The shape can be also arbitrary, say square, triangular, or hexagonal. We use i to label the lattice points and s_i to denote the spin at point i. Each spin takes only two values, $s_i=+1$ (N-pole is up) or $s_i=-1$ (S-pole is up). In pictures, $s_i=+1$ and $s_i=-1$ are often depicted by upward and downward arrows, respectively. When all the spins align in the same direction, they form a strong magnet as a whole. If two directions are mixed, they cancel with each other and become a weak magnet. When the numbers of up and down spin are the same, it is not a magnet macroscopically.

Spins sitting nearby interact with each other. To make the story simple, let us assume that only the nearest-neighbor spins interact. Then, the energy of the system is given by

$$E(\{s\})=-J\sum_{\langle i,j\rangle}s_i s_j-h\sum_i s_i . \tag{6.45}$$

Here $\langle i,\ j\rangle$ stands for a pair of $i,\ j$ sitting next to each other. The parameter J describes the interaction. As we will see, when J is a large positive value the spins take the same direction and the system as a whole becomes a strong magnet. When J is negative, various interesting physics appear depending on the shape of the lattice. The other parameter h is the external magnetic field. When the magnetic field is strong (i.e., h

is a large positive value or large negative value), spins tend to align to the direction of the magnetic field.

Each configuration is realized with the probability

$$P(\{s\}) = \frac{e^{-\beta E(\{s\})}}{Z}, \tag{6.46}$$

where Z is the partition function which is defined by

$$Z = \sum_{\{s\}} e^{-\beta E(\{s\})}. \tag{6.47}$$

The parameter β is related to the absolute temperature T by[8]

$$\beta = \frac{1}{T}. \tag{6.48}$$

Below, we will explain how to construct a sequence of configurations $\{s^{(0)}\} \to \{s^{(1)}\} \to \{s^{(2)}\} \to \cdots \to \{s^{(k)}\} \to \{s^{(k+1)}\} \to \cdots$ that follows the statistical distribution (6.46) via MCMC.

6.2.1 *Ising Model via Metropolis*

The simplest implementation of the Metropolis algorithm is as follows:

Ising model via Metropolis algorithm

1. Choose a lattice point i randomly.
2. Flip the i-th spin, leaving all other spins untouched: $s_i' = -s_i^{(k)}$, $s_j' = s_j^{(k)}$ $(j \neq i)$. Propose this $\{s'\}$ as a candidate for $\{s^{(k+1)}\}$.
3. By using the difference of the energy before and after flipping the i-th spin, i.e., $\Delta E = E(\{s'\}) - E(\{s^{(k)}\})$, the proposal is accepted with the probability $\min(1, e^{-\beta \Delta E})$ and the configuration is updated as $\{s^{(k+1)}\} = \{s'\}$. Otherwise the proposal is rejected, i.e., $\{s^{(k+1)}\} = \{s^{(k)}\}$.

Note that ΔE depends only on s_i and neighboring spins, and hence the computational cost for each update is small. We can also choose the lattice points in some specific order. Or we can do more acrobatic things like choosing a positive integer n randomly and then flipping randomly chosen n spins simultaneously. It is not wrong, although probably there is no advantage.

[8] More precisely, $\beta = \frac{1}{k_B T}$, where k_B is the Boltzmann constant. Note that the Boltzmann constant can be set to 1, just by changing the definition of temperature by an overall constant.

6.2.2 Ising Model via Gibbs Sampling (Heat Bath)

We can also use the Gibbs sampling algorithm introduced in Sect. 5.2.

Ising model via Gibbs sampling (heat bath)

1. Choose a lattice point i randomly.
2. Consider two configurations $s_i' = \pm 1$, $s_j' = s_j^{(k)}$ $(j \neq i)$, and calculate the energies $E_\pm$ corresponding to $s_i' = \pm 1$.
3. New configuration $\{s^{(k+1)}\}$ is $s_i^{(k+1)} = +1$ with the probability $\frac{e^{-\beta E_+}}{e^{-\beta E_+} + e^{-\beta E_-}}$ and $s_i^{(k+1)} = -1$ with the probability $\frac{e^{-\beta E_-}}{e^{-\beta E_+} + e^{-\beta E_-}}$. All the spins but the i-th one are left untouched, i.e., $s_j^{(k+1)} = s_j^{(k)} (j \neq i)$.

The update probability $\frac{e^{-\beta E_\pm}}{e^{-\beta E_+} + e^{-\beta E_-}}$ is the conditional probability that the i-th spin is $s_i = \pm 1$ when all other spins are fixed. Therefore, this is the same as the Gibbs sampling. Actually, the name "Gibbs sampling" comes from the fact that this probability distribution is called the Gibbs distribution (or also the Boltzmann distribution) in physics. To calculate $\delta \equiv E_+ - E_-$, we only have to take into account the interaction with the spins sitting next to the i-th point. By using δ, the update probability can be written as $\frac{e^{-\beta E_+}}{e^{-\beta E_+} + e^{-\beta E_-}} = \frac{e^{-\beta\delta}}{e^{-\beta\delta} + 1}$ and $\frac{e^{-\beta E_-}}{e^{-\beta E_+} + e^{-\beta E_-}} = \frac{1}{e^{-\beta\delta} + 1}$. Hence, the computational cost for update probability is small.

Readers with some knowledge of thermodynamics and statistical physics would have noticed that the conditional probability used above is the equilibrium distribution, where all spins but the i-th one are regarded as the heat bath. This interpretation is valid not just for the Ising model but also for any examples to which the Gibbs sampling algorithm can be applied. For this reason, in the physics community, the Gibbs sampling algorithm is also called the heat bath algorithm. This point of view enables us to better understand both physics and the Gibbs sampling algorithm.

6.2.3 Simulation of Two-Dimensional Ising Model

Let us consider the Ising model on the two-dimensional square lattice. In this case, the analytic solution is known thanks to Onsager [8]. We denote the lattice points by (i_x, i_y), where i_x and i_y take integer values from 1 to n. We impose the periodic boundary condition in order to reduce the finite-volume effect, namely, $i_x = n$ and $i_x = 1$ are taken to be next to each other, so as $i_y = n$ and $i_y = 1$. (Equivalently, we identify $n + 1$ and 1. See Fig. 6.7.)

For each lattice site, we can assign a serial number $i = n(i_x - 1) + i_y$, which runs from 1 to n^2. Let us generate a random number r between 0 and n^2, and vary the spin sitting on the i-th lattice point if $i - 1 \leq r < i$.

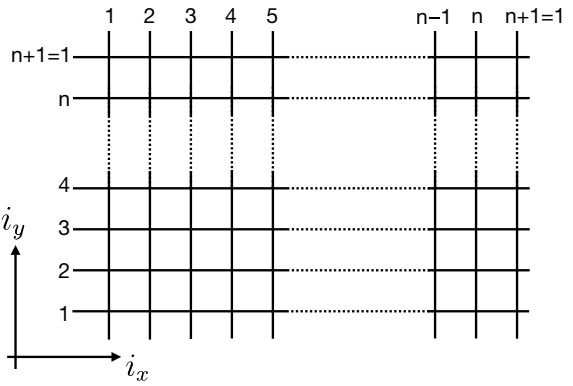

Fig. 6.7 Two-dimensional square lattice with periodic boundary condition. $i_x = n+1$ and $i_x = 1$, and $i_y = n+1$ and $i_y = 1$ are identified

In the Metropolis algorithm, when we vary s_{i_x,i_y}, the change of the energy ΔE depends only on s_{i_x,i_y} itself and the spins on the neighboring lattice points, $s_{i_x\pm1,i_y}$, $s_{i_x,i_y\pm1}$. More concretely, we only have to calculate the change of

$$\left(-J\sum_{\pm1}\left(s_{i_x\pm1,i_y}+s_{i_x,i_y\pm1}\right)-h\right)s_{i_x,i_y}. \tag{6.49}$$

Because the spin at (i_x, i_y) is flipped as $s_{i_x,i_y} \to s'_{i_x,i_y} = -s_{i_x,i_y}$, the change of the energy is

$$\Delta E = 2\left(J\sum_{\pm1}\left(s_{i_x\pm1,i_y}+s_{i_x,i_y\pm1}\right)+h\right)s_{i_x,i_y}. \tag{6.50}$$

Zero External Magnetic Field

Suppose that the external magnetic field h is zero. Then, the action is $S = \frac{E}{T} = -\frac{J}{T}\sum_{\langle i,j\rangle} s_i s_j$ and hence the ratio $\frac{J}{T}$ is the only independent parameter. The specific heat C is obtained by taking the derivative of the expectation value of the energy $\langle E\rangle$ with respect to temperature T:

$$\begin{aligned} C &\equiv \frac{\partial\langle E\rangle}{\partial T} \\ &= \frac{\partial}{\partial T}\frac{\sum_{s_i=\pm1} E e^{-\frac{E}{T}}}{Z}. \\ &= \frac{1}{Z}\cdot\frac{\partial}{\partial T}\left(\sum_{s_i=\pm1} E e^{-\frac{E}{T}}\right)+\left(\sum_{s_i=\pm1} E e^{-\frac{E}{T}}\right)\cdot\frac{\partial}{\partial T}\left(\frac{1}{Z}\right) \end{aligned}$$

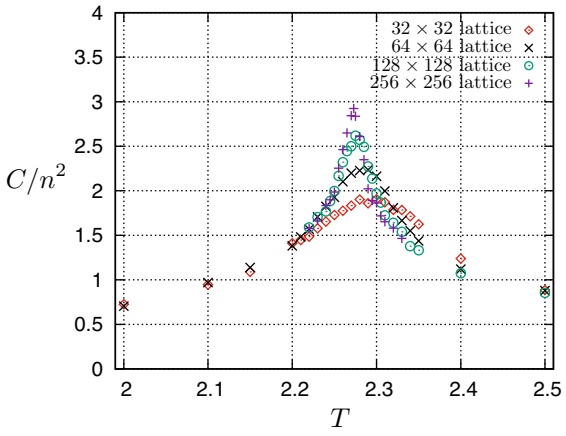

Fig. 6.8 The specific heat per volume, $\frac{C}{n^2}$, in the two-dimensional Ising model at $J = 1, h = 0$. As the volume increases, $\frac{C}{n^2}$ diverges at $T \simeq 2.269$. To avoid the critical slowing down, we used the Wolff algorithm which is introduced in Sect. 6.2.4. The error bars are omitted in this plot

$$= \frac{1}{Z} \cdot \left(\sum_{s_i = \pm 1} \frac{E^2}{T^2} e^{-\frac{E}{T}} \right) + \left(\sum_{s_i = \pm 1} E e^{-\frac{E}{T}} \right) \cdot \left(-\frac{1}{Z^2} \frac{\partial Z}{\partial T} \right)$$

$$= \frac{1}{T^2} \left(\langle E^2 \rangle - (\langle E \rangle)^2 \right). \tag{6.51}$$

The energy of the system is approximately proportional to the volume (the number of lattice points) n^2. Hence, the property of the system can be better understood by looking at the energy and specific heat per volume, $\frac{E}{n^2}$ and $\frac{C}{n^2}$. In the large-volume limit $n \to \infty$, which is also called the thermodynamic limit, $\frac{C}{n^2}$ diverges. In physics, this is called the second-order phase transition. The phase transition temperature is $T \simeq 2.269\,\mathrm{J}$ (more precisely, $\sinh\left(\frac{2J}{T}\right) = 1$) [8]. In Fig. 6.8, $\frac{C}{n^2}$ is plotted as a function of temperature. (The simulation was done by using the Wolff algorithm, which is explained later.) We can see that $\frac{C}{n^2}$ diverges at $T \simeq 2.269\,\mathrm{J}$ as the volume becomes large.

At zero temperature, all spins align upward or downward. Even at finite temperature, if the temperature is sufficiently low, mostly up and mostly down configurations are dominant. Therefore, two peaks appear in the distribution of total spin $\sum_{i=1}^{n^2} s_i$. In such a case, the simulation can be trapped in one of the peaks, which can lead to tricky problems. This issue has been discussed in Sect. 4.4. The same problems happen very often in the optimization problems discussed in Sect. 6.3. However, in the case of $h = 0$, two peaks have exactly the same properties except for the sign of the spin, and hence it is sufficient to study only one of the two peaks. Rather, another difficult problem—the critical slowing down—arises when two peaks merge. We will discuss the critical slowing down in Sect. 6.2.4.

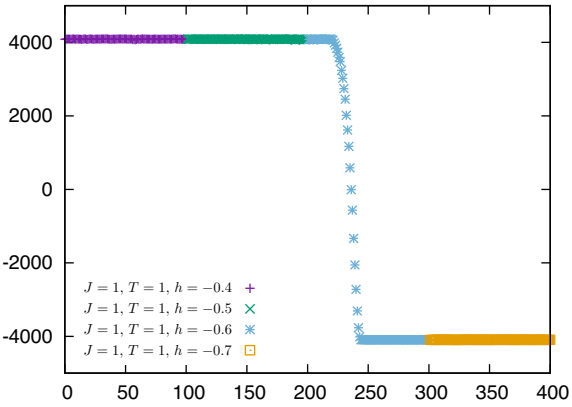

Fig. 6.9 Total spin of two-dimensional Ising model $\sum_{i=1}^{n^2} s_i$ obtained by using the Gibbs sampling algorithm. The lattice size is $n^2 = 64^2 = 4096$, and one sample is collected every $10n^2 = 40960$ steps. The coupling constant and temperature are $J = 1$ and $T = 1$, respectively, and the external magnetic field is varied every 100 samples. The initial condition is all-up ($s_i = +1$ for all i)

Nonzero External Magnetic Field

Next, let us vary the external magnetic field h, fixing the coupling constant and temperature to be $J = 1$ and $T = 1$. When h is zero, there are two ground states: all-up ($s_i = +1$ for all i) and all-down ($s_i = -1$ for all i). When h is not zero, this degeneracy is resolved, and the ground state is all-up for $h > 0$ and all-down for $h < 0$.

Let us focus on the case of $h < 0$ below. We start from $h = -0.4$ with the all-up initial condition and perform the simulation with the Gibbs sampling algorithm. The lattice size is $n^2 = 64^2 = 4096$, and we sample one configuration every $10n^2 = 40960$ steps. We lower the value of h by 0.1 for every 100 samples. The result obtained this way is shown in Fig. 6.9. The vertical axis is the total spin. At first, the sum is close to 4096, which means almost all spins are up. This is a similar situation to Fig. 4.9: it is hard to transit from one of two peaks (almost-up and almost-down) to the other. This kind of situation is seen when the external magnetic field h is small and the temperature T is low. If we wait a long time, eventually a small cluster of down spins is formed, then it spreads quickly and the transition to the true vacuum is realized. In the current example, at $h = -0.6$ it is easier for such a transition to take place, and the almost-down phase appears. In this way, depending on the parameters and initial condition, wrong peaks (which are called the metastable states in physics) can be sampled. Often we cannot notice such a problem just by looking at the history of the simulation and hence we have to be extra cautious. We will have the same sort of headache when we study the optimization problems in Sect. 6.3.

The rate that the transition takes place depends on the detail of the algorithm. With the Wolff algorithm, the transition takes place immediately whatever small h and T are; see Sect. 6.2.4.

6.2.4 Critical Slowing Down and Cluster Algorithm

If you actually performed simulations for the Ising model, you might have had a hard time studying parameter region near the phase transition via the Metropolis algorithm or Gibbs sampling algorithm, because autocorrelation becomes larger. The reason that we see such an increase in autocorrelation is that we are using a local update procedure (i.e., only one spin is flipped at each time) while a global phenomenon (the phase transition), which changes the property of the system drastically, is taking place. To understand it more precisely and more intuitively, let us see what is happening near the phase transition point.

In Fig. 6.10, typical configurations at several different temperatures are shown for $J = 1$, $h = 0$ and lattice size 512×512. (These plots are obtained by using the Wolff algorithm.) Up and down spins are denoted by white and black, respectively. At low temperatures, almost all points are white (spin $+1$) and black dots (spin -1) are sparsely scattered. As the temperature is raised, clusters of down spins become bigger. Eventually, both up and down spins form big clusters. Above the phase transition temperature, neither up spins nor down spins form a big cluster, and they are mixed well. So we can see that the phase transition is not realized by flipping each spin. Rather, large blocks of up or down spin are formed, and then they gradually become smaller.

This is why the simulation slows down near the phase transition if we use the Metropolis algorithm or Gibbs sampling algorithm. At low temperatures, almost all spins take the same direction, so we can efficiently collect independent samples just by flipping one spin at each time. (Note, however, that it is hard to go from almost-up to almost-down and vice versa.) However, as the temperature goes up and big clusters are formed, it becomes harder to get independent samples unless many spins in a big cluster are flipped simultaneously. With the local update procedure (i.e., updating only one spin at each time), it takes a very long time to reach the target distribution, and long autocorrelation is inevitable. This phenomenon is the critical slowing down. At sufficiently high temperatures, up and down spins are mixed well, so the local update procedure can work efficiently again.

The plots obtained via the Gibbs sampling algorithm are shown in the first row of Fig. 6.11. The lattice size is $n^2 = 64^2 = 4096$, and the coupling constant and the external magnetic fields are chosen to be $J = 1$ and $h = 0$, respectively. As the initial condition, we chose all spins to be up. One configuration is sampled at every 40960 steps, and 10,000 configurations are collected in total. The vertical axis is the total spin. At low temperature ($T = 2.20$), the simulation is trapped in one of the two ground states. This is problematic in the sense that the right distribution is not realized, but not particularly a problem in the sense that it reflects the spontaneous breaking of the symmetry at low temperature, which is an important physical phenomenon.[9] Near the phase transition ($T = 2.35$), the total spin goes back and force between the positive and negative values, but the autocorrelation is rather large. If we go even closer to the phase transition ($T = 2.30$), the autocorrelation increases further. This

[9] Similar situations are discussed in Sect. 6.3 in detail.

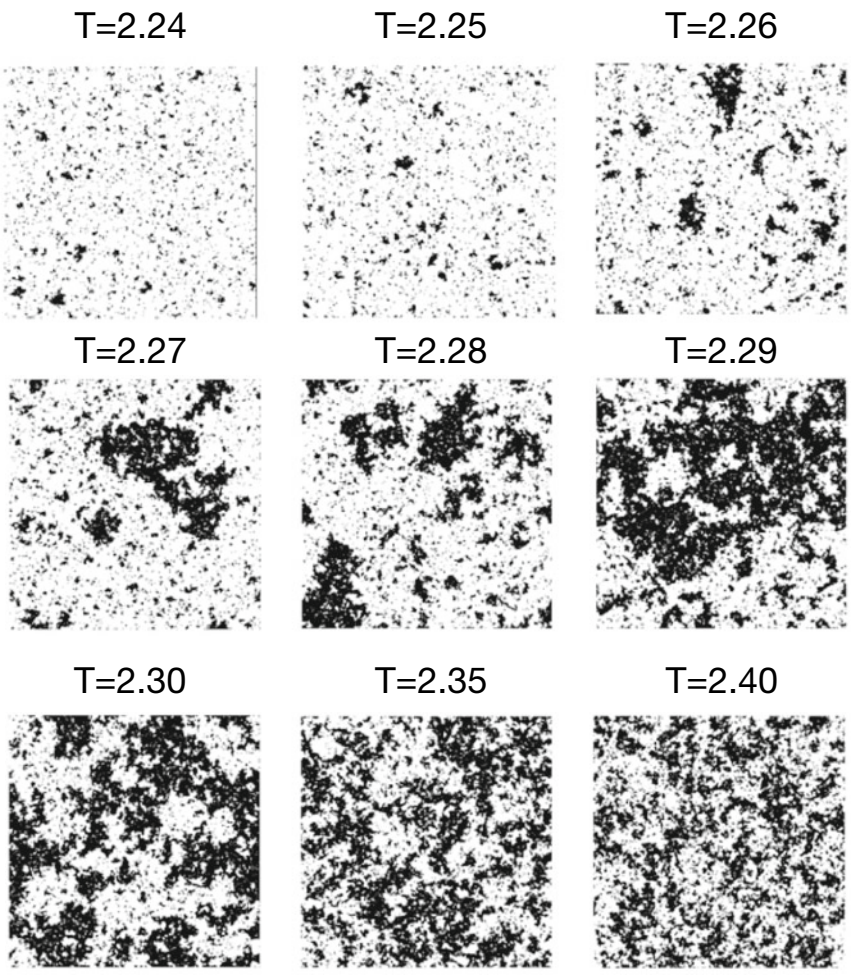

Fig. 6.10 Typical configurations in two-dimensional Ising model at $J = 1, h = 0$ and various temperatures. The lattice size is 512×512, and the white and black regions are spin $+1$ and -1, respectively. The Wolff algorithm was used for the simulation. (Note that, because $h = 0$, the configurations with the opposite sign can also appear with the same probabilities.)

is evidence for the critical slowing down. (As we have stated before, the critical temperature in the large-volume limit is $T \simeq 2.269\,\mathrm{J}$.)

The critical slowing down can be avoided by using the cluster algorithm, in which a big cluster of spins is chosen and flipped simultaneously. The bottom row of Fig. 6.11 is obtained by using the Wolff algorithm, which is a particular version of the cluster algorithm. We can see the power of the cluster algorithm very clearly: we cannot see the autocorrelation at all, even near the phase transition temperature.

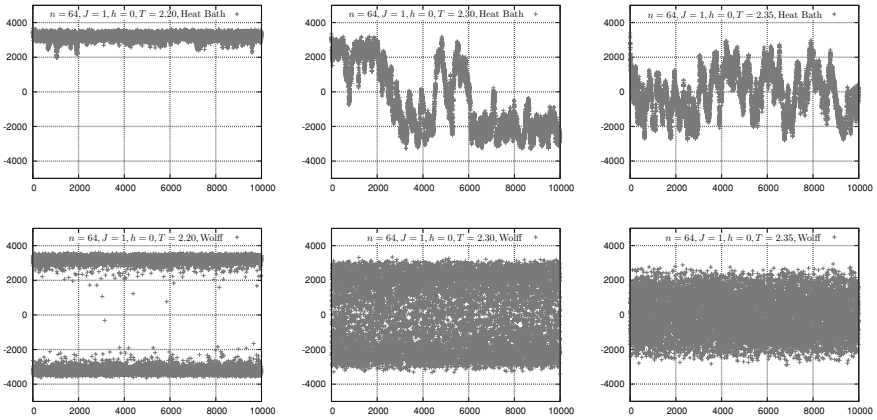

Fig. 6.11 Two-dimensional Ising model, lattice size $n^2 = 64^2 = 4096$, coupling $J = 1$, and external magnetic field $h = 0$. [Top] The Gibbs sampling algorithm (equivalently, the heat bath algorithm), sampled every $10n^2 = 40960$ steps. [Bottom] The Wolff algorithm, sampled every 10 steps. The vertical axis is the total spin, $\sum_{i=1}^{n^2} s_i$. From left to right, $T = 2.20$, 2.30, and 2.35

6.2.5 *Wolff Algorithm*

When the external magnetic field h is zero, the Wolff algorithm [9] works as follows:

Ising model via the Wolff algorithm (when $h = 0$)

1. Choose a lattice point i randomly and add it to the "cluster". We use red to denote the points belonging to the cluster.
2. If a spin sitting next to the cluster is the same as the spin of the cluster, it is connected to the cluster via a blue link or green link with the probability K or $1 - K$, respectively, where $K = 1 - e^{-2J/T}$.
3. The points connected to the cluster via a blue link are added to the cluster.
4. Repeat the above as long as the cluster can grow.
5. Flip all the spins in the cluster simultaneously.

An example of the construction of the cluster is shown in Fig. 6.12. All the spins in the cluster constructed in this manner are flipped as shown in Fig. 6.13.

The Wolff algorithm can be regarded as a special kind of the Metropolis-Hastings algorithm (see Sect. 5.3). New configurations are proposed with a very well-crafted probability $f(\{s\} \to \{s'\})$ such that the acceptance rate (5.43) becomes 1:

$$\frac{e^{-S(\{s'\})} f(\{s'\} \to \{s\})}{e^{-S(\{s\})} f(\{s\} \to \{s'\})} = 1. \tag{6.52}$$

Let us confirm the relation (6.52). As an example, we take the left and right of Fig. 6.13 to be the configurations $\{s\}$ and $\{s'\}$. The initial point in the construction of

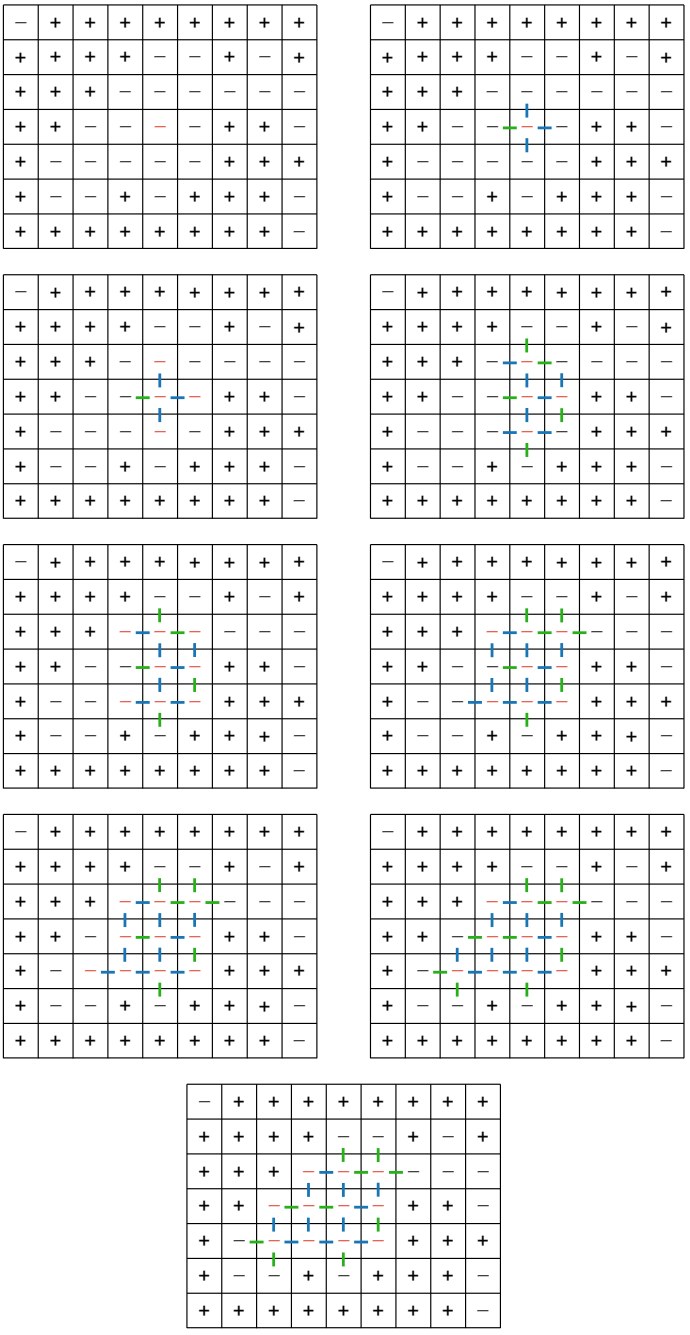

Fig. 6.12 An example of the construction of the cluster in the two-dimensional Ising model. The cluster is formed starting from a randomly chosen spin. We used red to denote the spins added to the cluster

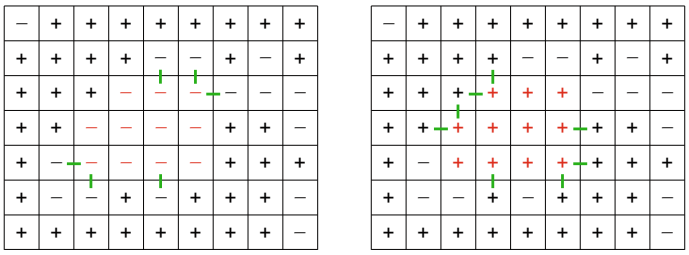

Fig. 6.13 Two configurations related to each other via the flip of the spins in the clusters

the cluster can be anything, and the links which are not shown explicitly in Fig. 6.13 can be anything, as long as the cluster is formed. Then, modulo the common overall factor, the probabilities $f(\{s\} \to \{s'\})$ and $f(\{s'\} \to \{s\})$ are determined by the probabilities that the green links explicitly shown in Fig. 6.13 are chosen. Let the numbers of green links in the left and right panels be m and n ($m = 6$ and $n = 8$ in this specific example), then

$$f(\{s\} \to \{s'\}) \propto (1-K)^m, \qquad f(\{s'\} \to \{s\}) \propto (1-K)^n. \tag{6.53}$$

Therefore, (6.52) can be rewritten as

$$e^{-S(\{s'\})+S(\{s\})} = (1-K)^{m-n}. \tag{6.54}$$

Because we are assuming $h = 0$, the left-hand side can be calculated by looking at the interactions at the boundary of the cluster. In the left panel of Fig. 6.13, the green links and the rest contribute to the energy by $-mJ$ and $+nJ$, respectively. In the right panel, the green links and the rest contribute by $-nJ$ and $+mJ$, respectively. Therefore, the change of the energy associated with the flip of the spins $\{s\} \to \{s'\}$ is $2(m-n)J$, and the relation (6.54) can be rewritten further as

$$e^{-2(m-n)J/T} = (1-K)^{m-n}. \tag{6.55}$$

Because we chose $K = 1 - e^{-2J/T}$, this relation is satisfied.

The cluster algorithms are designed by using the specific features of the systems under consideration. There is no known way to resolve the critical slowing down in arbitrary systems.

Nonzero External Magnetic Field

Suppose that the external magnetic field h is not zero. Then, if we construct the cluster as before, we obtain

$$\frac{e^{-S(\{s'\})} f(\{s'\} \to \{s\})}{e^{-S(\{s\})} f(\{s\} \to \{s'\})} = e^{h \times (\text{change of total spin})/T}. \tag{6.56}$$

Here, (change of total spin) $= \pm 2 \times$ (cluster size). We choose the positive sign when $\downarrow$ is flipped to $\uparrow$, and we choose the negative sign otherwise. Therefore, we have to perform the Metropolis test when the cluster is flipped. Following the general strategy of the Metropolis-Hastings algorithm, the proposal $\{s\} \to \{s'\}$ is accepted with the probability $\min\left(1, e^{h \times (\text{change of total spin})/T}\right)$.

Sample Program

Concerning the coding, the only nontrivial issue is the construction of the cluster. To construct the cluster, we prepare an integer-valued variable n_{cluster} that counts the number of lattice points belonging to the cluster and integer-valued variables $i_k (k = 0, 1, 2, \ldots)$ that are used for recording the lattice points in the cluster.

Construction of the cluster in the Ising model

1. Choose a lattice point i_0 randomly and add it to the cluster. Set $n_{\text{cluster}} = 1$, $k = 0$.
2. Among the lattice points next to i_k, study the ones not yet added to the cluster. If the spin is the same as s_{i_0}, add it to the cluster with probability $1 - e^{-2J/T}$, store the number labeling this lattice point in $i_{n_{\text{cluster}}}$, and increase the value of n_{cluster} by 1.
3. Increase k by 1.
4. If $k < n_{\text{cluster}}$, go back to step 2.

Let us see some sample code written in C++. We focus on the two-dimensional model, and instead of i_k we use $i_{\text{cluster}}(k, 0)$ and $i_{\text{cluster}}(k, 1)$ to store the x- and y-coordinates, respectively. Because the algorithm is not so complicated, let us just write everything manifestly:

```
int make_cluster(const int spin[nx][ny],const double coupling_J,
  const double temperature,int& n_cluster,int (&i_cluster)[nx*ny][2]){

  int in_or_out[nx][ny];
  for(int ix=0; ix!=nx; ix++){
    for(int iy=0; iy!=ny; iy++){
      in_or_out[ix][iy]=1;
    }
  }
  //in_or_out[ix][iy] = 1 -> not in the cluster; 0 -> in the cluster.
  //choose a point randomly.
  double rand_site = (double)rand()/RAND_MAX;
  rand_site=rand_site*nx*ny;
  int ix=(int)rand_site/ny;
  int iy=(int)rand_site
  in_or_out[ix][iy]=0;
```

```
i_cluster[0][0]=ix;
i_cluster[0][1]=iy;
int spin_cluster=spin[ix][iy];
n_cluster=1;
double probability=1e0-exp(-2e0*coupling_J/temperature);
int k=0;
while(k < n_cluster){
  ix=i_cluster[k][0];
  iy=i_cluster[k][1];
  //be careful about the boundary condition.
  int ixp1=(ix+1)
  int iyp1=(iy+1)
  int ixm1=(ix-1+nx)
  int iym1=(iy-1+ny)

  if(spin[ixp1][iy]==spin_cluster){
    if(in_or_out[ixp1][iy]==1){
      if((double)rand()/RAND_MAX < probability){
        i_cluster[n_cluster][0]=ixp1;
        i_cluster[n_cluster][1]=iy;
        n_cluster=n_cluster+1;
        in_or_out[ixp1][iy]=0;
      }
    }
  }
  if(spin[ix][iyp1]==spin_cluster){
    if(in_or_out[ix][iyp1]==1){
      if((double)rand()/RAND_MAX < probability){
        i_cluster[n_cluster][0]=ix;
        i_cluster[n_cluster][1]=iyp1;
        n_cluster=n_cluster+1;
        in_or_out[ix][iyp1]=0;
      }
    }
  }
  if(spin[ixm1][iy]==spin_cluster){
    if(in_or_out[ixm1][iy]==1){
      if((double)rand()/RAND_MAX < probability){
        i_cluster[n_cluster][0]=ixm1;
        i_cluster[n_cluster][1]=iy;
        n_cluster=n_cluster+1;
        in_or_out[ixm1][iy]=0;
      }
    }
  }
  if(spin[ix][iym1]==spin_cluster){
    if(in_or_out[ix][iym1]==1){
      if((double)rand()/RAND_MAX < probability){
        i_cluster[n_cluster][0]=ix;
        i_cluster[n_cluster][1]=iym1;
        n_cluster=n_cluster+1;
        in_or_out[ix][iym1]=0;
      }
    }
  }
```

```
    k=k+1;
  }
  return spin_cluster;
}
```

6.3 Combinatorial Optimization and Traveling Salesman Problem

The traveling salesman problem is a typical combinatorial optimization problem. The goal of this section is to solve this problem via Markov Chain Monte Carlo.

Suppose a salesman starts his trip from the city where the headquarter of his company is located, visits $N-1$ cities, and comes back to the headquarter. We use the numbers $1, 2, \ldots, N$ to label the cities. The headquarter is in the city 1. We assume that the distances between the cities are known; r_{ij} is the distance between the city i and the city j. The salesman has to visit each city once and only once in such a way that the total distance is minimum. Our task is to design such a trip for him. This is the traveling salesman problem.

By choosing an ordering of numbers from 2 to N as $i_2 \to i_3 \to \cdots \to i_N$, the route of the trip and the total distance can be specified. To simplify the notation, we set $i_1 = i_{N+1} = 1$. Then the total distance is expressed as

$$\begin{aligned} r_{\text{total}}(i_1 \to i_2 \to \cdots \to i_N \to i_{N+1}) &= r_{i_1 i_2} + r_{i_2 i_3} + \cdots + r_{i_{N-1} i_N} + r_{i_N i_{N+1}} \\ &= \sum_{k=1}^{N} r_{i_k i_{k+1}}. \end{aligned} \tag{6.57}$$

We want to find the ordering $i_1 \to i_2 \to \cdots \to i_N \to i_{N+1}$ which minimizes r_{total}. When N is small, we can check all possible orderings. However, the number of orderings increases as[10] $(N-1)!$ and hence this naive approach does not work for large values of N.

Of course, we can reduce the computational cost by removing such routes that are obviously non-optimal. For example, if there is an intersection, then by resolving it the distance can be made shorter, so any route with at least one intersection is not optimal. Also, if one cuts out a part of the route and finds a detour in that piece, such a route cannot be optimal. By utilizing such properties systematically, it is possible to find the optimal solution even when N is a few thousand.

In this section, we introduce methods based on MCMC that are applicable to various other optimization problems as well.

[10] The opposite ordering gives the same distance, but still, we have to check $\frac{(N-1)!}{2}$ combinations.

6.3.1 Minimization and Local Optimum

To develop the right intuition, we start with the minimization of a function $f(x)$. The simplest algorithm would be something like this:

A naive algorithm for minimizing a function

1. Choose Δx randomly and vary x a little bit as $x \to x' = x + \Delta x$.
2. Accept this change and update x to x' if $f(x') < f(x)$.
3. Repeat it until the acceptance rate becomes very small.

This method is not efficient when there are multiple variables. The gradient descent method is more convenient:

Gradient descent method (univariate version)

1. Vary x a little bit as $x \to x' = x - f'(x) \times \epsilon$. Here, $f'(x)$ is the derivative of $f(x)$ and ϵ is a parameter controlling the amount of the variation.
2. Repeat it until $f'(x)$ becomes almost zero.

Note that the accept/reject test via the comparison of $f(x)$ and $f(x')$ is not there anymore. With this method, if ϵ is sufficiently small, almost certainly $f(x') < f(x)$. The multivariate version of the gradient descent method is as follows:

Gradient descent method (multivariate version)

1. Vary x_i a little bit as $x_i \to x_i' = x_i - \frac{\partial f}{\partial x_i} \times \epsilon$.
2. Repeat it until $\frac{\partial f}{\partial x_i}$ becomes almost zero for all i's.

Here, $\frac{\partial f}{\partial x_i}$ means the partial derivative of f with respect to x_i that is the slope along the direction of x_i. Intuitively, we regard $f(x)$ as a height and go down the slope as shown in Fig. 6.14.

A naive method applies also to the traveling salesman problem:

A naive algorithm for the traveling salesman problem

1. Propose a new route by changing the ordering $i_1 \to i_2 \to \cdots \to i_N$ a little bit. For example, choose two numbers $2 \leq k < l \leq N$ randomly and exchange the k-th and l-th cities.
2. Accept the proposal if r_{total} decreases. Otherwise reject the proposal.
3. Repeat it until the acceptance rate becomes zero.

If there is only one minimum like in Fig. 6.14, the minimum can be found via such a simple method. However, if there are multiple minima like in Fig. 6.15, we may get a wrong answer (local but not global minimum) depending on the initial condition. This is the problem of local optimum.

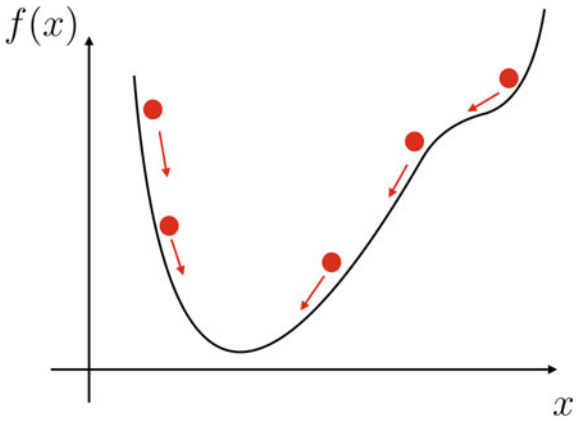

Fig. 6.14 A schematic picture of the gradient descent method. If we go down the slope, we should be able to reach the minimum

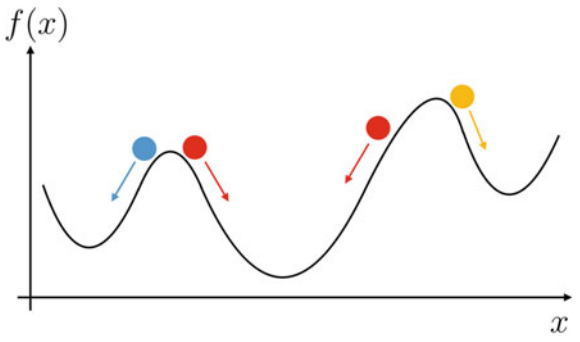

Fig. 6.15 The problem of local optimum in the gradient descent method. The true minimum (global minimum) is obtained if we start with a red point, but a wrong answer (local but not global minimum) is obtained if we start with a yellow or blue point

6.3.2 Simulated Annealing Algorithm

The simulated annealing algorithm [10] uses an idea from physics to avoid the problem of local optimum.

It is instructive to recall the Ising model. The weight was $e^{-\frac{E}{T}}$, where E and T are energy and temperature, respectively. When the Metropolis algorithm is used, at a very low temperature, even a tiny increase of E leads to a huge increase of $\frac{E}{T}$ and gets rejected via the Metropolis test. Therefore, the energy decreases monotonically toward the local minimum (equivalently, local optimum), which may or may not be the global minimum. Once reaching the minimum, there is no update anymore. On the other hand, at finite temperature, even if the simulation is captured at a local minimum, there is a chance to escape from there after a sufficiently long time (Fig. 6.16). By performing a long enough simulation decreasing temperature sufficiently slowly, we can find the true, global minimum at zero temperature. This is the simulated annealing method.

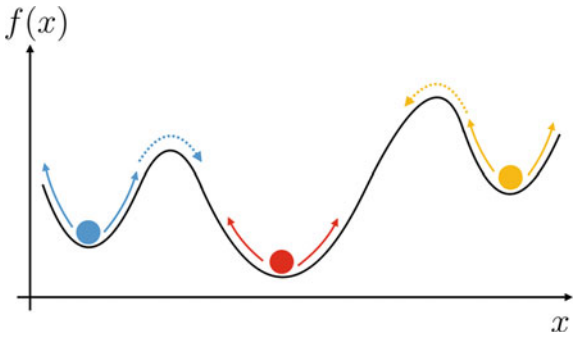

Fig. 6.16 Schematic picture of the simulated annealing algorithm. It is possible to escape from the local minima (equivalently, local optima) because of thermal fluctuation. The global minimum can be found by lowering temperature sufficiently slowly and continuing the simulation for a sufficiently long time. However, it is not easy to find the optimal rate of decreasing temperature. Furthermore, it is not clear whether the global minimum can be found in realistic simulation time

In order to find the minimum of a generic function $f(x)$, we regard $f(x)$ as the energy, introduce temperature T, and perform the simulated annealing with the weight $e^{-\frac{f(x)}{T}}$.

Simulated annealing algorithm

1. Introduce sufficiently high temperature T_0.
2. Perform MCMC with the weight $P_0(x) \propto e^{-S_0(x)}$, where $S_0(x) = f(x)/T_0$.
3. Lower temperature a little bit: $T_0 \to T_1$.
4. Perform MCMC the weight $P_1(x) \propto e^{-S_1(x)}$, where $S_1(x) = f(x)/T_1$.
5. Lower temperature a little bit: $T_1 \to T_2$.
6. Perform MCMC with the weight $P_2(x) \propto e^{-S_2(x)}$, where $S_2(x) = f(x)/T_2$.
7. Repeat it until the temperature becomes zero.

Via the simulated annealing algorithm, the global minimum can be found by lowering temperature sufficiently slowly and continuing the simulation for a sufficiently long time. However, it is difficult to know how long simulation time is needed; if the temperature becomes too low, it is impossible to escape from the wrong minima (local but not global minima) within a realistic simulation time. Another subtle issue is whether we can tell if the result of the simulation is the global minimum. We can get a certain level of confidence if we obtain the same answer from several different simulations with different initial configurations, but we cannot be 100% sure.

If a truly global minimum is not needed, the simulation can be terminated when a sufficiently good local minimum is obtained.

6.3.3 Replica Exchange Algorithm

The replica exchange algorithm [11] (also known as the parallel tempering algorithm) improves the simulated annealing algorithm. Suppose that two different probability distributions $P_1(\{x\})$ and $P_2(\{x\})$ are defined for the same set of variables $\{x\}$. We distinguish two copies (replicas) of the set as $\{x\}_1$ and $\{x\}_2$, respectively. We consider the product distribution defined by

$$P(\{x\}_1, \{x\}_2) = P_1(\{x\}_1) \times P_2(\{x\}_2). \tag{6.58}$$

By construction, if we do not care about $\{x\}_2$ and see only the distribution of $\{x\}_1$, we obtain $P_1(\{x\}_1)$. Also, if we do not care about $\{x\}_1$ and see only the distribution of $\{x\}_2$, we obtain $P_2(\{x\}_2)$. The replica exchange algorithm improves the simulated annealing algorithm by using this trivial property in a very interesting and nontrivial manner.

Let $f(X)$ be the function we want to minimize.[11] We introduce two different values of "temperature" T_1 and T_2 ($T_1 > T_2$), and define the probability distributions as $P_1(X) \propto e^{-f(X)/T_1}$ and $P_2(X) \propto e^{-f(X)/T_2}$. We construct $P(X_1, X_2) = P_1(X_1) \times P_2(X_2)$ in the following way:

Replica exchange algorithm

1. Update X_1 via a usual MCMC method, e.g., Metropolis or HMC. (Whether we use $P_1(X_1)$ or $P(X_1, X_2)$ does not matter.)
2. Update X_2 via a usual MCMC method, e.g., Metropolis or HMC. (Whether we use $P_2(X_2)$ or $P(X_1, X_2)$ does not matter.)
3. Exchange replicas as $X_1 \to X_1' = X_2$ and $X_2 \to X_2' = X_1$ with the probability $\min(1, e^{-\Delta S})$, where ΔS denotes the change of the action $S(X_1, X_2) = \frac{f(X_1)}{T_1} + \frac{f(X_2)}{T_2}$, i.e., $\Delta S \equiv S(X_1', X_2') - S(X_1, X_2) = \left(\frac{1}{T_2} - \frac{1}{T_1}\right)(f(X_1) - f(X_2))$.
4. Repeat step 1, step 2, and step 3.

In principle, we can get the target distribution just from step 1 and step 2. The third step just exchanges X_1 and X_2 via the Metropolis test; mathematically, it does not affect the resulting distribution at all. However, this step has a very important meaning which improves the efficiency of the simulation. The exchange of the replicas $X_1 \leftrightarrow X_2$ is equivalent to the exchange of the temperatures $T_1 \leftrightarrow T_2$. Even when X_2 is trapped in a local minimum, the temperature can go up from T_2 to T_1 so that there is a bigger chance of escaping from a local minimum.

The frequency of the third step (replica exchange) can be chosen arbitrarily. It does not have to be 1, 2, 3, 1, 2, 3, ... Other choices such as 1, 2, 1, 2, 3, 1, 2, 1, 2, 3,... are equally fine. The efficiency of the simulation may change depending on the frequency of the third step.

[11] To make the equations less messy, we introduced the notation $X = \{x\}$.

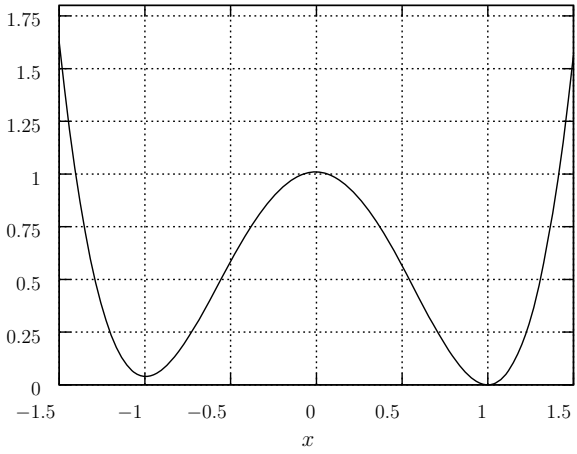

Fig. 6.17 $f(x) = (x-1)^2 \cdot \left((x+1)^2 + 0.01\right)$

We can repeat the same argument with more replicas. We prepare M temperatures $T_1 > T_2 > \cdots > T_M$ and replicas $X_1, X_2, \ldots, X_M$, and define the action as $S(X_1, X_2, \ldots, X_M) = \sum_{m=1}^{M} \frac{f(X_m)}{T_m}$.

Replica exchange algorithm with M replicas

1. Update $X_1, X_2, \ldots, X_M$ via a usual MCMC method, e.g., Metropolis or HMC.
2. For $m = 1, 2, \ldots, M-1$, exchange the replicas as $X_m \to X'_m = X_{m+1}$, $X_{m+1} \to X'_{m+1} = X_m$ with the probability $\min(1, e^{-\Delta S})$. Here, $\Delta S = \left(\frac{1}{T_{m+1}} - \frac{1}{T_m}\right)(f(X_m) - f(X_{m+1}))$.
3. Repeat step 1 and step 2.

Again, the exchange of the replicas $X_m \leftrightarrow X_{m+1}$ is equivalent to the exchange of the temperatures $T_m \leftrightarrow T_{m+1}$. If the difference between neighboring temperatures is sufficiently small, the temperature can go up or go down frequently, and then it is easier to escape from local minima. Therefore, we have a bigger chance of finding the global minimum within a realistic simulation time.

In the replica exchange algorithm, the simulation for each replica (step 1) can be done independently for each replica, and hence the parallelization is straightforward. This is a nice feature when we want to study a big system.

Replica Exchange Algorithm Applied to a Simple Function

Let us consider a function $f(x) = (x-1)^2 \cdot \left((x+1)^2 + 0.01\right)$. This function has two minima as we can see from Fig. 6.17: the global minimum at $x = +1$ and the local but not global minimum near $x = -1$. This function resembles the situation in the Ising model with a very weak external magnetic field h: the distribution proportional

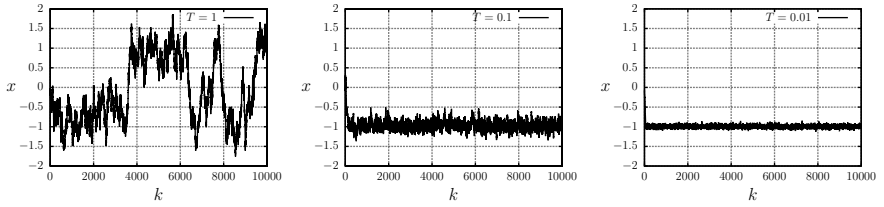

Fig. 6.18 Metropolis simulation with step size 0.1 and the weight proportional to $e^{-f(x)/T}$. From left to right, $T = 1.0$, 0.1, and 0.01. The horizontal axis is the number of steps, k. The initial value is chosen to be $x = 0$, and the simulation results which were trapped in a wrong minimum $x \simeq -1$ are shown. If the temperature is sufficiently high, it is possible to escape from the wrong minimum. However, if the temperature is low, it is impossible to escape from the wrong minimum within a reasonable simulation time

to $e^{-f(x)/T}$ has two peaks around $x = \pm 1$, and as temperature T approaches zero the true minimum $x = 1$ becomes more and more dominant.

Let us first try a naive Metropolis simulation with the weight proportional to $e^{-f(x)/T}$. We take the initial value to be $x = 0$. Then, depending on the random numbers, the simulation may be trapped in a wrong minimum at $x \simeq -1$ as shown in Fig. 6.18. When the temperature is not too low ($T = 1.0$ in Fig. 6.18), the fluctuation is sufficiently large and the value of x goes back and force between two minima, such that the correct distribution with two peaks around $x = \pm 1$ is obtained. However, at low temperature ($T = 0.1$ and 0.01 in Fig. 6.18), it is difficult to escape from the local minimum.

Next, we use the replica exchange algorithm. (As for sample code, see Appendix A.7.1.) We introduce 2000 replicas by taking the inverse temperature $\beta = \frac{1}{T}$ from $\beta = 0.5$ ($T = 2.0$) to $\beta = 1000$ ($T = 0.001$) with the spacing $\Delta\beta = 0.5$. The result of the simulation is shown in Fig. 6.19. As we can see from the Monte Carlo history (top row of Fig. 6.19), it takes some time for thermalization, but eventually the correct distribution is obtained.[12] The distribution of x at each temperature after the thermalization (from the 500,000-th step to 1,000,000-th step) is shown in the bottom row. Very good agreement with the target distribution can be seen. The replica exchange algorithm is working perfectly here!

We chose the parameters used here without any logical reason. Apparently, the fluctuation is suppressed at low temperature, so the simulation can be made more efficient by adjusting the step size for the Metropolis part depending on temperature.

[12] We can see that the right distribution is obtained immediately at $T = 0.001$, while at $T = 0.01$ and $T = 0.1$ it takes some time for the convergence. The reason is as follows. Because we chose $x = 0$ as the initial value, at the beginning of the simulation half of the replicas are at $x > 0$ and the other half are at $x < 0$. Then the replicas are exchanged, and roughly speaking, the low-temperature half of the replicas go around the true minimum $x = 1$, while the high-temperature half go around $x = -1$. This is more or less the correct answer at very low temperatures, but not at high temperatures. As we continue the simulation, the fluctuations generated in the high-temperature region propagate to the entire system, and the correct distribution is achieved at all temperatures.

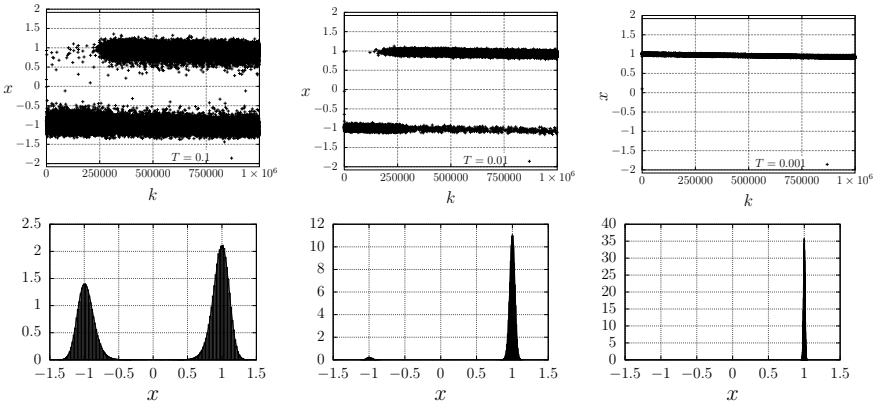

Fig. 6.19 The result of the replica exchange simulation at $T = 0.1, 0.01$, and 0.001. [Top] The Monte Carlo history. The horizontal axis is the number of steps, k. [Bottom] The distribution of x in the second half of the simulation. The target distributions are drawn as well, but it is hard to distinguish them from the histograms because the result of the replica exchange simulation is very precise

Furthermore, we do not have to take the spacing between $\beta = 1/T$ to be uniform. The frequency of the exchange of the replicas can also be tuned.

Let us use the replica exchange algorithm for the traveling salesman problem. We prepare M replicas labeled by $m = 1, 2, \ldots, M$. Each replica specifies a route $i_1^{(m)} = 1 \to i_2^{(m)} \to \cdots \to i_N^{(m)} \to i_{N+1}^{(m)} = 1$. From the ordering $I^{(m)} = \left\{ i_1^{(m)}, i_2^{(m)}, \ldots, i_N^{(m)} \right\}$, the total distance $r_{\text{total}}^{(m)}$ can be calculated.

Traveling salesman problem via Replica exchange

1. For each replica, choose $2 \le k < l \le N$ randomly and exchange k-th and l-th cities as a candidate for a new route: $i_k^{(m)} \to i_k'^{(m)} = i_l^{(m)}$, $i_l^{(m)} \to i_l'^{(m)} = i_k^{(m)}$. This candidate is accepted with a probability $\min(1, e^{-\Delta r_{\text{total}}^{(m)} / T_m})$, where $\Delta r_{\text{total}}^{(m)}$ is the change of the total distance $r_{\text{total}}^{(m)}$.
2. For each m, the replicas are exchanged $(I^{(m)} \to I'^{(m)} = I^{(m+1)}$, $I^{(m+1)} \to I'^{(m+1)} = I^{(m)})$ with a probability $\min(1, e^{-\Delta S})$, where $\Delta S = \left(\frac{1}{T_{m+1}} - \frac{1}{T_m} \right) \left(r_{\text{total}}^{(m)} - r_{\text{total}}^{(m+1)} \right)$.
3. Repeat.

We study the case of $N = 100$. We regard 100 pairs of random numbers (x_i, y_i) $(i = 1, 2, \ldots, 100)$ as the locations of 100 cities; see Fig. 6.20. The symbol + is (x_1, y_1), which is the city where the headquarter is located. The distance between two cities i and j is given by[13]

[13] To simplify the setup, we ignore the fact that the earth is not flat.

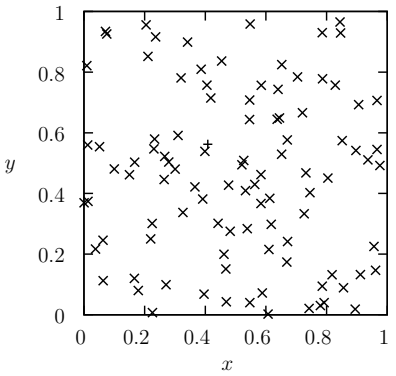

Fig. 6.20 100 pairs of random numbers (x_i, y_i) $(i = 1, 2, \ldots, 100)$ are regarded as the locations of cities. The symbol + is (x_1, y_1)

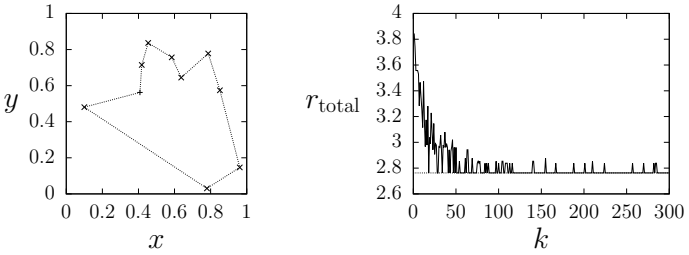

Fig. 6.21 10 pairs of random numbers (x_i, y_i) $(i = 1, 2, \ldots, 10)$ are regarded as the locations of cities. The symbol + is (x_1, y_1). [Left] The shortest route. [Right] The total distance r_{total} calculated via the replica exchange algorithm. The value at $\beta = 100$ is shown. The horizontal axis is the number of steps, k. The dotted line is the optimal, shortest distance. We can see that the optimal path is found quickly

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{6.59}$$

Because the initial condition can be arbitrary, we simply use $i_2^{(m)} = 2, i_3^{(m)} = 3, \ldots, i_{100}^{(m)} = 100$ for all m's. As for the inverse temperature $\beta = \frac{1}{T}$, we choose $\beta_m = \frac{1}{T_m} = m \cdot \Delta\beta$. We use $M = 200$ and $\Delta\beta = 0.5$ (again, without deep reasons).

Before tackling $N = 100$, let us study an easier case, $N = 10$. There are $(N - 1)! = 9! = 362,880$ routes in total; this is not a very large number, we can find the answer by brute force. Hence, we can check if the correct answer is obtained via the replica exchange algorithm. We use (x_i, y_i) $(i = 1, 2, \ldots, 10)$ as the locations of ten cities. The outcome of the replica exchange simulation is shown in Fig. 6.21. We can see that the optimal route is found rather quickly.

Next let us study the case of $N = 100$ (Fig. 6.22). We calculate r_{total} at $\beta = 100$ every 500 steps. The shortest route found by that step is shown by a solid line. The

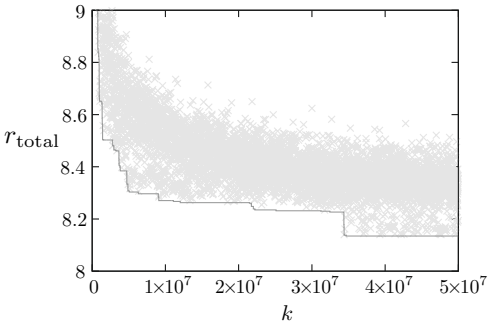

Fig. 6.22 The outcome of the replica exchange simulation for the traveling salesman problem, $N = 100$. The total distance r_{total} is calculated at $\beta = 100$ every 500 steps. The horizontal axis is the number of Monte Carlo steps, k. The solid line is the minimum distance found by then. The cross marks are r_{total} shown every 10,000 steps

values of r_{total} are shown at every 10,000 steps by gray cross marks. It looks that the simulation is thermalized after 35,000,000 steps or so. In Fig. 6.23, the optimal routes found by the 5,000-th, 50,000-th, 50,0000-th, and 50,000,000-th step are depicted. The last one is a nice, single-stroke shape without a crossing.

First-Order Phase Transition

We studied a simple example with two minima, $f(x) = (x-1)^2 \cdot \left((x+1)^2 + 0.01\right)$. Similar examples appear in physics, associated with first-order phase transitions. A first-order phase transition is a sudden, discontinuous change of the property of a system when the parameters of the theory such as temperature are varied. It happens when the importance of the local optimal solutions changes with parameters and the global optimum changes at a certain point (Fig. 6.24). For example, in the Ising model, if we change the magnetic field h at a low-temperature region, a first-order phase transition takes place at $h = 0$.

An interesting example of the first-order transition is related to the thermodynamics of a black hole. You might have heard that a black hole swallows everything and even light cannot escape. However, Hawking considered the quantum mechanical effects near a black hole and predicted that black hole gradually emits particles and eventually evaporates [12].[14] According to the holographic principle (see Sect. 6.4), the properties of a black hole can be studied by using certain gauge theories. To see the creation and evaporation of a black hole, we should study the phase transition of gauge theory by changing temperature. In many cases, there is a first-order phase transition called the Hawking-Page transition [13, 14]. Associated with the

[14] With the current technology, it is impossible to detect the evaporation of large black holes observed in the universe because it is very slow.

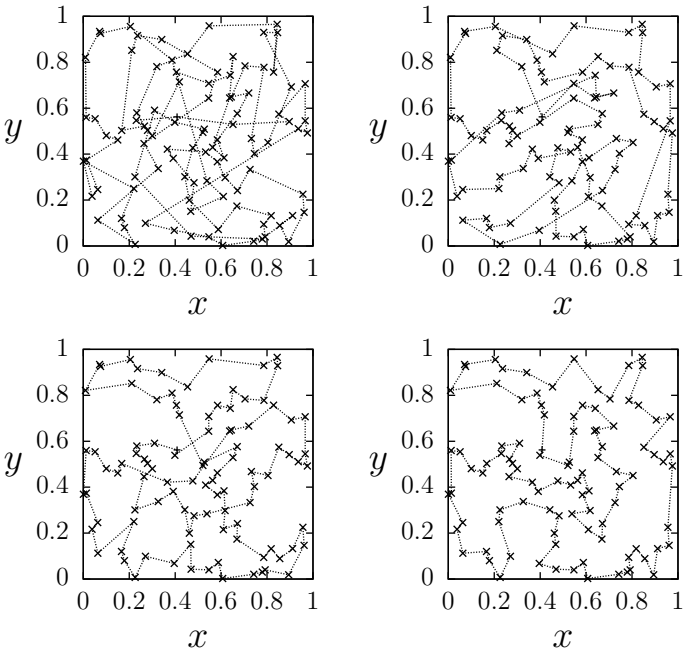

Fig. 6.23 The shortest route found with the parameter choice explained in the main text. [Top, left] by the 5,000-th step, [top, right] by the 50,000-th step, [bottom, left] by the 500,000-th step, [bottom, right] by the 50,000,000-th step

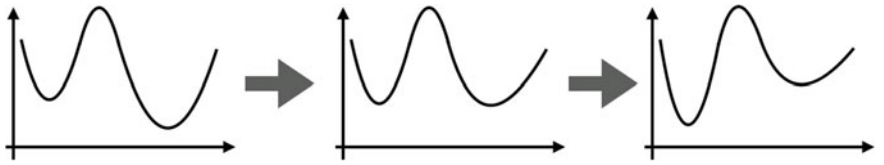

Fig. 6.24 Schematic picture of a first-order phase transition. As the parameter is varied (e.g., as the temperature is lowered), the global optimum changes at a certain point

Hawking-Page transition, not just the optimal solutions but also the dip between the peaks in the probability distribution should have interesting information about black hole [15, 16]. The Hawking-Page transition has not been studied intensively so far because of the large cost of the simulations, but the detailed study via the replica exchange algorithm would be performed in near future.

6.4 Applications to High-Energy Physics

The last example is high-energy physics. Once you understand how the elementary particles are described in high-energy physics, you can see that MCMC is a very natural approach. The keywords are quantum field theory and Feynman path integral.

It is important to note that particles are not literally "particles" in quantum mechanics, rather they are "quanta" which are, in certain sense, spread over space. Rather than sitting at specific points in space, they are probabilistically spread out. Perhaps it does not make sense for you, but there is no need to be worried. As Feynman mentioned at the beginning of his famous lecture on quantum mechanics [17], no human could ever understand quantum mechanics intuitively.[15] However, there is a well-established theoretical framework to deal with such counter-intuitive "quanta": there is an object called "field", which is something like a wave spread over space, and the field is described by a probability distribution specified by a certain action. If we want to understand the behavior of elementary particles, we should understand the probability distribution of the fields. Such formulation is called the Feynman path integral [18, 19]. Markov Chain Monte Carlo can be used to describe particles via the Feynman path integral.

6.4.1 Quantum Chromodynamics (QCD)

Let us study physics inside nuclei. Nuclei are made of a class of particles called hadrons such as protons, neutrons, and pions. Hadrons are made of more elementary particles called quarks and gluons. There are six different types of quarks. Among them, we consider the two lightest ones, up quark and down quark. Elementary particles—quarks and gluons, in this case—are described by the fields. We denote quark and the gluon fields as $q_\alpha^{f,i}(x)$ and $G_\mu^a(x)$, respectively. Here we used "x" to denote time t and spatial coordinate x, y, z together. A small script f attached to quark field labels up quark ($f = 1$) or down quark ($f = 2$). Other indices $i = 1, \ldots, 3$ and $\alpha = 1, \ldots, 4$ represent the degrees of freedom called color and spin, respectively. From now on, we use $I = (i, \alpha)$ to simplify the notation and write the quark field as $q_{f,I}(x)$. Gluons intermediate the force connecting quarks. The gluon field has indices $a = 1, \ldots, 8$ and $\mu = 0, \ldots, 3$. The former is related to the colors of quarks connected by gluon. The label μ is needed because gluon field is a vector that has the temporal ($\mu = 0$) and spatial ($\mu = 1, 2, 3$) components.

---

[15] Feynman said that "*Because atomic behavior is so unlike ordinary experience, it is very difficult to get used to, and it appears peculiar and mysterious to everyone—both to the novice and to the experienced physicist. Even the experts do not understand it the way they would like to, and it is perfectly reasonable that they should not, because all of direct, human experience and of human intuition applies to large objects*". Quantum field theory is an even more crazy beast.

QCD (Quantum Chromodynamics) is the quantum field theory that describes the interaction between these fields. Concerning the use of MCMC, the key point is that the action $S_{\mathrm{QCD}}[G, q]$ is explicitly given. Schematically, it takes the following form:

$$S_{\mathrm{QCD}}[G, q] = \int d^4x \Big(\frac{1}{4} \sum_{\mu,\nu,a} F^a_{\mu\nu}(x)^2 + \sum_f \sum_{I,J} q^*_{f,I}(x) D(G)_{IJ} q_{f,J}(x) \Big). \quad (6.60)$$

Here $F^a_{\mu\nu}(x)$ is a quantity called field strength which can be calculated from the gluon field $G^a_\mu(x)$. A matrix of differential operators $D(G)_{IJ}$, which is called Dirac operator, can also be determined from the gluon field. $\int d^4x \cdots$ is the four-dimensional integral over time t and space x, y, z. We ignored the difference between the mass of up quark and down quark, so that we can use the same Dirac operator for them. This is not a bad approximation, and the simulation becomes much simpler with this approximation.

In QCD,[16] the probability distribution of the fields is proportional to $e^{-S_{\mathrm{QCD}}[G,q]}$. We can understand the interaction between elementary particles if we can calculate this probability distribution. Markov chain Monte Carlo is a natural option for this purpose.

However, there are two problems. Firstly, the fields are functions defined in spacetime. Because there are infinitely many points in spacetime, we have to consider the probability distribution of infinitely many degrees of freedom. Secondly, quark fields are represented by Grassmann numbers, i.e., when the order of the product is flipped, a negative sign appears as $q_{f,I}(x) q_{f',I'}(x') = -q_{f',I'}(x') q_{f,I}(x)$.

One of the resolutions of the first problem, which is particularly suitable for MCMC simulations, is to approximate the continuum spacetime by a discrete lattice at large but finite volume [20]. There is only a finite number of points on such a lattice, so the fields are expressed by a finite number of variables. Such discretized theory is called lattice field theory. The lattice version of QCD is called lattice QCD. With more variables the calculation is more costly, however as long as it is finite the MCMC algorithms can be applied. By making the lattice finer and finer, we can learn what happens in the continuum limit. For a technical reason, it is convenient to put quarks on lattice sites and gluons on the links connecting the sites. The action of lattice QCD is schematically expressed as

$$S^{\mathrm{lat}}_{\mathrm{QCD}}(G, q) = S(G) + \sum_f \sum_{I,J,x,y} q^*_{f,I,x} D_{Ix;Jy}(G) q_{f,J,y}, \quad (6.61)$$

where x and y stand for lattice points. The first term $S(G)$ is the lattice counterpart of the gluon action. In the second term, the Dirac operator has indices x and y because the continuum version of the Dirac operator contains a derivative. Since the derivative describes a change associated with an infinitesimal translation, its counterpart on a lattice is expressed by a difference between the values at two neighboring lattice

[16] Strictly speaking, in QCD on Euclidean spacetime.

sites. As a consequence, the Dirac operator becomes a huge matrix whose indices express color, spin, and spacetime coordinate.

Still, we have to resolve the second problem: the second term contains Grassmann numbers $q_{I,x}$, which cannot be handled efficiently on a computer. Luckily enough, regarding the quark fields, the action takes the Gaussian form. Therefore, we can integrate out the quark fields for fixed values of gluon field G_μ^a. From the usual Gaussian integral, the inverse of the determinant is obtained (see Appendix B.2.2), but from the Grassmannian version, the determinant itself is obtained. In the end, we obtain the probability distribution in terms of the gluon field,

$$P(G) \propto \det(D(G) \cdot D^\dagger(G)) \cdot e^{-S(G)}. \tag{6.62}$$

One has to generate this distribution via MCMC.[17]

HMC with Pseudofermion

In principle, the MCMC methods we learned in this book can be applied to the target probability distribution (6.62). However, the calculation of the determinant $\det(DD^\dagger)$ is so costly that the simulation is impossible with realistic computational resources. Therefore, we introduce a complex vector F as an auxiliary field and rewrite the determinant $\det(DD^\dagger)$ as

$$\det(DD^\dagger) \propto \int dF \exp\left(-F^\dagger (DD^\dagger)^{-1} F\right). \tag{6.63}$$

Then, to get (6.62), the action should be modified to

$$\tilde{S}(G, F) \equiv S(G) + F^\dagger (DD^\dagger)^{-1} F, \tag{6.64}$$

and the simulation should be done by using $e^{-\tilde{S}}$ as the weight.[18] This weight can be simulated by combining the HMC algorithm and the Gibbs sampling algorithm.

The concrete procedures are as follows [21]. Firstly, F is updated via the Gibbs sampling algorithm for fixed G. Because of $F^\dagger (DD^\dagger)^{-1} F = F^\dagger (D^\dagger)^{-1} D^{-1} F = (D^{-1}F)^\dagger (D^{-1}F)$, in terms of Φ defined by

[17] We used one approximation at this stage. If we integrate out the quark fields in (6.61), we obtain $\det(DD)$ rather than $\det(DD^\dagger)$. The determinant of the Dirac operator is not necessarily non-negative, often it becomes complex. But unless $\det(DD) = (\det(D))^2$ is non-negative, $\det(D(G) \cdot D(G)) \cdot e^{-S(G)}$ cannot be regarded as a probability and the MCMC techniques cannot be applied. Therefore, we neglected the phase factor and replaced $\det(DD)$ with $\det(DD^\dagger)$. If the physical phenomena under consideration are sensitive to this phase, we have to take the phase into account. Fortunately, in lattice QCD, many important phenomena can be studied in a setup without a phase.

[18] In physics, this F is called pseudofermion. This is because F is a boson that leads to the fermion determinant.

$$\Phi \equiv D^{-1}F \tag{6.65}$$

the distribution is Gaussian, $e^{-\Phi^\dagger \Phi}$. This Φ can be generated by using the Box-Muller method. Once Φ is given, F is obtained by solving $D\Phi = F$.

Next, G is updated via the HMC algorithm for fixed F. The Hamiltonian is defined by

$$H = \frac{1}{2}\mathrm{Tr}PP^\dagger + \tilde{S}[G, F]. \tag{6.66}$$

The conjugate momentum of G is taken to be $P^\dagger$. The Hamilton equation is

$$\frac{dG_{ij}}{d\tau} = P_{ij}, \tag{6.67}$$

$$\frac{dP_{ij}}{d\tau} = -\frac{\partial \tilde{S}}{\partial G^\dagger_{ij}} = -\frac{\partial S}{\partial G^\dagger_{ij}} + \chi^\dagger \frac{\partial (DD^\dagger)}{\partial G^\dagger_{ij}}\chi, \tag{6.68}$$

where $\chi = (DD^\dagger)^{-1}F$. This χ is obtained by solving $(DD^\dagger)\chi = F$.

Here is a summary of the procedures:

Lattice QCD simulation via HMC with pseudofermion

1. Generate Φ with the Gaussian weight.
2. Derive F by using $D\Phi = F$.
3. Fix F and update G via HMC.
4. Repeat.

Most of the computational time is spent obtaining χ by solving $(DD^\dagger)\chi = F$. Although such a computation is usually very hard, in lattice QCD the solutions can be obtained efficiently by using the conjugate gradient method (CG method) because the Dirac operator D is a sparse matrix. In this way, the simulation is much more efficient than a naive method which requires the calculation of the determinant. As for the CG method, see Appendix E.

As an example of the lattice QCD simulation via the HMC algorithm, the mass spectrum of the hadrons is shown in Fig. 6.25.[19] N stands for nuclei (proton and neutron). By tuning the parameters of the theory such that the mass of π, K, and Ω agree with the experimental values, the mass spectrum of all other particles can be calculated numerically. The values obtained via Monte Carlo simulation agree excellently with the results of the accelerator experiments.

[19] Strictly speaking, up and down quarks are treated via the HMC algorithm, and the strange quark is incorporated via another method. Note also that some features of the real world are neglected, such as the difference of the mass of up and down quarks and the electromagnetic interaction.

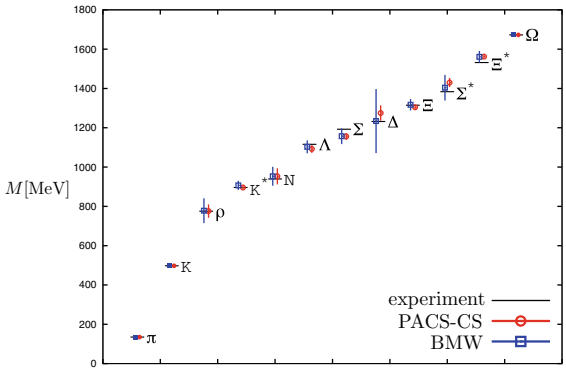

Fig. 6.25 Hadron mass spectrum from lattice QCD obtained by using the HMC algorithm. This plot is based on the data in Ref. [22] (PACS-CS collaboration) and Ref. [23] (BMW collaboration)

6.4.2 Superstring Theory and Holographic Principle

Superstring theory is a candidate for the unified theory of gravity (general relativity) and quantum mechanics. (The standard textbooks on superstring theory include Refs. [24–27].) The history of superstring theory is very complicated. When it was first proposed in the 1960s, the motivation was to explain the strong interaction: it was hoped that various hadrons might be understood as different vibration modes of a string. This idea was so attractive that many researchers studied it until the early 1970s. However, superstring theory turned out to have a few fatal problems as a theory of the strong interaction. In the meantime, QCD turned out to be a better candidate, and by the mid-1970s most researchers left superstring theory. But then it was revived as a theory of quantum gravity: some properties that were serious flaws as a theory of the strong interaction turned out to be advantageous as a theory of quantum gravity. Since then, superstring theory has been actively studied to date.

Nobody knows what the ultimate theory of quantum gravity would look like, but there is a hint from a black hole. A black hole is a solution of the Einstein equation equipped with an extreme spacetime structure. Hawking studied how quantum mechanics affects the properties of a black hole, and discovered that a black hole has temperature and entropy. Roughly speaking, entropy is the amount of hidden information. To specify the macroscopic state of the air filling our room, we need only a few parameters such as temperature and pressure. However, at microscopic level, a lot of molecules are moving around and colliding with each other. There are many different microscopic states which look the same macroscopically. Entropy characterizes the number of microscopic states that cannot be distinguished macroscopically. A black hole can have various microscopic states, depending on the detail of its history. The entropy of a black hole can be interpreted as the amount of such hidden information.

Peculiarly, the entropy of a black hole is proportional to its surface area [12, 28]. Usually, entropy is proportional to volume; in the case of the air, if the volume is doubled then the number of molecules is also doubled, which leads to twice as much information.[20] A black hole becomes bigger when it swallows something, then one would naively think that the increment of the entropy is proportional to that of the volume, but, in fact, it is not the case.

Because general relativity is a theory of spacetime itself, we expect that the complete theory of quantum gravity has some fundamental degrees of freedom which are the building blocks of spacetime. That the entropy of a black hole is proportional to the surface area suggests that those fundamental degrees of freedom are distributed on a surface surrounding spacetime. Based on such reasoning, 't Hooft and Susskind conjectured that the building blocks of spacetime are living on the "surface" rather than filling the spacetime volume, and that gravity and spacetime are a sort of hologram. This conjecture is called holographic principle [29, 30].

At first, many people thought that holographic principle was just an abstract slogan. The situation changed in 1997, due to the discovery of gauge/gravity duality by Maldacena [31]. Maldacena studied an object called D-brane in superstring theory. He argued that a system of D-branes admits two descriptions—superstring theory and a non-gravitational theory (supersymmetric gauge theory) defined on a "surface"—in a certain parameter region, and hence these two theories should be equivalent. Supersymmetric gauge theory is similar to QCD and both of them belong to a class of quantum field theories called gauge theory. At first, many people thought that the gauge/gravity duality is not true, because it was a very surprising proposal that claims superstring theory, which contains gravity, and supersymmetric gauge theory, which does not contain gravity, should be equivalent. However, although many nontrivial tests have been performed for more than 20 years, no contradiction was found. Rather, there is overwhelming evidence supporting the correctness of the duality. Today, almost nobody doubts its validity. Of course, it does not immediately mean that the universe we live in is a hologram, but it is certainly a big step.

Gauge/gravity duality claims that superstring theory is described by supersymmetric gauge theory via holographic principle. In other words, we can learn about the properties of superstring theory by studying supersymmetric gauge theory. Because supersymmetric gauge theory is similar to QCD, similar simulation techniques can be used. In particular, we can use MCMC to study supersymmetric gauge theory and translate the results into the statements regarding quantum gravity.[21]

[20] To specify the locations and velocities of N molecules, $6N$ numbers are needed. If there are $2N$ molecules, $12N$ numbers are needed.

[21] This is the reason why the authors learned MCMC.

Many powerful algorithms are used for the Monte Carlo study of supersymmetric gauge theory. We do not have enough space to explain the details, but let us introduce a part of them.

RHMC Algorithm

In lattice QCD, there was a huge matrix called Dirac operator D and we had to tame $\det(DD^\dagger)$. The Dirac operator appears in supersymmetric gauge theory as well. Therefore, a fractional power of the determinant can appear in the weight, for example,

$$P(G) \propto \left(\det(DD^\dagger)\right)^{1/4} e^{-S(G)}. \tag{6.69}$$

In this case, the pseudofermion can be introduced as in QCD. The determinant part can be rewritten as $\left(\det(DD^\dagger)\right)^{1/4} \propto \int dF \exp\left(-F^\dagger (DD^\dagger)^{-1/4} F\right)$, hence the action can be taken as

$$\tilde{S}(G, F) \equiv S(G) + F^\dagger (DD^\dagger)^{-1/4} F, \tag{6.70}$$

and the simulation with the weight $e^{-\tilde{S}}$ should be performed.

If we simulate this weight naively using the fractional power of the matrix D, the simulation can be very complicated and costly. To avoid the use of fractional power, the following rational approximations can be used:

$$x^{1/8} \simeq a_0 + \sum_{i=1}^{Q} \frac{a_i}{x + b_i}, \quad x^{-1/4} \simeq a'_0 + \sum_{i=1}^{Q'} \frac{a'_i}{x + b'_i}. \tag{6.71}$$

For any given range in $x > 0$, by taking Q and Q' sufficiently large and by choosing the coefficients appropriately, a very good approximation can be achieved.[22]

Let us start with the update of F via the Gibbs sampling algorithm. As before, we generate $\Phi = (DD^\dagger)^{-1/8} F$ with the Gaussian weight. When we obtain F from Φ, we use (6.71) and evaluate

$$F = (DD^\dagger)^{1/8} \Phi \simeq a_0 \Phi + \sum_{i=1}^{Q} a_i (DD^\dagger + b_i)^{-1} \Phi. \tag{6.72}$$

[22] Kate Clark, who is one of the inventors of the RHMC algorithm, offers a free program to obtain these coefficients via the Remez algorithm at https://github.com/maddyscientist/AlgRemez.

We do not have to repeat similar calculations to determine $\chi_i \equiv (DD^\dagger + b_i)^{-1}\Phi$ for each i, because there is a convenient tool called the multi-mass solver that solves Q equations

$$\left(DD^\dagger + b_i\right)\chi_i = \Phi \tag{6.73}$$

simultaneously without increasing the cost. See Appendix E.2 for details.

When we fix F and update G via the HMC algorithm, by using (6.71) we approximate the Hamiltonian as

$$H = \frac{1}{2}\mathrm{Tr}(PP^\dagger) + S(G) + a_0' F^\dagger F + \sum_{i=1}^{Q'} a_i' F^\dagger (DD^\dagger + b_i')^{-1} F. \tag{6.74}$$

We can solve

$$\left(DD^\dagger + b_i'\right)\chi_i' = F \tag{6.75}$$

simultaneously via the multi-mass solver, and the Hamiltonian can be expressed as

$$H = \frac{1}{2}\mathrm{Tr}(PP^\dagger) + S(G) + a_0' F^\dagger F + \sum_{i=1}^{Q'} a_i' \chi_i'^\dagger (DD^\dagger + b_i')\chi_i'. \tag{6.76}$$

Then the Hamilton equation can be written as

$$\frac{dG_{ij}}{d\tau} = P_{ij}, \tag{6.77}$$

$$\frac{dP_{ij}}{d\tau} = -\frac{\partial \tilde{S}}{\partial G_{ij}^\dagger} = -\frac{\partial S}{\partial G_{ij}^\dagger} + \sum_{k=1}^{Q'} a_k' \chi_k'^\dagger \frac{\partial (DD^\dagger)}{\partial G_{ij}^\dagger}\chi_k', \tag{6.78}$$

which is almost the same as the Hamilton equation in the lattice QCD simulation.

This algorithm is called RHMC [32, 33] because it is a combination of the rational approximation and the HMC algorithm. Let us summarize the procedures:

RHMC simulation of supersymmetric gauge theory

1. Generate Φ with the Gaussian weight.
2. Solve $\left(DD^\dagger + b_i\right)\chi_i = \Phi$ simultaneously and derive χ_i. Then F is obtained via $F = a_0\Phi + \sum_{i=1}^{Q} a_i\chi_i$.
3. Solve $\left(DD^\dagger + b_i'\right)\chi_i' = F$ simultaneously and derive χ_i'. Calculate H_i via (6.76).
4. Fix F and let G evolve via leapfrog. The time derivatives are given by (6.77) and (6.78). Note that, although F is fixed, χ_i' changes as G evolves. Therefore, we have to solve $\left(DD^\dagger + b_i'\right)\chi_i' = F$ and obtain χ_i' at each step of the leapfrog.
5. Solve $\left(DD^\dagger + b_i'\right)\chi_i' = F$ simultaneously and derive χ_i'. Calculate H_f via (6.76).
6. Perform the Metropolis test and accept or reject $G(\tau = \tau_f)$ obtained via the leapfrog.
7. Repeat.

By using the RHMC algorithm, we can study supersymmetric gauge theories that describe black hole via holography. In Fig. 6.26, the simulation result of one of such theories (the maximally supersymmetric matrix quantum mechanics) is shown. The horizontal and vertical axes are temperature and energy, respectively. The energy is identified with the mass of the black hole via a famous relation $E = mc^2$ discovered by Einstein. The black hole corresponding to this gauge theory was identified in Ref. [34]. The solid line is the energy of this black hole calculated by using general relativity (i.e., Einstein's theory of gravity). Holographic principle predicts that the agreement between gauge theory and general relativity is better at lower temperatures. We can confirm this prediction in Fig. 6.26, i.e., we can see that the simulation data points approach the green line. The dashed line is a fit of the simulation results that takes the string-theoretic corrections into account. Monte Carlo simulation provided us with the evidence supporting the correctness of gauge/gravity duality.

6.4.3 Further Optimization

Multi-time-Step Method

In the HMC and RHMC algorithms, the discretization error associated with the leapfrog time evolution becomes large when the force $-\frac{\partial S}{\partial G}$ is large. Then the change of the Hamiltonian becomes large as well, and the acceptance rate goes down. To avoid this problem, we can use a smaller step size for a variable with a larger force.

The multi-time-step method [36] is based on a very clever idea: *use multiple step sizes for the same variable*. Suppose the Hamiltonian is written as

$$H = \sum P^2 + S_1[G] + S_2[G], \tag{6.79}$$

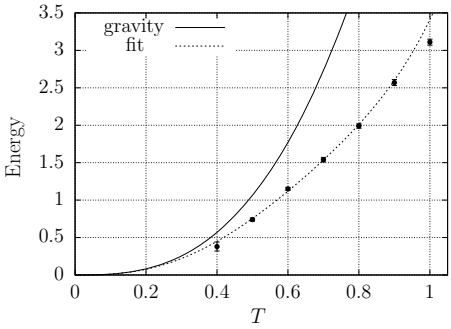

Fig. 6.26 The energy of the maximally supersymmetric matrix quantum mechanics as a function of temperature T calculated by using the RHMC algorithm. The horizontal and vertical axes are temperature and energy, respectively. The solid line is the energy of this black hole calculated by using gravity (general relativity), and the dashed line is a fit of the simulation results that takes the string-theoretic corrections into account. This plot was created based on Ref. [35]

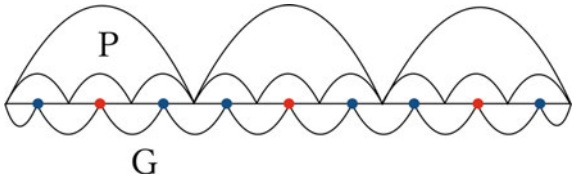

Fig. 6.27 Multi-time-step leapfrog. $N_\tau = 3, l = 1$

and the force originating from S_1 is much larger than that from S_2. We introduce two different step sizes $\Delta\tau_1$ and $\Delta\tau_2$ for S_1 and S_2 satisfying

$$\Delta\tau_2 = (2l+1)\Delta\tau_1. \tag{6.80}$$

Here l is a natural number. The leapfrog time evolution is defined as shown in Fig. 6.27. The total number of steps is $(2l+1)N_\tau$. We always use the time step $\Delta\tau_1$, and take the force for the time evolution of P to be $-\frac{\partial S_1}{\partial G}$ at blue points and $-\frac{\partial S_1}{\partial G} - (2l+1)\frac{\partial S_2}{\partial G}$ at red points. Then, effectively, the forces coming from S_1 and S_2 are incorporated every $\Delta\tau_1$ and $\Delta\tau_2$, respectively. In QCD and supersymmetric gauge theory, we can take $S_1 = S(G)$, and S_2 can be taken as the term arising from the determinant $\det(DD^\dagger)$. Often, the force coming from S_1 is large and hence we have to calculate it frequently, but the cost per calculation is very small. On the other hand, the force coming from S_2 is small but the cost per calculation is huge, so we want to avoid calculating them too often. The multi-time-step method allows us to reduce the discretization error without increasing the computational cost significantly.

n-th Root Trick

When the force coming from the determinant is too large, a naive multi-time-step method does not work. The low-temperature region of supersymmetric gauge theory has this problem. The n-th root trick is effective in such cases.

Let us introduce n pseudofermions. Then

$$\left(\det(DD^\dagger)\right)^{1/4} = \int dF_1 dF_1^* \cdots dF_n dF_n^* \exp\left(-\sum_{i=1}^{n} F_i^\dagger (D^\dagger D)^{-1/4n} F_i\right). \quad (6.81)$$

We can simulate a theory with this action by using the RHMC algorithm. (We only have to change (6.71) to the rational approximations of $x^{1/8n}$ and $x^{-1/4n}$.) Although we have to repeat the same calculation n times because there are n pseudofermions, the force coming from the determinant becomes much smaller (roughly the n-th root), and hence we can take the step size $\Delta\tau_2$ large and reduce the computational cost.

References

1. H. Cramer, *Mathematical Methods of Statistics* (Princeton University Press, 1946)
2. G. Young, R. Smith, *Essentials of Statistical Inference*, vol. 16 (Cambridge University Press, 2005)
3. G. Casella, R.L. Berger, *Statistical Inference*, 2nd edn. (Duxbury, 2001)
4. W. Lenz, Beitrag zum verständnis der magnetischen erscheinungen in festen körpern. Z. Phys. **21**, 613–615 (1920)
5. E. Ising, Beitrag zur theorie des ferromagnetismus. Z. Phys. **31**(1), 253–258 (1925)
6. M. Newman, G. Barkema, *Monte Carlo Methods in Statistical Physics* (Oxford University Press, 1999)
7. D. Landau, K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge University Press, 2000)
8. L. Onsager, Crystal statistics. i. a two-dimensional model with an order-disorder transition. Phys. Rev. **65**(3–4), 117 (1944)
9. U. Wolff, Collective Monte Carlo updating for spin systems. Phys. Rev. Lett. **62**(4), 361 (1989)
10. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
11. R.H. Swendsen, J.-S. Wang, Replica Monte Carlo simulation of spin-glasses. Phys. Rev. Lett. **57**(21), 2607 (1986)
12. S.W. Hawking, Particle creation by Black Holes. Commun. Math. Phys. **43**, 199–220 (1975). https://doi.org/10.1007/BF02345020. [Erratum: Commun. Math. Phys. 46, 206 (1976)]
13. S.W. Hawking, D.N. Page, Thermodynamics of black holes in anti-de sitter space. Commun. Math. Phys. **87**(4), 577–588 (1983)
14. E. Witten, Anti-de Sitter space, thermal phase transition, and confinement in gauge theories. Adv. Theor. Math. Phys. **2**, 505–532 (1998). https://doi.org/10.4310/ATMP.1998.v2.n3.a3
15. M. Hanada, G. Ishiki, H. Watanabe, Partial deconfinement. J. High Energy Phys. **2019**(3), 1–25 (2019). [Erratum: J. High Energy Phys. **2019**(10), 029 (2019)]
16. M. Hanada, A. Jevicki, C. Peng, N. Wintergerst, Anatomy of deconfinement. J. High Energy Phys. **2019**(12), 1–21 (2019)

17. R.P. Feynman, R.B. Leighton, M. Sands, Feynman Lect. Phys. https://www.feynmanlectures.caltech.edu
18. R.P. Feynman, Space-time approach to non-relativistic quantum mechanics. Rev. Mod. Phys. **20**, 367–387 (1948)
19. R.P. Feynman, Space-time approach to quantum electrodynamics. Phys. Rev. **76**(6), 769 (1949)
20. K.G. Wilson, Confinement of quarks. Phys. Rev. D **10**(8), 2445 (1974)
21. S. Duane, A.D. Kennedy, B.J. Pendleton, D. Roweth, Hybrid Monte Carlo. Phys. Lett. B **195**(2), 216–222 (1987)
22. S. Aoki, K.-I. Ishikawa, N. Ishizuka, T. Izubuchi, D. Kadoh, K. Kanaya, Y. Kuramashi, Y. Namekawa, M. Okawa, Y. Taniguchi et al., 2+ 1 flavor lattice qcd toward the physical point. Phys. Rev. D **79**(3), 034503 (2009)
23. S. Dürr, Z. Fodor, J. Frison, C. Hoelbling, R. Hoffmann, S. Katz, S. Krieg, T. Kurth, L. Lellouch, T. Lippert et al., Ab initio determination of light hadron masses. Science **322**(5905), 1224–1227 (2008)
24. M.B. Green, J.H. Schwarz, E. Witten, *Superstring Theory. Vol. 1: Introduction; Vol. 2: Loop Amplitudes, Anomalies and Phenomenology* (Cambridge University Press, 1987)
25. J. Polchinski, *String Theory. Vol. 1: An Introduction to the Bosonic String; Vol. 2: Superstring Theory and Beyond* (Cambridge University Press, 1998)
26. K. Becker, M. Becker, J.H. Schwarz, *String Theory and M-theory: A Modern Introduction* (Cambridge University Press, 2006)
27. B. Zwiebach, *A First Course in String Theory* (Cambridge University Press, 2004)
28. J.D. Bekenstein, Black holes and entropy. Phys. Rev. D **7**, 2333–2346 (1973). https://doi.org/10.1103/PhysRevD.7.2333
29. G. 't Hooft, Dimensional reduction in quantum gravity. Conf. Proc. C **930308**, 284–296 (1993)
30. L. Susskind, The world as a hologram. J. Math. Phys. **36**(11), 6377–6396 (1995)
31. J. Maldacena, The large-n limit of superconformal field theories and supergravity. Int. J. Theor. Phys. **38**(4), 1113–1133 (1999)
32. M.A. Clark, A.D. Kennedy, The RHMC algorithm for two flavors of dynamical staggered fermions. Nucl. Phys. B Proc. Suppl. **129**, 850–852 (2004). https://doi.org/10.1016/S0920-5632(03)02732-4
33. M.A. Clark, The rational hybrid Monte Carlo algorithm. PoS **LAT2006**, 004 (2006). https://doi.org/10.22323/1.032.0004
34. N. Itzhaki, J.M. Maldacena, J. Sonnenschein, S. Yankielowicz, Supergravity and the large n limit of theories with sixteen supercharges. Phys. Rev. D **58**(4), 046004 (1998)
35. E. Berkowitz, E. Rinaldi, M. Hanada, G. Ishiki, S. Shimasaki, P. Vranas, Precision lattice test of the gauge/gravity duality at large n. Phys. Rev. D **94**(9), 094501 (2016)
36. J. Sexton, D. Weingarten, Hamiltonian evolution for the hybrid Monte Carlo algorithm. Nucl. Phys. B **380**(3), 665–677 (1992)

Appendix A
List of Sample Code

Sample code can be obtained from

https://github.com/masanorihanada/MCMC-Sample-Codes

No special package is used, so they can be compiled just by using the standard C/C++ compilers. Some sample code in Python 3 can also be found there.

A.1 Naive Monte Carlo Method (*Not* MCMC)

A.1.1 Calculation of π (Sect. 2.2.1)

<File name> `pi_MC.c`
<Parameter> `niter`
<Output> Number of trials (equivalently, number of samples), the approximate value of the area of the fan-shaped region
<How it works>
In this code, the area of the fan-shaped region is estimated by scattering the points randomly and counting how many of them fall into this region. Please vary the number of trials `niter` and see if the result of the simulation is close to the exact analytic value $\frac{\pi}{4}$.

A.1.2 Calculation of π (Sect. 2.2.2)

<File name> `pi_MC_integral.c`
<Parameter> `niter`
<Output> Number of trials, the approximate value of $\int_0^1 dx\sqrt{1-x^2}$

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7

<How it works>

This code estimates the integral $\int_0^1 dx\sqrt{1-x^2}$. The approximate value converges to $\frac{\pi}{4}$ as `niter` becomes large.

A.1.3 Calculation of the Volume of a Ball $x^2 + y^2 + z^2 \leq 1$ (Sect. 2.3)

<File name> `three_sphere.c`
<Parameter> `niter`
<Output> Number of trials, the approximate value of the volume of a ball $x^2 + y^2 + z^2 \leq 1$
<How it works>

This code calculates the volume of a ball with radius one. The approximate value converges to the analytic value $\frac{4\pi}{3}$ as `niter` becomes large.

A.2 Metropolis Algorithm

A.2.1 Univariate Gaussian Integration via Metropolis (Sect. 4.2)

<File name> `Gaussian_Matropolis.c`
<Parameters> `niter`, `step_size`
<Output> x, acceptance rate
<How it works>

This code generates the Gaussian distribution $P(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$ by using the Metropolis algorithm. The acceptance rate and $x^{(1)}, x^{(2)}, \ldots, x^{(k)}, \ldots$ are shown as output. The initial configuration for the simulation is $x^{(0)} = 0$. By changing $S(x)$ to other functions, various other probability distributions can be generated. (`action_init` = ... and `action_fin` = ... should be modified for that purpose.) Please try several different choices of the number of samples `niter` and step size `step_size` and see how the result changes.

A.2.2 *Bivariate Gaussian Integration via Metropolis (Sect. 4.8.1)*

<File name> `Gaussian_Metropolis_2variables.c`
<Parameters> `niter`, `step_size_x`, `step_size_y`
<Output> x, y, acceptance rate
<How it works>

This code generates the bivariate Gaussian distribution,

$$P(x, y) \propto e^{-\frac{x^2+y^2+xy}{2}}. \tag{A.1}$$

By changing $S(x, y)$ to other functions, various other probability distributions can be studied. (`action_init` = ... and `action_fin` = ... should be modified for that purpose.) Depending on the detail of the distribution, it might be better to choose different values for the step size for x (`step_size_x`) and that for y (`step_size_y`).

A.2.3 *Two-Dimensional Ising Model via Metropolis (Sect. 6.2.3)*

<File name> `2d_Ising_Metropolis.cpp`
<Parameters> `niter`, `nx`, `ny`, `coupling_J`, `coupling_h`, `temperature`, `nskip`, `nconfig`
<Output>

`output.txt`: Total spin, energy, acceptance rate
`output_config.txt`: Final configuration (x coordinate, y coordinate, spin)
<How it works>

This is a simulation code for the two-dimensional Ising model via the Metropolis algorithm. The size of the lattice is specified by the number of points in x and y directions, `nx` and `ny`. Parameters `coupling_J` and `coupling_h` appear in the definition of the energy (J and h in Eq. (6.45)), and `temperature` is temperature T. Total spin, energy, acceptance rate are calculated every `nskip` steps and written in the output file `output.txt`.

The Metropolis algorithm suffers from very large autocorrelation near the phase transition point. Therefore, the final configuration is saved in the file `output_config.txt`, so that the simulation can be continued further by using it as the initial configuration.

`nconfig` is used to specify the initial configuration. If `nconfig` $= 1$ or `nconfig` $= -1$, all spins are taken to be $+1$ or -1, respectively. If you want to start the simulation by using the existing configuration (the final configuration `output_config.txt` from another run), please change the file name to `input_config.txt` and set `nconfig` $= 0$.

A.2.4 Bayesian Updating for Coin Toss via Metropolis (Sect. 6.1.5)

<File name> `Bayes_coin_toss_metropolis.c`
<Parameters> `niter`, `step_size`
<Output> p, acceptance rate
<How it works>

This code is for the Bayesian updating for the coin-toss problem via the Metropolis algorithm. The action used here is based on the example in the main text, $n = 1000$, $k = 515$, $P(p) \propto e^{-100\left(p-\frac{9}{10}\right)^2}$. For details, see the description around (6.40). To study other cases, please rewrite S in the right-hand side of `action_init` = ... and `action_fin` = ... accordingly. `niter` is the number of iterations (number of configurations collected in the simulation), and `step_size` is the step size for p.

A.2.5 Bayesian Updating for Bivariate Gaussian Distribution via Metropolis (Sect. 6.1.5)

<File name> `Bayes_Gaussian_Metropolis.cpp`
<Parameters> `niter`, `av_x`, `av_y`, `av_xx`, `av_yy`, `av_xy`, `nsample`, `step_A`, `step_mu`, `nskip`
<Output> A_{11}, A_{22}, A_{12}, μ_1, μ_2, acceptance rate
<How it works>

The Bayesian updating for the bivariate Gaussian distribution is performed via the Metropolis algorithm. We assume that `nsample` $= n$ samples $\{x^{(1)}, y^{(1)}\}$, $\ldots$, $\{x^{(n)}, y^{(n)}\}$ were obtained via some experiment and the average values `av_x` $= \overline{x}$, `av_y` $= \overline{y}$, `av_xx` $= \overline{xx}$, `av_yy` $= \overline{yy}$ and `av_xy` $= \overline{xy}$ were obtained. By using them as input, and by using $P(\{A_{ij}, \mu_i\}) \propto e^{-\frac{1}{2}\sum_{i,j}|A_{ij}|^2 - \frac{1}{2}\sum_i |\mu_i|^2}$ as the prior distribution, the Bayesian updating is performed. `niter` is the number of iterations, and the values of A_{11}, A_{22}, A_{12}, μ_1 and μ_2 are shown every `nskip` steps (hence the number of configurations collected in the simulation is `niter`/`nskip`). `step_A` and `step_mu` are the step sizes for A and μ, respectively. Please make sure about the difference between `niter` and `nsample`. (Sorry for confusing names!)

A.3 HMC Algorithm

A.3.1 Univariate Gaussian Integration via HMC (Sect. 5.1.4)

<File name> `Gaussian_HMC.cpp`
<Parameters> `niter`, `ntau`, `dtau`

<Output> x, $\langle x^2 \rangle$, acceptance rate
<How it works>

This code generates the Gaussian distribution $P(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ via HMC. It is instructive to see how the acceptance rate and the autocorrelation length change when `ntau` and `dtau` are varied.

Various other probability distributions can be studied by modifying `calc_action` and `calc_delh`.

A.3.2 Multivariate Gaussian Integration via HMC (Sect. 5.1.5)

<File name> `Gaussian_HMC_multi_variables.cpp`
<Parameters> `niter`, `ntau`, `dtau`, `ndim`
<Output> x, y, z, acceptance rate
<How it works>

This code generates the multivariate Gaussian distribution, $P(\{x\}) \propto e^{-\frac{1}{2}\sum_{i,j} A_{ij} x_i x_j}$, via HMC. The number of variables `ndim` can be chosen arbitrarily. (`ndim` means the number of dimensions.) The values of A_{ij} are set explicitly in `main`. As a default setup, we chose `ndim` $= 3$, $P(x, y, z) \propto e^{-\frac{x^2+2y^2+2z^2+2xy+2yz+2zx}{2}}$. As for the simulation result with this default setup, see Figs. 5.8 and 5.9.

Various other probability distributions can be studied by modifying `calc_action` and `calc_delh`.

A.3.3 Bayesian Updating for Bivariate Gaussian Distribution via HMC (Sect. 6.1.5)

<File name> `Bayes_Gaussian_HMC_2variables.cpp`
<Parameters> `niter`, `av_x`, `av_y`, `av_xx`, `av_yy`, `av_xy`, `nsample`, `ntau` `dtau_A`, `dtau_mu`, `nskip`
<Output> A_{11}, A_{22}, A_{12}, μ_1, μ_2, acceptance rate
<How it works>

This code performs the Bayesian updating for the bivariate Gaussian distribution via the HMC algorithm. We use the averages `av_x` $= \overline{x}$, `av_y` $= \overline{y}$, `av_xx` $= \overline{xx}$, `av_yy` $= \overline{yy}$ and `av_xy` $= \overline{xy}$, which were obtained from `nsample` $= n$ samples $\{x^{(1)}, y^{(1)}\}, \ldots, \{x^{(n)}, y^{(n)}\}$. By using them as input, and by using $P(\{A_{ij}, \mu_i\}) \propto e^{-\frac{1}{2}\sum_{i,j}|A_{ij}|^2 - \frac{1}{2}\sum_i |\mu_i|^2}$ as the prior distribution, the Bayesian updating is performed. Different priors can be used by rewriting `calc_action` and `calc_delh`.

As for the step size $\Delta\tau$, we can use different values for A_{ij}, $p_{ij}^{(A)}$ and μ_i, $p_i^{(\mu)}$. We can use different values for A_{11}, A_{12} and A_{22}, but for simplicity we used the same value $\Delta\tau_A$ for all A_{ij}, $p_{ij}^{(A)}$. We used the same value $\Delta\tau_\mu$ for all μ_i, $p_i^{(\mu)}$ as well.

A.3.4 Matrix Integral via HMC (Sect. 5.1.5)

<File name> `matrix_HMC.cpp`
<Parameters> `nmat`, `niter`, `ninit`, `ntau`, `dtau`
<Output>
`output.txt`: $\langle S \rangle$ (the expectation value of the action), acceptance rate
`configuration.dat`: Final configuration
<How it works>

This code is for the matrix model whose action is specified in eq. (5.22). In `Molecular_Dynamics`, the auxiliary momenta are generated randomly following the Gaussian distribution, the configurations are evolved along the auxiliary time, and the Hamiltonian is calculated. `calc_hamiltonian`, `calc_delh` and `calc_action` calculate the Hamiltonian, 'force' acting on ϕ (namely $\frac{\partial H}{\partial \phi_{ji}} = \frac{\partial S}{\partial \phi_{ji}}$) and the action $S(\phi)$, respectively.

When the matrix size `nmat` is large, the simulation becomes time-consuming and we need to run multiple jobs successively. Therefore, the final configuration of each job is saved in `configuration.dat`, and when `ninit` $= 0$ the configuration in this file is used as the initial configuration.

In this code, we do not save the information regarding the pseudorandom numbers, we just set the seed every time by using the system clock. For more serious simulations, it is better to save the information regarding the pseudorandom numbers together with the configuration.

A.4 Gibbs Sampling Algorithm (Heat Bath Algorithm)

A.4.1 Multivariate Gaussian Integration via Gibbs Sampling (Sect. 5.2.3)

<File name> `Gaussian_Gibbs.cpp`
<Parameter> `niter`
<Output> x, y, z
<How it works>

This is a Gibbs-sampling code for the trivariate Gaussian distribution. The parameters A_{ij} can be set to any values in `main`.

A.4.2 Two-Dimensional Ising Model via Gibbs Sampling (Sect. 6.2.3)

<File name> `2d_Ising_Heat_Bath.cpp`
<Parameters> `niter`, `nx`, `ny`, `coupling_J`, `coupling_h`, `temperature`, `nskip`, `nconfig`
<Output>
`output.txt`: Total spin, energy
`output_config.txt`: Final configuration (x coordinate, y coordinate, spin)
<How it works>

This code studies the two-dimensional Ising model via the Gibbs sampling algorithm (equivalently, the heat bath algorithm). The parameters are the same as those used in Appendix A.2.3.

There is a problem of large autocorrelation near the phase transition point, just like the Metropolis algorithm. Therefore, the final configuration is saved in `output_config.txt` so that the simulation can be continued if necessary.

A.5 Combination of Different Algorithms

A.5.1 Bayesian Updating for Bivariate Gaussian Distribution via Gibbs Sampling and Metropolis (Sect. 6.1.5)

<File name> `Bayes_Gaussian_Gibbs.cpp`
<Parameters> `niter`, `av_x`, `av_y`, `av_xx`, `av_yy`, `av_xy`, `nsample`, `step_A`, `nskip`
<Output> A_{11}, A_{22}, A_{12}, μ_1, μ_2, acceptance rate
<How it works>

This code uses the Gibbs sampling algorithm for μ and the Metropolis algorithm for A. The averages `av_x` $= \overline{x}$, `av_y` $= \overline{y}$, `av_xx` $= \overline{xx}$, `av_yy` $= \overline{yy}$ and `av_xy` $= \overline{xy}$, which were obtained from `nsample` $= n$ samples $\{x^{(1)}, y^{(1)}\}, \ldots, \{x^{(n)}, y^{(n)}\}$, are used. By using them as input, and by using $P(\{A_{ij}, \mu_i\}) \propto e^{-\frac{1}{2}\sum_{i,j}|A_{ij}|^2 - \frac{1}{2}\sum_i |\mu_i|^2}$ as the prior distribution, the Bayesian updating is performed. The step size for Metropolis is `step_A`. The frequency of the sampling is specified by `nskip`, i.e., the output is shown every `nskip` steps.

A.6 Cluster Algorithm

A.6.1 Two-Dimensional Ising Model via the Wolff Algorithm (Sect. 6.2.4)

<File name> `2d_Ising_Wolff.cpp`
<Parameters> `niter`, `nx`, `ny`, `coupling_J`, `coupling_h`, `temperature`, `nskip`, `nconfig`
<Output>
`output.txt`: Total spin, energy
`output_config.txt`: Final configuration (x coordinate, y coordinate, spin)
<How it works>

This code studies the two-dimensional Ising model via the Wolff algorithm. Total spin and energy are calculated every `nskip` steps and recorded in `output.txt`. When `nconfig` is a positive integer, the configuration is saved in `output_config.txt` every `nconfig` steps. (Note that the meaning of `nconfig` is different from the Metropolis and Gibbs sampling codes.) The initial configuration is all-spin-up.

`make_cluster` is the most important part that constructs the cluster. The structure of `main` is extremely simple: the cluster is constructed by using `make_cluster`, and then the spins in the cluster are flipped or kept unchanged depending on the result of the Metropolis test.

A.7 Replica Exchange Algorithm

A.7.1 Simple Integral via Replica Exchange (Sect. 6.3.3)

<File name> `replica_simple_example.c`
<Parameters> `nbeta`, `niter`, `step_size`, `dbeta`
<Output> The values of x at `nbeta` $= 20$, 200 and 2000
<How it works>

Probability distribution $P(x) \propto e^{-f(x)/T}$ is generated for $f(x) = (x-1)^2 \cdot \left((x+1)^2 + 0.01\right)$, via the replica exchange algorithm. `nbeta` is the number of $\beta = \frac{1}{T}$ (number of replicas) and `dbeta` is the spacing between the values of β for replicas. The values of β are $\beta =$ `dbeta`, `dbeta` $\times 2, \ldots,$ `dbeta` $\times$ `nbeta`. Each replica is updated via the Metropolis algorithm with step size `step_size`, then the replicas are exchanged again via Metropolis.

A.7.2 Traveling Salesman Problem via Replica Exchange (Sect. 6.3.3)

<File name> `replica_salesman.c`
<Parameters> `nbeta`, `niter`, `step_size`, `dbeta`
<Output> `output.txt`: Number of iterations, total distance r_{total} at the lowest temperature replica, the smallest r_{total} found by that stage are shown. Then, at the end, the coordinates of the cities are shown along the shortest route found during the simulation.

`output_config.txt`: Final configuration (location of the cities, the route at each replica)
<How it works>

This is a code for the traveling salesman problem via the replica exchange algorithm.

`nbeta` is the number of $\beta = \frac{1}{T}$ (number of replicas) and the values of β are $\beta =$ `dbeta`, `dbeta` $\times 2, \ldots,$ `dbeta` $\times$ `nbeta`. `ncity` is the number of cities.

If `ninit` is 0, then the locations of the cities are read from `100_cities.txt` at the beginning of the simulation. If `ninit` is 1, then the configuration (location of the cities and the route at each replica) are read from `input_config.txt`. If `ninit` is 2, then the locations of the cities are set by using the uniform random numbers at $0 < x < 1$ and $0 < y < 1$.

`calc_distance` calculates the total distance along the route, r_{total}.

In order to calculate the change of the total distance Δr_{total} when ordering of visiting two cities are exchanged ($i_k^{(m)} \to i_k'^{(m)} = i_l^{(m)}$, $i_l^{(m)} \to i_l'^{(m)} = i_k^{(m)}$), we do not have to calculate r_{total} every time. However, because the authors were lazy and did not spend too much time for coding, Δr_{total} is obtained by calculating r_{total} before and after exchanging the ordering, and then by taking the difference. If you improve this part, the simulation becomes much faster.

In this code, two steps explained in the main text (MCMC at each replica and the exchange of replicas) are performed alternately. You can change this part, say to repeat the first step several times and then to perform the second step once.

The final configuration is saved in `output_config.txt`. By changing the file name to `input_config.txt` and choosing `ninit` to be 1, the simulation can be continued.

Appendix B
Miscellaneous Math Topics

B.1 Matrix

Let us consider an $m \times n$ matrix M. If we write all matrix entries explicitly, there are m rows and n columns:

$$M = \begin{pmatrix} M_{11} & M_{12} & \cdots & M_{1n} \\ M_{21} & M_{22} & \cdots & M_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1} & M_{m2} & \cdots & M_{mn} \end{pmatrix}. \tag{B.1}$$

M is called a square matrix if $m = n$. If all matrix entries are real numbers (respectively, complex numbers), it is called a real matrix (respectively, complex matrix).

Vector can be regarded as a special kind of matrix. An $m \times 1$ matrix is an m-component column vector and a $1 \times n$ matrix is an n-component row vector.

Summation, subtraction, and multiplication of scalar

The sum is taken component-by-component. Namely the (i, j)-component of $M + M'$ is $(M + M')_{ij} = M_{ij} + M'_{ij}$. The subtraction is taken component-by-component, too.

Multiplication of a scalar c and a matrix M, which is written as cM, means a matrix whose (i, j)-component is cM_{ij}, i.e., all matrix entries are multiplied by c.

Product of matrices

Let M and M' be $l \times m$ and $m \times n$ matrices, respectively. The product of M and M', denoted by MM', is defined by an $l \times n$ matrix whose (i, j)-component is

$$(MM')_{ij} = \sum_{k=1}^{m} M_{ik} M'_{kj}. \tag{B.2}$$

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7

In order to see the motivation for this definition, let us consider a simple example. We use a two-component vector $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$ to denote the coordinate in a plane. By using a polar coordinate, we write $\vec{v}$ as

$$\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix}. \tag{B.3}$$

To this vector, we multiply a 2×2 square matrix M defined by

$$M(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}. \tag{B.4}$$

Then, we obtain

$$M(\phi) \cdot \vec{v} = \begin{pmatrix} r\cos(\theta+\phi) \\ r\sin(\theta+\phi) \end{pmatrix}. \tag{B.5}$$

We can see that the multiplication of $M(\phi)$ corresponds to a rotation with the angle ϕ. If we perform two rotations with the angle ϕ_1 and ϕ_2, we should get a rotation with the angle $\phi_1 + \phi_2$. This is naturally realized as a product of matrices $M(\phi_2) \cdot M(\phi_1) = M(\phi_1 + \phi_2)$. Other manipulations such as rotations in higher dimensional space or stretching/shrinking to some directions can also be described by matrices, and a product of two matrices is interpreted as a combination of two manipulations.

Inverse matrix

For a square matrix M, a matrix M^{-1} that satisfies $M^{-1}M = \mathbf{1}$ is called the inverse of M. Here $\mathbf{1}$ is the identity matrix, i.e., all diagonal entries are 1 and all off-diagonal entries are 0. If M represents a rotation, then M^{-1} is simply the rotation to the opposite direction. When the inverse matrix M^{-1} exists, M is said to be invertible (also regular or non-singular).

Transpose and Hermitian conjugate

For an $m \times n$ matrix M, the transpose M^{T} is defined by an $n \times m$ matrix whose entries are $\left(M^{\mathrm{T}}\right)_{ij} = M_{ji}$.

For an $m \times n$ complex matrix M, the Hermitian conjugate $M^\dagger$ is defined by an $n \times m$ matrix whose entries are $\left(M^\dagger\right)_{ij} = M^*_{ji}$, where $*$ means the complex conjugate. Because $M^\dagger = \left(M^{\mathrm{T}}\right)^*$, the Hermitian conjugation is a combination of transpose and complex conjugation.

A matrix M is "symmetric" if $M^{\mathrm{T}} = M$, and "Hermitian" if $M^\dagger = M$.

Determinant

For an $n \times n$ square matrix M, the determinant is defined by

$$\det M = \sum_{\sigma} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} M_{i\sigma(i)}. \tag{B.6}$$

σ runs through all permutations of $1, 2, \ldots, n$, and $\mathrm{sgn}(\sigma)$ is the signature of the permutation σ.

For example, when $n = 3$, there are six permutations that map $(1, 2, 3)$ to $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$ or $(3, 2, 1)$.

Any permutation can be obtained by repeating the permutations of two numbers ('transpositions'). For example, $(1, 2, 3) \to (2, 3, 1)$ is made of two transpositions, $(1, 2, 3) \to (2, 1, 3) \to (2, 3, 1)$. The signature $\mathrm{sgn}(\sigma)$ is $+1$ when σ consists of an even number of transpositions, and -1 when σ consists of an odd number of transpositions.

The determinant $\det M$ represents the change of the volume via the multiplication of M.

By calculating honestly following the definition, we see that the product of determinants is the determinant of the product:

$$\det\left(MM'\right) = \det M \cdot \det M'. \tag{B.7}$$

This relation, together with the fact that the determinant of $\mathbf{1}$ is 1, leads to

$$\det M^{-1} = \frac{1}{\det M}. \tag{B.8}$$

Therefore, invertible matrices have nonzero determinants.

When M can be diagonalized by using an invertible matrix A as $A^{-1}MA = \mathrm{diag}(d_1, \ldots, d_n)$ (these $d_1, d_2, \ldots, d_n$ are called eigenvalues), the determinant of M is the product of $d_1, d_2, \ldots, d_n$,

$$\det M = d_1 \times d_2 \times \cdots \times d_n, \tag{B.9}$$

because

$$\det(A^{-1}MA) = \det A^{-1} \cdot \det M \cdot \det A = \det M. \tag{B.10}$$

Logarithm and exponential

The exponential of a square matrix M is defined by the Taylor expansion as

$$e^M = \sum_{k=0}^{\infty} \frac{M^k}{k!}. \tag{B.11}$$

When M can be diagonalized as $A^{-1}MA = \mathrm{diag}(d_1, \ldots, d_n)$, e^M can also be diagonalized as $e^M = A \times \mathrm{diag}(e^{d_1}, \ldots, e^{d_n}) \times A^{-1}$.

The logarithm is the inverse function of the exponential ($\log e^x = x$). When all eigenvalues of M are positive, the logarithm of M can be obtained as $\log M = A \times \mathrm{diag}(\log d_1, \ldots, \log d_n) \times A^{-1}$. Also, if all eigenvalues of M has the magnitudes smaller than 1, it can be defined via the Taylor series

$$\log(\mathbf{1} + M) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} M^n}{n}. \tag{B.12}$$

Because the diagonalization of a big matrix is numerically expensive, when we want to save the computational cost we truncate the Taylor series at some order, making sure that the higher order terms are sufficiently small.

Let us show an important relation used in Sects. 6.1.2 and 6.1.5,

$$\frac{\partial \mathrm{Tr} \log M}{\partial M_{ij}} = M^{-1}_{ji}. \tag{B.13}$$

For that, we use the Taylor series

$$\log M = \log(\mathbf{1} - (\mathbf{1} - M)) = -\sum_{k=1}^{\infty} \frac{(\mathbf{1} - M)^k}{k}. \tag{B.14}$$

From this, we can easily show that

$$\frac{\partial \mathrm{Tr} \log M}{\partial M_{ij}} = \sum_{k=1}^{\infty} (\mathbf{1} - M)^{k-1}_{ji} = (\mathbf{1} - (\mathbf{1} - M))^{-1}_{ji} = M^{-1}_{ji}. \tag{B.15}$$

Let us also show that

$$\log \det M = \mathrm{Tr} \log M. \tag{B.16}$$

When M can be diagonalized, we can immediately see that both left and right-hand sides are $\sum_{i=1}^{n} \log d_i$. Even when M is not diagonalizable, it can be triangulated. Namely, by using an appropriate invertible matrix A, it can be written as $M = AUA^{-1}$, where U is an upper-triangular matrix (i.e., $U_{ij} = 0$ for $i < j$). Then, by using the diagonal entries of U denoted by $U_{ii} = u_i$, the determinant is written as $\det M = \prod_i u_i$, and hence,

$$\log \det M = \sum_{i=1}^{n} \log u_i. \tag{B.17}$$

Also, by noticing that the diagonal elements of U^k are $(U^k)_{ii} = u_i^k$, we obtain

$$\begin{aligned}\operatorname{Tr}\log M &= \operatorname{Tr}\log U \\ &= -\sum_{k=1}^{\infty}\frac{\operatorname{Tr}\left[(\mathbf{1}-U)^k\right]}{k} \\ &= -\sum_{k=1}^{\infty}\sum_{i=1}^{n}\frac{(1-u_i)^k}{k} \\ &= \sum_{i=1}^{n}\log u_i . \end{aligned} \tag{B.18}$$

B.2 Gaussian Integral

B.2.1 Univariate Gaussian Integration

Let us start with a proof of the most basic formula,

$$\int_{-\infty}^{\infty} dx e^{-\frac{x^2}{2}} = \sqrt{2\pi}. \tag{B.19}$$

By using $I \equiv \int dx e^{-\frac{x^2}{2}}$, we can easily obtain

$$I^2 = \int_{-\infty}^{\infty} dx e^{-\frac{x^2}{2}} \times \int_{-\infty}^{\infty} dy e^{-\frac{y^2}{2}} = \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy e^{-\frac{x^2+y^2}{2}}. \tag{B.20}$$

We can rewrite is by using the polar coordinates $x = r\cos\theta$, $y = r\sin\theta$ as

$$I^2 = \int_0^{2\pi} d\theta \int_0^{\infty} dr r e^{-\frac{r^2}{2}} = 2\pi \cdot \left[-e^{-\frac{r^2}{2}}\right]_0^{\infty} = 2\pi. \tag{B.21}$$

Therefore, $I = \sqrt{2\pi}$.

By using this formula, we can immediately show $\int_{-\infty}^{\infty} dx e^{-\frac{x^2}{2\sigma^2}} = \sqrt{2\pi}\sigma$. Indeed, by using $y = \frac{x}{\sigma}$, we have $\int_{-\infty}^{\infty} dx e^{-\frac{x^2}{2\sigma^2}} = \sigma \int_{-\infty}^{\infty} dy e^{-\frac{y^2}{2}} = \sqrt{2\pi}\sigma$. Therefore,

$$\int_{-\infty}^{\infty} \frac{dx}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} = 1. \tag{B.22}$$

We can also show that

$$\int_{-\infty}^{\infty} \frac{dx}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = 1 \tag{B.23}$$

just by shifting the integration variable.

B.2.2 Multivariate Gaussian Integration

We start with a normalized Gaussian distribution consisting of d variables $y_1, y_2, \ldots, y_d$,

$$\rho(y_1, \ldots, y_d) = \frac{e^{-\frac{1}{2}\sum_{i=1}^{d}(y_i-\nu_i)^2}}{(2\pi)^{d/2}}. \tag{B.24}$$

We define new variables x_i and μ_i by using an invertible $N \times N$ matrix M as $x_i = \sum_j M_{ij}^{-1} y_j$ and $\mu_i = \sum_j M_{ij}^{-1} \nu_j$, respectively. Then, via $A = M^T M$, the distribution can be expressed as

$$e^{-\frac{1}{2}\sum_{i=1}^{d}(y_i-\nu_i)^2} = e^{-\frac{1}{2}\sum_{i,j=1}^{d} A_{ij}(x_i-\mu_i)(x_j-\mu_j)}. \tag{B.25}$$

By taking into account the change of the integration measure as well, we can show that the normalized distribution satisfying $\int_{-\infty}^{\infty} dx_1 \cdots \int_{-\infty}^{\infty} dx_d \rho(x_1, \ldots, x_d) = 1$ is

$$\rho(x_1, \ldots, x_d) = \sqrt{\frac{\det A}{(2\pi)^d}} e^{-\frac{1}{2}\sum_{i,j=1}^{d} A_{ij}(x_i-\mu_i)(x_j-\mu_j)}. \tag{B.26}$$

Therefore, we regard (B.26) as a generalized Gaussian distribution. If all the eigenvalues of A are positive, we can change the variables such that the standard form (B.24) is obtained from (B.26). Unless all the eigenvalues of A are positive, the integral does not converge and the distribution cannot be interpreted as a probability.

Why do we consider a complicated form (B.26) instead of a simple standard form (B.24)? Because there are advantages, of course. For example, if x_1 and x_2 are the numbers of dogs and cats, y_1 and y_2 would be something like $0.1 \times$ (number of dogs) $+ 1.2356 \times$ (number of cats), which are hard to understand intuitively. Then we can understand the meaning of the distribution more easily by using (B.26). Note also that, via the method explained in Sect. 6.1, from the data x_i we can estimate A_{ij} and μ_i; we cannot get the standard form (B.24) directly.

Appendix C
Hamilton Equation

We briefly explain the Hamilton equation (5.2), in order to make it easier to understand the HMC algorithm. We use the symbol V instead of S so that people with basic knowledge of physics can follow the argument more easily. Then, the Hamiltonian H is written as

$$H = \frac{1}{2}\sum_{i=1}^{k} p_i^2 + V(x_1, \ldots, x_k). \tag{C.1}$$

Suppose there are k particles, and x_i and p_i are the coordinate and momentum of the i-th particle. The Hamiltonian can be identified with the energy of this system. The first term on the right-hand side is the kinetic energy, assuming that the mass of the particles m is 1. The second term is the potential energy. The Hamilton equation (5.2) is

$$\frac{dp_i}{d\tau} = -\frac{\partial H}{\partial x_i} = -\frac{\partial V}{\partial x_i}, \qquad \frac{dx_i}{d\tau} = \frac{\partial H}{\partial p_i} = p_i. \tag{C.2}$$

Note that $-\frac{\partial V}{\partial x_i}$ is the force acting on the i-th particle; hence the first equation is nothing but the Newton's law, "change of momentum = force". Momentum p and velocity v are related by $p = mv$. If $m = 1$, we simply have $p = v$. Hence, the second equation is just the definition of velocity, "rate of the change of coordinate = velocity". Therefore, the Hamilton equation is the equation of motion.

Because the Hamiltonian is the same as energy, it is conserved under the time evolution controlled by the equation of motion. To check it manifestly by using math, we use

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7

$$\frac{dH}{d\tau} = \sum_i \left(\frac{dx_i}{d\tau} \frac{\partial H}{\partial x_i} + \frac{dp_i}{d\tau} \frac{\partial H}{\partial p_i} \right). \tag{C.3}$$

By combining it with the Hamilton equation, we obtain

$$\frac{dH}{d\tau} = \sum_i \left(\frac{\partial H}{\partial p_i} \frac{\partial H}{\partial x_i} - \frac{\partial H}{\partial x_i} \frac{\partial H}{\partial p_i} \right) = 0. \tag{C.4}$$

Appendix D
Jackknife Method in Generic Cases

When the Jackknife method was introduced in Sect. 4.3.3, it was assumed that the quantity of interest can be calculated sample-by-sample. Below, we extend the Jackknife method to apply to more general situations, e.g., when we want to estimate the error bar for the variance $\left\langle (x - \langle x \rangle)^2 \right\rangle$.

Let f be the quantity we want to estimate. As we have done in Sect. 4.3.3, we divide the configurations into groups. Suppose that each group consists of w configurations and there are n groups in total. The first group is $\{x^{(1)}, x^{(2)}, \ldots, x^{(w)}\}$, the second group is $\{x^{(w+1)}, x^{(w+2)}, \ldots, x^{(2w)}\}$, and so on. We calculate f by removing the k-th group, and call it $\overline{f}^{(k,w)}$:

$$\overline{f}^{(k,w)} \equiv (\text{the value of } f \text{ calculated by removing the } k\text{-th group}) . \tag{D.1}$$

For example, when f is the variance and 1,000 configurations are divided to $n = 10$ groups, for each of $k = 1, 2, \ldots, n = 10$ we calculate the variance by using 900 configurations. We employ the average of $\overline{f}^{(k,w)}$'s ($k = 1, 2, \ldots, n$), which we call $\overline{f}$, as the value of f:

$$\overline{f} \equiv \frac{1}{n} \sum_k \overline{f}^{(k,w)} . \tag{D.2}$$

The Jackknife error is defined by

$$\Delta_w \equiv \sqrt{\frac{n-1}{n} \sum_k \left(\overline{f}^{(k,w)} - \overline{f} \right)^2} . \tag{D.3}$$

When f can be calculated sample-by-sample, this definition is the same as the one given in Sect. 4.3.3.

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7

Appendix E
Conjugate Gradient Method (CG Method)

We explain the Conjugate Gradient method (CG method), which we mentioned in Sect. 6.4.2, following Ref. [1]. The CG method is used to solve the linear equation

$$A\vec{x} = \vec{b}, \tag{E.1}$$

where A is a positive-definite (i.e., all the eigenvalues are positive) and Hermitian.[1] A sequence of approximate solutions $\vec{x}_1, \vec{x}_2, \ldots$, which eventually converges to the exact solution, is obtained via the CG method.

Let us first see the procedures before explaining the logic:

Conjugate Gradient method (CG method)

1. Choose a candidate of the solution $\vec{x}_1$. This $\vec{x}_1$ can be arbitrary. From this, we define $\vec{r}_1$ and $\vec{p}_1$ as $\vec{r}_1 = \vec{p}_1 = \vec{b} - A\vec{x}_1$.
   Then, $\vec{x}_k$'s are constructed as follows, for $k = 1, 2, \ldots$:
2. $\alpha_k = \frac{\vec{r}_k^\dagger \cdot \vec{r}_k}{\vec{p}_k^\dagger \cdot A\vec{p}_k}$.
3. $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$.
4. $\vec{r}_{k+1} = \vec{r}_k - \alpha_k A\vec{p}_k$.
5. $\beta_k = \frac{\vec{r}_{k+1}^\dagger \cdot \vec{r}_{k+1}}{\vec{r}_k^\dagger \cdot \vec{r}_k}$.
6. $\vec{p}_{k+1} = \vec{r}_{k+1} + \beta_k \vec{p}_k$.

By combining step 3 and step 4, we get $\vec{r}_{k+1} + A\vec{x}_{k+1} = \vec{r}_k + A\vec{x}_k$. This relation holds for any k, so together with step 1 we obtain $\vec{r}_k = \vec{b} - A\vec{x}_k$. Therefore, $\vec{r}_k$ represents the deviation from the exact solution that is called the residual vector. A good approximate solution is obtained by terminating the calculation when $\vec{r}_k$ gets sufficiently small.

[1] The matrix A appearing in the Gaussian distribution satisfies this condition.

M. Hanada and S. Matsuura, *MCMC from Scratch*,
https://doi.org/10.1007/978-981-19-2715-7

Why does $\vec{x}_k$ converge to the solution of $A\vec{x} = \vec{b}$? Equivalently, why does the residual vector $\vec{r}_k$ converge to zero? To answer this question, we rewrite the problem a little bit. Because A is a positive-definite Hermitian matrix, $\vec{v}^\dagger A^{-1}\vec{v} \geq 0$ holds for any vector $\vec{v}$ and the equality holds only when $\vec{v} = 0$. Therefore, the exact solution is obtained if $\vec{r}_k^\dagger A^{-1}\vec{r}_k$ converges to zero. So we should minimize $\vec{r}_k^\dagger A^{-1}\vec{r}_k$, which is equivalent to minimizing

$$\vec{x}^\dagger A\vec{x} - \vec{b}^\dagger\vec{x} - \vec{x}^\dagger\vec{b}. \tag{E.2}$$

Because the derivative of this quantity with respect to $\vec{x}^\dagger$ is $A\vec{x} - \vec{b}$, the gradient vector at $\vec{x} = \vec{x}_k$ is $A\vec{x}_k - \vec{b} = -\vec{r}_k$.[2] Therefore, if we construct the sequence as $\vec{x}_k \to \vec{x}_k + \epsilon\vec{r}_k$, it is the same as the gradient descent method. However, as we can see from the step 3 and step 6, in the CG method x_k is updated by using p_k rather than $\vec{r}_k$.

Let us assume that $\vec{p}_i^\dagger A\vec{p}_j = 0$ holds when $i \neq j$. (We will show that it is actually the case.) Such $\vec{p}_i$'s are linearly independent and the solution can be written as

$$\vec{x} \equiv \vec{x}_1 + \sum_{i=1}^{D} \alpha_i \vec{p}_i, \tag{E.3}$$

where D is the number of variables. If we truncate the sum as

$$\vec{x}_k \equiv \vec{x}_1 + \sum_{i=1}^{k} \alpha_i \vec{p}_i, \tag{E.4}$$

then the sequence $\vec{x}_1 \to \vec{x}_2 \to \cdots \vec{x}_k \to \cdots \to \vec{x}_D$ converges to the solution. Note that $\vec{x}_D$ is the exact solution. We show that the procedures explained above actually lead to such a convergent series.

The first step of the proof is to show that $\vec{r}_i^\dagger\vec{r}_j = 0$ and $\vec{p}_i^\dagger A\vec{p}_j = 0$ hold for $i \neq j$. We show these relations via the mathematical induction. That is, we show that, if these relations hold for $i, j \leq k$, then they hold also for $i, j \leq k+1$. For $i \leq k$, we have

$$\begin{aligned}
\vec{r}_i^\dagger\vec{r}_{k+1} &= \vec{r}_i^\dagger(\vec{r}_k - \alpha_k A\vec{p}_k) \\
&= \vec{r}_i^\dagger\vec{r}_k - \alpha_k\vec{r}_i^\dagger A\vec{p}_k \\
&= \vec{r}_i^\dagger\vec{r}_k - \alpha_k\left(\vec{p}_i^\dagger - \beta_{i-1}\vec{p}_{i-1}^\dagger\right)A\vec{p}_k \\
&= \vec{r}_i^\dagger\vec{r}_k - \alpha_k\vec{p}_i^\dagger A\vec{p}_k.
\end{aligned} \tag{E.5}$$

[2] Because the gradient can be zero only at the exact solution, there is no problem of a wrong local minimum.

This is zero for $i < k$ by assumption for the induction, and zero for $i = k$ as well due to the definition of α_k. In the same way, we can easily show

$$\begin{aligned}\vec{p}_i^{\,\dagger} A \vec{p}_{k+1} &= \vec{p}_i^{\,\dagger} A \left(\vec{r}_{k+1} + \beta_k \vec{p}_k\right) \\ &= \vec{p}_i^{\,\dagger} A \vec{r}_{k+1} + \beta_k \vec{p}_i^{\,\dagger} A \vec{p}_k \\ &= \frac{1}{\alpha_i}\left(\vec{r}_i^{\,\dagger} - \vec{r}_{i+1}^{\,\dagger}\right)\vec{r}_{k+1} + \beta_k \vec{p}_i^{\,\dagger} A \vec{p}_k \\ &= -\frac{\vec{r}_{i+1}^{\,\dagger}\vec{r}_{k+1}}{\alpha_i} + \beta_k \vec{p}_i^{\,\dagger} A \vec{p}_k, \end{aligned} \tag{E.6}$$

which is zero for $i < k$ by assumption, and zero for $i = k$ as well due to the definitions of α_k and β_k. Hence, we could justify the expansion (E.3).

Next, we show that the expansion (E.3), with our specific choice of α_i, converges to the solution we want. From the step 4, $\alpha_k A \vec{p}_k = \vec{r}_k - \vec{r}_{k+1}$ holds, from which we can immediately get $\sum_{k=1}^{D} \alpha_k A \vec{p}_k = \vec{r}_1$. By combining it with the step 1, we obtain $\sum_{k=1}^{D} \alpha_k A \vec{p}_k = \vec{b} - A\vec{x}_1$. This is equivalent to $A\left(\vec{x}_1 + \sum_{k=1}^{D} \alpha_k \vec{p}_k\right) = \vec{b}$, and hence, $\vec{x} = \vec{x}_1 + \sum_{k=1}^{D} \alpha_k \vec{p}_k$ is indeed the solution. For practical purpose, we terminate the calculation when the residual vector became sufficiently small.

In the CG method, the coefficient of $\vec{p}_k$ is completely determined at each stage; once it is fixed, it does not change later. This means that, when $\vec{x}_{k+1}$ is obtained, the remaining directions to be studied are reduced to the $D - k$-dimensional space spanned by $\vec{p}_{k+1}, \vec{p}_{k+2}, \ldots, \vec{p}_D$. The solution can be obtained efficiently due to this property.

The most costly part of the CG method is the multiplications of A to the vectors. If A is sparse (i.e., if many components of A are zero), the computational cost can be reduced by skipping the multiplication of zero. Furthermore, the parallelization is not too difficult.

E.1 BiCG Method

The CG method provides us with an efficient way to solve $A\vec{x} = \vec{b}$ when A is a sparse, positive-definite Hermitian matrix. To solve $M\vec{x} = \vec{b}$ for a generic sparse matrix M, it is convenient to use the biconjugate gradient method (biCG method).[3]

[3] Because $A = MM^\dagger$ is positive-definite and Hermitian, it is also possible to solve $(MM^\dagger)\vec{y} = \vec{b}$ via the CG method and get $\vec{y} = (MM^\dagger)^{-1}\vec{b} = (M^\dagger)^{-1}M^{-1}\vec{b}$, and then multiply $M^\dagger$ to get $M^\dagger\vec{y} = M^{-1}\vec{b} = \vec{x}$.

BiCG method

1. Choose a candidate of the solution $\vec{x}_1$. This $\vec{x}_1$ can be arbitrary. From this, we define $\vec{r}_1, \vec{\bar{r}}_1, \vec{p}_1$ and $\vec{\bar{p}}_1$ as $\vec{r}_1 = \vec{\bar{r}}_1 = \vec{p}_1 = \vec{\bar{p}}_1 = \vec{b} - M\vec{x}_1$.
   Then, $\vec{x}_k$'s are constructed as follows:
2. $\alpha_k = \frac{\vec{\bar{r}}_k^{\mathrm{T}} \cdot \vec{r}_k}{\vec{\bar{p}}_k^{\mathrm{T}} \cdot M\vec{p}_k}$.
3. $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$.
4. $\vec{r}_{k+1} = \vec{r}_k - \alpha_k M\vec{p}_k$,
   $\vec{\bar{r}}_{k+1} = \vec{\bar{r}}_k - \alpha_k M^{\mathrm{T}} \vec{\bar{p}}_k$.
5. $\beta_k = \frac{\vec{\bar{r}}_{k+1}^{\mathrm{T}} \cdot \vec{r}_{k+1}}{\vec{\bar{r}}_k^{\mathrm{T}} \cdot \vec{r}_k}$.
6. $\vec{p}_{k+1} = \vec{r}_{k+1} + \beta_k \vec{p}_k$,
   $\vec{\bar{p}}_{k+1} = \vec{\bar{r}}_{k+1} + \beta_k \vec{\bar{p}}_k$.

Note that the transpose $^{\mathrm{T}}$ is used instead of the Hermitian conjugate †. The residual vector is $\vec{r}_k = \vec{b} - M\vec{x}_k$, and the calculation is terminated when it becomes sufficiently small.

The validity of this method can be checked as in the case of the CG method. Via the induction, we can show that $\vec{\bar{r}}_i^{\mathrm{T}} \vec{r}_j$ and $\vec{\bar{p}}_i^{\mathrm{T}} M \vec{p}_j$ are zero for $i \neq j$. From the latter, we can show that $\vec{p}_i$'s are linearly independent and that the solution $\vec{x}$ can be written as a linear combination of $\vec{p}_i$. Finally, the coefficients are α_i that can be checked as before.

E.2 Multi-mass CG Method

We introduce the "multi-mass" CG method [2], which is a crucial piece of the RHMC algorithm. For positive definite and Hermitian matrix A and a real number $\sigma > 0$, we define A_σ as

$$A_\sigma = A + \sigma \cdot \mathbf{1}. \tag{E.7}$$

Via the multi-mass CG method,

$$A_\sigma \vec{x} = \vec{b} \tag{E.8}$$

can be solved simultaneously for many different values of σ, with almost no extra cost. This σ is analogous to the mass in physics problems. Because many equations are solved simultaneously for multiple values of mass, the adjective "multi-mass" is used.

The key idea in the multi-mass CG method is that, from the recurrence relations

$$\begin{aligned}\vec{r}_{k+1} &= \vec{r}_k - \alpha_k A\vec{p}_k,\\ \vec{p}_{k+1} &= \vec{r}_{k+1} + \beta_k \vec{p}_k\end{aligned} \tag{E.9}$$

that appear in the usual CG method, the recurrence relations for A_σ, namely

$$\begin{aligned}\vec{r}^{\,\sigma}_{k+1} &= \vec{r}^{\,\sigma}_k - \alpha^\sigma_k A_\sigma \vec{p}^{\,\sigma}_k,\\ \vec{p}^{\,\sigma}_{k+1} &= \vec{r}^{\,\sigma}_{k+1} + \beta^\sigma_k \vec{p}^{\,\sigma}_k,\end{aligned} \tag{E.10}$$

can be obtained. Here, the coefficient with σ is defined as[4]

$$\begin{aligned}\vec{r}^{\,\sigma}_k &= \zeta^\sigma_k \vec{r}_k,\\ \alpha^\sigma_k &= \alpha_k \cdot \frac{\zeta^\sigma_{k+1}}{\zeta^\sigma_k},\\ \beta^\sigma_k &= \beta_k \cdot \left(\frac{\zeta^\sigma_{k+1}}{\zeta^\sigma_k}\right)^2,\\ \zeta^\sigma_{k+1} &= \frac{\zeta^\sigma_k \zeta^\sigma_{k-1}\alpha_{k-1}}{\alpha_{k-1}\zeta^\sigma_{k-1}(1+\alpha_k\sigma) + \alpha_k\beta_{k-1}(\zeta^\sigma_{k-1} - \zeta^\sigma_k)}.\end{aligned} \tag{E.12}$$

Furthermore, we need to choose the initial condition as

$$\begin{aligned}&\vec{x}_1 = \vec{x}^{\,\sigma}_1 = \vec{p}_0 = \vec{p}^{\,\sigma}_0 = \vec{0},\\ &\vec{r}_1 = \vec{r}^{\,\sigma}_1 = \vec{r}_0 = \vec{r}^{\,\sigma}_0 = \vec{p}_1 = \vec{p}^{\,\sigma}_1 = \vec{b},\\ &\zeta^\sigma_0 = \zeta^\sigma_1 = \alpha_0 = \alpha^\sigma_0 = \beta_0 = \beta^\sigma_0 = 1.\end{aligned} \tag{E.13}$$

Note that (E.12) does not necessarily hold if a different initial condition is used.

In summary, we modify the usual CG method as follows:

[4] From (E.9),

$$\vec{r}_{k+1} = \left(1 + \frac{\alpha_k\beta_{k-1}}{\alpha_{k-1}}\right)\vec{r}_k - \alpha_k A\vec{r}_k - \frac{\alpha_k\beta_{k-1}}{\alpha_{k-1}}\vec{r}_{k-1} \tag{E.11}$$

follows. A similar relation can be obtained from (E.10). By comparing the coefficients in these relations, we obtain (E.12).

Multi-mass CG method

1. Choose the initial condition following (E.13).
   Then $\vec{x}_k$'s are constructed as follows:
2. $\alpha_k = \frac{\vec{r}_k^\dagger \cdot \vec{r}_k}{\vec{p}_k^\dagger \cdot A\vec{p}_k}$.
3. Calculate ζ_{k+1}^σ and α_k^σ by using (E.12).
4. $\vec{x}_{k+1}^\sigma = \vec{x}_k^\sigma + \alpha_k^\sigma \vec{p}_k^\sigma$.
5. $\vec{r}_{k+1} = \vec{r}_k - \alpha_k A\vec{p}_k$.
6. $\beta_k = \frac{\vec{r}_{k+1}^\dagger \cdot \vec{r}_{k+1}}{\vec{r}_k^\dagger \cdot \vec{r}_k}$.
7. Calculate β_k^σ by using (E.12).
8. $\vec{p}_{k+1} = \vec{r}_{k+1} + \beta_k \vec{p}_k$,
   $\vec{p}_{k+1}^\sigma = \vec{r}_{k+1}^\sigma + \beta_k^\sigma \vec{p}_k^\sigma$.

$\vec{x}_k^\sigma$ obtained this way is the approximate solution with the residual vector $\vec{r}_k^\sigma$:

$$\vec{r}_k^\sigma = \vec{b} - A_\sigma \vec{x}_k^\sigma . \tag{E.14}$$

References

1. W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing* (Cambridge University Press, 2007)
2. B. Jegerlehner, Krylov space solvers for shifted linear systems (1996). arXiv preprint arXiv:hep-lat/9612014