# Preface

This book is intended for those who are interested in the use of Monte Carlo simulations in classical statistical mechanics. It would be suitable for use in a course on simulation methods or on statistical physics. It would also be a good choice for those who wish to teach themselves about Monte Carlo methods, or for experienced researchers who want to learn more about some of the sophisticated new simulation techniques which have appeared in the last decade or so.

The primary goal of the book is to explain how to perform Monte Carlo simulations efficiently. For many people, Monte Carlo simulation just means applying the Metropolis algorithm to the problem in hand. Although this famous algorithm is very easy to program, it is rarely the most efficient way to perform a simulation. The Metropolis algorithm is certainly important and we do discuss it in some detail (Chapter 3 is devoted to it), but we also show that for most problems a little work with a pencil and paper can usually turn up a better algorithm, in some cases thousands or millions of times faster. In recent years there has been quite a flurry of interesting new Monte Carlo algorithms described in the literature, many of which are specifically designed to accelerate the simulation of particular classes of problems in statistical physics. Amongst others, we describe cluster algorithms, multigrid methods, non-local algorithms for conserved-order-parameter models, entropic sampling, simulated tempering and continuous time Monte Carlo. The book is divided into parts covering equilibrium and non-equilibrium simulations, and throughout we give pointers to how the algorithms can be most efficiently implemented. At the end of the book we include a number of chapters on general implementation issues for Monte Carlo simulations. We also cover data analysis methods in some detail, including generic methods for estimating observable quantities, equilibration and correlation times, correlation functions, and standard errors, as well as a number of techniques which are specific to Monte Carlo simulation, such as the single and multiple histogram methods, finite-size scaling and the Monte Carlo renormalization group.

The *modus operandi* of this book is teaching by example. We have tried

to include as many as possible of the important Monte Carlo algorithms in use today, and each one we introduce in the context of a particular model or models. For example, we illustrate the Metropolis algorithm by applying it to the simulation of the Ising model. We have not assumed however that the reader is familiar with the models studied, and give a brief outline of the physics behind each one at the start of the corresponding chapter. All we assume is that the reader has a working knowledge of basic statistical mechanics and thermodynamics at the level typical of a student beginning graduate study in physics. Reasonable fluency in a computer programming language such as FORTRAN or C will certainly be necessary if you actually want to write a Monte Carlo program yourself, but the book can be understood without it. We have avoided giving examples of actual computer code in the body of the book. There are a couple of reasons for this. Doing so would require the reader to know a particular language, or to learn that language if he or she was not already familiar with it. Furthermore, we do not believe that inclusion of the code helps much with the understanding of an algorithm. It is better to get a clear idea of the principles behind an algorithm by working through the physics and mathematics involved than to try to learn by reading someone else's program. With a clear understanding of the principles, you should be able to write your own program with little difficulty. However, inspecting other people's programs can be useful in one respect: it is a good way to learn programming tricks and techniques for writing efficient code. For this reason we have included programs for some of the more common Monte Carlo algorithms in an appendix at the end of the book. The programs are written in C, which is fast replacing FORTRAN as the most commonly used language for scientific programming.

We have also included a number of problems at the end of each chapter for the reader to work through if he or she wishes. Some of these are purely analytic and can be done on paper. Others ask the reader to write a short computer program. Answers to the analytic problems are given at the end of the book. The ones which require you to write a computer program have many equally good solutions, so we have by and large resorted to giving hints rather than answers for these problems.

There are many people and organizations who have assisted us in the writing of this book. We would like to thank our editors Sönke Adlung, Donald Degenhardt and Julia Tompson at Oxford University Press for their help and patience. We are also grateful to a number of institutions who have offered us hospitality during the four-year process of preparing the manuscript, including Cornell University, Oxford University, the Institute for Advanced Study in Princeton, Forschungszentrum Jülich, the Santa Fe Institute and Utrecht University. Finally, we would like to thank the many colleagues and friends who have offered suggestions and encouragement, including James Binney, Geoffrey Chester, Eytan Domany, Peter Grass-

berger, Daniel Kandel, Jim Louck, Jon Machta, Nick Metropolis, Cris Moore, Richard Palmer, Gunter Schütz, Jim Sethna, Alan Sokal and Ben Widom. Responsibility for any mistakes which may lurk in the text rests of course with the authors. We would be very grateful to learn from our keen-eyed readers of any such problems.

June 1998

Mark Newman
*Santa Fe, New Mexico, USA*
Gerard Barkema
*Utrecht, The Netherlands*

# Contents

# Part I

# Equilibrium Monte Carlo simulations

# 1

# Introduction

This book is about the use of computers to solve problems in statistical physics. In particular, it is about **Monte Carlo methods**, which form the largest and most important class of numerical methods used for solving statistical physics problems. In this opening chapter of the book we look first at what we mean by statistical physics, giving a brief overview of the discipline we call **statistical mechanics**. Whole books have been written on statistical mechanics, and our synopsis takes only a few pages, so we must necessarily deal only with the very basics of the subject. We are assuming that these basics are actually already familiar to you, but writing them down here will give us a chance to bring back to mind some of the ideas that are most relevant to the study of Monte Carlo methods. In this chapter we also look at some of the difficulties associated with solving problems in statistical physics using a computer, and outline what Monte Carlo techniques are, and why they are useful. In the last section of the chapter, purely for fun, we give a brief synopsis of the history of computational physics and Monte Carlo methods.

## 1.1 Statistical mechanics

Statistical mechanics is primarily concerned with the calculation of properties of condensed matter systems. The crucial difficulty associated with these systems is that they are composed of very many parts, typically atoms or molecules. These parts are usually all the same or of a small number of different types and they often obey quite simple equations of motion so that the behaviour of the entire system can be expressed mathematically in a straightforward manner. But the sheer number of equations—just the magnitude of the problem—makes it impossible to solve the mathematics exactly. A standard example is that of a volume of gas in a container. One

litre of, say, oxygen at standard temperature and pressure consists of about $3 \times 10^{22}$ oxygen molecules, all moving around and colliding with one another and the walls of the container. One litre of air under the same conditions contains the same number of molecules, but they are now a mixture of oxygen, nitrogen, carbon dioxide and a few other things. The atmosphere of the Earth contains $4 \times 10^{21}$ litres of air, or about $1 \times 10^{44}$ molecules, all moving around and colliding with each other and the ground and trees and houses and people. These are large systems. It is not feasible to solve Hamilton's equations for these systems because there are simply too many equations, and yet when we look at the macroscopic properties of the gas, they are very well-behaved and predictable. Clearly, there is something special about the behaviour of the solutions of these many equations that "averages out" to give us a predictable behaviour for the entire system. For example, the pressure and temperature of the gas obey quite simple laws although both are measures of rather gross average properties of the gas. Statistical mechanics attempts to side-step the problem of solving the equations of motion and cut straight to the business of calculating these gross properties of large systems by treating them in a probabilistic fashion. Instead of looking for exact solutions, we deal with the probabilities of the system being in one state or another, having this value of the pressure or that—hence the name *statistical* mechanics. Such probabilistic statements turn out to be extremely useful, because we usually find that for large systems the range of behaviours of the system that are anything more than phenomenally unlikely is very small; all the reasonably probable behaviours fall into a narrow range, allowing us to state with extremely high confidence that the real system will display behaviour within that range. Let us look at how statistical mechanics treats these systems and demonstrates these conclusions.

The typical paradigm for the systems we will be studying in this book is one of a system governed by a Hamiltonian function $H$ which gives us the total energy of the system in any particular state. Most of the examples we will be looking at have discrete sets of states each with its own energy, ranging from the lowest, or ground state energy $E_0$ upwards, $E_1$, $E_2$, $E_3 \ldots$, possibly without limit. Statistical mechanics, and the Monte Carlo methods we will be introducing, are also applicable to systems with continuous energy spectra, and we will be giving some examples of such applications.

If our Hamiltonian system were all we had, life would be dull. Being a Hamiltonian system, energy would be conserved, which means that the system would stay in the same energy state all the time (or if there were a number of degenerate states with the same energy, maybe it would make transitions between those, but that's as far as it would get).[1] However,

[1] For a classical system which has a continuum of energy states there can be a continuous set of degenerate states through which the system passes, and an average over those states can sometimes give a good answer for certain properties of the system. Such sets of

there's another component to our paradigm, and that is the **thermal reservoir**. This is an external system which acts as a source and sink of heat, constantly exchanging energy with our Hamiltonian system in such a way as always to push the temperature of the system—defined as in classical thermodynamics—towards the temperature of the reservoir. In effect the reservoir is a weak perturbation on the Hamiltonian, which we ignore in our calculation of the energy levels of our system, but which pushes the system frequently from one energy level to another. We can incorporate the effects of the reservoir in our calculations by giving the system a **dynamics**, a rule whereby the system changes periodically from one state to another. The exact nature of the dynamics is dictated by the form of the perturbation that the reservoir produces in the Hamiltonian. We will discuss many different possible types of dynamics in the later chapters of this book. However, there are a number of general conclusions that we can reach without specifying the exact form of the dynamics, and we will examine these first.

Suppose our system is in a state $\mu$. Let us define $R(\mu \to \nu)\,\mathrm{d}t$ to be the probability that it is in state $\nu$ a time $\mathrm{d}t$ later. $R(\mu \to \nu)$ is the **transition rate** for the transition from $\mu$ to $\nu$. The transition rate is normally assumed to be time-independent and we will make that assumption here. We can define a transition rate like this for every possible state $\nu$ that the system can reach. These transition rates are usually all we know about the dynamics, which means that even if we know the state $\mu$ that the system starts off in, we need only wait a short interval of time and it could be in any one of a very large number of other possible states. This is where our probabilistic treatment of the problem comes in. We define a set of weights $w_\mu(t)$ which represent the probability that the system will be in state $\mu$ at time $t$. Statistical mechanics deals with these weights, and they represent our entire knowledge about the state of the system. We can write a **master equation** for the evolution of $w_\mu(t)$ in terms of the rates $R(\mu \to \nu)$ thus:[2]

$$\frac{\mathrm{d}w_\mu}{\mathrm{d}t} = \sum_\nu \bigl[w_\nu(t)R(\nu \to \mu) - w_\mu(t)R(\mu \to \nu)\bigr]. \tag{1.1}$$

The first term on the right-hand side of this equation represents the rate at which the system is undergoing transitions into state $\mu$; the second term is the rate at which it is undergoing transitions out of $\mu$ into other states. The probabilities $w_\mu(t)$ must also obey the sum rule

$$\sum_\mu w_\mu(t) = 1 \tag{1.2}$$

degenerate states are said to form a **microcanonical ensemble**. The more general case we consider here, in which there is a thermal reservoir causing the energy of the system to fluctuate, is known as a **canonical ensemble**.

[2]The master equation is really a set of equations, one for each state $\mu$, although people always call it *the* master equation, as if there were only one equation here.

for all $t$, since the system must always be in *some* state. The solution of Equation (1.1), subject to the constraint (1.2), tells us how the weights $w_\mu$ vary over time.

And how are the weights $w_\mu$ related to the macroscopic properties of the system which we want to know about? Well, if we are interested in some quantity $Q$, which takes the value $Q_\mu$ in state $\mu$, then we can define the **expectation** of $Q$ at time $t$ for our system as

$$\langle Q \rangle = \sum_\mu Q_\mu w_\mu(t). \tag{1.3}$$

Clearly this quantity contains important information about the real value of $Q$ that we might expect to measure in an experiment. For example, if our system is definitely in one state $\tau$ then $\langle Q \rangle$ will take the corresponding value $Q_\tau$. And if the system is equally likely to be in any of perhaps three states, and has zero probability of being in any other state, then $\langle Q \rangle$ is equal to the mean of the values of $Q$ in those three states, and so forth. However, the precise relation of $\langle Q \rangle$ to the observed value of $Q$ is perhaps not very clear. There are really two ways to look at it. The first, and more rigorous, is to imagine having a large number of copies of our system all interacting with their own thermal reservoirs and whizzing between one state and another all the time. $\langle Q \rangle$ is then a good estimate of the number we would get if we were to measure the instantaneous value of the quantity $Q$ in each of these systems and then take the mean of all of them. People who worry about the conceptual foundations of statistical mechanics like to take this "many systems" approach to defining the expectation of a quantity.[3] The trouble with it however is that it's not very much like what happens in a real experiment. In a real experiment we normally only have one system and we make all our measurements of $Q$ on that system, though we probably don't just make a single instantaneous measurement, but rather integrate our results over some period of time. There is another way of looking at the expectation value which is similar to this experimental picture, though it is less rigorous than the many systems approach. This is to envisage the expectation as a *time average* of the quantity $Q$. Imagine recording the value of $Q$ every second for a thousand seconds and taking the average of those one thousand values. This will correspond roughly to the quantity calculated in Equation (1.3) as long as the system passes through a representative selection of the states in the probability distribution $w_\mu$ in those thousand seconds. And if we make ten thousand measurements of $Q$ instead of one thousand,

[3] In fact the word *ensemble,* as in the "canonical ensemble" which was mentioned in a previous footnote, was originally introduced by Gibbs to describe an ensemble of *systems* like this, and not an ensemble of, say, molecules, or any other kind of ensemble. These days however, use of this word no longer implies that the writer is necessarily thinking of a many systems formulation of statistical mechanics.

or a million or more, we will get an increasingly accurate fit between our experimental average and the expectation $\langle Q \rangle$.

Why is this a less rigorous approach? The main problem is the question of what we mean by a "representative selection of the states". There is no guarantee that the system will pass through anything like a representative sample of the states of the system in our one thousand seconds. It could easily be that the system only hops from one state to another every ten thousand seconds, and so turns out to be in the same state for all of our one thousand measurements. Or maybe it changes state very rapidly, but because of the nature of the dynamics spends long periods of time in small portions of the state space. This can happen for example if the transition rates $R(\mu \to \nu)$ are only large for states of the system that differ in very small ways, so that the only way to make a large change in the state of the system is to go through very many small steps. This is a very common problem in a lot of the systems we will be looking at in this book. Another potential problem with the time average interpretation of (1.3) is that the weights $w_\mu(t)$, which are functions of time, may change considerably over the course of our measurements, making the expression invalid. This can be a genuine problem in both experiments and simulations of non-equilibrium systems, which are the topic of the second part of this book. For equilibrium systems, as discussed below, the weights are by definition not time-varying, so this problem does not arise.

Despite these problems however, this time-average interpretation of the expectation value of a quantity is the most widely used and most experimentally relevant interpretation, and it is the one that we will adopt in this book. The calculation of expectation values is one of the fundamental goals of statistical mechanics, and of Monte Carlo simulation in statistical physics, and much of our time will be concerned with it.

## 1.2 Equilibrium

Consider the master equation (1.1) again. If our system ever reaches a state in which the two terms on the right-hand side exactly cancel one another for all $\mu$, then the rates of change $\mathrm{d}w_\mu/\mathrm{d}t$ will all vanish and the weights will all take constant values for the rest of time. This is an **equilibrium** state. Since the master equation is first order with real parameters, and since the variables $w_\mu$ are constrained to lie between zero and one (which effectively prohibits exponentially growing solutions to the equations) we can see that all systems governed by these equations must come to equilibrium in the end. A large part of this book will be concerned with Monte Carlo techniques for simulating equilibrium systems and in this section we develop some of the important statistical mechanical concepts that apply to these systems.

The transition rates $R(\mu \to \nu)$ appearing in the master equation (1.1)

do not just take *any* values. They take particular values which arise out of the thermal nature of the interaction between the system and the thermal reservoir. In the later chapters of this book we will have to choose values for these rates when we simulate thermal systems in our Monte Carlo calculations, and it is crucial that we choose them so that they mimic the interactions with the thermal reservoir correctly. The important point is that we know *a priori* what the equilibrium values of the weights $w_\mu$ are for our system. We call these equilibrium values the **equilibrium occupation probabilities** and denote them by

$$p_\mu = \lim_{t\to\infty} w_\mu(t). \tag{1.4}$$

It was Gibbs (1902) who showed that for a system in thermal equilibrium with a reservoir at temperature $T$, the equilibrium occupation probabilities are

$$p_\mu = \frac{1}{Z}\,\mathrm{e}^{-E_\mu/kT}. \tag{1.5}$$

Here $E_\mu$ is the energy of state $\mu$ and $k$ is Boltzmann's constant, whose value is $1.38 \times 10^{-23}$ J K$^{-1}$. It is conventional to denote the quantity $(kT)^{-1}$ by the symbol $\beta$, and we will follow that convention in this book. $Z$ is a normalizing constant, whose value is given by

$$Z = \sum_\mu \mathrm{e}^{-E_\mu/kT} = \sum_\mu \mathrm{e}^{-\beta E_\mu}. \tag{1.6}$$

$Z$ is also known as the **partition function**, and it figures a lot more heavily in the mathematical development of statistical mechanics than a mere normalizing constant might be expected to. It turns out in fact that a knowledge of the variation of $Z$ with temperature and any other parameters affecting the system (like the volume of the box enclosing a sample of gas, or the magnetic field applied to a magnet) can tell us virtually everything we might want to know about the macroscopic behaviour of the system. The probability distribution (1.5) is known as the **Boltzmann distribution**, after Ludwig Boltzmann, one of the pioneers of statistical mechanics. For a discussion of the origins of the Boltzmann distribution and the arguments that lead to it, the reader is referred to the exposition by Walter Grandy in his excellent book *Foundations of Statistical Mechanics* (1987). In our treatment we will take Equation (1.5) as our starting point for further developments.

From Equations (1.3), (1.4) and (1.5) the expectation of a quantity $Q$ for a system in equilibrium is

$$\langle Q\rangle = \sum_\mu Q_\mu p_\mu = \frac{1}{Z}\sum_\mu Q_\mu\,\mathrm{e}^{-\beta E_\mu}. \tag{1.7}$$

For example, the expectation value of the energy $\langle E \rangle$, which is also the quantity we know from thermodynamics as the internal energy $U$, is given by

$$U = \frac{1}{Z} \sum_{\mu} E_{\mu} \, \mathrm{e}^{-\beta E_{\mu}}. \tag{1.8}$$

From Equation (1.6) we can see that this can also be written in terms of a derivative of the partition function:

$$U = -\frac{1}{Z} \frac{\partial Z}{\partial \beta} = -\frac{\partial \log Z}{\partial \beta}. \tag{1.9}$$

The specific heat is given by the derivative of the internal energy:

$$C = \frac{\partial U}{\partial T} = -k\beta^2 \frac{\partial U}{\partial \beta} = -k\beta^2 \frac{\partial^2 \log Z}{\partial \beta^2}. \tag{1.10}$$

However, from thermodynamics we know that the specific heat is also related to the entropy:

$$C = T \frac{\partial S}{\partial T} = -\beta \frac{\partial S}{\partial \beta}, \tag{1.11}$$

and, equating these two expressions for $C$ and integrating with respect to $\beta$, we find the following expression for the entropy:

$$S = -k\beta \frac{\partial \log Z}{\partial \beta} + k \log Z. \tag{1.12}$$

(There is in theory an integration constant in this equation, but it is set to zero under the convention known as the third law of thermodynamics, which fixes the arbitrary origin of entropy by saying that the entropy of a system should tend to zero as the temperature does.) We can also write an expression for the (Helmholtz) free energy $F$ of the system, using Equations (1.9) and (1.12):

$$F = U - TS = -kT \log Z. \tag{1.13}$$

We have thus shown how $U$, $F$, $C$ and $S$ can all be calculated directly from the partition function $Z$. The last equation also tells us how we can deal with other parameters affecting the system. In classical thermodynamics, parameters and constraints and fields interacting with the system each have conjugate variables which represent the response of the system to the perturbation of the corresponding parameter. For example, the response of a gas system in a box to a change in the confining volume is a change in the pressure of the gas. The pressure $p$ is the conjugate variable to the parameter $V$. Similarly, the magnetization $M$ of a magnet changes in response

to the applied magnetic field $B$; $M$ and $B$ are conjugate variables. Thermodynamics tells us that we can calculate the values of conjugate variables from derivatives of the free energy:

$$p = -\frac{\partial F}{\partial V}, \tag{1.14}$$

$$M = \frac{\partial F}{\partial B}. \tag{1.15}$$

Thus, if we can calculate the free energy using Equation (1.13), then we can calculate the effects of parameter variations too.

In performing Monte Carlo calculations of the properties of equilibrium systems, it is sometimes appropriate to calculate the partition function and then evaluate other quantities from it. More often it is better to calculate the quantities of interest directly, but many times in considering the theory behind our simulations we will return to the idea of the partition function, because in principle the entire range of thermodynamic properties of a system can be deduced from this function, and any numerical method that can make a good estimate of the partition function is at heart a sound method.

### 1.2.1 Fluctuations, correlations and responses

Statistical mechanics can tell us about other properties of a system apart from the macroscopic ones that classical equilibrium thermodynamics deals with such as entropy and pressure. One of the most physically interesting classes of properties is **fluctuations** in observable quantities. We described in the first part of Section 1.1 how the calculation of an expectation could be regarded as a time average over many measurements of the same property of a single system. In addition to calculating the mean value of these many measurements, it is often useful also to calculate their standard deviation, which gives us a measure of the variation over time of the quantity we are looking at, and so tells us quantitatively how much of an approximation we are making by giving just the one mean value for the expectation. To take an example, let us consider the internal energy again. The mean square deviation of individual, instantaneous measurements of the energy away from the mean value $U = \langle E \rangle$ is

$$\langle (E - \langle E \rangle)^2 \rangle = \langle E^2 \rangle - \langle E \rangle^2. \tag{1.16}$$

We can calculate $\langle E^2 \rangle$ from derivatives of the partition function in a way similar to our calculation of $\langle E \rangle$:

$$\langle E^2 \rangle = \frac{1}{Z} \sum_{\mu} E_{\mu}^2 \mathrm{e}^{-\beta E_{\mu}} = \frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2}. \tag{1.17}$$

So

$$\langle E^2\rangle - \langle E\rangle^2 = \frac{1}{Z}\frac{\partial^2 Z}{\partial\beta^2} - \left[\frac{1}{Z}\frac{\partial Z}{\partial\beta}\right]^2 = \frac{\partial^2 \log Z}{\partial\beta^2}. \tag{1.18}$$

Using Equation (1.10) to eliminate the second derivative, we can also write this as

$$\langle E^2\rangle - \langle E\rangle^2 = \frac{C}{k\beta^2}. \tag{1.19}$$

And the standard deviation of $E$, the RMS fluctuation in the internal energy, is just the square root of this expression.

This result is interesting for a number of reasons. First, it gives us the magnitude of the fluctuations in terms of the specific heat $C$ or alternatively in terms of $\log Z = -\beta F$. In other words we can calculate the fluctuations entirely from quantities that are available within classical thermodynamics. However, this result could never have been derived within the framework of thermodynamics, since it depends on microscopic details that thermodynamics has no access to. Second, let us look at what sort of numbers we get out for the size of the energy fluctuations of a typical system. Let us go back to our litre of gas in a box. A typical specific heat for such a system is $1\ \mathrm{J\,K^{-1}}$ at room temperature and atmospheric pressure, giving RMS energy fluctuations of about $10^{-18}$ J. The internal energy itself on the other hand will be around $10^2$ J, so the fluctuations are only about one part in $10^{20}$. This lends some credence to our earlier contention that statistical treatments can often give a very accurate estimate of the expected behaviour of a system. We see that in the case of the internal energy at least, the variation of the actual value of $U$ around the expectation value $\langle E\rangle$ is tiny by comparison with the kind of energies we are considering for the whole system, and probably not within the resolution of our measuring equipment. So quoting the expectation value gives a very good guide to what we should expect to see in an experiment. Furthermore, note that, since the specific heat $C$ is an extensive quantity, the RMS energy fluctuations, which are the square root of Equation (1.19), scale like $\sqrt{V}$ with the volume $V$ of the system. The internal energy itself on the other hand scales like $V$, so that the relative size of the fluctuations compared to the internal energy decreases as $1/\sqrt{V}$ as the system becomes large. In the limit of a very large system, therefore, we can ignore the fluctuations altogether. For this reason, the limit of a large system is called the **thermodynamic limit**. Most of the questions we would like to answer about condensed matter systems are questions about behaviour in the thermodynamic limit. Unfortunately, in Monte Carlo simulations it is often not feasible to simulate a system large enough that its behaviour is a good approximation to a large system. Much of the effort we put into designing algorithms will be aimed at making them efficient enough that we can simulate the largest systems possible in the available computer time, in

the hope of getting results which are at least a reasonable approximation to the thermodynamic limit.

What about fluctuations in other thermodynamic variables? As we discussed in Section 1.2, each parameter of the system that we fix, such as a volume or an external field, has a conjugate variable, such as a pressure or a magnetization, which is given as a derivative of the free energy by an equation such as (1.14) or (1.15). Derivatives of this general form are produced by terms in the Hamiltonian of the form $-XY$, where $Y$ is a "field" whose value we fix, and $X$ is the conjugate variable to which it couples. For example, the effect of a magnetic field on a magnet can be accounted for by a magnetic energy term in the Hamiltonian of the form $-MB$, where $M$ is the magnetization of the system, and $B$ is the applied magnetic field. We can write the expectation value of $X$ in the form of Equations (1.14) and (1.15) thus:

$$\langle X\rangle = \frac{1}{\beta Z}\sum_\mu X_\mu \mathrm{e}^{-\beta E_\mu} = \frac{1}{\beta Z}\frac{\partial}{\partial Y}\sum_\mu \mathrm{e}^{-\beta E_\mu}, \tag{1.20}$$

since $E_\mu$ now contains the term $-X_\mu Y$ which the derivative acts on. Here $X_\mu$ is the value of the quantity $X$ in the state $\mu$. We can then write this in terms of the free energy thus:

$$\langle X\rangle = \frac{1}{\beta}\frac{\partial \log Z}{\partial Y} = -\frac{\partial F}{\partial Y}. \tag{1.21}$$

This is a useful technique for calculating the thermal average of a quantity, even if no appropriate field coupling to that quantity appears in the Hamiltonian. We can simply make up a fictitious field which couples to our quantity in the appropriate way—just add a term to the Hamiltonian anyway to allow us to calculate the expectation of the quantity we are interested in—and then set the field to zero after performing the derivative, making the fictitious term vanish from the Hamiltonian again. This is a very common trick in statistical mechanics.

Another derivative of $\log Z$ with respect to $Y$ produces another factor of $X_\mu$ in the sum over states, and we find

$$-\frac{1}{\beta}\frac{\partial^2 F}{\partial Y^2} = \frac{1}{\beta}\frac{\partial\langle X\rangle}{\partial Y} = \langle X^2\rangle - \langle X\rangle^2, \tag{1.22}$$

which we recognize as the mean square fluctuation in the variable $X$. Thus we can find the fluctuations in all sorts of quantities from second derivatives of the free energy with respect to the appropriate fields, just as we can find the energy fluctuations from the second derivative with respect to $\beta$. The derivative $\partial\langle X\rangle/\partial Y$, which measures the strength of the response of $X$ to changes in $Y$ is called the **susceptibility** of $X$ to $Y$, and is usually denoted by $\chi$:

$$\chi \equiv \frac{\partial\langle X\rangle}{\partial Y}. \tag{1.23}$$

Thus the fluctuations in a variable are proportional to the susceptibility of that variable to its conjugate field. This fact is known as the **linear response theorem** and it gives us a way to calculate susceptibilities within Monte Carlo calculations by measuring the size of the fluctuations of a variable.

Extending the idea of the susceptibility, and at the same time moving a step further from the realm of classical thermodynamics, we can also consider what happens when we change the value of a parameter or field at one particular position in our system and ask what effect that has on the conjugate variable at other positions. To study this question we will consider for the moment a system on a lattice. Similar developments are possible for continuous systems like gases, but most of the examples considered in this book are systems which fall on lattices, so it will be of more use to us to go through this for a lattice system here. The interested reader might like to develop the corresponding theory for a continuous system as an exercise.

Let us then suppose that we now have a field which is spatially varying and takes the value $Y_i$ on the $i^{\text{th}}$ site of the lattice. The conjugate variables to this field[4] are denoted $x_i$, and the two are linked via a term in the Hamiltonian $-\sum_i x_i Y_i$. Clearly if we set $Y_i = Y$ and $x_i = X/N$ for all sites $i$, where $N$ is the total number of sites on the lattice, then this becomes equal once more to the homogeneous situation we considered above. Now in a direct parallel with Equation (1.20) we can write the average value of $x_i$ as

$$\langle x_i \rangle = \frac{1}{Z} \sum_\mu x_i^\mu \mathrm{e}^{-\beta E_\mu} = \frac{1}{\beta} \frac{\partial \log Z}{\partial Y_i}, \tag{1.24}$$

where $x_i^\mu$ is the value of $x_i$ in state $\mu$. Then we can define a generalized susceptibility $\chi_{ij}$ which is a measure of the response of $\langle x_i \rangle$ to a variation of the field $Y_j$ at a different lattice site:

$$\chi_{ij} = \frac{\partial \langle x_i \rangle}{\partial Y_j} = \frac{1}{\beta} \frac{\partial^2 \log Z}{\partial Y_i \partial Y_j}. \tag{1.25}$$

Again the susceptibility is a second derivative of the free energy. If we make the substitution $Z = \sum_\mu \mathrm{e}^{-\beta E_\mu}$ again (Equation (1.6)), we see that this is also equal to

$$\begin{aligned} \chi_{ij} &= \frac{\beta}{Z} \sum_\mu x_i^\mu x_j^\mu \mathrm{e}^{-\beta E_\mu} - \beta \left[ \frac{1}{Z} \sum_\mu x_i^\mu \mathrm{e}^{-\beta E_\mu} \right] \left[ \frac{1}{Z} \sum_\nu x_j^\nu \mathrm{e}^{-\beta E_\nu} \right] \\ &= \beta(\langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle) = \beta G_c^{(2)}(i,j). \end{aligned} \tag{1.26}$$

[4] We use lower-case $x_i$ to denote an intensive variable. $X$ by contrast was extensive, i.e., its value scales with the size of the system. We will use this convention to distinguish intensive and extensive variables throughout much of this book.

The quantity $G_c^{(2)}(i,j)$ is called the **two-point connected correlation function** of $x$ between sites $i$ and $j$, or just the connected correlation, for short. The superscript (2) is to distinguish this function from higher order correlation functions, which are discussed below. As its name suggests, this function is a measure of the correlation between the values of the variable $x$ on the two sites; it takes a positive value if the values of $x$ on those two sites fluctuate in the same direction together, and a negative one if they fluctuate in opposite directions. If their fluctuations are completely unrelated, then its value will be zero. To see why it behaves this way consider first the simpler **disconnected correlation function** $G^{(2)}(i,j)$ which is defined to be

$$G^{(2)}(i,j) \equiv \langle x_i x_j \rangle. \tag{1.27}$$

If the variables $x_i$ and $x_j$ are fluctuating roughly together, around zero, both becoming positive at once and then both becoming negative, at least most of the time, then all or most of the values of the product $x_i x_j$ that we average will be positive, and this function will take a positive value. Conversely, if they fluctuate in opposite directions, then it will take a negative value. If they sometimes fluctuate in the same direction as one another and sometimes in the opposite direction, then the values of $x_i x_j$ will take a mixture of positive and negative values, and the correlation function will average out close to zero. This function therefore has pretty much the properties we desire of a correlation function, and it can tell us a lot of useful things about the behaviour of our system. However, it is not perfect, because we must also consider what happens if we apply our field $Y$ to the system. This can have the effect that the mean value of $x$ at a site $\langle x_i \rangle$ can be non-zero. The same thing can happen even in the absence of an external field if our system undergoes a phase transition to a **spontaneously symmetry broken state** where a variable such as $x$ spontaneously develops a non-zero expectation value. (The Ising model of Section 1.2.2, for instance, does this.) In cases like these, the disconnected correlation function above can have a large positive value simply because the values of the variables $x_i$ and $x_j$ are always either both positive or both negative, even though this has nothing to do with them being correlated to one another. The *fluctuations* of $x_i$ and $x_j$ can be completely unrelated and still the disconnected correlation function takes a non-zero value. To obviate this problem we define the connected correlation function as above:

$$\begin{aligned} G_c^{(2)}(i,j) &\equiv \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle \\ &= \big\langle (x_i - \langle x_i \rangle) \times (x_j - \langle x_j \rangle) \big\rangle. \end{aligned} \tag{1.28}$$

When the expectations $\langle x_i \rangle$ and $\langle x_j \rangle$ are zero and $x_i$ and $x_j$ are just fluctuating around zero, this function is exactly equal to the disconnected correlation function. But when the expectations are non-zero, the connected correlation

function correctly averages only the fluctuations about those expectations—the term we subtract exactly takes care of any trivial contribution arising because of external fields or spontaneous symmetry breaking. If such trivial contributions are the only reason why $G^{(2)}$ is non-zero then $G_c^{(2)}$ will be zero, which is what we would like. If it is not zero, then we have a genuine correlation between the fluctuations of $x_i$ and $x_j$.

Although they are not often used in the sorts of systems we will be studying in this book and we will not have call to calculate their values in any of the calculations we will describe here, it is worth mentioning, in case you ever need to use them, that there are also higher-order connected correlation functions, defined by generalizing Equation (1.25) like this:

$$\begin{aligned} G_c^{(3)}(i,j,k) &= \frac{1}{\beta^3}\frac{\partial^3 \log Z}{\partial Y_i \partial Y_j \partial Y_k}, \\ G_c^{(4)}(i,j,k,l) &= \frac{1}{\beta^4}\frac{\partial^4 \log Z}{\partial Y_i \partial Y_j \partial Y_k \partial Y_l}, \end{aligned} \tag{1.29}$$

and so on. These are measures of the correlation between simultaneous fluctuations on three and four sites respectively. For a more detailed discussion of these correlation functions and other related ones, see for example Binney *et al.* (1992).

### 1.2.2 An example: the Ising model

To try to make all of this a bit more concrete, we now introduce a particular model which we can try these concepts out on. That model is the Ising model, which is certainly the most thoroughly researched model in the whole of statistical physics. Without doubt more person-hours have been spent investigating the properties of this model than any other, and although an exact solution of its properties in three dimensions still eludes us, despite many valiant and increasingly sophisticated attempts, a great deal about it is known from computer simulations, and also from approximate methods such as series expansions and $\epsilon$-expansions. We will spend three whole chapters of this book (Chapters 3, 4 and 10) discussing Monte Carlo techniques for studying the model's equilibrium and non-equilibrium properties. Here we will just introduce it briefly and avoid getting too deeply into the discussion of its properties.

The Ising model is a model of a magnet. The essential premise behind it, and behind many magnetic models, is that the magnetism of a bulk material is made up of the combined magnetic dipole moments of many atomic spins within the material. The model postulates a lattice (which can be of any geometry we choose—the simple cubic lattice in three dimensions is a common choice) with a magnetic dipole or spin on each site. In the Ising model

these spins assume the simplest form possible, which is not particularly realistic, of scalar variables $s_i$ which can take only two values $\pm 1$, representing up-pointing or down-pointing dipoles of unit magnitude. In a real magnetic material the spins interact, for example through exchange interactions or RKKY interactions (see, for instance, Ashcroft and Mermin 1976), and the Ising model mimics this by including terms in the Hamiltonian proportional to products $s_i s_j$ of the spins. In the simplest case, the interactions are all of the same strength, denoted by $J$ which has the dimensions of an energy, and are only between spins on sites which are nearest neighbours on the lattice. We can also introduce an external magnetic field $B$ coupling to the spins. The Hamiltonian then takes the form

$$H = -J \sum_{\langle ij \rangle} s_i s_j - B \sum_i s_i, \tag{1.30}$$

where the notation $\langle ij \rangle$ indicates that the sites $i$ and $j$ appearing in the sum are nearest neighbours.[5] The minus signs here are conventional. They merely dictate the choice of sign for the interaction parameter $J$ and the external field $B$. With the signs as they are here, a positive value of $J$ makes the spins want to line up with one another—a ferromagnetic model as opposed to an anti-ferromagnetic one which is what we get if $J$ is negative—and the spins also want to line up in the same direction as the external field—they want to be positive if $B > 0$ and negative if $B < 0$.

The states of the Ising system are the different sets of values that the spins can take. Since each spin can take two values, there are a total of $2^N$ states for a lattice with $N$ spins on it. The partition function of the model is the sum

$$Z = \sum_{s_1=\pm 1} \sum_{s_2=\pm 1} \cdots \sum_{s_N=\pm 1} \exp\Bigl[\beta J \sum_{\langle ij \rangle} s_i s_j + \beta B \sum_i s_i\Bigr]. \tag{1.31}$$

To save the eyes, we'll write this in the shorter notation

$$Z = \sum_{\{s_i\}} \mathrm{e}^{-\beta H}. \tag{1.32}$$

If we can perform this sum, either analytically or using a computer, then we can apply all the results of the previous sections to find the internal energy, the entropy, the free energy, the specific heat, and so forth. We can also calculate the mean magnetization $\langle M \rangle$ of the model from the partition

[5] This notation is confusingly similar to the notation for a thermal average, but unfortunately both are sufficiently standard that we feel compelled to use them here. In context it is almost always possible to tell them apart because one involves site labels and the other involves physical variables appearing in the model.

function using Equation (1.15), although as we will see it is usually simpler to evaluate $\langle M \rangle$ directly from an average over states:

$$\langle M \rangle = \Big\langle \sum_i s_i \Big\rangle. \tag{1.33}$$

Often, in fact, we are more interested in the mean magnetization per spin $\langle m \rangle$, which is just

$$\langle m \rangle = \frac{1}{N} \Big\langle \sum_i s_i \Big\rangle. \tag{1.34}$$

(In the later chapters of this book, we frequently use the letter $m$ alone to denote the average magnetization per spin, and omit the brackets $\langle \ldots \rangle$ around it indicating the average. This is also the common practice of many other authors. In almost all cases it is clear from the context when an average over states is to be understood.)

We can calculate fluctuations in the magnetization or the internal energy by calculating derivatives of the partition function. Or, as we mentioned in Section 1.2.1, if we have some way of calculating the size of the fluctuations in the magnetization, we can use those to evaluate the **magnetic susceptibility**

$$\frac{\partial \langle M \rangle}{\partial B} = \beta(\langle M^2 \rangle - \langle M \rangle^2). \tag{1.35}$$

(See Equation (1.22).) Again, it is actually more common to calculate the magnetic susceptibility per spin:

$$\chi = \frac{\beta}{N}(\langle M^2 \rangle - \langle M \rangle^2) = \beta N(\langle m^2 \rangle - \langle m \rangle^2). \tag{1.36}$$

(Note the leading factor of $N$ here, which is easily overlooked when calculating $\chi$ from Monte Carlo data.) Similarly we can calculate the specific heat per spin $c$ from the energy fluctuations thus:

$$c = \frac{k\beta^2}{N}(\langle E^2 \rangle - \langle E \rangle^2). \tag{1.37}$$

(See Equation (1.19).)

We can also introduce a spatially varying magnetic field into the Hamiltonian thus:

$$H = -J \sum_{\langle ij \rangle} s_i s_j - \sum_i B_i s_i. \tag{1.38}$$

This gives us a different mean magnetization on each site:

$$\langle m_i \rangle = \langle s_i \rangle = \frac{1}{\beta} \frac{\partial \log Z}{\partial B_i}, \tag{1.39}$$

and allows us to calculate the connected correlation function

$$G_c^{(2)}(i,j) = \frac{1}{\beta^2}\frac{\partial^2 \log Z}{\partial B_i \partial B_j}. \tag{1.40}$$

When we look at the equilibrium simulation of the Ising model in Chapters 3 and 4, all of these will be quantities of interest, and relations like these between them give us useful ways of extracting good results from our numerical data.

## 1.3 Numerical methods

While the formal developments of statistical mechanics are in many ways very elegant, the actual process of calculating the properties of a particular model is almost always messy and taxing. If we consider calculating the partition function $Z$, from which, as we have shown, a large number of interesting properties of a system can be deduced, we see that we are going to have to perform a sum over a potentially very large number of states. Indeed, if we are interested in the thermodynamic limit, the sum is over an infinite number of states, and performing such sums is a notoriously difficult exercise. It has been accomplished exactly for a number of simple models with discrete energy states, most famously the Ising model in two dimensions (Onsager 1944). This and other exact solutions are discussed at some length by Baxter (1982). However, for the majority of models of interest today, it has not yet proved possible to find an exact analytic expression for the partition function, or for any other equivalent thermodynamic quantity. In the absence of such exact solutions a number of approximate techniques have been developed including series expansions, field theoretical methods and computational methods. The focus of this book is on the last of these, the computational methods.

The most straightforward computational method for solving problems in statistical physics is to take the model we are interested in and put it on a lattice of finite size, so that the partition function becomes a sum with a finite number of terms. (Or in the case of a model with a continuous energy spectrum it becomes an integral of finite dimension.) Then we can employ our computer to evaluate that sum (or integral) numerically, by simply evaluating each term in turn and adding them up. Let's see what happens when we apply this technique to the Ising model of Section 1.2.2.

If we were really interested in tackling an unsolved problem, we might look at the Ising model in three dimensions, whose exact properties have not yet been found by any method. However, rather than jump in at the deep end, let's first look at the two-dimensional case. For a system of a given linear dimension, this model will have fewer energy states than the three-dimensional one, making the sum over states simpler and quicker to perform,

and the model has the added pedagogical advantage that its behaviour has been solved exactly, so we can compare our numerical calculations with the exact solution. Let's take a smallish system to start with, of 25 spins on a square lattice in a $5 \times 5$ arrangement. By convention we apply periodic boundary conditions, so that there are interactions between spins on the border of the array and the opposing spins on the other side. We will also set the external magnetic field $B$ to zero, to make things simpler still.

With each spin taking two possible states, represented by $\pm 1$, our 25 spin system has a total of $2^{25} = 33\,554\,432$ possible states. However, we can save ourselves from summing over half of these, because the system has up/down symmetry, which means that for every state there is another one in which every spin is simply flipped upside down, which has exactly the same energy in zero magnetic field. So we can simplify the calculation of the partition function by just taking one out of every pair of such states, for a total of 16 777 216 states, and summing up the corresponding terms in the partition function, Equation (1.6), and then doubling the sum.[6]

In Figure 1.1 we show the mean magnetization per spin and the specific heat per spin for this $5 \times 5$ system, calculated from Equations (1.10) and (1.34). On the same axes we show the exact solutions for these quantities on an infinite lattice, as calculated by Onsager. The differences between the two are clear, and this is precisely the difference between our small finite-sized system and the infinite thermodynamic-limit system which we discussed in Section 1.2.1. Notice in particular that the exact solution has a non-analytic point at about $kT = 2.3J$ which is not reproduced even moderately accurately by our small numerical calculation. This point is the so-called "critical temperature" at which the length-scale $\xi$ of the fluctuations in the magnetization, also called the "correlation length", diverges. (This point is discussed in more detail in Section 3.7.1.) Because of this divergence of the length-scale, it is never possible to get good results for the behaviour of the system at the critical temperature out of any calculation performed on a finite lattice—the lattice is never large enough to include all of the important physics of the critical point. Does this mean that calculations on finite lattices are useless? No, it certainly does not. To start with, at temperatures well away from the critical point the problems are much less severe, and the numerical calculation and the exact solution agree better,

[6] If we were really serious about this, we could save ourselves further time by making use of other symmetries too. For example the square system we are investigating here also has a reflection symmetry and a four-fold rotational symmetry (the symmetry group is $C_4$), meaning that the states actually group into sets of 16 states (including the up–down symmetry pairs), all of which have the same energy. This would reduce the number of terms we have to evaluate to 2 105 872. (The reader may like to ponder why this number is not exactly $2^{25}/16$, as one might expect.) However, such efforts are not really worthwhile, since, as we will see very shortly, this direct evaluation of the partition function is not a promising method for solving models.

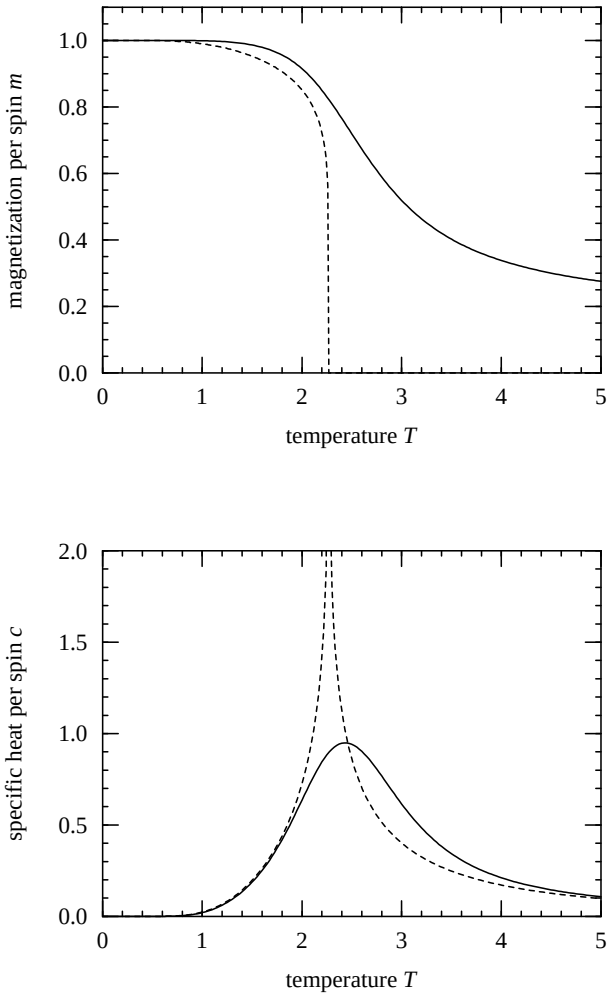


FIGURE 1.1 Top: the mean magnetization per spin $m$ of a $5 \times 5$ Ising model on a square lattice in two dimensions (solid line) and the same quantity on an infinitely big square lattice (dashed line). Bottom: the specific heat per spin $c$ for the same two cases.

as we can see in the figure. If we are interested in physics in this regime, then a calculation on a small lattice may well suffice. Second, the technique of "finite size scaling", which is discussed in Section 8.3, allows us to extrapolate results for finite lattices to the limit of infinite system size, and extract good results for the behaviour in the thermodynamic limit. Another technique, that of "Monte Carlo renormalization", discussed in Section 8.4, provides us with a cunning indirect way of calculating some of the features of the critical regime from just the short length-scale phenomena that we get out of a calculation on a small lattice, even though the direct cause of the features that we are interested in is the large length-scale fluctuations that we mentioned.

However, although these techniques can give answers for the critical properties of the system, the accuracy of the answers they give still depends on the size of the system we perform the calculation on, with the answers improving steadily as the system size grows. Therefore it is in our interest to study the largest system we can. However, the calculation which appears as the solid lines in Figure 1.1 took eight hours on a moderately powerful computer. The bulk of this time is spent running through the terms in the sum (1.6). For a system of $N$ spins there are $2^N$ terms, of which, as we mentioned, we only need actually calculate a half, or $2^{N-1}$. This number increases exponentially with the size of the lattice, so we can expect the time taken by the program to increase very rapidly with lattice size. The next size of square lattice up from the present one would be $6 \times 6$ or $N = 36$, which should take about $2^{36-1}/2^{25-1} = 2048$ times as long as the previous calculation, or about two years. Clearly this is an unacceptably long time to wait for the answer to this problem. If we are interested in results for any system larger than $5 \times 5$, we are going to have to find other ways of getting them.

### 1.3.1 Monte Carlo simulation

There is essentially only one known numerical method for calculating the partition function of a model such as the Ising model on a large lattice, and that method is Monte Carlo simulation, which is the subject of this book. The basic idea behind Monte Carlo simulation is to simulate the random thermal fluctuation of the system from state to state over the course of an experiment. In Section 1.1 we pointed out that for our purposes it is most convenient to regard the calculation of an expectation value as a time average over the states that a system passes through. In a Monte Carlo calculation we directly simulate this process, creating a model system on our computer and making it pass through a variety of states in such a way that the probability of it being in any particular state $\mu$ at a given time $t$ is equal to the weight $w_\mu(t)$ which that state would have in a real system.

In order to achieve this we have to choose a dynamics for our simulation—a rule for changing from one state to another during the simulation—which results in each state appearing with exactly the probability appropriate to it. In the next chapter we will discuss at length a number of strategies for doing this, but the essential idea is that we try to simulate the physical processes that give rise to the master equation, Equation (1.1). We choose a set of rates $R(\mu \to \nu)$ for transitions from one state to another, and we choose them in such a way that the equilibrium solution to the corresponding master equation is precisely the Boltzmann distribution (1.5). Then we use these rates to choose the states which our simulated system passes through during the course of a simulation, and from these states we make estimates of whatever observable quantities we are interested in.

The advantage of this technique is that we need only sample quite a small fraction of the states of the system in order to get accurate estimates of physical quantities. For example, we do not need to include every state of the system in order to get a decent value for the partition function, as we would if we were to evaluate it directly from Equation (1.6). The principal disadvantage of the technique is that there are statistical errors in the calculation due to this same fact that we don't include every state in our calculation, but only some small fraction of the states. In particular this means that there will be statistical noise in the partition function. Taking the derivative of a noisy function is always problematic, so that calculating expectation values from derivatives of the partition function as discussed in Section 1.2 is usually not a good way to proceed. Instead it is normally better in Monte Carlo simulations to calculate as many expectations as we can directly, using equations such as (1.34). We can also make use of relations such as (1.36) to calculate quantities like susceptibilities without having to evaluate a derivative.

In the next chapter we will consider the theory of Monte Carlo simulation in equilibrium thermal systems, and the rest of the first part of the book will deal with the design of algorithms to investigate these systems. In the second part of the book we look at algorithms for non-equilibrium systems.

## 1.4 A brief history of the Monte Carlo method

In this section we outline the important historical developments in the evolution of the Monte Carlo method. This section is just for fun; feel free to skip over it to the next chapter if you're not interested.

The idea of Monte Carlo calculation is a lot older than the computer. The name "Monte Carlo" is relatively recent—it was coined by Nicolas Metropolis in 1949—but under the older name of "statistical sampling" the method has a history stretching back well into the last century, when numerical calculations were performed by hand using pencil and paper and perhaps

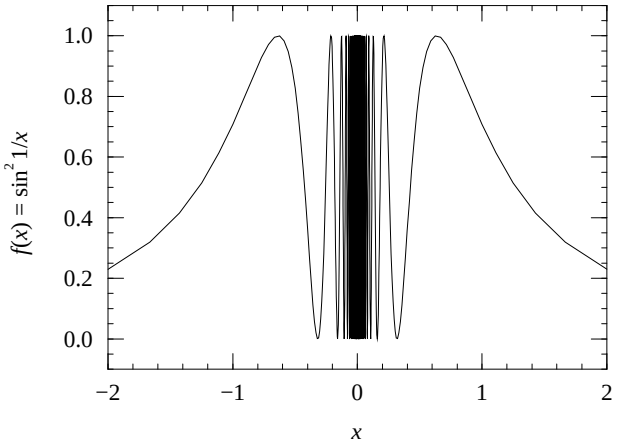


FIGURE 1.2 The pathological function $f(x) \equiv \sin^2 \frac{1}{x}$, whose integral with respect to $x$, though hard to evaluate analytically, can be evaluated in a straightforward manner using the Monte Carlo integration technique described in the text.

a slide-rule. As first envisaged, Monte Carlo was not a method for solving problems in physics, but a method for estimating integrals which could not be performed by other means. Integrals over poorly-behaved functions and integrals in high-dimensional spaces are two areas in which the method has traditionally proved profitable, and indeed it is still an important technique for problems of these types. To give an example, consider the function

$$f(x) \equiv \sin^2 \frac{1}{x} \tag{1.41}$$

which is pictured in Figure 1.2. The values of this function lie entirely between zero and one, but it is increasingly rapidly varying in the neighbourhood of $x = 0$. Clearly the integral

$$I(x) \equiv \int_0^x f(x')\,\mathrm{d}x' \tag{1.42}$$

which is the area under this curve between 0 and $x$, takes a finite value somewhere in the range $0 < I(x) < x$, but it is not simple to calculate this value exactly because of the pathologies of the function near the origin. However, we can make an estimate of it by the following method. If we choose a random real number $h$, uniformly distributed between zero and $x$, and another $v$ between zero and one and plot on Figure 1.2 the point for which these are the horizontal and vertical coordinates, the probability that

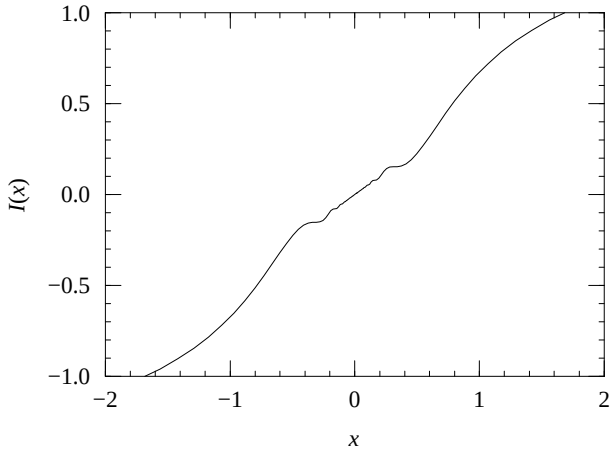


FIGURE 1.3 The function $I(x)$, calculated by Monte Carlo integration as described in the text.

this point will be below the line of $f(x)$ is just $I(x)/x$. It is easy to determine whether the point is in fact below the line: it is below it if $h < f(v)$. Thus if we simply pick a large number $N$ of these random points and count up the number $M$ which fall below the line, we can estimate $I(x)$ from

$$I(x) = \lim_{N\to\infty} \frac{Mx}{N}. \tag{1.43}$$

You can get an answer accurate to one figure by taking a thousand points, which would be about the limit of what one could have reasonably done in the days before computers. Nowadays, even a cheap desktop computer can comfortably run through a million points in a few seconds, giving an answer accurate to about three figures. In Figure 1.3 we have plotted the results of such a calculation for a range of values of $x$. The errors in this calculation are smaller than the width of the line in the figure.[7]

A famous early example of this type of calculation is the experiment known as "Buffon's needle" (Dörrie 1965), in which the mathematical constant $\pi$ is determined by repeatedly dropping a needle onto a sheet of paper ruled with evenly spaced lines. The experiment is named after Georges-Louis Leclerc, Comte de Buffon who in 1777 was the first to show that if we throw a needle of length $l$ completely at random onto a sheet of paper ruled with lines a distance $d$ apart, then the chances that the needle will fall so as to

[7] In fact there exist a number of more sophisticated Monte Carlo integration techniques which give more accurate answers than the simple "hit or miss" method we have described here. A discussion can be found in the book by Kalos and Whitlock (1986).

intersect one of the lines is $2l/\pi d$, provided that $d \geq l$. It was Laplace in 1820 who then pointed out that if the needle is thrown down $N$ times and is observed to land on a line $M$ of those times, we can make an estimate of $\pi$ from

$$\pi = \lim_{N\to\infty} \frac{2Nl}{Md}. \tag{1.44}$$

(Perhaps the connection between this and the Monte Carlo evaluation of integrals is not immediately apparent, but it will certainly become clear if you try to derive Equation (1.44) for yourself, or if you follow Dörrie's derivation.) A number of investigators made use of this method over the years to calculate approximate values for $\pi$. The most famous of these is Mario Lazzarini, who in 1901 announced that he had calculated a value of 3.1415929 for $\pi$ from an experiment in which a $2\frac{1}{2}$ cm needle was dropped 3408 times onto a sheet of paper ruled with lines 3 cm apart. This value, accurate to better than three parts in ten million, would be an impressive example of the power of the statistical sampling method were it not for the fact that it is almost certainly faked. Badger (1994) has demonstrated extremely convincingly, that, even supposing Lazzarini had the technology at his disposal to measure the length of his needle and the spaces between his lines to a few parts in $10^7$ (a step necessary to ensure the accuracy of Equation (1.44)), still the chances of his finding the results he did were poorer than three in a million; Lazzarini was imprudent enough to publish details of the progress of the experiment through the 3408 castings of the needle, and it turns out that the statistical "fluctuations" in the numbers of intersections of the needle with the ruled lines are much smaller than one would expect in a real experiment. All indications are that Lazzarini forged his results. However, other, less well known attempts at the experiment were certainly genuine, and yielded reasonable figures for $\pi$: 3.1596 (Wolf 1850), 3.1553 (Smith 1855). Apparently, performing the Buffon's needle experiment was for a while quite a sophisticated pastime amongst Europe's intellectual gentry.

With the advent of mechanical calculating machines at the end of the nineteenth century, numerical methods took a large step forward. These machines increased enormously the number and reliability of the arithmetic operations that could be performed in a numerical "experiment", and made the application of statistical sampling techniques to research problems in physics a realistic possibility for the first time. An early example of what was effectively a Monte Carlo calculation of the motion and collision of the molecules in a gas was described by William Thomson (later Lord Kelvin) in 1901. Thomson's calculations were aimed at demonstrating the truth of the equipartition theorem for the internal energy of a classical system. However, after the fashion of the time, he did not perform the laborious analysis himself, and a lot of the credit for the results must go to Thomson's

secretary, William Anderson, who apparently solved the kinetic equations for more than five thousand molecular collisions using nothing more than a pencil and a mechanical adding machine.

Aided by mechanical calculators, numerical methods, particularly the method of finite differences, became an important tool during the First World War. The authors recently heard the intriguing story of the Herculean efforts of French mathematician Henri Soudée, who in 1916 calculated firing tables for the new 400 mm cannons being set up at Verdun, directly from his knowledge of the hydrodynamic properties of gases. The tables were used when the cannons were brought to bear on the German-occupied Fort de Douaumont, and as a result the fort was taken by the allies. Soudée was later honoured by the French. By the time of the Second World War the mechanical calculation of firing angles for large guns was an important element of military technology. The physicist Richard Feynman tells the story of his employment in Philadelphia during the summer of 1940 working for the army on a mechanical device for predicting the trajectories of planes as they flew past (Feynman 1985). The device was to be used to guide anti-aircraft guns in attacking the planes. Despite some success with the machine, Feynman left the army's employ after only a few months, joking that the subject of mechanical computation was too difficult for him. He was shrewd enough to realize he was working on a dinosaur, and that the revolution of electronic computing was just around the corner. It was some years however before that particular dream would become reality, and before it did Feynman had plenty more chance to spar with the mechanical calculators. As a group leader during the Manhattan Project at Los Alamos he created what was effectively a highly pipelined human CPU, by employing a large number of people armed with Marchant mechanical adding machines in an arithmetic assembly line in which little cards with numbers on were passed from one worker to the next for processing on the machines. A number of numerical calculations crucial to the design of the atomic bomb were performed in this way.

The first real applications of the statistical sampling method to research problems in physics seem to have been those of Enrico Fermi, who was working on neutron diffusion in Rome in the early 1930s. Fermi never published his numerical methods—apparently he considered only the results to be of interest, not the methods used to obtain them—but according to his influential student and collaborator Emilio Segrè those methods were, in everything but name, precisely the Monte Carlo methods later employed by Ulam and Metropolis and their collaborators in the construction of the hydrogen bomb (Segrè 1980).

So it was that when the Monte Carlo method finally caught the attention of the physics community, it was again as the result of armed conflict. The important developments took place at the Los Alamos National Laboratory

in New Mexico, where Nick Metropolis, Stanislaw Ulam and John von Neumann gathered in the last months of the Second World War shortly after the epochal bomb test at Alamagordo, to collaborate on numerical calculations to be performed on the new ENIAC electronic computer, a mammoth, room-filing machine containing some 18 000 triode valves, whose construction was nearing completion at the University of Pennsylvania. Metropolis (1980) has remarked that the technology that went into the ENIAC existed well before 1941, but that it took the pressure of America's entry into the war to spur the construction of the machine.

It seems to have been Stan Ulam who was responsible for reinventing Fermi's statistical sampling methods. He tells of how the idea of calculating the average effect of a frequently repeated physical process by simply simulating the process over and over again on a digital computer came to him whilst huddled over a pack of cards, playing patience[8] one day. The game he was playing was "Canfield" patience, which is one of those forms of patience where the goal is simply to turn up every card in the pack, and he wondered how often on average one could actually expect to win the game. After abandoning the hopelessly complex combinatorics involved in answering this question analytically, it occurred to him that you could get an approximate answer simply by playing a very large number of games and seeing how often you win. With his mind never far from the exciting new prospect of the ENIAC computer, the thought immediately crossed his mind that he might be able to get the machine to play these games for him far faster than he ever could himself, and it was only a short conceptual leap to applying the same idea to some of the problems of the physics of the hydrogen bomb that were filling his work hours at Los Alamos. He later described his idea to John von Neumann who was very enthusiastic about it, and the two of them began making plans to perform actual calculations. Though Ulam's idea may appear simple and obvious to us today, there are actually many subtle questions involved in this idea that a physical problem with an exact answer can be approximately solved by studying a suitably chosen random process. It is a tribute to the ingenuity of the early Los Alamos workers that, rather than plunging headlong into the computer calculations, they considered most of these subtleties right from the start.

The war ended before the first Monte Carlo calculations were performed on the ENIAC. There was some uncertainty about whether the Los Alamos laboratory would continue to exist in peacetime, and Edward Teller, who was leading the project to develop the hydrogen bomb, was keen to apply the power of the computer to the problems of building the new bomb, in order to show that significant work was still going on at Los Alamos. Von Neumann developed a detailed plan of how the Monte Carlo method could be

[8] Also called "solitaire" in the USA.

implemented on the ENIAC to solve a number of problems concerned with neutron transport in the bomb, and throughout 1947 worked with Metropolis on preparations for the calculations. They had to wait to try their ideas out however, because the ENIAC was to be moved from Philadelphia where it was built to the army's Ballistics Research Laboratory in Maryland. For a modern computer this would not be a problem, but for the gigantic ENIAC, with its thousands of fragile components, it was a difficult task, and there were many who did not believe the computer would survive the journey. It did, however, and by the end of the year it was working once again in its new home. Before von Neumann and the others put it to work on the calculations for the hydrogen bomb, Richard Clippinger of the Ballistics Lab suggested a modification to the machine which allowed it to store programs in its electronic memory. Previously a program had to be set up by plugging and unplugging cables at the front of the machine, an arduous task which made the machine inflexible and inconvenient to use. Von Neumann was in favour of changing to the new "stored program" model, and Nick Metropolis and von Neumann's wife, Klari, made the necessary modifications to the computer themselves. It was the end of 1947 before the machine was at last ready, and Metropolis and von Neumann set to work on the planned Monte Carlo calculations.

The early neutron diffusion calculations were an impressive success, but Metropolis and von Neumann were not able to publish their results, because they were classified as secret. Over the following two years however, they and others, including Stan Ulam and Stanley Frankel, applied the new statistical sampling method to a variety of more mundane problems in physics, such as the calculation of the properties of hard-sphere gases in two and three dimensions, and published a number of papers which drew the world's attention to this emerging technique. The 1949 paper by Metropolis and Ulam on statistical techniques for studying integro-differential equations is of interest because it contained in its title the first use of the term "Monte Carlo" to describe this type of calculation. Also in 1949 the first conference on Monte Carlo methods was held in Los Alamos, attracting more than a hundred participants. It was quickly followed by another similar meeting in Gainesville, Florida.

The calculations received a further boost in 1948 with the arrival at Los Alamos of a new computer, humorously called the MANIAC. (Apparently the name was suggested by Enrico Fermi, who was tiring of computers with contrived acronymics—he claimed that it stood for "Metropolis and Neumann Invent Awful Contraption". Nowadays, with all our computers called things like XFK–23/z we would no doubt appreciate a few pronounceable names.) Apart from the advantage of being in New Mexico rather than Maryland, the MANIAC was a significant technical improvement over the ENIAC which Presper Eckert (1980), its principal architect, refers to as a

"hastily built first try". It was faster and contained a larger memory (40 kilobits, or 5 kilobytes in modern terms). It was built under the direction of Metropolis, who had been lured back to Los Alamos after a brief stint on the faculty at Chicago by the prospect of the new machine. The design was based on ideas put forward by John von Neumann and incorporated a number of technical refinements proposed by Jim Richardson, an engineer working on the project. A still more sophisticated computer, the MANIAC 2, was built at Los Alamos two years later, and both machines remained in service until the late fifties, producing a stream of results, many of which have proved to be seminal contributions to the field of Monte Carlo simulation. Of particular note to us is the publication in 1953 of the paper by Nick Metropolis, Marshall and Arianna Rosenbluth, and Edward and Mici Teller, in which they describe for the first time the Monte Carlo technique that has come to be known as the Metropolis algorithm. This algorithm was the first example of a thermal "importance sampling" method, and it is to this day easily the most widely used such method. We will be discussing it in some detail in Chapter 3. Also of interest are the Monte Carlo studies of nuclear cascades performed by Antony Turkevich and Nick Metropolis, and Edward Teller's work on phase changes in interacting hard-sphere gases using the Metropolis algorithm.

The exponential growth in computer power since those early days is by now a familiar story to us all, and with this increase in computational resources Monte Carlo techniques have looked deeper and deeper into the subject of statistical physics. Monte Carlo simulations have also become more accurate as a result of the invention of new algorithms. Particularly in the last twenty years, many new ideas have been put forward, of which we describe a good number in the rest of this book.

## Problems

**1.1** "If a system is in equilibrium with a thermal reservoir at temperature $T$, the probability of its having a total energy $E$ varies with $E$ in proportion to $\mathrm{e}^{-\beta E}$." True or false?

**1.2** A certain simple system has only two energy states, with energies $E_0$ and $E_1$, and transitions between the two states take place at rates $R(0 \to 1) = R_0 \exp[-\beta(E_1 - E_0)]$ and $R(1 \to 0) = R_0$. Solve the master equation (1.1) for the probabilities $w_0$ and $w_1$ of occupation of the two states as a function of time with the initial conditions $w_0 = 0$, $w_1 = 1$. Show that as $t \to \infty$ these solutions tend to the Boltzmann probabilities, Equation (1.5).

**1.3** A slightly more complex system contains $N$ distinguishable particles, each of which can be in one of two boxes. The particles in the first box have energy $E_0 = 0$ and the particles in the second have energy $E_1$, and particles

are allowed to move back and forward between the boxes under the influence of thermal excitations from a reservoir at temperature $T$. Find the partition function for this system and then use this result to calculate the internal energy.

**1.4** Solve the Ising model, whose Hamiltonian is given in Equation (1.30), in one dimension for the case where $B = 0$ as follows. Define a new set of variables $\sigma_i$ which take values 0 and 1 according to $\sigma_i = \frac{1}{2}(1 - s_i s_{i+1})$ and rewrite the Hamiltonian in terms of these variables for a system of $N$ spins with periodic boundary conditions. Show that the resulting system is equivalent to the one studied in Problem 1.3 and hence calculate the internal energy as a function of temperature.

# 2

# The principles of equilibrium thermal Monte Carlo simulation

In Section 1.3.1 we looked briefly at the general ideas behind equilibrium thermal Monte Carlo simulations. In this chapter we discuss these ideas in more detail in preparation for the discussion in the following chapters of a variety of specific algorithms for use with specific problems. The three crucial ideas that we introduce in this chapter are "importance sampling", "detailed balance" and "acceptance ratios". If you know what these phrases mean, you can understand most of the thermal Monte Carlo simulations that have been performed in the last thirty years.

## 2.1 The estimator

The usual goal in the Monte Carlo simulation of a thermal system is the calculation of the expectation value $\langle Q \rangle$ of some observable quantity $Q$, such as the internal energy in a model of a gas, or the magnetization in a magnetic model. As we showed in Section 1.3, the ideal route to calculating such an expectation, that of averaging the quantity of interest over all states $\mu$ of the system, weighting each with its own Boltzmann probability

$$\langle Q \rangle = \frac{\sum_\mu Q_\mu \, \mathrm{e}^{-\beta E_\mu}}{\sum_\mu \mathrm{e}^{-\beta E_\mu}} \tag{2.1}$$

is only tractable in the very smallest of systems. In larger systems, the best we can do is average over some subset of the states, though this necessarily introduces some inaccuracy into the calculation. Monte Carlo techniques work by choosing a subset of states at random from some probability distribution $p_\mu$ which we specify. Suppose we choose $M$ such states $\{\mu_1 \ldots \mu_M\}$.

Our best estimate of the quantity $Q$ will then be given by

$$Q_M = \frac{\sum_{i=1}^{M} Q_{\mu_i} p_{\mu_i}^{-1} \mathrm{e}^{-\beta E_{\mu_i}}}{\sum_{j=1}^{M} p_{\mu_j}^{-1} \mathrm{e}^{-\beta E_{\mu_j}}}. \tag{2.2}$$

$Q_M$ is called the **estimator** of $Q$. It has the property that, as the number $M$ of states sampled increases, it becomes a more and more accurate estimate of $\langle Q \rangle$, and when $M \to \infty$ we have $Q_M = \langle Q \rangle$.

The question we would like to answer now is how should we choose our $M$ states in order that $Q_M$ be an accurate estimate of $\langle Q \rangle$? In other words, how should we choose the probability distribution $p_\mu$? The simplest choice is to pick all states with equal probability; in other words make all $p_\mu$ equal. Substituting this choice into Equation (2.2), we get

$$Q_M = \frac{\sum_{i=1}^{M} Q_{\mu_i} \mathrm{e}^{-\beta E_{\mu_i}}}{\sum_{j=1}^{M} \mathrm{e}^{-\beta E_{\mu_j}}}. \tag{2.3}$$

It turns out however, that this is usually a rather poor choice to make. In most numerical calculations it is only possible to sample a very small fraction of the total number of states. Consider, for example, the Ising model of Section 1.2.2 again. A small three-dimensional cubic system of $10 \times 10 \times 10$ Ising spins would have $2^{1000} \simeq 10^{300}$ states, and a typical numerical calculation could only hope to sample up to about $10^8$ of those in a few hours on a good computer, which would mean we were only sampling one in every $10^{292}$ states of the system, a very small fraction indeed. The estimator given above is normally a poor guide to the value of $\langle Q \rangle$ under these circumstances. The reason is that one or both of the sums appearing in Equation (2.1) may be dominated by a small number of states, with all the other states, the vast majority, contributing a negligible amount even when we add them all together. This effect is often especially obvious at low temperatures, where these sums may be dominated by a hundred states, or ten states, or even one state, because at low temperatures there is not enough thermal energy to lift the system into the higher excited states, and so it spends almost all of its time sitting in the ground state, or one of the lowest of the excited states. In the example described above, the chances of one of the $10^8$ random states we sample in our simulation being the ground state are one in $10^{292}$, which means there is essentially no chance of our picking it, which makes $Q_M$ a very inaccurate estimate of $\langle Q \rangle$ if the sums are dominated by the contribution from this state.

On the other hand, if we had some way of knowing which states made the important contributions to the sums in Equation (2.1) and if we could pick our sample of $M$ states from just those states and ignore all the others, we could get a very good estimate of $\langle Q \rangle$ with only a small number of terms. This is the essence of the idea behind thermal Monte Carlo methods. The

technique for picking out the important states from amongst the very large number of possibilities is called **importance sampling**.

## 2.2 Importance sampling

As discussed in Section 1.1, we can regard an expectation value as a time average over the states that a system passes through during the course of a measurement. We do not assume that the system passes through every state during the measurement, even though every state appears in the sums of Equation (2.1). When you count how many states a typical system has you realize that this would never be possible. For instance, consider again the example we took in the last chapter of a litre container of gas at room temperature and atmospheric pressure. Such a system contains on the order of $10^{22}$ molecules. Typical speeds for these molecules are in the region of 100 $\mathrm{m\,s^{-1}}$, giving them a de Broglie wavelength of around $10^{-10}$ m. Each molecule will then have about $10^{27}$ different quantum states within the one litre box, and the complete gas will have around $(10^{27})^{10^{22}}$ states, which is a spectacularly large number.[1] The molecules will change from one state to another when they undergo collisions with one another or with the walls of the container, which they do at a rate of about $10^9$ collisions per second, or $10^{31}$ changes of state per second for the whole gas. At this rate, it will take about $10^{10^{23}}$ times the lifetime of the universe for our litre of gas to move through every possible state. Clearly then, our laboratory systems are only sampling the tiniest portion of their state spaces during the time that we conduct our experiments on them. In effect, real systems are carrying out a sort of Monte Carlo calculation of their own properties; they are "analogue computers" which evaluate expectations by taking a small but representative sample of their own states and averaging over that sample.[2] So it should not come as a great surprise to learn that we can also perform a reasonable calculation of the properties of a system using a simulation which only samples a small fraction of its states.

In fact, our calculations are often significantly better than this simple argument suggests. In Section 1.2.1 we showed that the range of energies of the states sampled by a typical system is very small compared with the total

[1] Actually, this is probably an overestimate, since it counts states which are classically distinguishable but quantum mechanically identical. For the purpose of the present rough estimation however, it will do fine.

[2] There are some systems which, because they have certain conservation laws, will not in fact sample their state spaces representatively, and this can lead to discrepancies between theory and experiment. Special Monte Carlo techniques exist for simulating these "conservative" systems, and we will touch on one or two of them in the coming chapters. For the moment, however, we will make the assumption that our system takes a representative sample of its own states.

energy of the system—the ratio was about $10^{-20}$ in the case of our litre of gas, for instance. Similar arguments can be used to show that systems sample very narrow ranges of other quantities as well. The reason for this, as we saw, is that the system is not sampling all states with equal probability, but instead sampling them according to the Boltzmann probability distribution, Equation (1.5). If we can mimic this effect in our simulations, we can exploit these narrow ranges of energy and other quantities to make our estimates of such quantities very accurate. For this reason, we normally try to take a sample of the states of the system in which the likelihood of any particular one appearing is proportional to its Boltzmann weight. This is the most common form of importance sampling, and most of the algorithms in this book make use of this idea in one form or another.

Our strategy then is this: instead of picking our $M$ states in such a way that every state of the system is as likely to get chosen as every other, we pick them so that the probability that a particular state $\mu$ gets chosen is $p_\mu = Z^{-1}\mathrm{e}^{-\beta E_\mu}$. Then our estimator for $\langle Q \rangle$, Equation (2.2), becomes just

$$Q_M = \frac{1}{M}\sum_{i=1}^{M} Q_{\mu_i}. \tag{2.4}$$

Notice that the Boltzmann factors have now cancelled out of the estimator, top and bottom, leaving a particularly simple expression. This definition of $Q_M$ works much better than (2.3), especially when the system is spending the majority of its time in a small number of states (such as, for example, the lowest-lying ones when we are at low temperatures), since these will be precisely the states that we pick most often, and the relative frequency with which we pick them will exactly correspond to the amount of time the real system would spend in those states.

The only remaining question is *how* exactly we pick our states so that each one appears with its correct Boltzmann probability. This is by no means a simple task. In the remainder of this chapter we describe the standard solution to the problem, which makes use of a "Markov process".

### 2.2.1 Markov processes

The tricky part of performing a Monte Carlo simulation is the generation of an appropriate random set of states according to the Boltzmann probability distribution. For a start, one cannot simply choose states at random and accept or reject them with a probability proportional to $\mathrm{e}^{-\beta E_\mu}$. That would be no better than our original scheme of sampling states at random; we would end up rejecting virtually all states, since the probabilities for their acceptance would be exponentially small. Instead, almost all Monte Carlo schemes rely on **Markov processes** as the generating engine for the set of states used.

For our purposes, a Markov process is a mechanism which, given a system in one state $\mu$, generates a new state of that system $\nu$. It does so in a random fashion; it will not generate the same new state every time it is given the initial state $\mu$. The probability of generating the state $\nu$ given $\mu$ is called the **transition probability** $P(\mu \to \nu)$ for the transition from $\mu$ to $\nu$, and for a true Markov process all the transition probabilities should satisfy two conditions: (1) they should not vary over time, and (2) they should depend only on the properties of the current states $\mu$ and $\nu$, and not on any other states the system has passed through. These conditions mean that the probability of the Markov process generating the state $\nu$ on being fed the state $\mu$ is the same every time it is fed the state $\mu$, irrespective of anything else that has happened. The transition probabilities $P(\mu \to \nu)$ must also satisfy the constraint

$$\sum_{\nu} P(\mu \to \nu) = 1, \tag{2.5}$$

since the Markov process must generate *some* state $\nu$ when handed a system in the state $\mu$. Note however, that the transition probability $P(\mu \to \mu)$, which is the probability that the new state generated will be the same as the old one, need not be zero. This amounts to saying there may be a finite probability that the Markov process will just stay in state $\mu$.

In a Monte Carlo simulation we use a Markov process repeatedly to generate a **Markov chain** of states. Starting with a state $\mu$, we use the process to generate a new one $\nu$, and then we feed *that* state into the process to generate another $\lambda$, and so on. The Markov process is chosen specially so that when it is run for long enough starting from any state of the system it will eventually produce a succession of states which appear with probabilities given by the Boltzmann distribution. (We call the process of reaching the Boltzmann distribution "coming to equilibrium", since it is exactly the process that a real system goes through with its "analogue computer" as it reaches equilibrium at the ambient temperature.) In order to achieve this, we place two further conditions on our Markov process, in addition to the ones specified above, the conditions of "ergodicity" and "detailed balance".

### 2.2.2 Ergodicity

The **condition of ergodicity** is the requirement that it should be possible for our Markov process to reach any state of the system from any other state, if we run it for long enough. This is necessary to achieve our stated goal of generating states with their correct Boltzmann probabilities. Every state $\nu$ appears with some non-zero probability $p_\nu$ in the Boltzmann distribution, and if that state were inaccessible from another state $\mu$ no matter how long we continue our process for, then our goal is thwarted if we start in state $\mu$: the probability of finding $\nu$ in our Markov chain of states will be zero, and

not $p_\nu$ as we require it to be.

The condition of ergodicity tells us that we are allowed to make some of the transition probabilities of our Markov process zero, but that there must be at least one path of non-zero transition probabilities between any two states that we pick. In practice, most Monte Carlo algorithms set almost all of the transition probabilities to zero, and we must be careful that in so doing we do not create an algorithm which violates ergodicity. For most of the algorithms we describe in this book we will explicitly prove that ergodicity is satisfied before making use of the algorithm.

### 2.2.3 Detailed balance

The other condition we place on our Markov process is the **condition of detailed balance**. This condition is the one which ensures that it is the Boltzmann probability distribution which we generate after our system has come to equilibrium, rather than any other distribution. Its derivation is quite subtle. Consider first what it means to say that the system is in equilibrium. The crucial defining condition is that the rate at which the system makes transitions into and out of any state $\mu$ must be equal. Mathematically we can express this as[3]

$$\sum_\nu p_\mu P(\mu \to \nu) = \sum_\nu p_\nu P(\nu \to \mu). \tag{2.6}$$

Making use of the sum rule, Equation (2.5), we can simplify this to

$$p_\mu = \sum_\nu p_\nu P(\nu \to \mu). \tag{2.7}$$

For any set of transition probabilities satisfying this equation, the probability distribution $p_\mu$ will be an equilibrium of the dynamics of the Markov process. Unfortunately, however, simply satisfying this equation is not sufficient to guarantee that the probability distribution will tend to $p_\mu$ from any state of the system if we run the process for long enough. We can demonstrate this as follows.

The transition probabilities $P(\mu \to \nu)$ can be thought of as the elements of a matrix $\mathbf{P}$. This matrix is called the **Markov matrix** or the **stochastic matrix** for the Markov process. Let us return to the notation of Section 1.1, in which we denoted by $w_\mu(t)$, the probability that our system is in a state $\mu$ at time $t$. If we measure time in steps along our Markov chain, then the

[3]This equation is essentially just a discrete-time version of the one we would get if we were to set the derivative in the master equation, Equation (1.1), to zero.

probability $w_\nu(t+1)$ of being in state $\nu$ at time $t+1$ is given by[4]

$$w_\nu(t+1) = \sum_\mu P(\mu \to \nu)\, w_\mu(t). \tag{2.8}$$

In matrix notation, this becomes

$$\mathbf{w}(t+1) = \mathbf{P} \cdot \mathbf{w}(t), \tag{2.9}$$

where $\mathbf{w}(t)$ is the vector whose elements are the weights $w_\mu(t)$. If the Markov process reaches a simple equilibrium state $\mathbf{w}(\infty)$ as $t \to \infty$, then that state satisfies

$$\mathbf{w}(\infty) = \mathbf{P} \cdot \mathbf{w}(\infty). \tag{2.10}$$

However, it is also possible for the process to reach a **dynamic equilibrium** in which the probability distribution $\mathbf{w}$ rotates around a number of different values. Such a rotation is called a **limit cycle**. In this case $\mathbf{w}(\infty)$ would satisfy

$$\mathbf{w}(\infty) = \mathbf{P}^n \cdot \mathbf{w}(\infty), \tag{2.11}$$

where $n$ is the length of the limit cycle. If we choose our transition probabilities (or equivalently our Markov matrix) to satisfy Equation (2.7) we guarantee that the Markov chain will have a simple equilibrium probability distribution $p_\mu$, but it may also have any number of limit cycles of the form (2.11). This means that there is no guarantee that the actual states generated will have anything like the desired probability distribution.

We get around this problem by applying an additional condition to our transition probabilities thus:

$$p_\mu P(\mu \to \nu) = p_\nu P(\nu \to \mu). \tag{2.12}$$

This is the condition of detailed balance. It is clear that any set of transition probabilities which satisfy this condition also satisfy Equation (2.6). (To prove it, simply sum both sides of Equation (2.12) over $\nu$.) We can also show that this condition eliminates limit cycles. To see this, look first at the left-hand side of the equation, which is the probability of being in a state $\mu$ multiplied by the probability of making a transition from that state to another state $\nu$. In other words, it is the overall rate at which transitions from $\mu$ to $\nu$ happen in our system. The right-hand side is the overall rate for the reverse transition. The condition of detailed balance tells us that on average the system should go from $\mu$ to $\nu$ just as often as it goes from $\nu$ to $\mu$. In a limit cycle, in which the probability of occupation of some or all of the states changes in a cyclic fashion, there must be states for which this

[4] This equation is also closely related to Equation (1.1). The reader may like to work out how the one can be transformed into the other.

condition is violated on any particular step of the Markov chain; in order for the probability of occupation of a particular state to increase, for instance, there must be more transitions into that state than out of it, on average. The condition of detailed balance forbids dynamics of this kind and hence forbids limit cycles.

Once we remove the limit cycles in this way, it is straightforward to show that the system will always tend to the probability distribution $p_\mu$ as $t \to \infty$. As $t \to \infty$, $\mathbf{w}(t)$ will tend exponentially towards the eigenvector corresponding to the largest eigenvalue of $\mathbf{P}$. This may be obvious to you if you are familiar with stochastic matrices. If not, we prove it in Section 3.3.2. For the moment, let us take it as given. Looking at Equation (2.10) we see that the largest eigenvalue of the Markov matrix must in fact be one.[5] If limit cycles of the form (2.11) were present, then we could also have eigenvalues which are complex roots of one, but the condition of detailed balance prevents this from happening. Now look back at Equation (2.7) again. We can express this equation in matrix notation as

$$\mathbf{p} = \mathbf{P} \cdot \mathbf{p}. \tag{2.13}$$

In other words, if Equation (2.7) (or equivalently the condition of detailed balance) holds for our Markov process, then the vector $\mathbf{p}$ whose elements are the probabilities $p_\mu$ is precisely the one correctly normalized eigenvector of the Markov matrix which has eigenvalue one. Putting this together with Equation (2.10) we see that the equilibrium probability distribution over states $\mathbf{w}(\infty)$ is none other than $\mathbf{p}$, and hence $\mathbf{w}(t)$ must tend exponentially to $\mathbf{p}$ as $t \to \infty$.

There is another reason why detailed balance makes sense for Monte Carlo simulations: the "analogue computers" which constitute the real physical systems we are trying to mimic almost always obey the condition of detailed balance. The reason is that they are based on standard quantum or classical mechanics, which is time-reversal symmetric. If they did not obey detailed balance, then in equilibrium they could have one or more limit cycles around which the system passes in one particular direction. If we take such a system and reverse it in time, the motion around this cycle is also reversed, and it becomes clear that the dynamics of the system in equilibrium is not the same forward as it is in reverse. Such a violation of time-reversal symmetry is forbidden for most systems, implying that they must satisfy detailed balance. Although this does not mean that we are necessarily obliged

[5] All Markov matrices have at least one eigenvector with corresponding eigenvalue one, a fact which is easily proven since Equation (2.5) implies that the vector $(1, 1, 1, \ldots)$ is a left eigenvector of $\mathbf{P}$ with eigenvalue one. It is possible to have more than one eigenvector with eigenvalue one if the states of the system divide into two or more mutually inaccessible subsets. However, if the condition of ergodicity is satisfied then such subsets are forbidden and hence there is only one such eigenvector.

to enforce detailed balance in our simulations as well, it is helpful if we do, because it makes the behaviour of our model system more similar to that of the real one we are trying to understand.

So, we have shown that we can arrange for the probability distribution of states generated by our Markov process to tend to any distribution $p_\mu$ we please by choosing a set of transition probabilities which satisfy Equation (2.12). Given that we wish the equilibrium distribution to be the Boltzmann distribution,[6] clearly we want to choose the values of $p_\mu$ to be the Boltzmann probabilities, Equation (1.5). The detailed balance equation then tells us that the transition probabilities should satisfy

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{p_\nu}{p_\mu} = \mathrm{e}^{-\beta(E_\nu - E_\mu)}. \tag{2.14}$$

This equation and Equation (2.5) are the constraints on our choice of transition probabilities $P(\mu \to \nu)$. If we satisfy these, as well as the condition of ergodicity, then the equilibrium distribution of states in our Markov process will be the Boltzmann distribution. Given a suitable set of transition probabilities, our plan is then to write a computer program which implements the Markov process corresponding to these transition probabilities so as to generate a chain of states. After waiting a suitable length of time[7] to allow the probability distribution of states $w_\mu(t)$ to get sufficiently close to the Boltzmann distribution, we average the observable $Q$ that we are interested in over $M$ states and we have calculated the estimator $Q_M$ defined in Equation (2.4). A number of refinements on this outline are possible and we will discuss some of those in the remainder of this chapter and in later chapters of the book, but this is the basic principle on which virtually all modern equilibrium Monte Carlo calculations are based.

Our constraints still leave us a good deal of freedom over how we choose the transition probabilities. There are many ways in which to satisfy them. One simple choice for example is

$$P(\mu \to \nu) \propto \mathrm{e}^{-\frac{1}{2}\beta(E_\mu - E_\nu)}, \tag{2.15}$$

although as we will show in Section 3.1 this choice is not a very good one. There are some other choices which are known to work well in many cases, such as the "Metropolis algorithm" proposed by Metropolis and co-workers in 1953, and we will discuss the most important of these in the coming chapters. However, it must be stressed—and this is one of the most important

[6] Occasionally, in fact, we want to generate equilibrium distributions other than the Boltzmann distribution. An example is the entropic sampling algorithm of Section 6.3. In this case the arguments here still apply. We simply feed our required distribution into the condition of detailed balance.

[7] Exactly how long we have to wait can be a difficult thing to decide. A number of possible criteria are discussed in Section 3.2.

things this book has to say—that the standard algorithms are *very rarely* the best ones for solving new problems with. In most cases they will work, and in some cases they will even give quite good answers, but you can almost always do a better job by giving a little extra thought to choosing the best set of transition probabilities to construct an algorithm that will answer the particular questions that you are interested in. A purpose-built algorithm can often give a much faster simulation than an equivalent standard algorithm, and the improvement in efficiency can easily make the difference between finding an answer to a problem and not finding one.

## 2.3 Acceptance ratios

Our little summary above makes rather light work of the problems of constructing a Monte Carlo algorithm. Given a desired set of transition probabilities $P(\mu \to \nu)$ satisfying the conditions (2.5) and (2.14), we say, we simply concoct some Markov process that generates states with exactly those transition probabilities, and *presto!* we produce a string of states of our system with exactly their correct Boltzmann probabilities. However, it is often very far from obvious what the appropriate Markov process is that has the required transition probabilities, and finding one can be a haphazard, trial-and-error process. For some problems we can use known algorithms such as the Metropolis method (see Section 3.1), but for many problems the standard methods are far from ideal, and we will do much better if we can tailor a new algorithm to our specific needs. But though we may be able to suggest many candidate Markov processes—different ways of creating a new state $\nu$ from an old one $\mu$—still we may not find one which gives exactly the right set of transition probabilities. The good news however is that we don't have to. In fact it turns out that we can choose any algorithm we like for generating the new states, and still have our desired set of transition probabilities, by introducing something called an **acceptance ratio**. The idea behind the trick is this.

We mentioned in Section 2.2.1 that we are allowed to make the "stay-at-home" transition probability $P(\mu \to \mu)$ non-zero if we want. If we set $\nu = \mu$ in Equation (2.14), we get the simple tautology $1 = 1$, which means that the condition of detailed balance is always satisfied for $P(\mu \to \mu)$, no matter what value we choose for it. This gives us some flexibility about how we choose the other transition probabilities with $\mu \neq \nu$. For a start, it means that we can adjust the value of any $P(\mu \to \nu)$ and keep the sum rule (2.5) satisfied, by simply compensating for that adjustment with an equal but opposite adjustment of $P(\mu \to \mu)$. The only thing we need to watch is that $P(\mu \to \mu)$ never passes out of its allowed range between zero and one. If we make an adjustment like this in $P(\mu \to \nu)$, we can also arrange for Equation (2.14) to remain satisfied, by simultaneously making a change in

$P(\nu \to \mu)$, so that the ratio of the two is preserved.

It turns out that these considerations actually give us enough freedom that we can make the transition probabilities take any set of values we like by tweaking the values of the probabilities $P(\mu \to \mu)$. To see this, we break the transition probability down into two parts:

$$P(\mu \to \nu) = g(\mu \to \nu)\, A(\mu \to \nu). \tag{2.16}$$

The quantity $g(\mu \to \nu)$ is the **selection probability**, which is the probability, given an initial state $\mu$, that our algorithm will generate a new target state $\nu$, and $A(\mu \to \nu)$ is the acceptance ratio (sometimes also called the "acceptance probability"). The acceptance ratio says that if we start off in a state $\mu$ and our algorithm generates a new state $\nu$ from it, we should accept that state and change our system to the new state $\nu$ a fraction of the time $A(\mu \to \nu)$. The rest of the time we should just stay in the state $\mu$. We are free to choose the acceptance ratio to be any number we like between zero and one; choosing it to be zero for all transitions is equivalent to choosing $P(\mu \to \mu) = 1$, which is the largest value it can take, and means that we will never leave the state $\mu$. (Not a very desirable situation. We would never choose an acceptance ratio of zero for an actual calculation.)

This gives us complete freedom about how we choose the selection probabilities $g(\mu \to \nu)$, since the constraint (2.14) only fixes the ratio

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)}. \tag{2.17}$$

The ratio $A(\mu \to \nu)/A(\nu \to \mu)$ can take any value we choose between zero and infinity, which means that both $g(\mu \to \nu)$ and $g(\nu \to \mu)$ can take any values we like.

Our other constraint, the sum rule of Equation (2.5), is still satisfied, since the system must end up in *some* state after each step in the Markov chain, even if that state is just the state we started in.

So, in order to create our Monte Carlo algorithm what we actually do is think up an algorithm which generates random new states $\nu$ given old ones $\mu$, with some set of probabilities $g(\mu \to \nu)$, and then we accept or reject those states with acceptance ratios $A(\mu \to \nu)$ which we choose to satisfy Equation (2.17). This will then satisfy all the requirements for the transition probabilities, and so produce a string of states which, when the algorithm reaches equilibrium, will each appear with their correct Boltzmann probability.

This all seems delightful, but there is a catch which we must always bear in mind, and which is one of the most important considerations in the design of Monte Carlo algorithms. If the acceptance ratios for our moves are low, then the algorithm will on most time steps simply stay in the state

that it is in, and not go anywhere. The step on which it actually accepts a change to a new state will be rare, and this is wasteful of time. We want an algorithm that moves nimbly about state space and samples a wide selection of different states. We don't want to take a million time steps and find that our algorithm has only sampled a dozen states. The solution to this problem is to make the acceptance ratio as close to unity as possible. One way to do this is to note that Equation (2.17) fixes only the ratio $A(\mu \to \nu)/A(\nu \to \mu)$ of the acceptance ratios for the transitions in either direction between any two states. Thus we are free to multiply both $A(\mu \to \nu)$ and $A(\nu \to \mu)$ by the same factor, and the equation will still be obeyed. The only constraint is that both acceptance ratios should remain between zero and one. In practice then, what we do is to set the larger of the two acceptance ratios to one, and have the other one take whatever value is necessary for the ratio of the two to satisfy (2.17). This ensures that the acceptance ratios will be as large as they can be while still satisfying the relevant conditions, and indeed that the ratio in one direction will be unity, which means that in that direction at least, moves will always be accepted.

However, the best thing we can do to keep the acceptance ratios large is to try to embody in the selection probabilities $g(\mu \to \nu)$ as much as we can of the dependence of $P(\mu \to \nu)$ on the characteristics of the states $\mu$ and $\nu$, and put as little as we can in the acceptance ratio. The ideal algorithm is one in which the new states are selected with exactly the correct transition probabilities all the time, and the acceptance ratio is always one. A good algorithm is one in which the acceptance probability is usually close to one. Much of the effort invested in the algorithms described in this book is directed at making the acceptance ratios large.

## 2.4 Continuous time Monte Carlo

There is a another twist we can add to our Markov process to allow ourselves further freedom about the way in which we choose states, without letting the acceptance ratios get too low. It is called **continuous time Monte Carlo**, or sometimes the **BKL algorithm**, after Bortz, Kalos and Lebowitz (1975), who invented it. Continuous time Monte Carlo is not nearly as widely used as it ought to be; it is an important and powerful technique and many calculations can be helped enormously by making use of it.

Consider a system at low temperature. Such systems are always a problem where Monte Carlo methods are concerned—cool systems move from state to state very slowly in real life and the problem is no less apparent in simulations. A low-temperature system is a good example of the sort of problem system that was described in the last section. Once it reaches equilibrium at its low temperature it will spend a lot of its time in the ground state. Maybe it will spend a hundred consecutive time-steps of the simu-

lation in the ground state, then move up to the first excited state for one time-step and then relax back to the ground state again. Such behaviour is not unreasonable for a cold system but we waste a lot of computer time simulating it. Time-step after time-step our algorithm selects a possible move to some excited state, but the acceptance ratio is very low and virtually all of these possible moves are rejected, and the system just ends up spending most of its time in the ground state.

Well, what if we were to accept that this is the case, and take a look at the acceptance ratio for a move from the ground state to the first excited state, and say to ourselves, "Judging by this acceptance ratio, this system is going to spend a hundred time-steps in the ground state before it accepts a move to the first excited state". Then we could jump the gun by *assuming* that the system will do this, miss out the calculations involved in the intervening useless one hundred time-steps, and progress straight to the one time-step in which something interesting happens. This is the essence of the idea behind the continuous time method. In this technique, we have a time-step which corresponds to a varying length of time, depending on how long we expect the system to remain in its present state before moving to a new one. Then when we come to take the average of our observable $Q$ over many states, we weight the states in which the system spends longest the most heavily—the calculation of the estimator of $Q$ is no more than a time average, so each value $Q_\mu$ for $Q$ in state $\mu$ should be weighted by how long the system spends in that state.

How can we adapt our previous ideas concerning the transition probabilities for our Markov process to take this new idea into account? Well, assuming that the system is in some state $\mu$, we can calculate how long a time $\Delta t$ (measured in steps of the simulation) it will stay there for before a move to another state is accepted by considering the "stay-at-home" probability $P(\mu \to \mu)$. The probability that it is still in this same state $\mu$ after $t$ time-steps is just

$$[P(\mu \to \mu)]^t = \mathrm{e}^{t \log P(\mu \to \mu)}, \tag{2.18}$$

and so the time-scale $\Delta t$ is

$$\begin{aligned} \Delta t &= -\frac{1}{\log P(\mu \to \mu)} = -\frac{1}{\log[1 - \sum_{\nu \neq \mu} P(\mu \to \nu)]} \\ &\simeq \frac{1}{\sum_{\nu \neq \mu} P(\mu \to \nu)}. \end{aligned} \tag{2.19}$$

So, if we can calculate this quantity $\Delta t$, then rather than wait this many time-steps for a Monte Carlo move to get accepted, we can simply pretend that we have done the waiting and go right ahead and change the state of the system to a new state $\nu \neq \mu$. Which state should we choose for $\nu$? We should choose one at random, but in proportion to $P(\mu \to \nu)$. Thus our continuous time Monte Carlo algorithm consists of the following steps:

1. We calculate the probabilities $P(\mu \to \nu)$ for transitions to all states which can be reached in one Monte Carlo step from the current state $\mu$. We choose a new state $\nu$ with probability proportional to $P(\mu \to \nu)$ and change the state of the system to $\nu$.

2. Using our values for the $P(\mu \to \nu)$ we also calculate the time interval $\Delta t$. Notice that we have to recalculate $\Delta t$ at each step, since in general it will change from one step to the next.

3. We increment the time $t$ by $\Delta t$, to mimic the effect of waiting $\Delta t$ Monte Carlo steps. The variable $t$ keeps a record of how long the simulation has gone on for in "equivalent Monte Carlo steps".

While this technique is in many respects a very elegant solution to the problem of simulating a system at low temperatures (or any other system which has a low acceptance ratio), it does suffer from one obvious drawback, which is that step (1) above involves calculating $P(\mu \to \nu)$ for every possible state $\nu$ which is accessible from $\mu$. There may be very many such states (for some systems the number goes up exponentially with the size of the system), and so this step may take a very long time. However, in some cases, it turns out that the set of transition probabilities is very similar from one step of the algorithm to the next, only a few of them changing at each step, and hence it is possible to keep a table of probabilities and update only a few entries at each step to keep the table current. In cases such as these the continuous time method becomes very efficient and can save us a great deal of CPU time, despite being more complex than the accept/reject method discussed in the previous section. One example of a continuous time Monte Carlo algorithm is presented in Section 5.2.1 for the conserved-order-parameter Ising model.

In the next few chapters, we will examine a number of common models used for calculating the equilibrium properties of condensed matter systems, and show how the general ideas presented in this chapter can be used to find efficient numerical solutions to these physical problems.

## Problems

**2.1** Derive Equation (2.8) from Equation (1.1).

**2.2** Consider a system which has just three energy states, with energies $E_0 < E_1 < E_2$. Suppose that the only allowed transitions are ones of the form $\mu \to \nu$, where $\nu = (\mu + 1) \bmod 3$. Such a system cannot satisfy detailed balance. Show nonetheless that it is possible to choose the transition probabilities $P(\mu \to \nu)$ so that the Boltzmann distribution is an equilibrium of the dynamics.

# 3

# The Ising model and the Metropolis algorithm

In Section 1.2.2 we introduced the Ising model, which is one of the simplest and best studied of statistical mechanical models. In this chapter and the next we look in detail at the Monte Carlo methods that have been used to investigate the properties of this model. As well as demonstrating the application of the basic principles described in the last chapter, the study of the Ising model provides an excellent introduction to the most important Monte Carlo algorithms in use today. Along the way we will also look at some of the tricks used for implementing Monte Carlo algorithms in computer programs and at some of the standard techniques used to analyse the data those programs generate.

To recap briefly, the Ising model is a simple model of a magnet, in which dipoles or "spins" $s_i$ are placed on the sites $i$ of a lattice. Each spin can take either of two values: $+1$ and $-1$. If there are $N$ sites on the lattice, then the system can be in $2^N$ states, and the energy of any particular state is given by the Ising Hamiltonian:

$$H = -J \sum_{\langle ij \rangle} s_i s_j - B \sum_i s_i, \tag{3.1}$$

where $J$ is an interaction energy between nearest-neighbour spins $\langle ij \rangle$, and $B$ is an external magnetic field. We are interested in simulating an Ising system of finite size using Monte Carlo methods, so that we can estimate the values of quantities such as the magnetization $m$ (Equation (1.34)) or the specific heat $c$ (Equation (1.37)) at any given temperature. Most of the interesting questions concerning the Ising model can be answered by performing simulations in zero magnetic field $B = 0$, so for the moment at least we will concentrate on this case.

## 3.1 The Metropolis algorithm

The very first Monte Carlo algorithm we introduce in this book is the most famous and widely used algorithm of them all, the **Metropolis algorithm**, which was introduced by Nicolas Metropolis and his co-workers in a 1953 paper on simulations of hard-sphere gases (Metropolis *et al.* 1953). We will use this algorithm to illustrate many of the general concepts involved in a real Monte Carlo calculation, including equilibration, measurement of expectation values, and the calculation of errors. First however, let us see how the algorithm is arrived at, and how one might go about implementing it on a computer.

The derivation of the Metropolis algorithm follows exactly the plan we outlined in Section 2.3. We choose a set of selection probabilities $g(\mu \to \nu)$, one for each possible transition from one state to another, $\mu \to \nu$, and then we choose a set of acceptance probabilities $A(\mu \to \nu)$ such that Equation (2.17) satisfies the condition of detailed balance, Equation (2.14). The algorithm works by repeatedly choosing a new state $\nu$, and then accepting or rejecting it at random with our chosen acceptance probability. If the state is accepted, the computer changes the system to the new state $\nu$. If not, it just leaves it as it is. And then the process is repeated again and again.

The selection probabilities $g(\mu \to \nu)$ should be chosen so that the condition of ergodicity—the requirement that every state be accessible from every other in a finite number of steps—is fulfilled (see Section 2.2.2). This still leaves us a good deal of latitude about how they are chosen; given an initial state $\mu$ we can generate any number of candidate states $\nu$ simply by flipping different subsets of the spins on the lattice. However, as we demonstrated in Section 1.2.1, the energies of systems in thermal equilibrium stay within a very narrow range—the energy fluctuations are small by comparison with the energy of the entire system. In other words, the real system spends most of its time in a subset of states with a narrow range of energies and rarely makes transitions that change the energy of the system dramatically. This tells us that we probably don't want to spend much time in our simulation considering transitions to states whose energy is very different from the energy of the present state. The simplest way of achieving this in the Ising model is to consider only those states which differ from the present one by the flip of a single spin. An algorithm which does this is said to have **single-spin-flip dynamics**. The algorithm we describe in this chapter has single-spin-flip dynamics, although this is not what makes it the Metropolis algorithm. (As discussed below, it is the particular choice of acceptance ratio that characterizes the Metropolis algorithm. Our algorithm would still be a Metropolis algorithm even if it flipped many spins at once.)

Using single-spin-flip dynamics guarantees that the new state $\nu$ will have an energy $E_\nu$ differing from the current energy $E_\mu$ by at most $2J$ for each

bond between the spin we flip and its neighbours. For example, on a square lattice in two dimensions each spin has four neighbours, so the maximum difference in energy would be $8J$. The general expression is $2zJ$, where $z$ is the **lattice coordination number**, i.e., the number of neighbours that each site on the lattice has.[1] Using single-spin-flip dynamics also ensures that our algorithm obeys ergodicity, since it is clear that we can get from any state to any other on a finite lattice by flipping one by one each of the spins by which the two states differ.

In the Metropolis algorithm the selection probabilities $g(\mu \to \nu)$ for each of the possible states $\nu$ are all chosen to be equal. The selection probabilities of all other states are set to zero. Suppose there are $N$ spins in the system we are simulating. With single-spin-flip dynamics there are then $N$ different spins that we could flip, and hence $N$ possible states $\nu$ which we can reach from a given state $\mu$. Thus there are $N$ selection probabilities $g(\mu \to \nu)$ which are non-zero, and each of them takes the value

$$g(\mu \to \nu) = \frac{1}{N}. \tag{3.2}$$

With these selection probabilities, the condition of detailed balance, Equation (2.14), takes the form

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)} = \frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \mathrm{e}^{-\beta(E_\nu - E_\mu)}. \tag{3.3}$$

Now we have to choose the acceptance ratios $A(\mu \to \nu)$ to satisfy this equation. As we pointed out in Section 2.2.3, one possibility is to choose

$$A(\mu \to \nu) = A_0 \mathrm{e}^{-\frac{1}{2}\beta(E_\nu - E_\mu)}. \tag{3.4}$$

The constant of proportionality $A_0$ cancels out in Equation (3.3), so we can choose any value for it that we like, except that $A(\mu \to \nu)$, being a probability, should never be allowed to become greater than one. As we mentioned above, the largest difference in energy $E_\nu - E_\mu$ that we can have between our two states is $2zJ$, where $z$ is the lattice coordination number. That means that the largest value of $\mathrm{e}^{-\frac{1}{2}\beta(E_\nu - E_\mu)}$ is $\mathrm{e}^{\beta z J}$. Thus, in order to make sure $A(\mu \to \nu) \leq 1$ we want to choose

$$A_0 \leq \mathrm{e}^{-\beta z J}. \tag{3.5}$$

To make the algorithm as efficient as possible, we want the acceptance probabilities to be as large as possible, so we make $A_0$ as large as it is allowed to

[1] This is not the same thing as the "spin coordination number" which we introduce in Chapter 5. The spin coordination number is the number of spins $j$ neighbouring $i$ which have the same value as spin $i$: $s_j = s_i$.

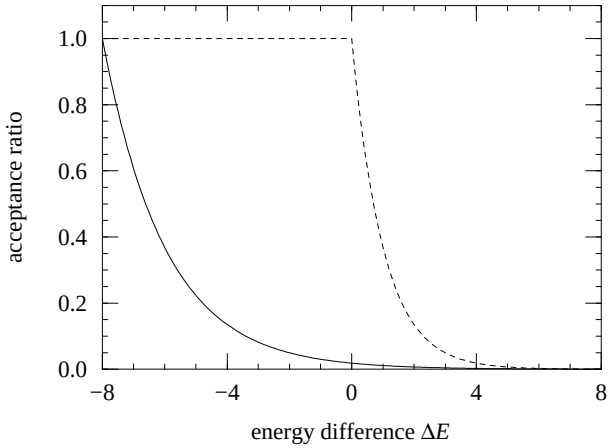


FIGURE 3.1 Plot of the acceptance ratio given in Equation (3.6) (solid line). This acceptance ratio gives rise to an algorithm which samples the Boltzmann distribution correctly, but is very inefficient, since it rejects the vast majority of the moves it selects for consideration. The Metropolis acceptance ratio (dashed line) is much more efficient.

be, which gives us

$$A(\mu \to \nu) = \mathrm{e}^{-\frac{1}{2}\beta(E_\nu - E_\mu + 2zJ)}. \tag{3.6}$$

This is not the Metropolis algorithm (we are coming to that), but using this acceptance probability we can perform a Monte Carlo simulation of the Ising model, and it will correctly sample the Boltzmann distribution. However, the simulation will be very inefficient, because the acceptance ratio, Equation (3.6), is very small for almost all moves. Figure 3.1 shows the acceptance ratio (solid line) as a function of the energy difference $\Delta E = E_\nu - E_\mu$ over the allowed range of values for a simulation with $\beta = J = 1$ and a lattice coordination number $z = 4$, as on a square lattice for example. As we can see, although $A(\mu \to \nu)$ starts off at 1 for $\Delta E = -8$, it quickly falls to only about 0.13 at $\Delta E = -4$, and to only 0.02 when $\Delta E = 0$. The chances of making any move for which $\Delta E > 0$ are pitifully small, and in practice this means that an algorithm making use of this acceptance ratio would be tremendously slow, spending most of its time rejecting moves and not flipping any spins at all. The solution to this problem is as follows.

In Equation (3.4) we have assumed a particular functional form for the acceptance ratio, but the condition of detailed balance, Equation (3.3), doesn't actually require that it take this form. Equation (3.3) only specifies the ratio of pairs of acceptance probabilities, which still leaves us quite a lot of room to manœuvre. In fact, as we pointed out in Section 2.3, when given a con-

straint like (3.3) the way to maximize the acceptance ratios (and therefore produce the most efficient algorithm) is always to give the larger of the two ratios the largest value possible—namely 1—and then adjust the other to satisfy the constraint. To see how that works out in this case, suppose that of the two states $\mu$ and $\nu$ we are considering here, $\mu$ has the lower energy and $\nu$ the higher: $E_\mu < E_\nu$. Then the larger of the two acceptance ratios is $A(\nu \to \mu)$, so we set that equal to one. In order to satisfy Equation (3.3), $A(\mu \to \nu)$ must then take the value $\mathrm{e}^{-\beta(E_\nu - E_\mu)}$. Thus the optimal algorithm is one in which

$$A(\mu \to \nu) = \begin{cases} \mathrm{e}^{-\beta(E_\nu - E_\mu)} & \text{if } E_\nu - E_\mu > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{3.7}$$

In other words, if we select a new state which has an energy lower than or equal to the present one, we should always accept the transition to that state. If it has a higher energy then we maybe accept it, with the probability given above. This is the Metropolis algorithm for the Ising model with single-spin-flip dynamics. It is Equation (3.7) which makes it the Metropolis algorithm. This is the part that was pioneered by Metropolis and co-workers in their paper on hard-sphere gases, and any algorithm, applied to any model, which chooses selection probabilities according to a rule like (3.7) can be said to be a Metropolis algorithm. At first, this rule may seem a little strange, especially the part about how we *always* accept a move that will lower the energy of the system. The first algorithm we suggested, Equation (3.6), seems much more natural in this respect, since it sometimes rejects moves to lower energy. However, as we have shown, the Metropolis algorithm satisfies detailed balance, and is by far the more efficient algorithm, so, natural or not, it has become the algorithm of choice in the overwhelming majority of Monte Carlo studies of simple statistical mechanical models in the last forty years. We have also plotted Equation (3.7) in Figure 3.1 (dashed line) for comparison between the two algorithms.

### 3.1.1 Implementing the Metropolis algorithm

Let us look now at how we would actually go about writing a computer program to perform a simulation of the Ising model using the Metropolis algorithm. For simplicity we will continue to focus on the case of zero magnetic field $B = 0$, although the generalization to the case $B \neq 0$ is not hard (see Problem 3.1). In fact almost all the past studies of the Ising model, including Onsager's exact solution in two dimensions, have looked only at the zero-field case.

First, we need an actual lattice of spins to work with, so we would define a set of $N$ variables—an array—which can take the values $\pm 1$. Probably we would use integer variables, so it would be an integer array. Normally, we

apply **periodic boundary conditions** to the array. That is, we specify that the spins on one edge of the lattice are neighbours of the corresponding spins on the other edge. This ensures that all spins have the same number of neighbours and local geometry, and that there are no special edge spins which have different properties from the others; all the spins are equivalent and the system is completely translationally invariant. In practice this considerably improves the quality of the results from our simulation.

A variation on the idea of periodic boundary conditions is to use "helical boundary conditions" which are only very slightly different from periodic ones and possess all the same benefits but are usually considerably simpler to implement and can make our simulation significantly faster. The various types of boundary conditions and their implementation are described in detail in Section 13.1, along with methods for representing most common lattice geometries using arrays.

Next we need to decide at what temperature, or alternatively at what value of $\beta$ we want to perform our simulation, and we need to choose some starting value for each of the spins—the initial state of the system. In a lot of cases, the initial state we choose is not particularly important, though sometimes a judicious choice can reduce the time taken to come to equilibrium (see Section 3.2). The two most commonly used initial states are the zero-temperature state and the infinite temperature state. At $T = 0$ the Ising model will be in its ground state. When the interaction energy $J$ is greater than zero and the external field $B$ is zero (as is the case in the simulations we will present in this chapter) there are actually two ground states. These are the states in which the spins are all up or all down. It is easy to see that these must be ground states, since in these states each pair of spins in the first term of Equation (3.1) contributes the lowest possible energy $-J$ to the Hamiltonian. In any other state there will be pairs of spins which contribute $+J$ to the Hamiltonian, so that its overall value will be higher. (If $B \neq 0$ then there will only be one ground state—the field ensures that one of the two is favoured over the other.) The other commonly used initial state is the $T = \infty$ state. When $T = \infty$ the thermal energy $kT$ available to flip the spins is infinitely larger than the energy due to the spin–spin interaction $J$, so the spins are just oriented randomly up or down in an uncorrelated fashion.

These two choices of initial state are popular because they each correspond to a known, well defined temperature, and they are easy to generate. There is, however, one other initial state which can sometimes be very useful, which we should mention. Often we don't just perform one simulation at a single temperature, but rather a set of simulations one after another at a range of different values of $T$, to probe the behaviour of the model with varying temperature. In this case it is often advantageous to us to choose as the initial state of our system the *final* state of the system for a simulation

at a nearby temperature. For example, suppose we are interested in probing a range of temperatures between $T = 1.0$ and $T = 2.0$ in steps of 0.1. (Here and throughout much of the rest of this book, we measure temperature in energy units, so that $k = 1$. Thus when we say $T = 2.0$ we mean that $\beta^{-1} = 2.0$.) Then we might start off by performing a simulation at $T = 1.0$ using the zero-temperature state with all spins aligned as our initial state. At the end of the simulation, the system will be in equilibrium at $T = 1.0$, and we can use the final state of that simulation as the initial state for the simulation at $T = 1.1$, and so on. The justification for doing this is clear: we hope that the equilibrium state at $T = 1.0$ will be more similar to that at $T = 1.1$ than will the zero-temperature state. In most cases this is a correct assumption and our system will come to equilibrium quicker with this initial state than with either a $T = 0$ or a $T = \infty$ one.

Now we start our simulation. The first step is to generate a new state—the one we called $\nu$ in the discussion above. The new state should differ from the present one by the flip of just one spin, and every such state should be exactly as likely as every other to be generated. This is an easy task to perform. We just pick a single spin $k$ at random from the lattice to be flipped. Next, we need to calculate the difference in energy $E_\nu - E_\mu$ between the new state and the old one, in order to apply Equation (3.7). The most straightforward way to do this would be to calculate $E_\mu$ directly by substituting the values $s_i^\mu$ of the spins in state $\mu$ into the Hamiltonian (3.1), then flip spin $k$ and calculate $E_\nu$, and take the difference. This, however, is not a very efficient way to do it. Even in zero magnetic field $B = 0$ we still have to perform the sum in the first term of (3.1), which has as many terms as there are bonds on the lattice, which is $\frac{1}{2}Nz$. But most of these terms don't change when we flip our single spin. The only ones that change are those that involve the flipped spin. The others stay the same and so cancel out when we take the difference $E_\nu - E_\mu$. The *change* in energy between the two states is thus

$$\begin{aligned} E_\nu - E_\mu &= -J \sum_{\langle ij \rangle} s_i^\nu s_j^\nu + J \sum_{\langle ij \rangle} s_i^\mu s_j^\mu \\ &= -J \sum_{i \text{ n.n. to } k} s_i^\mu (s_k^\nu - s_k^\mu). \end{aligned} \tag{3.8}$$

In the second line the sum is over only those spins $i$ which are nearest neighbours of the flipped spin $k$ and we have made use of the fact that all of these spins do not themselves flip, so that $s_i^\nu = s_i^\mu$. Now if $s_k^\mu = +1$, then after spin $k$ has been flipped we must have $s_k^\nu = -1$, so that $s_k^\nu - s_k^\mu = -2$. On the other hand, if $s_k^\mu = -1$ then $s_k^\nu - s_k^\mu = +2$. Thus we can write

$$s_k^\nu - s_k^\mu = -2 s_k^\mu, \tag{3.9}$$

and so

$$\begin{aligned} E_\nu - E_\mu &= 2J \sum_{i \text{ n.n. to } k} s_i^\mu s_k^\mu \\ &= 2J s_k^\mu \sum_{i \text{ n.n. to } k} s_i^\mu . \end{aligned} \tag{3.10}$$

This expression only involves summing over $z$ terms, rather than $\frac{1}{2}Nz$, and it doesn't require us to perform any multiplications for the terms in the sum, so it is much more efficient than evaluating the change in energy directly. What's more, it involves only the values of the spins in state $\mu$, so we can evaluate it *before* we actually flip the spin $k$.

The algorithm thus involves calculating $E_\nu - E_\mu$ from Equation (3.10) and then following the rule given in Equation (3.7): if $E_\nu - E_\mu \leq 0$ we definitely accept the move and flip the spin $s_k \to -s_k$. If $E_\nu - E_\mu > 0$ we still may want to flip the spin. The Metropolis algorithm tells us to flip it with probability $A(\mu \to \nu) = \mathrm{e}^{-\beta(E_\nu - E_\mu)}$. We can do this as follows. We evaluate the acceptance ratio $A(\mu \to \nu)$ using our value of $E_\nu - E_\mu$ from Equation (3.10), and then we choose a random number $r$ between zero and one. (Strictly the number can be equal to zero, but it must be less than one: $0 \leq r < 1$.) If that number is *less* than our acceptance ratio, $r < A(\mu \to \nu)$, then we flip the spin. If it isn't, we leave the spin alone.

And that is our complete algorithm. Now we just keep on repeating the same calculations over and over again, choosing a spin, calculating the energy change we would get if we flipped it, and then deciding whether to flip it according to Equation (3.7). Actually, there is one other trick that we can pull that makes our algorithm a bit faster still. (In fact, on most computers it will make it a lot faster.) One of the slowest parts of the algorithm as we have described it is the calculation of the exponential, which we have to perform if the energy of the new state we choose is greater than that of the current state. Calculating exponentials on a computer is usually done using a polynomial approximation which involves performing a number of floating-point multiplications and additions, and can take a considerable amount of time. We can save ourselves this effort, and thereby speed up our simulation, if we notice that the quantity, Equation (3.10), which we are calculating the exponential of, can only take a rather small number of values. Each of the terms in the sum can only take the values $+1$ and $-1$. So the entire sum, which has $z$ terms, can only take the values $-z, -z+2, -z+4 \ldots$ and so on up to $+z$—a total of $z+1$ possible values. And we only actually need to calculate the exponential when the sum is negative (see Equation (3.7) again), so in fact there are only $\frac{1}{2}z$ values of $E_\mu - E_\nu$ for which we ever need to calculate exponentials. Thus, it makes good sense to calculate the values of these $\frac{1}{2}z$ exponentials before we start the calculation proper, and store them in the computer's memory (usually in an array), where we can

simply look them up when we need them during the simulation. We pay the one-time cost of evaluating them at the beginning, and save a great deal more by never having to evaluate any exponentials again during the rest of the simulation. Not only does this save us the effort of evaluating all those exponentials, it also means that we hardly have to perform any floating-point arithmetic during the simulation. The only floating-point calculations will be in the generation of the random number $r$. (We discuss techniques for doing this in Chapter 16.) All the other calculations involve only integers, which on most computers are much quicker to deal with than real numbers.

## 3.2 Equilibration

So what do we do with our Monte Carlo program for the Ising model, once we have written it? Well, we probably want to know the answer to some questions like "What is the magnetization at such-and-such a temperature?", or "How does the internal energy behave with temperature over such-and-such a range?" To answer these questions we have to do two things. First we have to run our simulation for a suitably long period of time until it has come to equilibrium at the temperature we are interested in—this period is called the **equilibration time** $\tau_{\text{eq}}$—and then we have to measure the quantity we are interested in over another suitably long period of time and average it, to evaluate the estimator of that quantity (see Equation (2.4)). This leads us to several other questions. What exactly do we mean by "allowing the system to come to equilibrium"? And how long is a "suitably long" time for it to happen? How do we go about measuring our quantity of interest, and how long do we have to average over to get a result of a desired degree of accuracy? These are very general questions which we need to consider every time we do a Monte Carlo calculation. Although we will be discussing them here using our Ising model simulation as an example, the conclusions we will draw in this and the following sections are applicable to all equilibrium Monte Carlo calculations. These sections are some of the most important in this book.

As we discussed in Section 1.2, "equilibrium" means that the average probability of finding our system in any particular state $\mu$ is proportional to the Boltzmann weight $\mathrm{e}^{-\beta E_\mu}$ of that state. If we start our system off in a state such as the $T = 0$ or $T = \infty$ states described in the last section and we want to perform a simulation at some finite non-zero temperature, it will take a little time before we reach equilibrium. To see this, recall that, as we demonstrated in Section 1.2.1, a system at equilibrium spends the overwhelming majority of its time in a small subset of states in which its internal energy and other properties take a narrow range of values. In order to get a good estimate of the equilibrium value of any property of the system therefore, we need to wait until it has found its way to one of the states that

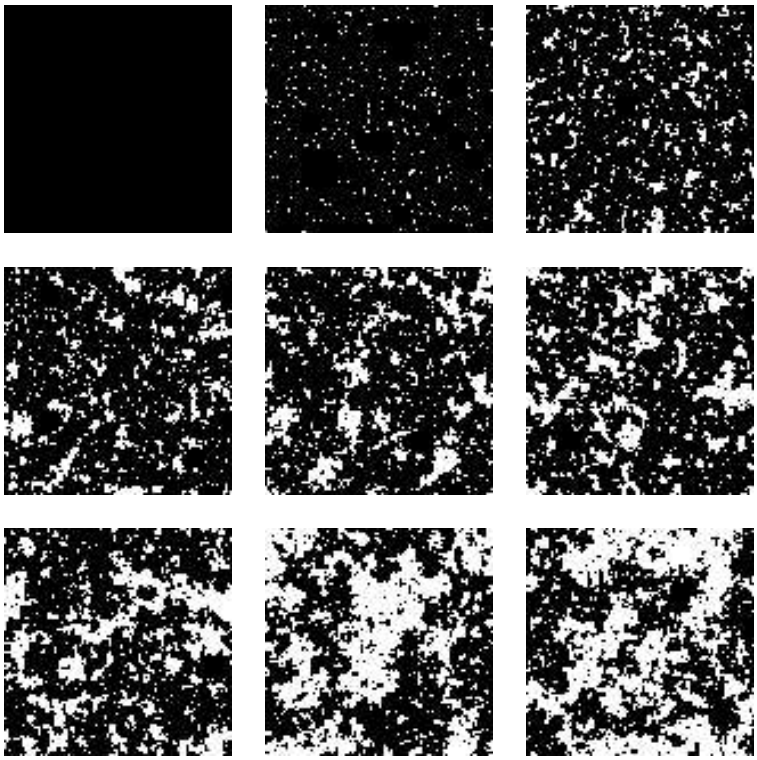

FIGURE 3.2 Nine snapshots of a $100 \times 100$ Ising model on a square lattice with $J = 1$ coming to equilibrium at a temperature $T = 2.4$ using the Metropolis algorithm. In these pictures the up-spins ($s_i = +1$) are represented by black squares and the down-spins ($s_i = -1$) by white ones. The starting configuration is one in which all the spins are pointing up. The progression of the figures is horizontally across the top row, then the middle row, then the bottom one. They show the lattice after 0, 1, 2, 4, 6, 10, 20, 40 and 100 times 100 000 steps of the simulation. In the last frame the system has reached equilibrium according to the criteria given in this section.

fall in this narrow range. Then, we assume, the Monte Carlo algorithm we have designed will ensure that it stays roughly within that range for the rest of the simulation—it should do since we designed the algorithm specifically to simulate the behaviour of the system at equilibrium. But it may take some time to find a state that lies within the correct range. In the version of the Metropolis algorithm which we have described here, we can only flip one spin at a time, and since we are choosing the spins we flip at random, it could take quite a while before we hit on the correct sequence of spins to

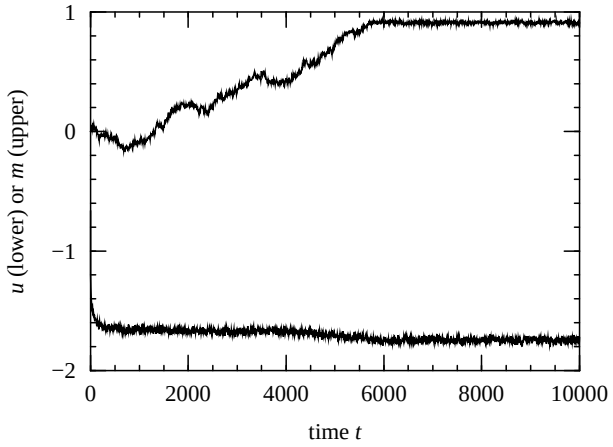


FIGURE 3.3 The magnetization (upper curve) and internal energy (lower curve) per site of a two-dimensional Ising model on a square lattice of $100 \times 100$ sites with $J = 1$ simulated using the Metropolis algorithm of Section 3.1. The simulation was started at $T = \infty$ (i.e., the initial states of the spins were chosen completely at random) and "cooled" to equilibrium at $T = 2.0$. Time is measured in Monte Carlo steps per lattice site, and equilibrium is reached after about 6000 steps per site (in other words, $6 \times 10^7$ steps altogether).

flip in order to get us to one of the states we want to be in. At the very least we can expect it to take about $N$ Monte Carlo steps to reach a state in the appropriate energy range, where $N$ is the number of spins on the lattice, since we need to allow every spin the chance to flip at least once. In Figure 3.2 we show a succession of states of a two-dimension Ising model on a square lattice of $100 \times 100$ spins with $J = 1$, as it is "warmed" up to a temperature $T = 2.4$ from an initial $T = 0$ state in which all the spins are aligned. In these pictures the $+1$ and $-1$ spins are depicted as black and white squares. By the time we reach the last frame out of nine, the system has equilibrated. The whole process takes on the order of $10^7$ steps in this case.

However looking at pictures of the lattice is not a reliable way of gauging when the system has come to equilibrium. A better way, which takes very little extra effort, is to plot a graph of some quantity of interest, like the magnetization per spin $m$ of the system or the energy of the system $E$, as a function of time from the start of the simulation. We have done this in Figure 3.3. (We will discuss the best ways of measuring these quantities in the next section, but for the moment let's just assume that we calculate them directly. For example, the energy of a given state can be calculated by

feeding all the values of the spins $s_i$ into the Hamiltonian, Equation (3.1).) It is not hard to guess simply by looking at this graph that the system came to equilibrium at around time $t = 6000$. Up until this point the energy and the magnetization are changing, but after this point they just fluctuate around a steady average value.

The horizontal axis in Figure 3.3 measures time in Monte Carlo steps *per lattice site*, which is the normal practice for simulations of this kind. The reason is that if time is measured in this way, then the average frequency with which any particular spin is selected for flipping is independent of the total number of spins $N$ on the lattice. This average frequency is called the "attempt frequency" for the spin. In the simulation we are considering here the attempt frequency has the value 1. It is natural that we should arrange for the attempt frequency to be independent of the lattice size; in an experimental system, the rate at which spins or atoms or molecules change from one state to another does not depend on how many there are in the whole system. An atom in a tiny sample will change state as often as one in a sample the size of a house. Attempt frequencies are discussed further in Section 11.1.1.

When we perform $N$ Monte Carlo steps—one for each spin in the system, on average—we say we have completed one **sweep** of the lattice. We could therefore also say that the time axis of Figure 3.3 was calibrated in sweeps.

Judging the equilibration of a system by eye from a plot such as Figure 3.3 is a reasonable method, provided we know that the system will come to equilibrium in a smooth and predictable fashion as it does in this case. The trouble is that we usually know no such thing. In many cases it is possible for the system to get stuck in some metastable region of its state space for a while, giving roughly constant values for all the quantities we are observing and so appearing to have reached equilibrium. In statistical mechanical terms, there can be a **local energy minimum** in which the system can remain temporarily, and we may mistake this for the **global energy minimum**, which is the region of state space that the equilibrium system is most likely to inhabit. (These ideas are discussed in more detail in the first few sections of Chapter 6.) To avoid this potential pitfall, we commonly adopt a different strategy for determining the equilibration time, in which we perform two different simulations of the same system, starting them in different initial states. In the case of the Ising model we might, for example, start one in the $T = 0$ state with all spins aligned, and one in the $T = \infty$ state with random spins. Or we could choose two different $T = \infty$ random-spin states. We should also run the two simulations with different "seeds" for the random number generator (see Section 16.1.2), to ensure that they take different paths to equilibrium. Then we watch the value of the magnetization or energy or other quantity in the two systems and when we see them reach the same approximately constant value, we deduce that

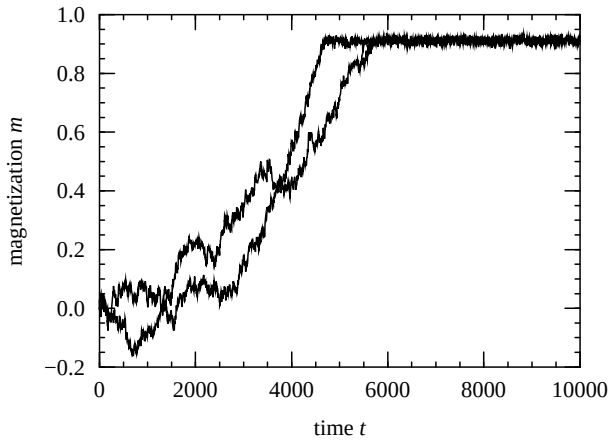


FIGURE 3.4 The magnetization of our $100 \times 100$ Ising model as a function of time (measured in Monte Carlo steps per lattice site) for two different simulations using the Metropolis algorithm. The two simulations were started off in two different $T = \infty$ (random-spin) states. By about time $t = 6000$ the two simulations have converged to the same value of the mean magnetization, within the statistical errors due to fluctuations, and so we conclude that both have equilibrated.

both systems have reached equilibrium. We have done this for two $100 \times 100$ Ising systems in Figure 3.4. Again, we clearly see that it takes about 6000 Monte Carlo steps for the two systems to reach a consensus about the value of the magnetization. This technique avoids the problem mentioned above, since if one of the systems finds itself in some metastable region, and the other reaches equilibrium or gets stuck in another metastable region, this will be apparent from the graph, because the magnetization (or other quantity) will take different values for the two systems. Only in the unlikely event that the two systems coincidentally become trapped in the same metastable region (for example, if we choose two initial states that are too similar to one another) will we be misled into thinking they have reached equilibrium when they haven't. If we are worried about this possibility, we can run *three* different simulations from different starting points, or four, or five. Usually, however, two is sufficient.

## 3.3 Measurement

Once we are sure the system has reached equilibrium, we need to measure whatever quantity it is that we are interested in. The most likely candidates

for the Ising model are the energy and the magnetization of the system. As we pointed out above, the energy $E_\mu$ of the current state $\mu$ of the system can be evaluated directly from the Hamiltonian by substituting in the values of the spins $s_i$ from our array of integers. However, this is not an especially efficient way of doing it, and there is a much better way. As part of our implementation of the Metropolis algorithm, you will recall we calculated the energy difference $\Delta E = E_\nu - E_\mu$ in going from state $\mu$ to state $\nu$ (see Equation (3.10)). So, if we know the energy of the current state $\mu$, we can calculate the new energy when we flip a spin, using only a single addition:

$$E_\nu = E_\mu + \Delta E. \tag{3.11}$$

So the clever thing to do is to calculate the energy of the system from the Hamiltonian at the very start of the simulation, and then every time we flip a spin calculate the new energy from Equation (3.11) using the value of $\Delta E$, which we have to calculate anyway.

Calculating the magnetization is even easier. The total magnetization $M_\mu$ of the whole system in state $\mu$ (as opposed to the magnetization *per spin*—we'll calculate that in a moment), is given by the sum

$$M_\mu = \sum_i s_i^\mu. \tag{3.12}$$

As with the energy, it is not a shrewd idea to evaluate the magnetization directly from this sum every time we want to know it. It is much better to notice that only one spin $k$ flips at a time in the Metropolis algorithm, so the change of magnetization from state $\mu$ to state $\nu$ is

$$\Delta M = M_\nu - M_\mu = \sum_i s_i^\nu - \sum_i s_i^\mu = s_k^\nu - s_k^\mu = 2s_k^\nu, \tag{3.13}$$

where the last equality follows from Equation (3.9). Thus, the clever way to evaluate the magnetization is to calculate its value at the beginning of the simulation, and then make use of the formula

$$M_\nu = M_\mu + \Delta M = M_\mu + 2s_k^\nu \tag{3.14}$$

every time we flip a spin.[2]

[2]However, to be absolutely fair, we should point out that doing this involves performing at least one addition operation every time we flip a spin, or one addition every $\bar{A}^{-1}$ steps, where $\bar{A}$ is the mean acceptance ratio. Direct evaluation of Equation (3.12) on the other hand involves $N$ additions every time we want to know the magnetization. Thus, if we want to make measurements less often than once every $N/\bar{A}$ steps, it may pay to use the direct method rather than employing Equation (3.14). Similar considerations apply to the measurement of the energy also.

Given the energy and the magnetization of our Ising system at a selection of times during the simulation, we can average them to find the estimators of the internal energy and average magnetization. Then dividing these figures by the number of sites $N$ gives us the internal energy and average magnetization per site.

We can also average the squares of the energy and magnetization to find quantities like the specific heat and the magnetic susceptibility:

$$c = \frac{\beta^2}{N}(\langle E^2\rangle - \langle E\rangle^2), \tag{3.15}$$

$$\chi = \beta N(\langle m^2\rangle - \langle m\rangle^2). \tag{3.16}$$

(See Equations (1.36) and (1.37). Note that we have set $k = 1$ again.)

In order to average quantities like $E$ and $M$, we need to know how long a run we have to average them over to get a good estimate of their expectation values. One simple solution would again be just to look at a graph like Figure 3.3 and guess how long we need to wait. However (as you might imagine) this is not a very satisfactory solution. What we really need is a measure of the **correlation time** $\tau$ of the simulation. The correlation time is a measure of how long it takes the system to get from one state to another one which is significantly different from the first, i.e., a state in which the number of spins which are the same as in the initial state is no more than what you would expect to find just by chance. (We will give a more rigorous definition in a moment.) There are a number of ways to estimate the correlation time. One that is sometimes used is just to assume that it is equal to the equilibration time. This is usually a fairly safe assumption:[3] usually the equilibration time is considerably longer than the correlation time, $\tau_{\text{eq}} > \tau$, because two states close to equilibrium are qualitatively more similar than a state far from equilibrium (like the $T = 0$ or $T = \infty$ states we suggested for starting this simulation with) and one close to equilibrium. However, this is again a rather unrigorous supposition, and there are more watertight ways to estimate $\tau$. The most direct of these is to calculate the "time-displaced autocorrelation function" of some property of the model.

### 3.3.1 Autocorrelation functions

Let us take the example of the magnetization $m$ of our Ising model. The **time-displaced autocorrelation** $\chi(t)$ of the magnetization is given by

$$\begin{aligned}\chi(t) &= \int \mathrm{d}t' \, [m(t') - \langle m\rangle][m(t'+t) - \langle m\rangle] \\ &= \int \mathrm{d}t' \, \bigl[m(t')m(t'+t) - \langle m\rangle^2\bigr].\end{aligned} \tag{3.17}$$

[3] In particular, it works fine for the Metropolis simulation of the Ising model which we are considering here.

where $m(t)$ is the instantaneous value of the magnetization at time $t$ and $\langle m \rangle$ is the average value. This is rather similar to the connected correlation function which we defined in Equation (1.26). That measured the correlation between the values of a quantity (such as the magnetization) on two different sites, $i$ and $j$, on the lattice. The autocorrelation gives us a similar measure of the correlation at two different times, one an interval $t$ later than the other. If we measure the difference between the magnetization $m(t')$ at time $t'$ and its mean value, and then we do the same thing at time $t' + t$, and we multiply them together, we will get a positive value if they were fluctuating in the same direction at those two times, and a negative one if they were fluctuating in opposite directions. If we then integrate over time as in Equation (3.17), then $\chi(t)$ will take a non-zero value if on average the fluctuations are correlated, or it will be zero if they are not. For our Metropolis simulation of the Ising model it is clear that if we measure the magnetization at two times just a single Monte Carlo step apart, the values we get will be very similar, so we will have a large positive autocorrelation. On the other hand, for two times a long way apart the magnetizations will probably be totally unrelated, and their autocorrelation will be close to zero. Ideally, we should calculate $\chi(t)$ by integrating over an infinite time, but this is obviously impractical in a simulation, so we do the best we can and just sum over all the measurements of $m$ that we have, from beginning to end of our run. Figure 3.5 shows the magnetization autocorrelation of our $100 \times 100$ Ising model at temperature $T = 2.4$ and interaction energy $J = 1$, calculated in exactly this manner using results from our Metropolis Monte Carlo simulation. As we can see, the autocorrelation does indeed drop from a significant non-zero value at short times $t$ towards zero at very long times. In this case, we have divided $\chi(t)$ by its value $\chi(0)$ at $t = 0$, so that its maximum value is one. The typical time-scale (if there is one) on which it falls off is a measure of the correlation time $\tau$ of the simulation. In fact, this is the *definition* of the correlation time. It is the typical time-scale on which the autocorrelation drops off; the autocorrelation is expected to fall off exponentially at long times thus:

$$\chi(t) \sim \mathrm{e}^{-t/\tau}. \tag{3.18}$$

(In the next section we show why it should take this form.) With this definition, we see that in fact there is still a significant correlation between two samples taken a correlation time apart: at time $t = \tau$ the autocorrelation function, which is a measure of the similarity of the two states, is only a factor of $1/\mathrm{e}$ down from its maximum value at $t = 0$. If we want truly independent samples then, we may want to draw them at intervals of greater than one correlation time. In fact, the most natural definition of statistical independence turns out to be samples drawn at intervals of $2\tau$. We discuss this point further in Section 3.4.1.

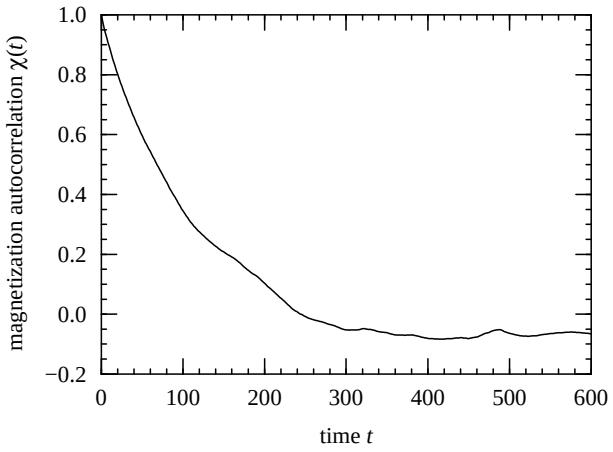


FIGURE 3.5 The magnetization autocorrelation function $\chi(t)$ for a two-dimensional Ising model at temperature $T = 2.4$ on a square lattice of $100 \times 100$ sites with $J = 1$ simulated using the Metropolis algorithm of Section 3.1. Time is measured in Monte Carlo steps per site.

We can make a reasonable estimate of $\tau$ by eye from Figure 3.5. At a guess we'd probably say $\tau$ was about 100 in this case. This is an accurate enough figure for estimating how long a Monte Carlo run we need to do in order to get decent statistics. It tells us that we expect to get a new independent spin configuration about once every $2\tau = 200$ sweeps of the lattice in our simulation. So if we want, say, 10 independent measurements of the magnetization, we need to run our Metropolis algorithm for about 2000 sweeps after equilibration, or $2 \times 10^7$ Monte Carlo steps. If we want 100 measurements we need to do $2 \times 10^8$ steps. In general, if a run lasts a time $t_{\max}$, then the number of independent measurements we can get out of the run, after waiting a time $\tau_{\rm eq}$ for equilibration, is on the order of

$$n = \frac{t_{\max}}{2\tau}. \tag{3.19}$$

It is normal practice in a Monte Carlo simulation to make measurements at intervals of less than the correlation time. For example, in the case of the Metropolis algorithm we might make one measurement every sweep of the lattice. Thus the total number of measurements we make of magnetization or energy (or whatever) during the run is usually greater than the number of independent measurements. There are a number of reasons why we do it this way. First of all, we usually don't know what the correlation time is until after the simulation has finished, or at least until after it has run

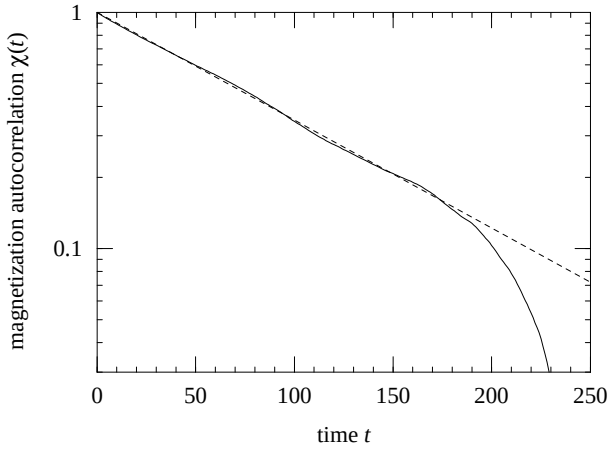


FIGURE 3.6 The autocorrelation function of Figure 3.5 replotted on semi-logarithmic axes. The dashed line is a straight line fit which yields a figure of $\tau = 95 \pm 5$ in this case. Note that the horizontal scale is not the same as in Figure 3.5.

for a certain amount of time, and we want to be sure of having at least one measurement every two correlation times. Another reason is that we want to be able to calculate the autocorrelation function for times less than a correlation time, so that we can use it to make an accurate estimate of the correlation time. If we only had one measurement every $2\tau$, we wouldn't be able to calculate $\tau$ with any accuracy at all.

If we want a more reliable figure for $\tau$, we can replot our autocorrelation function on semi-logarithmic axes as we have done in Figure 3.6, so that the slope of the line gives us the correlation time. Then we can estimate $\tau$ by fitting the straight-line portion of the plot using a least-squares method. The dotted line in the figure is just such a fit and its slope gives us a figure of $\tau = 95 \pm 5$ for the correlation time in this case.

An alternative is to calculate the **integrated correlation time**.[4] If we assume that Equation (3.18) is accurate for all times $t$ then

$$\int_0^\infty \frac{\chi(t)}{\chi(0)}\,\mathrm{d}t = \int_0^\infty \mathrm{e}^{-t/\tau}\,\mathrm{d}t = \tau. \tag{3.20}$$

This form has a number of advantages. First, it is often easier to apply Equation (3.20) than it is to perform the exponential fit to the autocorrelation

[4]This is a rather poor name for this quantity, since it is not the correlation time that is integrated but the autocorrelation function. However, it is the name in common use so we use it here too.

function. Second, the method for estimating $\tau$ illustrated in Figure 3.6 is rather sensitive to the range over which we perform the fit. In particular, the very long time behaviour of the autocorrelation function is often noisy, and it is important to exclude this portion from the fitted data. However, the exact point at which we truncate the fit can have quite a large effect on the resulting value for $\tau$. The integrated correlation time is much less sensitive to the way in which the data are truncated, although it is by no means perfect either, since, as we will demonstrate in Section 3.3.2, Equation (3.18) is only strictly correct for long times $t$, and we introduce an uncontrolled error by assuming it to be true for all times. On the other hand, the direct fitting method also suffers from this problem, unless we only perform our fit over the exact range of times for which true exponential behaviour is present. Normally, we don't know what this range is, so the fitting method is no more accurate than calculating the integrated correlation time. Usually then, Equation (3.20) is the method of choice for calculating $\tau$. Applying it to the data from Figure 3.6 gives a figure of $\tau = 86 \pm 5$, which is in moderately good agreement with our previous figure.

The autocorrelation function in Figure 3.5 was calculated directly from a discrete form of Equation (3.17). If we have a set of samples of the magnetization $m(t)$ measured at evenly-spaced times up to some maximum time $t_{\max}$, then the correct formula for the autocorrelation function is[5]

$$\begin{aligned}\chi(t) = {} & \frac{1}{t_{\max}-t}\sum_{t'=0}^{t_{\max}-t} m(t')\,m(t'+t) \\ & - \frac{1}{t_{\max}-t}\sum_{t'=0}^{t_{\max}-t} m(t') \times \frac{1}{t_{\max}-t}\sum_{t'=0}^{t_{\max}-t} m(t'+t).\end{aligned} \tag{3.21}$$

Notice how we have evaluated the mean magnetization $m$ in the second term using the same subsets of the data that we used in the first term. This is not strictly speaking necessary, but it makes $\chi(t)$ a little better behaved. In Figure 3.5 we have also normalized $\chi(t)$ by dividing throughout by $\chi(0)$, but this is optional. We've just done it for neatness.

Note that one should be careful about using Equation (3.21) to evaluate $\chi(t)$ at long times. When $t$ gets close to $t_{\max}$, the upper limit of the sums becomes small and we end up integrating over a rather small time interval to get our answer. This means that the statistical errors in $\chi(t)$ due to the random nature of the fluctuations in $m(t)$ may become large. A really satisfactory simulation would always run for many correlation times, in which case we will probably not be interested in the very tails of $\chi(t)$, since the correlations will have died away by then, by definition. However, it is not

[5] In fact, this formula differs from (3.17) by a multiplicative constant, but this makes no difference as far as the calculation of the correlation time is concerned.

always possible, because we only have limited computer resources, to perform simulations as long as we would like, and one should always be aware that errors of this type can crop up with shorter data sets.

Calculating the autocorrelation function from Equation (3.21) takes time of order $n^2$, where $n$ is the number of samples. For most applications, this is not a problem. Even for simulations where several thousand samples are taken, the time needed to evaluate the autocorrelation is only a few seconds on a modern computer, and the simplicity of the formulae makes their programming very straightforward, which is a big advantage—people-time is usually more expensive than computer-time. However, sometimes it is desirable to calculate the autocorrelation function more quickly. This is particularly the case when one needs to do it very often during the course of a calculation, for some reason. If you need to calculate an autocorrelation a thousand times, and each time takes a few seconds on the computer, then the seconds start to add up. In this case, at the expense of rather greater programming effort, we can often speed up the process by the following trick. Instead of calculating $\chi(t)$ directly, we calculate the Fourier transform $\widetilde{\chi}(\omega)$ of the autocorrelation and then we invert the Fourier transform to get $\chi(t)$. The Fourier transform is related to the magnetization as follows:

$$\begin{aligned}\widetilde{\chi}(\omega) &= \int \mathrm{d}t\, \mathrm{e}^{\mathrm{i}\omega t} \int \mathrm{d}t'\, [m(t') - \langle m\rangle][m(t'+t) - \langle m\rangle] \\ &= \int \mathrm{d}t \int \mathrm{d}t'\, \mathrm{e}^{-\mathrm{i}\omega t'} [m(t') - \langle m\rangle]\, \mathrm{e}^{\mathrm{i}\omega(t'+t)} [m(t'+t) - \langle m\rangle] \\ &= \widetilde{m}'(\omega)\, \widetilde{m}'(-\omega) = \bigl|\widetilde{m}'(\omega)\bigr|^2, \end{aligned} \tag{3.22}$$

where $\widetilde{m}'(\omega)$ is the Fourier transform of $m'(t) \equiv m(t) - \langle m\rangle$.[6]

So all we need to do is calculate the Fourier transform of $m'(t)$ and feed it into this formula to get $\widetilde{\chi}(\omega)$. The advantage in doing this is that the Fourier transform can be evaluated using the so-called **fast Fourier transform** or FFT algorithm, which was given by Cooley and Tukey in 1965. This is a standard algorithm which can evaluate the Fourier transform in a time which goes like $n \log n$, where $n$ is the number of measurements of the magnetization.[7] Furthermore, there exist a large number of ready-made computer software packages that perform FFTs. These packages have

[6] Since $m(t)$ and $m'(t)$ differ only by a constant, $\widetilde{m}(\omega)$ and $\widetilde{m}'(\omega)$ differ only in their $\omega = 0$ component (which is zero in the latter case, but may be non-zero in the former). For this reason, it is often simplest to calculate $\widetilde{m}'(\omega)$ by first calculating $\widetilde{m}(\omega)$ and then just setting the $\omega = 0$ component to zero.

[7] On a technical note, if we simply apply the FFT algorithm directly to our magnetization data, the result produced is the Fourier transform of an infinite periodic repetition of the data set, which is not quite what we want in Equation (3.22). A simple way of getting around this problem is to add $n$ zeros to the end of the data set before we perform the transform. It can be shown that this then gives a good estimate of the autocorrelation function (Futrelle and McGinty 1971).

been written very carefully by people who understand the exact workings of the computer and are designed to be as fast as possible at performing this particular calculation, so it usually saves us much time and effort to make use of the programs in these packages. Having calculated $\widetilde{\chi}(\omega)$, we can then invert the Fourier transform, again using a streamlined inverse FFT routine which also runs in time proportional to $n \log n$, and so recover the function $\chi(t)$.

In fact, if we want to calculate the integrated correlation time, Equation (3.20), then we don't need to invert the Fourier transform, since

$$\widetilde{\chi}(0) = \int_0^\infty \chi(t)\,\mathrm{d}t. \tag{3.23}$$

Thus

$$\tau = \frac{\widetilde{\chi}(0)}{\chi(0)}, \tag{3.24}$$

and $\chi(0)$ is simply the fluctuation

$$\chi(0) = \langle m^2 \rangle - \langle m \rangle^2 \tag{3.25}$$

in the magnetization, which we can calculate directly in time proportional to $n$.

### 3.3.2 Correlation times and Markov matrices

The techniques outlined in the previous section are in most cases quite sufficient for estimating correlation times in Monte Carlo simulations. However, we have simplified the discussion somewhat by supposing there to be only one correlation time in the system. In real life there are as many correlation times as there are states of the system, and the interplay between these different times can sometimes cause the methods of the last section to give inaccurate results. In this section we look at the situation in a little more detail and show how one can extract accurate estimates of the correlation time $\tau$ from simulation data. The reader who is not (for the moment) interested in such accurate estimates, but only in getting a rougher measure of $\tau$, could reasonably skip this section.

In Section 2.2.3 we showed that the probabilities $w_\mu(t)$ and $w_\mu(t+1)$ of being in a particular state $\mu$ at consecutive Monte Carlo steps are related by the Markov matrix $\mathbf{P}$ for the algorithm. In matrix notation we wrote

$$\mathbf{w}(t+1) = \mathbf{P} \cdot \mathbf{w}(t), \tag{3.26}$$

where $\mathbf{w}$ is the vector whose elements are the probabilities $w_\mu$ (see Equation (2.9)). By iterating this equation from time $t = 0$ we can then show that

$$\mathbf{w}(t) = \mathbf{P}^t \cdot \mathbf{w}(0). \tag{3.27}$$

Now $\mathbf{w}(0)$ can be expressed as a linear combination of the right eigenvectors $\mathbf{v}_i$ of $\mathbf{P}$ thus:[8]

$$\mathbf{w}(0) = \sum_i a_i \mathbf{v}_i, \tag{3.28}$$

where the quantities $a_i$ are coefficients whose values depend on the configuration of the system at $t = 0$. Then

$$\mathbf{w}(t) = \mathbf{P}^t \cdot \sum_i a_i \mathbf{v}_i = \sum_i a_i \lambda_i^t \mathbf{v}_i, \tag{3.29}$$

where $\lambda_i$ is the eigenvalue of $\mathbf{P}$ corresponding to the eigenvector $\mathbf{v}_i$. As $t \to \infty$, the right-hand side of this equation will be dominated by the term involving the largest eigenvalue $\lambda_0$ of the Markov matrix. This means that in the limit of long times, the probability distribution $\mathbf{w}(t)$ becomes proportional to $\mathbf{v}_0$, the eigenvector corresponding to the largest eigenvalue. We made use of this result in Section 2.2.3 to demonstrate that $\mathbf{w}(t)$ tends to the Boltzmann distribution at long times.

Now suppose that we are interested in knowing the value of some observable quantity $Q$, such as the magnetization. The expectation value of this quantity at time $t$ can be calculated from the formula

$$Q(t) = \sum_\mu w_\mu(t)\, Q_\mu \tag{3.30}$$

or

$$Q(t) = \mathbf{q} \cdot \mathbf{w}(t), \tag{3.31}$$

where $\mathbf{q}$ is the vector whose elements are the values $Q_\mu$ of the quantity in the various states of the system. Substituting Equation (3.29) into (3.31) we then get

$$Q(t) = \sum_i a_i \lambda_i^t\, \mathbf{q} \cdot \mathbf{v}_i = \sum_i a_i \lambda_i^t\, q_i. \tag{3.32}$$

Here $q_i \equiv \mathbf{q} \cdot \mathbf{v}_i$ is the expectation value of $Q$ in the $i^{\text{th}}$ eigenstate. The long time limit $Q(\infty)$ of this expression is also dominated by the largest eigenvalue $\lambda_0$ and is proportional to $q_0$. If we now define a set of quantities $\tau_i$ thus:

$$\tau_i = -\frac{1}{\log \lambda_i} \tag{3.33}$$

for all $i \neq 0$, then Equation (3.32) can be written

$$Q(t) = Q(\infty) + \sum_{i \neq 0} a_i q_i \mathrm{e}^{-t/\tau_i}. \tag{3.34}$$

[8] $\mathbf{P}$ is in general not symmetric, so its right and left eigenvectors are not the same.

The quantities $\tau_i$ are the correlation times for the system, as we can demonstrate by showing that they also govern the decay of the autocorrelation function. Noting that the long-time limit $Q(\infty)$ of the expectation of $Q$ is none other than the equilibrium expectation $\langle Q \rangle$, we can write the autocorrelation of $Q$ as the correlation between the expectations at zero time and some later time $t$ thus:

$$\chi(t) = [Q(0) - Q(\infty)][Q(t) - Q(\infty)] = \sum_{i \neq 0} b_i \mathrm{e}^{-t/\tau_i}, \tag{3.35}$$

with

$$b_i = \sum_{j \neq 0} a_i a_j q_i q_j. \tag{3.36}$$

Equation (3.35) is the appropriate generalization of Equation (3.18) to all times $t$ (not just long ones).

As we said, there are as many correlation times as there are states of the system since that is the rank of the matrix $\mathbf{P}$. (Well, strictly there are as many of them as the rank of the matrix *less one*, since there is no $\tau_0$ corresponding to the highest eigenvalue. There are $2^N - 1$ correlation times in the case of the Ising model, for example. However, the rank of the matrix is usually very large, so let's not quibble over one correlation time.) The longest of these correlation times is $\tau_1$, the one which corresponds to the second largest eigenvalue of the matrix. This is the correlation time we called $\tau$ in the last section. Clearly, for large enough times $t$, this will be the only correlation time we need to worry about, since all the other terms in Equation (3.35) will have decayed away to insignificance. (This is how Equation (3.18) is derived.) However, depending on how close together the higher eigenvalues of the Markov matrix are, and how long our simulation runs for, we may or may not be able to extract reliable results for $\tau_1$ by simply ignoring all the other terms. In general the most accurate results are obtained by fitting our autocorrelation function to a sum of a small number of decaying exponentials, of the form of Equation (3.35), choosing values for the quantities $b_i$ by a least-squares or similar method. In work on the three-dimensional Ising model, for example, Wansleben and Landau (1991) showed that including three terms was sufficient to get a good fit to the magnetization autocorrelation function, and thus get an accurate measure of the longest correlation time $\tau \equiv \tau_1$. In studies of the dynamical properties of statistical mechanical models, this is the most correct way to measure the correlation time. Strictly speaking it gives only a lower bound on $\tau$ since it is always possible that correlations exist beyond the longest times that one can measure. However, in practice it usually gives good results.

## 3.4 Calculation of errors

Normally, as well as measuring expectation values, we also want to calculate the errors on those values, so that we have an idea of how accurate they are. As with experiments, the errors on Monte Carlo results divide into two classes: **statistical errors** and **systematic errors**.[9] Statistical errors are errors which arise as a result of random changes in the simulated system from measurement to measurement—thermal fluctuations, for example—and they can be estimated simply by taking many measurements of the quantity we are interested in and calculating the spread of the values. Systematic errors on the other hand, are errors due to the procedure we have used to make the measurements, and they affect the whole simulation. An example is the error introduced by waiting only a finite amount of time for our system to equilibrate. (Ideally, we should allow an infinite amount of time for this, in order to be sure the system has completely equilibrated. However, this, of course, is not practical.)

### 3.4.1 Estimation of statistical errors

In a Monte Carlo calculation the principal source of statistical error in the measured value of a quantity is usually the fluctuation of that quantity from one time step to the next. This error is inherent in the Monte Carlo method. As the name "Monte Carlo" itself makes clear, there is an innate randomness and statistical nature to Monte Carlo calculations. (In Chapter 6 on glassy spin models, we will see another source of statistical error: "sample-to-sample" fluctuations in the actual system being simulated. However, for the simple Ising model we have been considering, thermal fluctuations are the only source of statistical error. All other errors fall into the category of systematic errors.) It is often straightforward to estimate the statistical error in a measured quantity, since the assumption that the error is statistical—i.e., that it arises through random deviations in the measured value of the quantity—implies that we can estimate the true value by taking the mean of several different measurements, and that the error on that estimate is simply the error on the mean. Thus, if we are performing the Ising model simulation described in the last section, and we make $n$ measurements $m_i$ of the magnetization of the system during a particular run, then our best estimate of the true thermal average of the magnetization is the mean $\overline{m}$ of those $n$ measurements (which is just the estimator of $m$, as defined in Section 2.1),

[9] Monte Carlo simulations are in many ways rather similar to experiments. It often helps to regard them as "computer experiments", and analyse the results in the same way as we would analyse the results of a laboratory experiment.

and our best estimate of the standard deviation on the mean is given by[10]

$$\sigma = \sqrt{\frac{\frac{1}{n}\sum_{i=0}^{n}(m_i - \overline{m})^2}{n-1}} = \sqrt{\frac{1}{n-1}(\overline{m^2} - \overline{m}^2)}. \tag{3.37}$$

This expression assumes that our samples $m_i$ are statistically independent, which in general they won't be. As we pointed out in Section 3.3.1, it is normal to sample at intervals of less than a correlation time, which means that successive samples will in general be correlated. A simple and usually perfectly adequate solution to this problem is to use the value of $n$ given by Equation (3.19), rather than the actual number of samples taken. In fact, it can be shown (Müller-Krumbhaar and Binder 1973) that the correct expression for $\sigma$ in terms of the actual number of samples is

$$\sigma = \sqrt{\frac{1 + 2\tau/\Delta t}{n-1}(\overline{m^2} - \overline{m}^2)}, \tag{3.38}$$

where $\tau$ is the correlation time and $\Delta t$ is the time interval at which the samples were taken. Clearly this becomes equal to Equation (3.37) when $\Delta t \gg \tau$, but more often we have $\Delta t \ll \tau$. In this case, we can ignore the 1 in the numerator of Equation (3.38). Noting that for a run of length $t_{\max}$ (after equilibration) the interval $\Delta t$ is related to the total number of samples by

$$n = \frac{t_{\max}}{\Delta t}, \tag{3.39}$$

we then find that for large $n$

$$\sigma = \sqrt{\frac{2\tau}{t_{\max}}(\overline{m^2} - \overline{m}^2)}, \tag{3.40}$$

which is the same result as we would get by simply using Equation (3.19) for $n$ in Equation (3.37). This in fact was the basis for our assertion in Section 3.3.1 that the appropriate sampling interval for getting independent samples was twice the correlation time. Note that the value of $\sigma$ in Equation (3.40) is independent of the value of $\Delta t$, which means we are free to choose $\Delta t$ in whatever way is most convenient.

### 3.4.2 The blocking method

There are some cases where it is either not possible or not straightforward to estimate the error in a quantity using the direct method described in the last

[10] The origin of the $n-1$ in this and following expressions for error estimates is a little obscure. The curious reader is referred to any good book on data analysis for an explanation, such as Bevington and Robinson (1992).

section. This happens when the result we want is not merely the average of some measurement repeated many times over the course of the simulation, as the magnetization is, but is instead derived in some more complex way from measurements we make during the run. An example is the specific heat $c$, Equation (3.15), which is inherently an average macroscopic quantity. Unlike the magnetization, the specific heat is not defined at a single time step in the simulation. It is only defined in terms of averages of many measurements of $E$ and $E^2$ over a longer period of time. We might imagine that we could calculate the error on $\langle E \rangle$ and the error on $\langle E^2 \rangle$ using the techniques we employed for the magnetization, and then combine them in some fashion to give an estimate of the error in $c$. But this is not as straightforward as it seems at first, since the errors in these two quantities are correlated—when $\langle E \rangle$ goes up, so does $\langle E^2 \rangle$. It is possible to do the analysis necessary to calculate the error on $c$ in this fashion. However, it is not particularly simple, and there are other more general methods of error estimation which lend themselves to this problem. As the quantities we want to measure become more complex, these methods—"blocking", the "bootstrap" and the "jackknife"—will save us a great deal of effort in estimating errors. We illustrate these methods here for the case of the specific heat, though it should be clear that they are applicable to almost any quantity that can be measured in a Monte Carlo simulation.

The simplest of our general-purpose error estimation methods is the **blocking method**. Applied to the specific heat, the idea is that we take the measurements of $E$ that we made during the simulation and divide them into several groups, or **blocks**. We then calculate $c$ separately for each block, and the spread of values from one block to another gives us an estimate of the error. To see how this works, suppose we make 200 measurements of the energy during our Ising model simulation, and then split those into 10 groups of 20 measurements. We can evaluate the specific heat from Equation (3.15) for each group and then find the mean of those 10 results exactly as we did for the magnetization above. The error on the mean is given again by Equation (3.37), except that $n$ is now replaced by the number $n_b$ of blocks, which would be 10 in our example. This method is intuitive, and will give a reasonable estimate of the order of magnitude of the error in a quantity such as $c$. However, the estimates it gives vary depending on the number of different blocks you divide your data up into, with the smallest being associated with large numbers of blocks, and the largest with small numbers of blocks, so it is clearly not a very rigorous method. A related but more reliable method, which can be used for error estimation in a wide variety of different circumstances, is the **bootstrap method**, which we now describe.

### 3.4.3 The bootstrap method

The bootstrap method is a **resampling method**. Applied to our problem of calculating the specific heat for the Ising model, it would work like this. We take our list of measurements of the energy of the model and from the $n$ numbers in this list we pick out $n$ at random. (Strictly, $n$ should be the number of *independent* measurements. In practice the measurements made are usually not all independent, but luckily it transpires that the bootstrap method is not much affected by this difference, a point which is discussed further below.) We specifically allow ourselves to pick the same number twice from the list, so that we end up with $n$ numbers each of which appears on the original list, and some of which may be duplicates of one another. (In fact, if you do your sums, you can show that about a fraction $1-1/\mathrm{e} \simeq 63\%$ of the numbers will be duplicates.) We calculate the specific heat from these $n$ numbers just as we would normally, and then we repeat the process, picking (or **resampling**) another $n$ numbers at random from the original measurements. It can be shown (Efron 1979) that after we have repeated this calculation several times, the standard deviation of the distribution in the results for $c$ is a measure of the error in the value of $c$. In other words if we make several of these "bootstrap" calculations of the specific heat, our estimate of the error $\sigma$ is given by

$$\sigma = \sqrt{\overline{c^2} - \overline{c}^2}. \tag{3.41}$$

Notice that there is no extra factor of $1/(n-1)$ here as there was in Equation (3.37). (It is clear that the latter would not give a correct result, since it would imply that our estimate of the error could be reduced by simply resampling our data more times.)

As we mentioned, it is not necessary for the working of the bootstrap method that all the measurements made be independent in the sense of Section 3.3.1 (i.e., one every two correlation times or more). As we pointed out earlier, it is more common in a Monte Carlo simulation to make measurements at comfortably short intervals throughout the simulation so as to be sure of making at least one every correlation time or so and then calculate the number of independent measurements made using Equation (3.19). Thus the number of samples taken usually exceeds the number which actually constitute independent measurements. One of the nice things about the bootstrap method is that it is not necessary to compensate for this difference in applying the method. You get fundamentally the same estimate of the error if you simply resample your $n$ measurements from the entire set of measurements that were made. In this case, still about 63% of the samples will be duplicates of one another, but many others will effectively be duplicates as well because they will be measurements taken at times less than a correlation time apart. Nonetheless, the resulting estimate of $\sigma$ is the

same.

The bootstrap method is a good general method for estimating errors in quantities measured by Monte Carlo simulation. Although the method initially met with some opposition from mathematicians who were not convinced of its statistical validity, it is now widely accepted as giving good estimates of errors (Efron 1979).

### 3.4.4 The jackknife method

A slightly different but related method of error estimation is the **jackknife**. For this method, unlike the bootstrap method, we really do need to choose $n$ independent samples out of those that were made during the run, taking one approximately every two correlation times or more. Applying the jackknife method to the case of the specific heat, we would first use these samples to calculate a value $c$ for the specific heat. Now however, we also calculate $n$ other estimates $c_i$ as follows. We take our set of $n$ measurements, and we remove the first one, leaving $n-1$, and we calculate the specific heat $c_1$ from that subset. Then we put the first one back, but remove the second and calculate $c_2$ from that subset, and so forth. Each $c_i$ is the specific heat calculated with the $i^{\text{th}}$ measurement of the energy removed from the set, leaving $n-1$ measurements. It can then be shown that an estimate of the error in our value of $c$ is

$$\sigma = \sqrt{\sum_{i=1}^{n}(c_i - c)^2}, \tag{3.42}$$

where $c$ is our estimate of the specific heat using all the data.[11]

Both the jackknife and the bootstrap give good estimates of errors for large data sets, and as the size of the data set becomes infinite they give exact estimates. Which one we choose in a particular case usually depends on how much work is involved applying them. In order to get a decent error estimate from the bootstrap method we usually need to take at least 100 resampled sets of data, and 1000 would not be excessive. (100 would give the error to a bit better than 10% accuracy.) With the jackknife we have to recalculate the quantity we are interested in exactly $n$ times to get the error estimate. So, if $n$ is much larger than 100 or so, the bootstrap is probably the more efficient. Otherwise, we should use the jackknife.[12]

[11] In fact, it *is* possible to use the jackknife method with samples taken at intervals $\Delta t$ less than $2\tau$. In this case we just reduce the sum inside the square root by a factor of $\Delta t/2\tau$ to get an estimate of $\sigma$ which is independent of the sampling interval.

[12] For a more detailed discussion of these two methods, we refer the interested reader to the review article by Efron (1979).

### 3.4.5 Systematic errors

Just as in experiments, systematic errors are much harder to gauge than statistical ones; they don't show up in the fluctuations of the individual measurements of a quantity. (That's why they are systematic errors.) The main source of systematic error in the Ising model simulation described in this chapter is the fact that we wait only a finite amount of time for the system to equilibrate. There is no good general method for estimating systematic errors; each source of error has to be considered separately and a strategy for estimating it evolved. This is essentially what we were doing when we discussed ways of estimating the equilibration time for the Metropolis algorithm. Another possible source of systematic error would be not running the simulation for a long enough time after equilibration to make good independent measurements of the quantities of interest. When we discussed methods for estimating the correlation time $\tau$, we were dealing with this problem. In the later sections of this chapter, and indeed throughout this book, we will discuss methods for estimating and controlling systematic errors as they crop up in various situations.

## 3.5 Measuring the entropy

We have described how we go about measuring the internal energy, specific heat, magnetization and magnetic susceptibility from our Metropolis Monte Carlo simulation of the Ising model. However, there are three quantities of interest which were mentioned in Chapter 1 which we have not yet discussed: the free energy $F$, the entropy $S$ and the correlation function $G_c^{(2)}(i,j)$. The correlation function we will consider in the next section. The other two we consider now.

The free energy and the entropy are related by

$$F = U - TS \tag{3.43}$$

so that if we can calculate one, we can easily find the other using the known value of the total internal energy $U$. Normally, we calculate the entropy, which we do by integrating the specific heat over temperature as follows.

We can calculate the specific heat of our system from the fluctuations in the internal energy as described in Section 3.3. Moreover, we know that the specific heat $C$ is equal to

$$C = T\frac{\mathrm{d}S}{\mathrm{d}T}. \tag{3.44}$$

Thus the entropy $S(T)$ at temperature $T$ is

$$S(T) = S(T_0) + \int_{T_0}^{T} \frac{C}{T}\,\mathrm{d}T. \tag{3.45}$$

If we are only interested in how $S$ varies with $T$, then it is not necessary to know the value of the integration constant $S(T_0)$ and we can give it any value we like. If we want to know the absolute value of $S$, then we have to fix $S(T_0)$ by choosing $T_0$ to be some temperature at which we know the value of the entropy. The conventional choice, known as the third law of thermodynamics, is to make the entropy zero[13] when $T_0 = 0$. In other words

$$S(T) = \int_0^T \frac{C}{T}\,\mathrm{d}T. \tag{3.46}$$

As with the other quantities we have discussed, we often prefer to calculate the entropy per spin $s(T)$ of our system, which is given in terms of the specific heat per spin $c$ by

$$s(T) = \int_0^T \frac{c}{T}\,\mathrm{d}T. \tag{3.47}$$

Of course, evaluating either of these expressions involves calculating the specific heat over a range of temperatures up to the temperature we are interested in, at sufficiently small intervals that its variation with $T$ is well approximated. Then we have to perform a numerical integral using, for example, the trapezium rule or any of a variety of more sophisticated integration techniques (see, for instance, Press *et al.* 1988). Calculating the entropy (or equivalently the free energy) of a system is therefore a more complex task than calculating the internal energy, and may use up considerably more computer time. On the other hand, if we are interested in probing the behaviour of our system over a range of temperatures anyway (as we often are), we may as well make use of the data to calculate the entropy; the integration is a small extra computational effort to make by comparison with the simulation itself.

The integration involved in the entropy calculation can give problems. In particular, if there is any very sharp behaviour in the curve of specific heat as a function of temperature, we may miss it in our simulation, which would give the integral the wrong value. This is a particular problem near "phase transitions", where the specific heat often diverges (see Section 3.7.1).

## 3.6 Measuring correlation functions

One other quantity which we frequently want to measure is the two-point connected correlation function $G_c^{(2)}(i,j)$. Let us see how we would go about

[13] Systems which have more than one ground state may violate the third law. This point is discussed in more detail in Section 7.1.2.

calculating this correlation function for the Ising model. The most straightforward way is to evaluate it directly from the definition, Equation (1.26), using the values of the spins $s_i$ from our simulation:

$$G_c^{(2)}(i,j) = \langle s_i s_j \rangle - \langle s_i \rangle \langle s_j \rangle = \langle s_i s_j \rangle - m^2. \tag{3.48}$$

(Here $m$ denotes the expectation value of the magnetization.) In fact, since our Ising system is translationally invariant, $G_c^{(2)}(i,j)$ is dependent only on the displacement $\mathbf{r}$ between the sites $i$ and $j$, and not on exactly where they are. In other words, if $\mathbf{r}_i$ is the position vector of the $i^{\text{th}}$ spin, then we should have

$$G_c^{(2)}(\mathbf{r}_i, \mathbf{r}_i + \mathbf{r}) = G_c^{(2)}(\mathbf{r}) \tag{3.49}$$

independent of the value of $\mathbf{r}_i$. This means that we can improve our estimate of the correlation function $G_c^{(2)}(\mathbf{r})$ by averaging its value over the whole lattice for all pairs of spins separated by a displacement $\mathbf{r}$:

$$G_c^{(2)}(\mathbf{r}) = \frac{1}{N} \sum_{\substack{i,j \text{ with} \\ \mathbf{r}_j - \mathbf{r}_i = \mathbf{r}}} \big[ \langle s_i s_j \rangle - m^2 \big]. \tag{3.50}$$

If, as with our Ising model simulation, the system we are simulating has periodic boundary conditions (see Section 3.1), then $G_c^{(2)}(\mathbf{r})$ will not die away for very large values of $\mathbf{r}$. Instead, it will be periodic, dying away for values of $\mathbf{r}$ up to half the width of the lattice, and then building up again to another maximum when we have gone all the way across the lattice and got back to the spin we started with.

In order to evaluate $G_c^{(2)}(\mathbf{r})$ using Equation (3.50) we have to record the value of every single spin on the lattice at intervals during the simulation. This is not usually a big problem given the generous amounts of storage space provided by modern computers. However, if we want to calculate $G_c^{(2)}(\mathbf{r})$ for every value of $\mathbf{r}$ on the lattice, this kind of direct calculation does take an amount of time which scales with the number $N$ of spins on the lattice as $N^2$. As with the calculation of the autocorrelation function in Section 3.3.1, it actually turns out to be quicker to calculate the Fourier transform of the correlation function instead.

The spatial Fourier transform $\widetilde{G}_c^{(2)}(\mathbf{k})$ is defined by

$$\begin{aligned}
\widetilde{G}_c^{(2)}(\mathbf{k}) &= \sum_{\mathbf{r}} \mathrm{e}^{\mathrm{i}\mathbf{k}\cdot\mathbf{r}}\, G_c^{(2)}(\mathbf{r}) \\
&= \frac{1}{N} \sum_{\mathbf{r}} \sum_{\substack{i,j \text{ with} \\ \mathbf{r}_j - \mathbf{r}_i = \mathbf{r}}} \mathrm{e}^{\mathrm{i}\mathbf{k}\cdot(\mathbf{r}_j - \mathbf{r}_i)} \big[ \langle s_i s_j \rangle - m^2 \big] \\
&= \frac{1}{N} \Big\langle \sum_{\mathbf{r}_i} \mathrm{e}^{-\mathrm{i}\mathbf{k}\cdot\mathbf{r}_i} (s_i - m) \sum_{\mathbf{r}_j} \mathrm{e}^{\mathrm{i}\mathbf{k}\cdot\mathbf{r}_j} (s_j - m) \Big\rangle
\end{aligned}$$

$$= \frac{1}{N}\langle|\tilde{s}'(\mathbf{k})|^2\rangle, \tag{3.51}$$

where $\tilde{s}'(\mathbf{k})$ is the Fourier transform of $s'_i \equiv s_i - m$. In other words, in order to calculate $\widetilde{G}_c^{(2)}(\mathbf{k})$, we just need to perform a Fourier transform of the spins at a succession of different times throughout the simulation and then feed the results into Equation (3.51). As in the case of the autocorrelation function of Section 3.2, this can be done using a standard FFT algorithm. To get the correlation function in real space we then have to use the algorithm again to Fourier transform back, but the whole process still only takes a time which scales as $N \log N$, and so for the large lattices of today's Monte Carlo studies it is usually faster than direct calculation from Equation (3.50).[14]

Occasionally we also need to calculate the disconnected correlation function defined in Section 1.2.1. The equivalent of (3.51) in this case is simply

$$\widetilde{G}^{(2)}(\mathbf{k}) = \frac{1}{N}\langle|\tilde{s}(\mathbf{k})|^2\rangle. \tag{3.52}$$

Note that $s_i$ and $s'_i$ differ only by the average magnetization $m$, which is a constant. As a result, $\widetilde{G}^{(2)}(\mathbf{k})$ and $\widetilde{G}_c^{(2)}(\mathbf{k})$ are in fact identical, except at $\mathbf{k} = 0$. For this reason, it is often simpler to calculate $\widetilde{G}_c^{(2)}(\mathbf{k})$ by first calculating $\widetilde{G}^{(2)}(\mathbf{k})$ and then just setting the $\mathbf{k} = 0$ component to zero.

## 3.7 An actual calculation

In this section we go through the details of an actual Monte Carlo simulation and demonstrate how the calculation proceeds. The example that we take is that of the simulation of the two-dimensional Ising model on a square lattice using the Metropolis algorithm. This system has the advantage that its properties in the thermodynamic limit are known exactly, following the analytic solution given by Onsager (1944). Comparing the results from our simulation with the exact solution will give us a feel for the sort of accuracy one can expect to achieve using the Monte Carlo method. Some of the results have already been presented (see Figures 3.3 and 3.5 for example). Here we

[14] Again we should point out that this does not necessarily mean that one should always calculate the correlation function this way. As with the calculation of the autocorrelation function, using the Fourier transform is a more complicated method than direct calculation of the correlation function, and if your goal is to get an estimate quickly, and your lattice is not very large, you may be better advised to go the direct route. However, the Fourier transform method is more often of use in the present case of the two-point correlation function, since in order to perform the thermal average appearing in Equations (3.50) and (3.51) we need to repeat the calculation about once every two correlation times throughout the entire simulation, which might mean doing it a hundred or a thousand times in one run. Under these circumstances the FFT method may well be advantageous.

describe in detail how these and other results are arrived at, and discuss what conclusions we can draw from them.

The first step in performing any Monte Carlo calculation, once we have decided on the algorithm we are going to use, is to write the computer program to implement that algorithm. The code used for our Metropolis simulation of the Ising model is given in Appendix B. It is written in the computer language C.

As a test of whether the program is correct, we have first used it to simulate a small $5 \times 5$ Ising model for a variety of temperatures between $T = 0$ and $T = 5.0$ with $J$ set equal to 1. For such a small system our program runs very fast, and the entire simulation only took about a second at each temperature. In Section 1.3 we performed an exact calculation of the magnetization and specific heat for the $5 \times 5$ Ising system by directly evaluating the partition function from a sum over all the states of the system. This gives us something to compare our Monte Carlo results with, so that we can tell if our program is doing the right thing. At this stage, we are not interested in doing a very rigorous calculation, only in performing a quick check of the program, so we have not made much effort to ensure the equilibration of the system or to measure the correlation time. Instead, we simply ran our program for 20 000 Monte Carlo steps per site (i.e., $20\,000 \times 25 = 500\,000$ steps in all), and averaged over the last 18 000 of these to measure the magnetization and the energy. Then we calculated $m$ from Equation (3.12) and $c$ from Equation (3.15). If the results do not agree with our exact calculation then it could mean either that there is a problem with the program, or that we have not waited long enough in either the equilibration or the measurement sections of the simulation. However, as shown in Figure 3.7, the numerical results agree rather well with the exact ones. Even though we have not calculated the statistical errors on our data in order to determine the degree of agreement, these results still give us enough confidence in our program to proceed with a more thorough calculation on a larger system.

For our large-scale simulation, we have chosen to examine a system of $100 \times 100$ spins on a square lattice. We started the program with randomly chosen values of all the spins—the $T = \infty$ state of Section 3.1.1—and ran the simulations at a variety of temperatures from $T = 0.2$ to $T = 5.0$ in steps of 0.2, for a total of 25 simulations in all.[15] Again we ran our simulations

[15]Note that it is not possible to perform a simulation at $T = 0$ because the acceptance ratio, Equation (3.7), for spin flips which increase the energy of the system becomes zero in this limit. This means that it is not possible to guarantee that the system will come to equilibrium, because the requirement of ergodicity is violated; there are some states which it is not possible to get to in a finite number of moves. It is true in general of thermal Monte Carlo methods that they break down at $T = 0$, and often they become very slow close to $T = 0$. The continuous time Monte Carlo method of Section 2.4 can sometimes by used to overcome this problem in cases where we are particularly interested

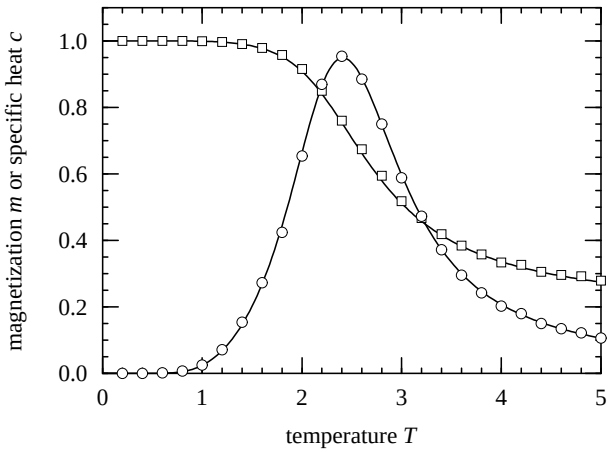


FIGURE 3.7 The magnetization (squares) and specific heat (circles) per spin of an Ising model in two dimensions on a $5 \times 5$ square lattice. The points are the results of the Metropolis Monte Carlo calculation described in the text. The lines are the exact calculations performed in Section 1.3, in which we evaluated the partition function by summing over all the states of the system.

for 20 000 Monte Carlo steps per lattice site. This is a fairly generous first run, and is only possible because we are looking at quite a small system still. In the case of larger or more complex models, one might well first perform a shorter run to get a rough measure of the equilibration and correlation times for the system, before deciding how long a simulation to perform. A still more sophisticated approach is to perform a short run and then store the configuration of the spins on the lattice before the program ends. Then, after deciding how long the entire calculation should last on the basis of the measurements during that short run, we can pick up exactly where we left off using the stored configuration, thus saving ourselves the effort of equilibrating the system twice.

Taking the data from our 25 simulations at different temperatures, we first estimate the equilibration times $\tau_{\text{eq}}$ at each temperature using the methods described in Section 3.2. In this case we found that all the equilibration times were less than about 1000 Monte Carlo steps per site, except for the simulations performed at $T = 2.0$ and $T = 2.2$, which both had equilibration times on the order of 6000 steps per site. (The reason for this anomaly is explained in the next section.) Allowing ourselves a margin for error in these estimates, we therefore took the data from time 2000 onwards as our equi-

in the behaviour of a model close to $T = 0$.

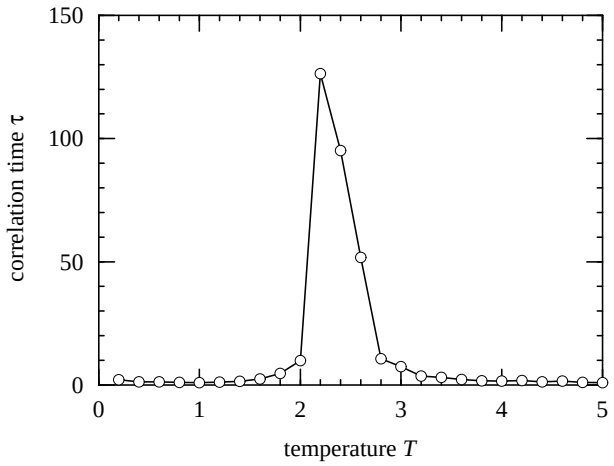


FIGURE 3.8 The correlation time for the $100 \times 100$ Ising model simulated using the Metropolis algorithm. The correlation time is measured in Monte Carlo steps per lattice site (i.e., in multiples of 10 000 Monte Carlo steps in this case). The straight lines joining the points are just to guide the eye.

librium measurements for all the temperatures except the two slower ones, for which we took the data for times 10 000 onwards.

Next we need to estimate how many independent measurements these data constitute, which means estimating the correlation time. To do this, we calculate the magnetization autocorrelation function at each temperature from Equation (3.21), for times $t$ up to 1000. (We must be careful only to use our equilibrium data for this calculation since the autocorrelation function is an equilibrium quantity. That is, we should not use the data from the early part of the simulation during which the system was coming to equilibrium.) Performing a fit to these functions as in Figure 3.6, we make an estimate of the correlation time $\tau$ at each temperature. The results are shown in Figure 3.8. Note the peak in the correlation time around $T = 2.2$. This effect is called "critical slowing down", and we will discuss it in more detail in Section 3.7.2. Given the length of the simulation $t_{\max}$ and our estimates of $\tau_{\text{eq}}$ and $\tau$ for each temperature, we can calculate the number $n$ of independent measurements to which our simulations correspond using Equation (3.19).

Using these figures we can now calculate the equilibrium properties of the $100 \times 100$ Ising model in two dimensions. As an example, we have calculated the magnetization and the specific heat again. Our estimate of the magnetization is calculated by averaging over the magnetization measurements from the simulation, again excluding the data from the early portion

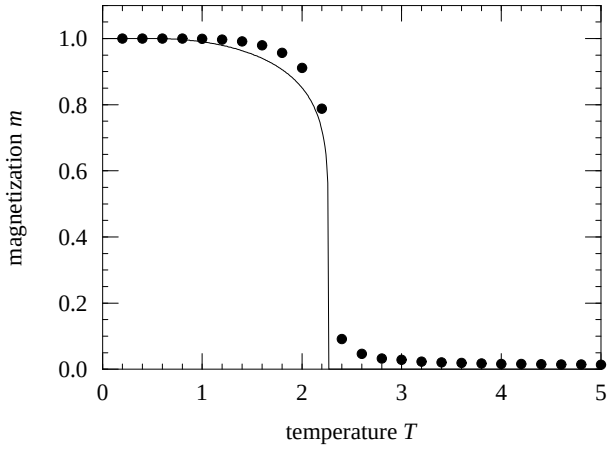


FIGURE 3.9 The magnetization per spin of the two-dimensional Ising model. The points are the results from our Monte Carlo simulation using the Metropolis algorithm. The errors are actually smaller than the points in this figure because the calculation is so accurate. The solid line is the known exact solution for the Ising model on an infinite two-dimensional square lattice.

of the run where the system was not equilibrated. The results are shown in Figure 3.9, along with the known exact solution for the infinite system. Calculating the errors on the magnetization from Equation (3.37), we find that the errors are so small that the error bars would be completely covered by the points themselves, so we have not bothered to put them in the figure. The agreement between the numerical calculation and the exact one for the infinite lattice is much better than it was for the smaller system in Figure 1.1, although there is still some discrepancy between the two. This discrepancy arises because the quantity plotted in the figure is in fact the average $\langle |m| \rangle$ of the *magnitude* of the magnetization, and not the average magnetization itself; we discuss our reasons for doing this in Section 3.7.1 when we examine the spontaneous magnetization of the Ising model.

The figure clearly shows the benefits of the Monte Carlo method. The calculation on the $5 \times 5$ system which we performed in Section 1.3 was exact, whereas the Monte Carlo calculation on the $100 \times 100$ system is not. However, the Monte Carlo calculation still gives a better estimate of the magnetization of the infinite system. The errors due to statistical fluctuations in our measurements of the magnetization are much smaller than the inaccuracy of working with a tiny $5 \times 5$ system.

Using the energy measurements from our simulation, we have also calculated the specific heat for our Ising model from Equation (3.15). To calculate

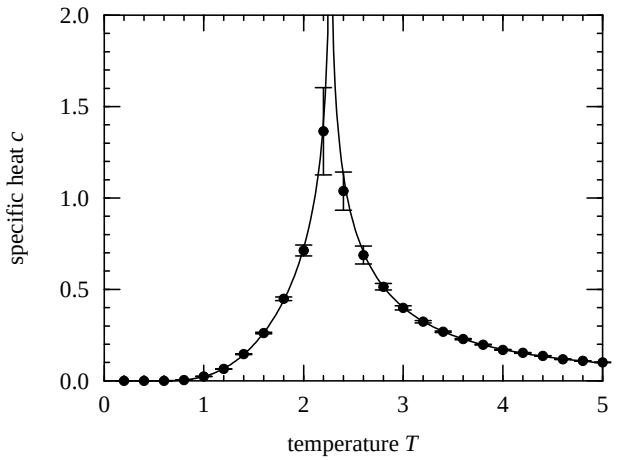


FIGURE 3.10 The specific heat per spin of the two-dimensional Ising model calculated by Monte Carlo simulation (points with error bars) and the exact solution for the same quantity (solid line). Note how the error bars get bigger close to the peak in the specific heat. This phenomenon is discussed in detail in the next section.

the errors on the resulting numbers we could use the blocking method of Section 3.4.2 to get a rough estimate. Here we are interested in doing a more accurate calculation, and for that we should use either the bootstrap or the jackknife method (see Sections 3.4.3 and 3.4.4). The number of independent samples $n$ is for most temperatures considerably greater than 100, so by the criterion given in Section 3.4.4, the bootstrap method is the more efficient one to use. In Figure 3.10 we show our results for the specific heat with error bars calculated from 200 bootstrap resamplings of the data (giving errors accurate to about 5%). The agreement here with the known exact result for the specific heat is excellent—better than for the magnetization—though the errors are larger, especially in the region close to the peak. If we were particularly interested to know the value of $c$ in this region it would make sense for us to go back and do a longer run of our program in this region to get more accurate data. For example, if we wanted to calculate the entropy difference from one side of the peak to the other using Equation (3.45), then large error bars in this region would make the integral inaccurate and we might well benefit from expending a little more effort in this region to get more accurate results.

### 3.7.1 The phase transition

It is not really the intention of this book to discuss the properties of the Ising model in detail, or the properties of any other model. However, there are a few things about the Ising model which we need to look into in a little more detail in order to understand the strengths and weaknesses of the Metropolis algorithm, and to explain why other algorithms may be better for certain calculations.

If we look at Figures 3.9 and 3.10, the most obvious feature that strikes us is the behaviour of the model around temperature $T = 2.2$ or so. The results shown in Figure 3.9 indicate that above this temperature the mean magnetization per site $m$ is quite small, whereas below it the magnetization is definitely non-zero and for the most part quite close to its maximum possible value of 1. This seems like a sensible way for the model to behave, since we know (see Section 3.1.1) that the $T = \infty$ state is one in which all the spins are randomly oriented up or down so that the net magnetization will be zero on average, and we know that the $T = 0$ state is one in which all the spins line up with one another, either all up or all down, so that the magnetization per site is either $+1$ or $-1$. If we only had results for a small Ising system to go by, like the ones depicted in Figure 3.7, we might imagine that the true behaviour of the system was simply that the magnetization rose smoothly from zero in the $T \to \infty$ limit to 1 as the temperature tends to zero. However, our results for the larger $100 \times 100$ system indicate that the transition from small $m$ to large $m$ becomes sharper as we go to larger systems, and in fact we know in the case of this particular model, because we have an exact solution, that the change from one regime to the other is actually infinitely sharp in the thermodynamic limit. This kind of change is called a **phase transition**. The Ising model is known to have a phase transition in two or more dimensions. The two regimes we observe are called the **phases** of the model. The phase transition between them takes place at a temperature $T_c$, which we call the **critical temperature**, whose value in the particular case of the two-dimensional Ising model is known to be

$$T_c = \frac{2J}{\log(1+\sqrt{2})} \simeq 2.269J. \tag{3.53}$$

Above this temperature the system is in the **paramagnetic phase**, in which the average magnetization is zero. Below it, the system is in the **ferromagnetic phase** and develops a **spontaneous magnetization** (i.e., most of the spins align one way or the other and the magnetization becomes non-zero all of its own accord without the application of a magnetic field to the model). This spontaneous magnetization rises from zero at the phase transition to unity at absolute zero. The magnetization is referred to as the **order parameter** of the Ising model because of this behaviour. In general, an order parameter is any quantity which is zero on one side of a phase transition and

non-zero on the other. A phase transition in which the order parameter is continuous at $T_c$, as it is here, is called a **continuous phase transition**.

In fact, to be strictly correct, the *mean* magnetization of the Ising model below the critical temperature is still zero, since the system is equally happy to have most of its spins pointing either down or up. Thus if we average over a long period of time we will find that the magnetization is close to $+1$ half the time and close to $-1$ for the other half, with occasional transitions between the two, so that the average is still close to zero. However, the average of the magnitude $|m|$ of the magnetization will be close to $+1$, whereas it will be close to zero above the phase transition. In Figure 3.9 we therefore actually plotted the average of $|m|$, and not $m$. This explains why the magnetization above the transition temperature is still slightly greater than zero. The average magnetization in this phase is definitely zero (give or take the statistical error) but the average of the *magnitude* of $m$ is always greater than zero, since we are taking the average of a number which is never negative. Still, as we go to the thermodynamic limit we expect this quantity to tend to zero, so that the numerical result and the exact solution should agree.[16]

We can look in detail at what happens to the spins in our Ising system as we pass through the phase transition from high to low temperatures by examining pictures such as those in Figure 3.2. At high temperatures the spins are random and uncorrelated, but as the temperature is lowered the interactions between them encourage nearby spins to point in the same direction, giving rise to correlations in the system. Groups of adjacent spins which are correlated in this fashion and tend to point in the same direction are called **clusters**.[17] As we approach $T_c$, the typical size $\xi$ of these clusters—also called the **correlation length**—diverges, so that when we are precisely at the transition, we may encounter arbitrarily large areas in which the spins are pointing mostly up or mostly down. Then, as we pass below the transition temperature, the system spontaneously chooses to have the majority of its spins in either the up or the down direction, and develops a non-zero magnetization in that direction. Which direction it chooses depends solely

[16] This also provides an explanation of why the agreement between the analytic solution and the Monte Carlo calculation was better for the specific heat, Figure 3.10, than it was for the magnetization. The process of taking the mean of the magnitude $|m|$ of the magnetization means that we consistently overestimate the magnetization above the critical temperature, and in fact this problem extends to temperatures a little below $T_c$ as well (see Figure 3.9). No such adjustments are necessary when calculating the specific heat, and as a result our simulation agrees much better with the known values for $c$, even though the error bars are larger in this case.

[17] A number of different mathematical definitions of a cluster are possible. Some of them require that all spins in the cluster point in the same direction whilst others are less strict. We discuss these definitions in detail in the next chapter, particularly in Sections 4.2 and 4.4.2.

on the random details of the thermal fluctuations it was going through as we passed the critical temperature, and so is itself completely random. As the temperature drops further towards $T = 0$, more and more of the spins line up in the same direction, and eventually as $T \to 0$ we get $|m| = 1$.

The study of phase transitions is an entire subject in itself and we refer the interested reader to other sources for more details of this interesting field. For our purposes the brief summary given above will be enough.

### 3.7.2 Critical fluctuations and critical slowing down

We are interested in the behaviour of the Ising model in the region close to $T_c$. This region is called the **critical region**, and the processes typical of the critical region are called **critical phenomena**. As we mentioned, the system tends to form into large clusters of predominantly up- or down-pointing spins as we approach the critical temperature from above. These clusters contribute significantly to both the magnetization and the energy of the system, so that, as they flip from one orientation to another, they produce large fluctuations in $m$ and $E$, often called **critical fluctuations**. As the typical size $\xi$ of the clusters diverges as $T \to T_c$, the size of the fluctuations does too. And since fluctuations in $m$ and $E$ are related to the magnetic susceptibility and the specific heat through Equations (3.15) and (3.16), we expect to get divergences in these quantities at $T_c$ also. This is what we see in Figure 3.10. These divergences are some of the most interesting of critical phenomena, and a lot of effort, particularly using Monte Carlo methods, has been devoted to investigating their exact nature. Many Monte Carlo studies of many different models have focused exclusively on the critical region to the exclusion of all else. Unfortunately, it is in precisely this region that our Metropolis algorithm is least accurate.

There are two reasons for this. The first has to do with the critical fluctuations. The statistical errors in the measured values of quantities like the magnetization and the internal energy are proportional to the size of these critical fluctuations (see Section 3.4) and so grow as we approach $T_c$. In a finite-sized system like the ones we study in our simulations, the size of the fluctuations never actually diverges—that can only happen in the thermodynamic limit—but they can become very large, and this makes for large statistical errors in the measured quantities.

What can we do about this? Well, recall that the error on, for example, the magnetization $m$ indeed increases with the size of the magnetization fluctuations, but it also decreases with the number of independent measurements of $m$ that we make during our simulation (see Equation (3.37)). Thus, in order to reduce the error bars on measurements close to $T_c$, we need to run our program for longer, so that we get a larger number of measurements. This however, is where the other problem with the Metropolis algorithm

comes in. As we saw in Figure 3.8, the correlation time $\tau$ of the simulation is also large in the region around $T_c$. In fact, like the susceptibility and the specific heat, the correlation time actually diverges at $T_c$ in the thermodynamic limit. For the finite-sized systems of our Monte Carlo simulations $\tau$ does not diverge, but it can still become very large in the critical region, and a large correlation time means that the number of independent measurements we can extract from a simulation of a certain length is small (see Equation (3.19)). This effect on its own would increase the size of the errors on measurements from our simulation, even without the large critical fluctuations. The combination of both effects is particularly unfortunate, because it means that in order to increase the number of independent measurements we make during our simulation, we have to perform a much longer run of the program; the computer time necessary to reduce the error bars to a size comparable with those away from $T_c$ increases very rapidly as we approach the phase transition.

The critical fluctuations which increase the size of our error bars are an innate physical feature of the Ising model. Any Monte Carlo algorithm which correctly samples the Boltzmann distribution will also give critical fluctuations. There is nothing we can do to change our algorithm which will reduce this source of error. However, the same is not true of the increase in correlation time. This effect, known as **critical slowing down**, is a property of the Monte Carlo algorithm we have used to perform the simulation, but not of the Ising model in general. Different algorithms can have different values of the correlation time at any given temperature, and the degree to which the correlation time grows as we approach $T_c$, if it grows at all, depends on the precise details of the algorithm. Therefore, if we are particularly interested in the behaviour of a model in the critical region, it may be possible to construct an algorithm which suffers less from critical slowing down than does the Metropolis algorithm, or even eliminates it completely, allowing us to achieve much greater accuracy for our measurements. In the next chapter we look at a number of other algorithms which do just this and which allow us to study the critical region of the Ising model more accurately.

## Problems

**3.1** Derive the appropriate generalization of Equation (3.10) for a simulation of an Ising model with non-zero external magnetic field $B$.

**3.2** Suppose we have a set of $n$ measurements $x_1 \dots x_n$ of a real quantity $x$. Find an approximate expression for the error on our best estimate of the mean of their squares. Take the numbers below and estimate this error.

| | | | |
|---|---|---|---|
| 20.27 | 19.61 | 20.06 | 20.73 |
| 20.09 | 20.68 | 19.37 | 20.40 |
| 19.95 | 20.55 | 19.64 | 19.94 |

Now estimate the same quantity for the same set of numbers using the jackknife method of Section 3.4.4.

**3.3** In this chapter we described methods for calculating a variety of quantities from Monte Carlo data, including internal energies, specific heats and entropies. Suggest a way in which we might measure the partition function using data from a Monte Carlo simulation.

**3.4** Write a computer program to carry out a Metropolis Monte Carlo simulation of the one-dimensional Ising model in zero field. Use it to calculate the internal energy of the model for a variety of temperatures and check the results against the analytic solution from Problem 1.4.

# 4

# Other algorithms for the Ising model

In the last chapter we saw that the Metropolis algorithm with single-spin-flip dynamics is an excellent Monte Carlo algorithm for simulating the Ising model when we are interested in temperatures well away from the critical temperature $T_c$. However, as we approach the critical temperature, the combination of large critical fluctuations and long correlation time makes the errors on measured quantities grow enormously. As we pointed out, there is little to be done about the critical fluctuations, since these are an intrinsic property of the model near its phase transition (and are, what's more, precisely the kind of interesting physical effect that we *want* to study with our simulations). On the other hand, the increase in the correlation time close to the phase transition is a function of the particular algorithm we are using—the Metropolis algorithm in this case—and it turns out that by using different algorithms we can greatly reduce this undesirable effect. In the first part of this chapter we will study one of the most widely used and successful such algorithms, the Wolff algorithm. Before introducing the algorithm however, we need to define a few terms.

## 4.1 Critical exponents and their measurement

As discussed in Section 3.7.1, the spins of an Ising model in equilibrium group themselves into clusters of a typical size $\xi$, called the correlation length, and this correlation length diverges as the temperature approaches $T_c$. Let us define a dimensionless parameter $t$, called the **reduced temperature**, which measures how far away we are from $T_c$:

$$t = \frac{T - T_c}{T_c}. \tag{4.1}$$

When $t = 0$, we are at the critical temperature. The divergence of the correlation length near the phase transition then goes like

$$\xi \sim |t|^{-\nu}. \tag{4.2}$$

The positive quantity $\nu$ is called a **critical exponent**. We take the absolute value $|t|$ of the reduced temperature so that the same expression can be used for temperatures both above and below $T_c$.[1]

It is not obvious that the divergence of $\xi$ has to take the form of Equation (4.2), and indeed for some models it does not; there are models in which the correlation length diverges in other ways near $T_c$—logarithmically, for instance, or exponentially. However, it turns out that Equation (4.2) is the correct form for the Ising model which we are studying here. And what's more, it is believed that the value of the critical exponent $\nu$ is a property of the Ising model itself, and is independent of such things as the value of the coupling $J$, or the type of lattice we are studying the model on (square or triangular for example). This property is known as **universality** and is one of the most important results to come out of the study of the renormalization group, though to prove it would take us a long way off our path here. In Monte Carlo calculations the value of $\nu$ is also independent of the algorithm we use to perform the simulation. (The value of $\nu$ does depend on the dimensionality of the lattice we are using. Ising models in two and three dimensions, for example, have different values for $\nu$.)

As we also discussed at the end of the last chapter, there are, along with the divergence of the correlation length, accompanying divergences of the magnetic susceptibility and the specific heat. Two other universal critical exponents govern these divergences:

$$\chi \sim |t|^{-\gamma}, \tag{4.3}$$

$$c \sim |t|^{-\alpha}. \tag{4.4}$$

To describe the divergence of the correlation time $\tau$ we define a slightly different sort of exponent, which we denote $z$:

$$\tau \sim |t|^{-z\nu}, \tag{4.5}$$

where $\tau$ is measured in Monte Carlo steps per lattice site. The exponent $z$ is often called the **dynamic exponent**. (It should not be confused with the lattice coordination number introduced in the last chapter, which was also

[1] We could have different values for $\nu$ above and below the transition temperature, but as it turns out we don't. Although it is beyond the scope of this book, it is relatively straightforward to show using renormalization group arguments that this and the other critical exponents defined here must take the same values above and below $T_c$ (see Binney *et al.* 1992). Note however that the constant of proportionality in Equation (4.2) need not be the same above and below the transition.

denoted $z$.) Its definition differs from that of the other exponents by the inclusion of the $\nu$, which is the same $\nu$ as appeared in Equation (4.2); the exponent is really $z\nu$, and not just $z$. We're not quite sure why it is defined this way, but it's the way everyone does it, so we just have to get used to it. It is convenient in one respect, as we will see in a moment, because of the way in which $z$ is measured.

The dynamic exponent gives us a way to quantify the critical slowing down effect. As we mentioned, the amount of critical slowing down we see in our simulations depends on what algorithm we use. This gives rise to different values of $z$ for different algorithms—$z$ is not a universal exponent in the way that $\nu$, $\alpha$ and $\gamma$ are.[2] A large value of $z$ means that $\tau$ becomes large very quickly as we approach the phase transition, making our simulation much slower and hence less accurate. A small value of $z$ implies relatively little critical slowing down and a faster algorithm near $T_c$. If $z = 0$, then there is no critical slowing down at all, and the algorithm can be used right up to the critical temperature without $\tau$ becoming large.[3]

The measurement of critical exponents is far from being a trivial exercise. Quite a lot of effort has in fact been devoted to their measurement, since their values have intrinsic interest to those concerned with the physics of phase transitions. At present we are interested in the value of the dynamic exponent $z$ primarily as a measure of the critical slowing down in our algorithms. In Chapter 8 we will discuss in some detail the techniques which have been developed for measuring critical exponents, but for the moment we will just run briefly though one simple technique—a type of "finite size scaling"—to give an idea of how we gauge the efficiency of these algorithms.

Combining Equations (4.2) and (4.5), we can write

$$\tau \sim \xi^z. \tag{4.6}$$

This equation tells us how the correlation time gets longer as the correlation length diverges near the critical point. However, in a system of finite size, which includes all the systems in our Monte Carlo simulations, the correlation length can never really diverge. Recall that the correlation length is the typical dimension of the clusters of correlated spins in the system. Once this size reaches the dimension, call it $L$, of the system being simulated it can get no bigger. A volume of $L^d$, where $d$ is the dimensionality of our system, is the largest cluster of spins we can have. This means that in fact

[2]The exponent $z$ is still independent of the shape of the lattice, the spin–spin interaction $J$ and so forth. Only changes in the dynamics affect its value. (Hence the name "dynamic exponent".)

[3]In fact, it is conventional to denote a logarithmic divergence $\tau \sim -\log|t|$ of the correlation time by $z = 0$, so a zero dynamic exponent does not necessarily mean that there is no critical slowing down. However, a logarithmic divergence is far less severe than a power-law one, so $z = 0$ is still a good thing.

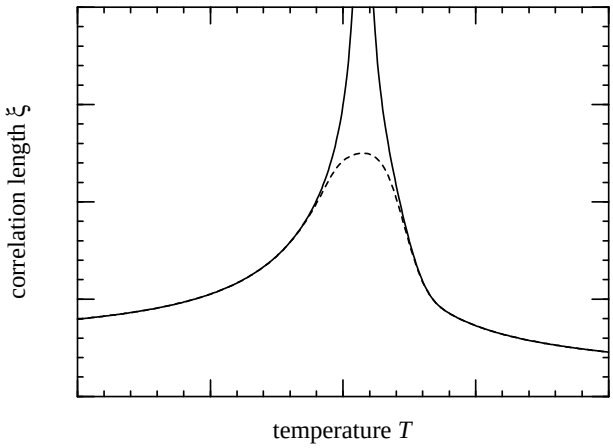


FIGURE 4.1 Schematic depiction of the cutoff in the divergence of the correlation length $\xi$ near the critical temperature for a system of finite size. The solid line indicates the way $\xi$ would diverge in an infinite system and the dashed one shows how it would be cut off around $\xi = L$ for a finite system of finite dimension $L$.

the divergence of the correlation length, and as a result that of the correlation time, is "cut off" in the region for which $\xi > L$. This effect is pictured schematically in Figure 4.1. Thus, for all temperatures sufficiently close to the critical one, and particularly when we are at the critical temperature, Equation (4.6) becomes

$$\tau \sim L^z. \tag{4.7}$$

Now suppose we *know* what the critical temperature $T_c$ of our model is, perhaps because, as in the 2D Ising model, an exact solution is available. (More generally, we don't know the critical temperature with any accuracy, in which case this method won't work, and we have to resort to some of the more sophisticated ones described in Chapter 8.) In that case, we can use Equation (4.7) to measure $z$. We simply perform a sequence of simulations at $T = T_c$ for systems of a variety of different sizes $L$, and then plot $\tau$ against $L$ on logarithmic scales. The slope of the resulting plot should be the exponent $z$. We have done exactly this in Figure 4.2 for the 2D Ising model simulated using the Metropolis algorithm. The line is the best fit through the points, given the errors, and its slope gives us a figure of $z = 2.09 \pm 0.06$ for the dynamic exponent. This calculation was rather a rough one, done on one of the authors' home computers. Much more thorough Monte Carlo calculations of $z$ have been performed using this and a number of other methods by a variety of people. At the time of the writing of this book the

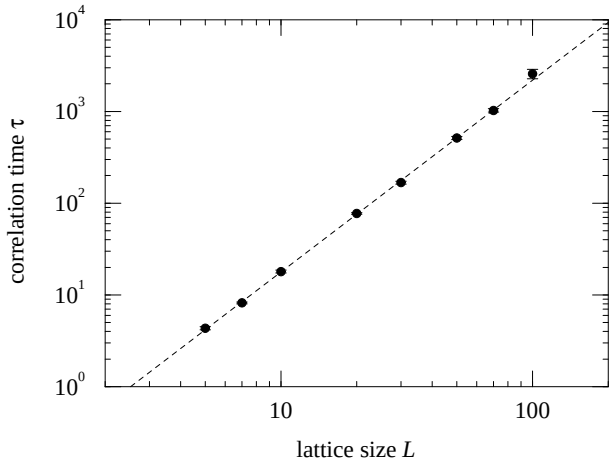


FIGURE 4.2 The correlation time $\tau$ for the 2D Ising model simulated using the Metropolis algorithm. The measurements were made at the critical temperature $T_c \simeq 2.269J$ for systems of a variety of different sizes $L \times L$, and then plotted on logarithmic scales against $L$. The slope $z = 2.09 \pm 0.06$ is an estimate of the dynamic exponent.

best figure available for this exponent was $z = 2.1665 \pm 0.0012$, obtained by Nightingale and Blöte (1996).

The value $z = 2.17$ is a fairly high one amongst Monte Carlo algorithms for the 2D Ising model, and indicates that the Metropolis algorithm is by no means the best algorithm for investigating the behaviour of the model near its phase transition. In the next section we will introduce a new algorithm which has a much lower dynamic exponent, albeit at the expense of some increase in complexity.

## 4.2 The Wolff algorithm

In the previous section we saw that the dynamic exponent $z$ of the two-dimensional Ising model simulated using the Metropolis algorithm has a value around 2. Equation (4.7) shows us that when we are at the critical temperature, the correlation time increases with system size as $L^z$. The CPU time taken to perform a certain number of Monte Carlo steps per site increases like the number of sites, or in other words as $L^d$, where $d$ is the dimensionality of the lattice. Thus the CPU time $\tau_{\mathrm{CPU}}$ taken to simulate one correlation time increases with system size as

$$\tau_{\mathrm{CPU}} \sim L^{d+z}, \tag{4.8}$$

which for the 2D Ising model is about $L^4$. This makes measurements on large systems extremely difficult in the critical region. (The simulations which went into Figure 4.2, for instance, took more than two weeks of CPU time, with the largest system size ($L = 100$) requiring 150 billion Monte Carlo steps to get a reasonably accurate value of $\tau$.)

The fundamental reason for the large value of $z$ in the Metropolis algorithm is the divergence of the correlation length and the critical fluctuations present near the phase transition. When the correlation length becomes large close to $T_c$, large regions form in which all the spins are pointing in the same direction. These regions are often called **domains**.[4] It is quite difficult for the algorithm to flip over one of these large domains because it has to do it spin by spin, and each move has quite a high probability of being rejected because of the ferromagnetic interactions between neighbouring spins. The chances of flipping over a spin in the middle of a domain are particularly low, because it is surrounded by four others pointing in the same direction. In two dimensions, flipping such a spin costs us $8J$ in energy and, using the value of $T_c \simeq 2.269J$ given in Equation (3.53), the probability of accepting such a move, when we are close to the critical temperature, is

$$A(\mu \to \nu) \simeq \mathrm{e}^{-8J/T_c} = 0.0294\ldots \tag{4.9}$$

or about three per cent, regardless of the value of $J$. The chance of flipping a spin on the edge of a domain is higher because it has a lower energy cost, and in fact it turns out that this process is the dominant one when it comes to flipping over entire domains. It can be very slow however, especially with the big domains that form near the critical temperature.

A solution to these problems was proposed in 1989 by Ulli Wolff, based on previous work by Swendsen and Wang (1987). The algorithm he suggested is now known as the **Wolff algorithm**. The basic idea is to look for clusters of similarly oriented spins and then flip them in their entirety all in one go, rather than trying to turn them over spin by painful spin. Algorithms of this type are referred to as **cluster-flipping algorithms**, or sometimes just **cluster algorithms**, and in recent years they have become popular for all sorts of problems, since it turns out that, at least in the case of the Ising model, they almost entirely remove the problem of critical slowing down.

How then do we go about finding these clusters of spins which we are going to flip? The simplest strategy which suggests itself, and the one which the Wolff algorithm uses, is just to pick a spin at random from the lattice and then look at its neighbours to see if any of them are pointing in the same

[4]Note that these domains are not quite the same thing as the clusters discussed in Section 3.7.1. By convention, the word "domain" refers to a group of adjacent spins which are pointing in the same direction, whereas the word "cluster" refers to a group of spins whose values are correlated with one another. We will shortly give a more precise definition of a cluster.

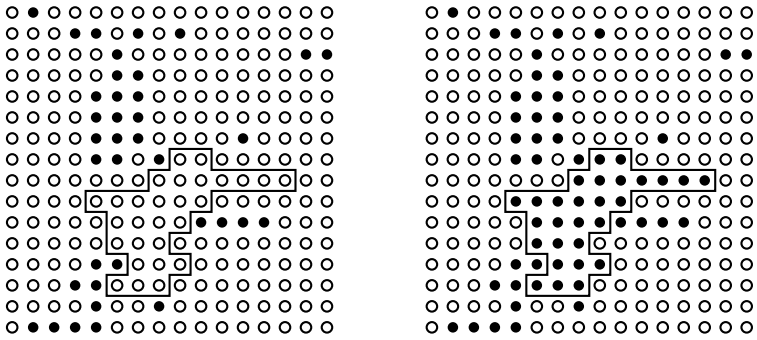

FIGURE 4.3 Flipping a cluster in a simulation of the 2D Ising model. The solid and open circles represent the up- and down-spins in the model.

direction. Then we can look at the neighbours of those neighbours, and so on, iteratively, until we have built up an entire cluster of spins. Clearly, however, we don't just want to flip over all of the spins which are pointing in the same direction as our first "seed" spin. How many we flip should depend on the temperature. We know for example that at high temperatures in the Ising model the spins tend to be uncorrelated with their neighbours, implying that they flip over singly, or in very small clusters. As we approach $T_c$, we know that the sizes of the clusters become much larger, and then, as we pass below $T_c$, ferromagnetism sets in, and most of the spins line up in the same direction, forming one big cluster that spans the entire lattice. In other words, the size of the clusters we flip should increase with decreasing temperature. We can achieve this if we build our clusters up from a seed, as described above, *but* we don't add every similarly oriented neighbour of the seed spin to the cluster. Instead, we have some probability $P_{\text{add}}$ of adding a spin, which goes up as the temperature falls. Then we look at the neighbours of the spins we have added to the cluster, and add them with probability $P_{\text{add}}$, and so forth. Then, when we finally run out of similarly oriented spins to add to the cluster, we stop and flip the cluster over with some acceptance ratio which depends on the energy cost of flipping it. The question then is, given a particular choice of $P_{\text{add}}$, what is the correct acceptance ratio to make the algorithm satisfy detailed balance, and what is the best choice of $P_{\text{add}}$ to make the average acceptance ratio as close to 1 as possible?

### 4.2.1 Acceptance ratio for a cluster algorithm

Let us work out what the acceptance ratio should be for this cluster-flipping algorithm so that detailed balance is obeyed. Consider two states of the system, $\mu$ and $\nu$, illustrated in Figure 4.3. They differ from one another by

the flipping of a single cluster of similarly oriented spins. The crucial thing to notice is the way the spins are oriented around the edge of the cluster (which is indicated by the line in the figure). Notice that in each of the two states, some of the spins just outside the cluster are pointing the same way as the spins in the cluster. The bonds between these spins and the ones in the cluster have to be "broken" when the cluster is flipped. Inevitably, the bonds which are *not* broken in going from $\mu$ to $\nu$ must be broken if we flip back again from $\nu$ to $\mu$.

Consider now a move which will take us from $\mu$ to $\nu$. There are in fact many such moves—we could choose any of the spins in the cluster as our seed spin, and then we could add the rest of the spins to it in a variety of orders. For the moment, however, let us just consider one particular move, starting with a particular seed spin and then adding the others to it in a particular order. Consider also the reverse move, which takes us back to $\mu$ from $\nu$, starting with exactly the same seed spin, and adding the others to it in exactly the same way as in the forward move. The probability of choosing the seed is exactly the same in the two directions, as is the probability of adding each spin to the cluster. The only thing that changes between the two is the probability of "breaking" bonds around the edge of the cluster, because the bonds which have to be broken are different in the two cases. Suppose that, for the forward move, there are $m$ bonds which have to be broken in order to flip the cluster. These broken bonds represent pairs of similarly oriented spins which were not added to the cluster by the algorithm. The probability of not adding such a spin is $1 - P_{\text{add}}$. Thus the probability of not adding all of them, which is proportional to the selection probability $g(\mu \to \nu)$ for the forward move, is $(1 - P_{\text{add}})^m$. If there are $n$ bonds which need to be broken in the reverse move then the probability of doing it will be $(1 - P_{\text{add}})^n$. The condition of detailed balance, Equation (2.14), along with Equation (2.17), then tells us that

$$\frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)} = (1 - P_{\text{add}})^{m-n} \frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \mathrm{e}^{-\beta(E_\nu - E_\mu)}, \tag{4.10}$$

where $A(\mu \to \nu)$ and $A(\nu \to \mu)$ are the acceptance ratios for the moves in the two directions. The change in energy $E_\nu - E_\mu$ between the two states also depends on the bonds which are broken. For each of the $m$ bonds which are broken in going from $\mu$ to $\nu$, the energy changes by $+2J$. For each of the $n$ bonds which are made, the energy changes by $-2J$. Thus

$$E_\nu - E_\mu = 2J(m - n). \tag{4.11}$$

Substituting into Equation (4.10) and rearranging we derive the following condition on the acceptance ratios:

$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = [\mathrm{e}^{2\beta J}(1 - P_{\text{add}})]^{n-m}. \tag{4.12}$$

But now we notice a delightful fact. If we choose

$$P_{\text{add}} = 1 - \mathrm{e}^{-2\beta J}, \tag{4.13}$$

then the right-hand side of Equation (4.12) is just 1, independent of any properties of the states $\mu$ and $\nu$, or the temperature, or anything else at all. With this choice, we can make the acceptance ratios for both forward and backward moves unity, which is the best possible value they could take. Every move we propose is accepted and the algorithm still satisfies detailed balance. The choice (4.13) defines the Wolff cluster algorithm for the Ising model, whose precise statement goes like this:

1. Choose a seed spin at random from the lattice.
2. Look in turn at each of the neighbours of that spin. If they are pointing in the same direction as the seed spin, add them to the cluster with probability $P_{\text{add}} = 1 - \mathrm{e}^{-2\beta J}$.
3. For each spin that was added in the last step, examine each of *its* neighbours to find the ones which are pointing in the same direction and add each of them to the cluster with the same probability $P_{\text{add}}$. (Notice that as the cluster becomes larger we may find that some of the neighbours are already members, in which case obviously we don't have to consider adding them again. Also, some of the spins may have been considered for addition before, as neighbours of other spins in the cluster, but rejected. In this case, they get another chance to be added to the cluster on this step.) This step is repeated as many times as necessary until there are no spins left in the cluster whose neighbours have not been considered for inclusion in the cluster.
4. Flip the cluster.

The implementation of this algorithm in an actual computer program is discussed in detail in Section 13.2.5.

As well as satisfying detailed balance, this algorithm clearly also satisfies the criterion of ergodicity (see Section 2.2.2). To see this, we need only note that there is a finite chance at any move that any spin will be chosen as the sole member of a cluster of one, which is then flipped. Clearly the appropriate succession of such moves will get us from any state to any other in a finite time, as ergodicity requires. Along with the condition of detailed balance, this then guarantees that our algorithm will generate a series of states of the lattice which (after equilibration) will appear with their correct Boltzmann probabilities.

Notice that the probability $P_{\text{add}}$, Equation (4.13), does indeed increase with decreasing temperature, from zero at $T = \infty$ to one at $T = 0$, just as we said it should, so that the sizes of the clusters we grow will increase

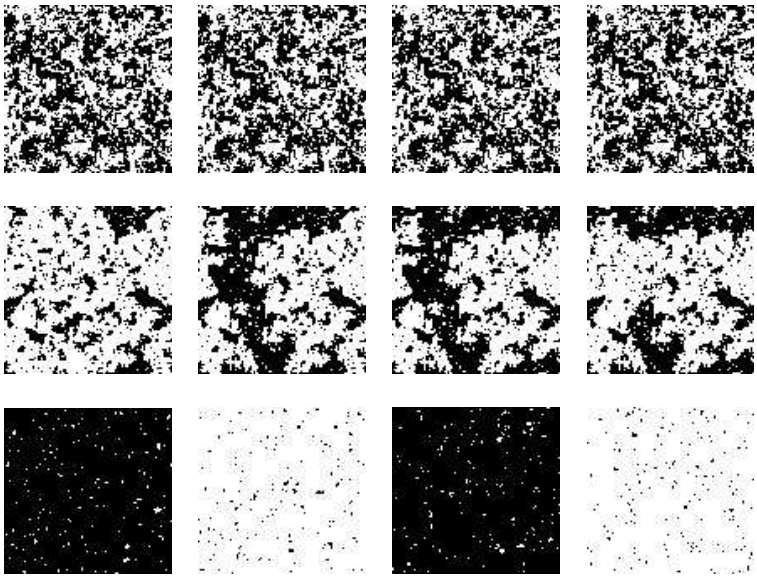

FIGURE 4.4 Consecutive states of a $100 \times 100$ Ising model in equilibrium simulated using the Wolff algorithm. The top row of four states are at a temperature $T = 2.8J$, which is well above the critical temperature, the middle row is close to the critical temperature at $T = 2.3J$, and the bottom row is well below the critical temperature at $T = 1.8J$.

as the temperature gets lower. The flipping of these clusters is a much less laborious task than in the case of the Metropolis algorithm—as we shall see it takes a time proportional to the size of the cluster to grow it and then turn it over—so we have every hope that the algorithm will have a lower dynamic exponent and less critical slowing down. In Section 4.3.1 we show that this is indeed the case. First we look in a little more detail at the way the Wolff algorithm works.

## 4.3 Properties of the Wolff algorithm

In this section we look at how the Wolff algorithm actually performs in practice. As we will see, it performs extremely well when we are near the critical temperature, but is actually a little slower than the Metropolis algorithm at very high or low temperatures.

Figure 4.4 shows a series of states generated by the algorithm at each of three temperatures, one above $T_c$, one close to $T_c$, and one below it. Up- and down-spins are represented by black and white dots. Consider first the middle set of states, the ones near $T_c$. If you examine the four frames, it

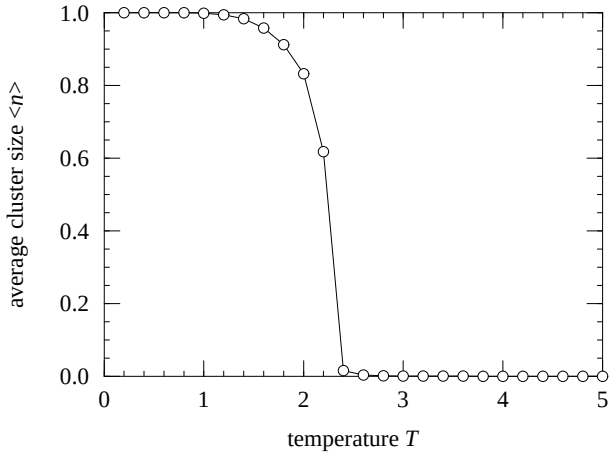


FIGURE 4.5 The average cluster size in the Wolff algorithm as a fraction of the size of the lattice measured as function of temperature. The error bars on the measurements are not shown, because they are smaller than the points. The lines are just a guide to the eye.

is not difficult to make out which cluster flipped at each step. Clearly the algorithm is doing its job, flipping large areas of spins when we are in the critical region. In the $T > T_c$ case, it is much harder to make out the changes between one frame and the next. The reason for this is that in this temperature region $P_{\text{add}}$ is quite small (see Equation (4.13)) and this in turn makes the clusters small, so it is hard to see when they flip over. Of course, this is exactly what the algorithm is supposed to do, since the correlation length here is small, and we don't expect to get large regions of spins flipping over together. In Figure 4.5, we have plotted the mean size of the clusters flipped by the Wolff algorithm over a range of temperatures, and, as we can see, they do indeed become small at large temperatures.

When the temperature gets sufficiently high (around $T = 10J$), the mean size of the clusters becomes hardly greater than one. In other words, the single seed spin for each cluster is being flipped over with probability one at each step, but none of its neighbours are. This is exactly what the Metropolis algorithm does in this temperature regime also. When $T$ is large, the Metropolis acceptance ratio, Equation (3.7), is 1, or very close to it, for any transition between two states $\mu$ and $\nu$. Thus in the limit of high temperatures, the Wolff algorithm and the Metropolis algorithm become the same thing. But notice that the Wolff algorithm will actually be the slower of the two in this case, because for each seed spin it has to go through the business of testing each of the neighbours for possible inclusion in the cluster, whereas

the Metropolis algorithm only has to decide whether to flip a single spin or not, a comparatively simple computational task. (To convince yourself of this, compare the programs for the two algorithms given in Appendix B. The Wolff algorithm is longer and more complex and will take more computer time per step than the Metropolis algorithm, even for a cluster with only one spin.) Thus, even if the Wolff algorithm is a good thing near the phase transition (and we will show that it is), there comes a point as the temperature increases where the Metropolis algorithm becomes better.

Now let us turn to the simulation at low temperature, the bottom row in Figure 4.4. The action of the algorithm is dramatically obvious in this case—almost every spin on the lattice is being flipped at each step. The reason for this is clear. When we are well below the critical temperature the Ising model develops a finite magnetization in one direction or the other, and the majority of the spins on the lattice line up in this direction, forming a "backbone" of similarly oriented spins which spans the entire lattice. When we choose our seed spin for the Wolff algorithm, it is likely that we will land on one of the spins comprising this backbone. Furthermore, the probability $P_{\text{add}}$, Equation (4.13), is large when $T$ is small, so the neighbours of the seed spin are not only very likely to be aligned with it, but are also very likely to be added to the growing cluster. The result is that the cluster grows (usually) to fill almost the entire backbone of spontaneously magnetized spins, and then they are all flipped over in one step. Such a lattice-filling cluster is said to be a **percolating cluster**.

On the face of it, this seems like a very inefficient way to generate states for the Ising model. After all, we know that what should really be happening in the Ising model at low temperature is that most of the spins should be lined up with one another, except for a few "excitation" spins, which are pointing the other way (see the Figure 4.4). Every so often, one of these excitation spins flips back over to join the majority pointing the other way, or perhaps one of the backbone spins gets flipped by a particularly enthusiastic thermal excitation and becomes a new excitation spin. This of course is exactly what the Metropolis algorithm does in this regime. The Wolff algorithm on the other hand is removing the single-spin excitations by the seemingly extravagant measure of *flipping all the other spins on the entire lattice* to point the same way as the single lonely excitation. It's like working out which string of your guitar is out of tune and then tuning the other five to that one. In fact, however, it's actually not such a stupid thing to do. First, let us point out that, since the Wolff algorithm never flips spins which are pointing in the opposite direction to the seed spin, all the excitation spins on the entire lattice end up pointing the same way as the backbone when the algorithm flips over a percolating cluster. Therefore, the Wolff algorithm gets rid of *all* the excitation spins on the lattice in a single step. Second, since we know that the Wolff algorithm generates states with the correct

Boltzmann probabilities, it must presumably create some new excitation spins at the same time as it gets rid of the old ones. And indeed it does do this, as is clear from the frames in the figure. Each spin in the backbone gets a number of different chances to be added to the cluster; for most of the spins this number is just the lattice coordination number $z$, which is four in the case of the square lattice. Thus the chance that a spin will *not* be added to the cluster is $(1-P_{\text{add}})^4$. This is a small number, but still non-zero. (It is 0.012 at the temperature $T = 1.8J$ used in the figure.) Thus there will be a small number of spins in the backbone which get left out of the percolating cluster and are not flipped along with the others. These spins become the new excitations on the lattice.

The net result of all this is that after only one Wolff Monte Carlo step, all the old excitations have vanished and a new set have appeared. This behaviour is clear in Figure 4.4. At low temperatures, the Wolff algorithm generates a complete new configuration of the model at every Monte Carlo step, though the payoff is that it has to flip very nearly every spin on every step. In the Metropolis algorithm at low temperatures it turns out that you have to do about one Monte Carlo step per site (each one possibly flipping one spin) to generate a new independent configuration of the lattice. To see this, we need only consider again the excitation spins. The average acceptance ratio for flipping these over will be close to 1 in the Metropolis algorithm, because the energy of the system is usually lowered by flipping them. (Only on the rare occasions when several of them happen to be close together might this not be true.) Thus, it will on average take $N$ steps, where $N$ is the number of sites on the lattice, before we choose any particular such spin and flip it—in other words, one Monte Carlo step per site. But we know that the algorithm correctly generates states with their Boltzmann probability, so, in the same $N$ steps that it takes to find all the excitation spins and flip them over to join the backbone, the algorithm must also choose a roughly equal number of new spins to excite out of the backbone, just as the Wolff algorithm also did. Thus, it takes one sweep of the lattice to replace all the excitation spins with a new set, generating an independent state of the lattice. (This is in fact the best possible performance that any Monte Carlo algorithm can have, since one has to allow at least one sweep of the lattice to generate a new configuration, so that each spin gets the chance to change its value.)

Once more then, we find that the Wolff algorithm and the Metropolis algorithm are roughly comparable, this time at low temperatures. Both need to consider about $N$ spins for flipping in order to generate an independent state of the lattice. Again, however, the additional complexity of the Wolff algorithm is its downfall, and the Metropolis algorithm has a slight edge in speed in this regime because of its extreme simplicity.

So, if the Metropolis algorithm beats the Wolff algorithm (albeit only

by a slim margin) at both high and low temperatures, that leaves only the intermediate regime close to $T_c$ in which the Wolff algorithm might be worthwhile. This of course is the regime in which we designed the algorithm to work well, so we have every hope that it will beat the Metropolis algorithm there, and indeed it does, very handily, as we can show by comparing the correlation times of the two algorithms.

### 4.3.1 The correlation time and the dynamic exponent

Before we measure the correlation time of the Wolff algorithm, we need to consider exactly how it should be defined. If we are going to compare the correlation times of the Wolff algorithm and the Metropolis algorithm near to the phase transition as a way of deciding which is the better algorithm in this region, it clearly would not be fair to measure it for both algorithms in terms of number of Monte Carlo steps (or steps per lattice site). A single Monte Carlo step in the Wolff algorithm is a very complicated procedure, flipping maybe hundreds of spins and potentially taking quite a lot of CPU time, whereas the Metropolis Monte Carlo step is a very simple, quick thing—we can do a million of them in a second on a good computer, though each one only flips at most one spin.

The time taken to complete one step of the Wolff algorithm is proportional to the number of spins $n$ in the cluster.[5] Such a cluster covers a fraction $n/L^d$ of the entire lattice, and thus it will on average take $\langle n\rangle/L^d$ steps to flip each spin once, where $\langle n\rangle$ is the mean cluster size in equilibrium. This is equivalent to one sweep of the lattice in the Metropolis algorithm, so the correct way to define the correlation time is to write $\tau \propto \tau_{\text{steps}}\langle n\rangle/L^d$, where $\tau_{\text{steps}}$ is the correlation time measured in steps (i.e., clusters flipped) in the Wolff algorithm. The conventional choice for the constant of proportionality is 1. This makes the correlation times for the Wolff and Metropolis algorithms equal in the limits of low and high temperature, for the reasons discussed in the last section. This is not quite fair, since, as we also pointed out in the last section, the Metropolis algorithm is slightly faster in these regimes because it is simpler and each step doesn't demand so much of the computer. However, this difference is slight compared with the enormous difference in performance between the two algorithms near the phase transition which we will witness in a moment, so for all practical purposes we can write

$$\tau = \tau_{\text{steps}}\frac{\langle n\rangle}{L^d}. \tag{4.14}$$

[5]This is fairly easy to see: for each spin we have to look at its neighbours to see if they should be included in the cluster, and then when the cluster is complete we have to flip all the spins. Since the same operations are performed for each spin in the cluster, the time taken to do one Monte Carlo step should scale with cluster size.

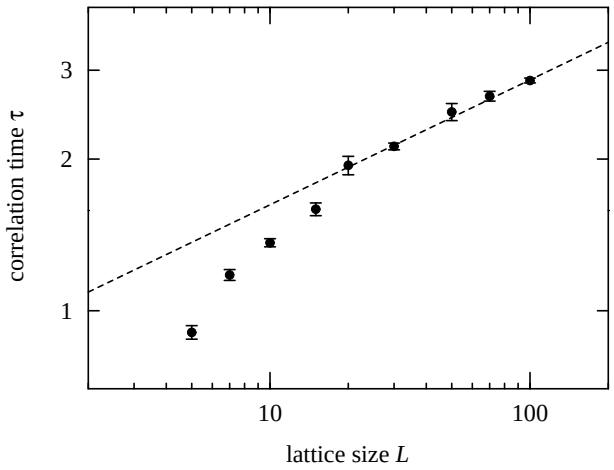


FIGURE 4.6 The correlation time $\tau$ for the 2D Ising model simulated using the Wolff algorithm. The measurements deviate from a straight line for small system sizes $L$, but a fit to the larger sizes, indicated by the dashed line, gives a reasonable figure of $z = 0.25 \pm 0.02$ for the dynamic exponent of the algorithm.

Using this definition, we can compare the performance of the two algorithms in the region close to the critical temperature, and we find that the Wolff algorithm does indeed dramatically outperform the Metropolis algorithm. For instance, in a $100 \times 100$ two-dimensional system right at the critical temperature, we measure the correlation time of the Wolff algorithm to be $\tau = 2.80 \pm 0.03$ spin-flips per site. The Metropolis algorithm by contrast has $\tau = 2570 \pm 330$. A factor of a thousand certainly outweighs any difference in the relative complexity of the two algorithms. It is this impressive performance on the part of the Wolff algorithm which makes it a worthwhile algorithm to use if we want to study the behaviour of the model close to $T_c$.

In Figure 4.6 we have plotted on logarithmic scales the correlation time of the Wolff algorithm for the two-dimensional Ising model at the critical temperature, over a range of different system sizes, just as we did for the Metropolis algorithm in Figure 4.2. Again, the slope of the line gives us an estimate of the dynamic exponent. Our best fit, given the errors on the data points, is $z = 0.25 \pm 0.02$. Again, this was something of a rough calculation, although our result is competitive with other more thorough ones. The best available figure at the time of writing was that of Coddington and Baillie (1992) who measured $z = 0.25 \pm 0.01$. This figure is clearly much lower than the $z = 2.17$ of the Metropolis algorithm, and gives us a quantitative measure of how much better the Wolff algorithm really is.

### 4.3.2 The dynamic exponent and the susceptibility

In most studies of the Wolff algorithm for the Ising model one does not actually make use of Equation (4.14) to calculate $\tau$. If we measure time in Monte Carlo steps (that is, simple cluster flips), we can define a corresponding dynamic exponent $z_{\text{steps}}$ in terms of the correlation time $\tau_{\text{steps}}$ of Equation (4.14) thus:

$$\tau_{\text{steps}} \sim \xi^{z_{\text{steps}}}. \tag{4.15}$$

We can measure this exponent in exactly the same way as we did before, and it turns out that it is related to the real dynamic exponent $z$ for the algorithm by

$$z = z_{\text{steps}} + \frac{\gamma}{\nu} - d, \tag{4.16}$$

where $\gamma$ and $\nu$ are the critical exponents governing the divergences of the magnetic susceptibility and the correlation length (see Equations (4.2) and (4.3)) and $d$ is the dimensionality of the model (which is 2 in the cases we have been looking at). If we know the values of $\nu$ and $\gamma$, as we do in the case of the 2D Ising model, then we can use Equation (4.16) to calculate $z$ without ever having to measure the mean cluster size in the algorithm, which eliminates one source of error in the measurement, thereby making the value of $z$ more accurate.[6]

The first step in demonstrating Equation (4.16) is to prove another useful result, about the magnetic susceptibility $\chi$. It turns out that, for temperatures $T \geq T_c$, the susceptibility is related to the mean size $\langle n \rangle$ of the clusters flipped by the Wolff algorithm thus:

$$\chi = \beta \langle n \rangle. \tag{4.17}$$

In many simulations of the Ising model using the Wolff algorithm, the susceptibility is measured using the mean cluster size in this way.

The demonstration of Equation (4.17) goes like this. Instead of carrying out the Wolff algorithm in the way described in this chapter, imagine instead doing it a slightly different way. Imagine that at each step we look at the whole lattice and for every pair of neighbouring spins which are pointing in the same direction, we make a "link" between them with probability $P_{\text{add}} = 1 - \mathrm{e}^{-2\beta J}$. When we are done, we will have divided the whole lattice into many different clusters of spins, as shown in Figure 4.7, each of which will be a correct Wolff cluster (since we have used the correct Wolff probability $P_{\text{add}}$ to make the links). Now we choose a single seed spin from the lattice at random, and flip the cluster to which it belongs. Then we throw away all the links we have made and start again. The only difference

[6] In cases such as the 3D Ising model, for which we don't know the exact values of the critical exponents, it may still be better to make use of Equation (4.14) and measure $z$ directly, as we did in Figure 4.6.

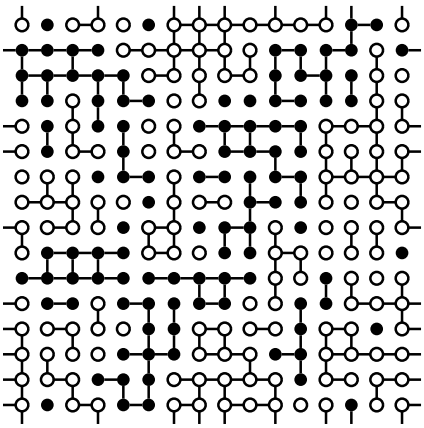

FIGURE 4.7 The whole lattice can be divided into clusters simply by putting "links" down with probability $P_{\text{add}}$ between any two neighbouring spins which are pointing in the same direction.

between this algorithm and the Wolff algorithm as we described it is that here we make the clusters first and then choose the seed spin afterwards, rather than the other way around. This would not be a very efficient way of implementing the Wolff algorithm, since it requires us to create clusters all over the lattice, almost all of which never get flipped, but it is a useful device for proving Equation (4.17). Why? Well, we can write the total magnetization $M$ of the lattice in a particular state as a sum over all the clusters on the lattice thus:

$$M = \sum_i S_i n_i. \tag{4.18}$$

Here $i$ labels the different clusters, $n_i$ is their size (a positive integer), and $S_i = \pm 1$ depending on whether the $i^{\text{th}}$ cluster is pointing up or down, thereby making either a positive or negative contribution to $M$. In order to calculate the mean square magnetization we now take the square of this expression and average it over a large number of spin configurations:

$$\langle M^2 \rangle = \Big\langle \sum_i S_i n_i \sum_j S_j n_j \Big\rangle = \Big\langle \sum_{i \neq j} S_i S_j n_i n_j \Big\rangle + \Big\langle \sum_i S_i^2 n_i^2 \Big\rangle. \tag{4.19}$$

Now, since the two values $\pm 1$ which the variables $S_i$ can take are equally likely on average, the first term in this expression is an average over a large number of quantities which are randomly either positive or negative and which will therefore tend to average out to zero. The second term on the other hand is an average over only positive quantities since $S_i^2 = +1$ for all

$i$. This term therefore is definitely not zero[7] and

$$\langle M^2 \rangle = \Big\langle \sum_i n_i^2 \Big\rangle. \tag{4.20}$$

And for the magnetization per spin we have

$$\langle m^2 \rangle = \frac{1}{N^2} \Big\langle \sum_i n_i^2 \Big\rangle. \tag{4.21}$$

Now consider the average $\langle n \rangle$ of the size of the clusters which get flipped in the Wolff algorithm. This is not quite the same thing as the average size of the clusters over the entire lattice, because when we choose our seed spin for the Wolff algorithm it is chosen at random from the entire lattice, which means that the probability $p_i$ of it falling in a particular cluster $i$ is proportional to the size of that cluster:

$$p_i = \frac{n_i}{N}. \tag{4.22}$$

The average cluster size in the Wolff algorithm is then given by the average of the probability of a cluster being chosen times the size of that cluster:

$$\langle n \rangle = \Big\langle \sum_i p_i n_i \Big\rangle = \frac{1}{N} \Big\langle \sum_i n_i^2 \Big\rangle = N \langle m^2 \rangle. \tag{4.23}$$

Now if we employ Equation (1.36), and recall that $\langle m \rangle = 0$ for $T \geq T_c$, we get

$$\chi = \beta \langle n \rangle \tag{4.24}$$

as promised. Thus, we can measure the magnetic susceptibility of the Ising model simply by counting the spins in our Wolff clusters and taking the average.[8]

The reason this result does not extend below the critical temperature is that for $T < T_c$ the average magnetization $\langle m \rangle$ is non-zero and must be included in Equation (1.36), but there is no simple expression for $\langle m \rangle$ in terms of the sizes of the clusters flipped. For $T \geq T_c$ however, we can use (4.24) to rewrite Equation (4.14) thus:

$$\tau = \tau_{\text{steps}} \frac{\chi}{\beta L^d}. \tag{4.25}$$

[7] Strictly, the second term scales like the number of spin configurations in the average and the first scales like its square root, so the second term dominates for a large number of configurations.

[8] In fact, measuring the susceptibility in this way is not only simpler than a direct measurement, it is also superior, giving, as it turns out, smaller statistical errors. For this reason the expression (4.24) is sometimes referred to as an **improved estimator** for the susceptibility (Sweeny 1983, Wolff 1989).

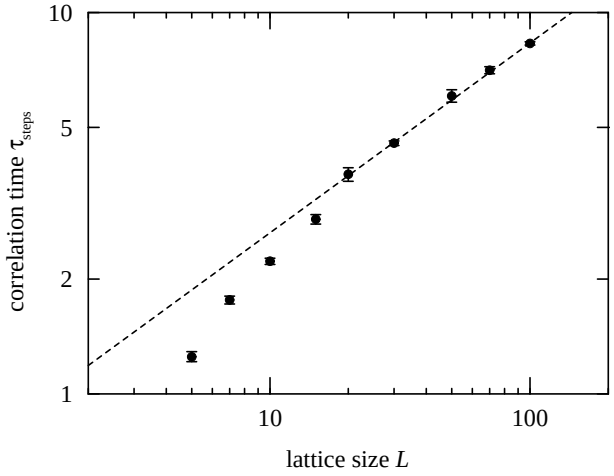


FIGURE 4.8 The correlation time $\tau_{\text{steps}}$ of the 2D Ising model simulated using the Wolff algorithm, measured in units of Monte Carlo steps (i.e., cluster flips). The fit gives us a value of $z_{\text{steps}} = 0.50 \pm 0.01$ for the corresponding dynamic exponent.

Now if we use Equations (4.2), (4.3), (4.6) and (4.15), this implies that slightly above $T_c$ we must have

$$\xi^z \sim \xi^{z_{\text{steps}}} \xi^{\gamma/\nu} L^{-d}. \tag{4.26}$$

Then, as we did in Section 4.1, we note that, close to $T_c$, the correlation length is just equal to the dimension $L$ of the system, so we replace $\xi$ everywhere by $L$, and we get

$$L^z \sim L^{z_{\text{steps}}} L^{\gamma/\nu} L^{-d}, \tag{4.27}$$

which then implies that

$$z = z_{\text{steps}} + \frac{\gamma}{\nu} - d \tag{4.28}$$

as we suggested earlier on.

In Figure 4.8 we have replotted the results from our Wolff algorithm simulations of the 2D Ising model using the correlation time measured in Monte Carlo steps (i.e., cluster flips). The best fit to the data gives a figure of $z_{\text{steps}} = 0.50 \pm 0.01$. Using the accepted values $\nu = 1$ and $\gamma = \frac{7}{4}$ for the 2D Ising model (Onsager 1944), and setting $d = 2$, Equation (4.28) then gives us $z = 0.25 \pm 0.01$, which is as good as the best published result for this exponent.

## 4.4 Further algorithms for the Ising model

We have looked at two algorithms for simulating the Ising model—the Metropolis algorithm and the Wolff algorithm—and for all practical purposes, these two are all we need for simulating the model in equilibrium. When we are interested in the behaviour of the model well away from the critical temperature, the Metropolis algorithm provides a simple and efficient way of getting results which is not bettered by any other algorithm. Close to $T_c$ the Wolff algorithm is a better choice than the Metropolis algorithm; although it is more complex than the Metropolis algorithm, the Wolff algorithm has a very small dynamic exponent, which means that the time taken to perform a simulation scales roughly like the size of the system, which is the best that we can hope for in any algorithm. The combination of these two algorithms allows us to study large systems at any temperature—calculations have been performed on systems with $L = 200$ or greater in both two and three dimensions.

However, many other algorithms have been suggested for the simulation of the Ising model. Although the algorithms studied so far pretty much cover the ground, it is still useful to know about some of these others, since many of them are in use and have been employed in published studies. In this section we talk about a few of these alternative algorithms.

### 4.4.1 The Swendsen–Wang algorithm

After the Metropolis and Wolff algorithms, probably the most important other algorithm is that of Swendsen and Wang (1987), which, like the Wolff algorithm, is a cluster-flipping algorithm. In fact this algorithm is very similar to the Wolff algorithm and Wolff took the idea for his algorithm directly from it. (Swendsen and Wang in turn got the idea from the work of Fortuin and Kasteleyn (1972) and Coniglio and Klein (1980).) Actually, we have seen the central idea behind the Swendsen–Wang algorithm already. In Section 4.3.2 we considered an alternative implementation of the Wolff algorithm in which the entire lattice of spins is divided up into clusters by making "links" with probability $P_{\text{add}} = 1 - \mathrm{e}^{-2\beta J}$ between similarly oriented neighbouring spins—any group of spins which is connected together by links is a cluster. We then imagined choosing a single spin from the lattice and flipping over the whole of the cluster to which it belongs. This procedure is clearly equivalent to the Wolff algorithm, although in practice it would be a very inefficient way of carrying it out.

The Swendsen–Wang algorithm divides the entire lattice into clusters in exactly the same way, with this same probability $P_{\text{add}}$ of making a link. But then, instead of flipping just one cluster, all the clusters are flipped with probability $\frac{1}{2}$. That is, for each cluster in turn, we decide independently

with probability $\frac{1}{2}$ whether to flip it or not. We notice the following facts about this algorithm:

1. The algorithm satisfies the requirement of ergodicity (Section 2.2.2). To see this, we note that there is always a finite chance that *no* links will be made on the lattice at all (in fact this is what usually happens when $T$ becomes very large), in which case the subsequent randomizing of the clusters is just equivalent to choosing a new state at random for the entire lattice. Thus, it is in theory possible to go from any state to any other in one step.
2. The algorithm also satisfies the condition of detailed balance. The proof of this fact is exactly the same as it was for the Wolff algorithm. If the number of links broken and made in performing a move are $m$ and $n$ respectively (and the reverse for the reverse move), then the energy change for the move is $2J(m-n)$ (or $2J(n-m)$ for the reverse move). The selection probabilities for choosing a particular set of links differ between forward and reverse moves only at the places where bonds are made or broken, so the ratio of the two selection probabilities is $(1-P_{\text{add}})^{m-n}$, just as it was before. By choosing $P_{\text{add}} = 1-\mathrm{e}^{-2\beta J}$, we then ensure, just as before, that the acceptance probability is independent of $m$ and $n$ and everything else, so any choice which makes it the same in each direction, such as flipping all clusters with probability $\frac{1}{2}$, will make the algorithm correct. Notice however that many other choices would also work. It doesn't matter how we choose to flip the clusters, though the choice made here is good because it minimizes the correlation between the direction of a cluster before and after a move, the new direction being chosen completely at random, regardless of the old one.
3. The algorithm updates the entire lattice on each move. In measuring correlation times for this algorithm, one should therefore measure them simply in numbers of Monte Carlo steps, and not steps per site as with the Metropolis algorithm. (In fact, on average, only half the spins get flipped on each move, but the number flipped scales like the size of the system, which is the important point.)
4. The Swendsen–Wang algorithm is essentially the same as the Wolff algorithm for low temperatures. Well below $T_c$, one of the clusters chosen by the algorithm will be the big percolating cluster, and the rest will correspond to the "excitations" discussed in Section 4.3, which will be very small. Ignoring these very small clusters then, the Swendsen–Wang algorithm will tend to turn over the percolating backbone of the lattice on average every two steps (rather than every step as in the Wolff algorithm—see Figure 4.4), but otherwise the two will behave almost identically. Thus, as with the Wolff algorithm, we can expect

the performance of the Swendsen–Wang algorithm to be similar to that of the Metropolis algorithm at low $T$, though probably a little slower on average due to the complexity of the algorithm.

5. At high temperatures, the Swendsen–Wang algorithm tends to divide the lattice into very small clusters because $P_{\text{add}}$ becomes small. As $T \to \infty$ the clusters will just be one spin each, and the algorithm will just change all the spins to new random values on each move. This is also what the Metropolis algorithm does at high temperatures in one sweep of the lattice, though again the Metropolis algorithm can be expected to be a little more efficient in this regime, since it is a simpler algorithm which takes few operations on the computer to flip each spin.

The combination of the last two points here implies that the only regime in which the Swendsen–Wang algorithm can be expected to outperform the Metropolis algorithm is the one close to the critical temperature. The best measurement of the dynamic exponent of the algorithm is that of Coddington and Baillie (1992), who found $z = 0.25 \pm 0.01$ in two dimensions, which is clearly much better than the Metropolis algorithm, and is in fact exactly the same as the result for the Wolff algorithm. So the Swendsen–Wang algorithm is a pretty good algorithm for investigating the 2D Ising model close to its critical point. However, the Wolff algorithm is always at least a factor of two faster, since in the Wolff algorithm every spin in every cluster generated is flipped, whereas the spins in only half of the clusters generated are flipped in the Swendsen–Wang algorithm. In addition, as Table 4.1 shows, for higher dimensions the Swendsen–Wang algorithm has a significantly higher dynamic exponent than the Wolff algorithm, making it slower close to $T_c$. The reason for this is that close to $T_c$ the properties of the Ising model are dominated by the fluctuations of large clusters of spins. As the arguments of Section 4.3.2 showed, the Wolff algorithm preferentially flips larger clusters because the chance of the seed spin belonging to any particular cluster is proportional to the size of that cluster. The Swendsen–Wang algorithm on the other hand treats all clusters equally, regardless of their size, and therefore wastes a considerable amount of effort on small clusters which make vanishingly little contribution to the macroscopic properties of the system for large system sizes. This, coupled with the fact that the Swendsen–Wang algorithm is slightly more complicated to program than the Wolff algorithm, makes the Wolff algorithm the algorithm of choice for most people.[9]

[9]There is one important exception to this rule; as discussed in Section 14.2.2, the Swendsen–Wang algorithm can be implemented more efficiently on a parallel computer than can the Wolff algorithm.

| dimension $d$ | Metropolis | Wolff | Swendsen–Wang |
|---|---|---|---|
| 2 | $2.167 \pm 0.001$ | $0.25 \pm 0.01$ | $0.25 \pm 0.01$ |
| 3 | $2.02 \pm 0.02$ | $0.33 \pm 0.01$ | $0.54 \pm 0.02$ |
| 4 | – | $0.25 \pm 0.01$ | $0.86 \pm 0.02$ |

TABLE 4.1 Comparison of the values of the dynamic exponent $z$ for the Metropolis, Wolff and Swendsen–Wang algorithms in various dimensions. The values are taken from Coddington and Baillie (1992), Matz *et al.* (1994), and Nightingale and Blöte (1996). To our knowledge, the dynamic exponent of the Metropolis algorithm has not been measured in four dimensions.

### 4.4.2 Niedermayer's algorithm

Another variation on the general cluster algorithm theme was proposed by Ferenc Niedermayer in 1988. His suggestion is really just an extension of the ideas used in the Wolff and Swendsen–Wang algorithms. Niedermayer's methods are very general and can be applied to all sorts of models, such as the glassy spin models that we will study in Chapter 6. Here we will just consider their application to the ordinary Ising model.

Niedermayer pointed out that it is not necessary to constrain the "links" with which we make clusters to be only between spins which are pointing in the same direction. In general, we can define two different probabilities for putting links between sites—one for parallel spins and one for anti-parallel ones. The way Niedermayer expressed it, he considered the energy contribution $E_{ij}$ that a pair of spins $i$ and $j$ makes to the Hamiltonian. In the case of the Ising model, for example,

$$E_{ij} = -Js_i s_j. \tag{4.29}$$

He then wrote the probability for making a link between two neighbouring spins as a function of this energy $P_{\text{add}}(E_{ij})$. In the Ising model $E_{ij}$ can only take two values $\pm J$, so the function $P_{\text{add}}(E_{ij})$ only needs to be defined at these points, but for some of the more general models Niedermayer considered it needs to be defined elsewhere as well. Clearly, if for the Ising model we make $P_{\text{add}}(-J) = 1 - \mathrm{e}^{-2\beta J}$ and $P_{\text{add}}(J) = 0$, then we recover the Wolff algorithm or the Swendsen–Wang algorithm, depending on whether we flip only a single cluster on each move, or many clusters over the entire lattice—Niedermayer's formalism is applicable in either case.

To be concrete about things, let us look at the case of the single-cluster, Wolff-type version of Niedermayer's algorithm. First, it is clear that for any choice of $P_{\text{add}}$ (except the very stupid choice $P_{\text{add}}(E) = 1$ for all $E$), the algorithm will satisfy the condition of ergodicity. Just as in the Wolff

algorithm, there is a finite probability that any spin on the lattice will find itself the sole member of a cluster of one. Flipping a succession of such clusters will clearly get us from any state to any other in a finite number of moves. Second, let us apply the condition of detailed balance to the algorithm. Consider, as we did in the case of the Wolff algorithm, two states of our system which differ by the flipping of a single cluster. (You can look again at Figure 4.3 if you like, but bear in mind that, since we are now allowing links between anti-parallel spins, not all the spins in the cluster need be pointing in the same direction.) As before, the probability of forming the cluster itself is exactly the same in the forward and reverse directions, except for the contributions which come from the borders. At the borders, there are some pairs of spins which are parallel and some which are anti-parallel. Suppose that in the forward direction there are $m$ pairs of parallel spins at the border—bonds which will be broken in flipping the cluster—and $n$ pairs which are anti-parallel—bonds which will be made. By definition, no links are made between the spins of any of these border pairs, and the probability of that happening is $[1-P_{\rm add}(-J)]^m[1-P_{\rm add}(J)]^n$. In the reverse direction the corresponding probability is $[1-P_{\rm add}(-J)]^n[1-P_{\rm add}(J)]^m$. Just as in the Wolff case, the energy cost of flipping the cluster from state $\mu$ to state $\nu$ is

$$E_\nu - E_\mu = 2J(m-n). \tag{4.30}$$

Thus, the appropriate generalization of the acceptance ratio relation, Equation (4.12), is

$$\frac{A(\mu\to\nu)}{A(\nu\to\mu)} = \left[\mathrm{e}^{2\beta J}\frac{1-P_{\rm add}(-J)}{1-P_{\rm add}(J)}\right]^{n-m}. \tag{4.31}$$

Any choice of acceptance ratios $A(\mu\to\nu)$ and $A(\nu\to\mu)$ which satisfies this relation will satisfy detailed balance. For the Wolff choice of $P_{\rm add}$ we get acceptance ratios which are always unity, but Niedermayer pointed out that there are other ways to achieve this. In fact, all we need to do is choose $P_{\rm add}$ to satisfy

$$\frac{1-P_{\rm add}(-E)}{1-P_{\rm add}(E)} = \mathrm{e}^{-2\beta E} \tag{4.32}$$

and we will get acceptance ratios which are always one. Niedermayer's solution to this equation was $P_{\rm add}(E) = 1-\exp[\beta(E-E_0)]$ where $E_0$ is a free parameter whose value we can choose as we like. Notice however that since $P_{\rm add}(E)$ is supposed to be a probability, it is not allowed to be less than zero. Thus the best expression we can write for the probability $P_{\rm add}(E_{ij})$ of adding a link between sites $i$ and $j$ is

$$P_{\rm add}(E_{ij}) = \begin{cases} 1-\mathrm{e}^{\beta(E_{ij}-E_0)} & \text{if } E_{ij}\le E_0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.33}$$

And this defines Niedermayer's algorithm. Notice the following things about this algorithm:

1. As long as all $E_{ij}$ on the lattice are less than or equal to $E_0$, the right-hand side of Equation (4.31) is always unity for the choice of $P_{\mathrm{add}}$ given in (4.33), so the two acceptance ratios can be chosen to be one for every move. Since we are at liberty to choose $E_0$ however we like, we can always satisfy this condition by making it greater than or equal to the largest value that $E_{ij}$ can take, which is $J$ in the case of the Ising model. This gives us a whole spectrum of Wolff-type algorithms for various values $E_0 \geq J$, which all have acceptance ratios of one. The Wolff algorithm itself is equivalent to $E_0 = J$. If we increase $E_0$ above this value, the probabilities $P_{\mathrm{add}}(E_{ij})$ tend closer and closer to one, making the clusters formed larger and larger. This gives us a way of controlling the sizes of the clusters formed in our algorithm, all the way up to clusters which encompass (almost) every spin on the lattice at every move.

2. If we choose $E_0$ to be less than the largest possible value of $E_{ij}$ then the right-hand side of Equation (4.31) is no longer equal to one, and we can no longer choose the acceptance ratios to be unity. For the Ising model, if we choose $-J \leq E_0 < J$ then we have

$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \left[\mathrm{e}^{2\beta J}\mathrm{e}^{-\beta(J+E_0)}\right]^{n-m} = \left[\mathrm{e}^{\beta(J-E_0)}\right]^{n-m}. \tag{4.34}$$

   Just as with the Metropolis algorithm, the optimal choice of acceptance ratios is then to make the larger of the two equal to one, and choose the smaller to satisfy this equation. If we do this, we again achieve detailed balance, and we now have an algorithm which, as $E_0$ is made smaller and smaller, produces smaller and smaller clusters, although it does so at the expense of an exponentially decreasing acceptance ratio for cluster moves which increase the energy of the system.

3. If $E_0$ is chosen to be smaller than the smallest possible value of $E_{ij}$, which is $-J$, then $P_{\mathrm{add}}(E_{ij}) = 0$ for all pairs of spins $i$, $j$, so every cluster formed has only one spin in it and the acceptance ratios are given by

$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \left[\mathrm{e}^{2\beta J}\right]^{n-m}. \tag{4.35}$$

   Bearing in mind that $m$ is the number of neighbours of this single spin which are pointing in the same direction as it, and $n$ is the number which point in the opposite direction, we can see that this is exactly the same as Equation (3.7) which defines the Metropolis algorithm.

Thus we see that by varying the parameter $E_0$, Niedermayer's cluster algorithm includes as special cases both the Metropolis algorithm and the Wolff algorithm, and interpolates smoothly from one to the other and beyond, varying the average cluster size from one spin all the way up to the entire lattice. The trouble with the algorithm is that no one really knows what value one should choose for $E_0$. Niedermayer himself conjectured that the Wolff choice $E_0 = J$ might not give the optimal correlation times and that shorter ones could be achieved by making other choices. He gave preliminary evidence in his 1988 paper that, in some cases at least, a smaller value of $E_0$ (i.e., a value which produces smaller clusters on average, at the expense of a lowered acceptance probability) gives a shorter correlation time. However, to our knowledge no one has performed an extensive study of the dynamic exponent of the algorithm as a function of $E_0$ so, for the moment at least, the algorithm remains an interesting extension of the Wolff idea which has yet to find use in large-scale simulations.

### 4.4.3 Multigrid methods

Also worthy of mention is a class of methods developed by Kandel *et al.* (1989, Kandel 1991), which are referred to as **multigrid methods**. These methods are also aimed at reducing critical slowing down and accelerating simulations at or close to the critical temperature. Multigrid methods have not been used to a very great extent in large-scale Monte Carlo simulations because they are considerably more complex to program than the cluster algorithms discussed above. However, they may yet prove useful in some contexts because they appear to be faster than cluster algorithms for simulating very large systems.

The fundamental idea behind these methods is the observation that, with the divergence of the correlation length at the critical temperature, we expect to see fluctuating clusters of spins of all sizes up to the size of the entire lattice. The multigrid methods therefore split the CPU time of the simulation up, spending varying amounts of time flipping blocks of spins of various sizes. This idea certainly has something in common with the ideas behind the Wolff and Swendsen–Wang algorithms, but the multigrid methods are more deliberate about flipping blocks of certain sizes, rather than allowing the sizes to be determined by the temperature and configuration of the lattice. Kandel and his co-workers gave a number of different, similar algorithms, which all fall under the umbrella of multigrid methods. Here, we describe one example, which is probably the simplest and most efficient such algorithm for the Ising model. The algorithm works by grouping the spins on the lattice into blocks and then treating the blocks as single spins and flipping them using a Metropolis algorithm. In detail, what one does is this.

In the Swendsen–Wang algorithm we made "links" between similarly oriented spins, which effectively tied those spins together into a cluster, so that they flipped as one. Any two spins which were not linked were free to flip separately—there was not even a ferromagnetic interaction between them to encourage them to point in the same direction. In the multigrid method, pairs of adjacent spins can be in three different configurations: they can be linked as in the Swendsen–Wang case, so that they must flip together, they can have no connection between them at all, so that they can flip however they like, or they can have a normal Ising interaction between them of strength $J$, which encourages them energetically to point in the same direction, but does not force them to as our links do. By choosing one of these three states for every pair of spins, the lattice is divided up into clusters of linked spins which either have interactions between them, or which are free to flip however they like. The algorithm is contrived so that only clusters of one or two spins are created. No clusters larger than two spins appear. The procedure for dividing up the lattice goes like this.

1. We take a spin on the lattice, and examine each of its neighbours in turn. If a neighbour is pointing in the opposite direction to the spin, then we leave it alone. In Kandel's terms, we "delete" the bond between the two spins, so that they are free to assume the same or different directions with no energy cost. If a neighbour is pointing in the same direction as our spin, then we make a link between the two with the same probability $P_{\text{add}} = 1 - \mathrm{e}^{-2\beta J}$ as we used in the Wolff and Swendsen–Wang algorithms. Kandel calls this "freezing" the bond between the spins.
2. Since we only want to create clusters of at most two spins, we stop looking at neighbours once we have created a link to any one of them. In fact, what we do is keep the normal Ising interactions between the spin and all the remaining neighbours that we have not yet looked at. We do this regardless of whether they are pointing in the same direction as our first spin or not. Kandel describes this as "leaving the bond active".
3. Now we move on to another spin and do the same thing, and in this way cover the entire lattice. Notice that, if we come to a spin which is adjacent to one we have considered before, then one or more of the spin's bonds will already have been frozen, deleted or marked as active. In this case we leave those bonds as they are, and only go to work on the others which have not yet been considered. Notice also that if we come to a spin and it has already been linked ("frozen") to another spin, then we know immediately that we need to leave the interactions on all the remaining bonds to that spin untouched.

In this way, we decide the fate of every spin on the lattice, dividing them

up into clusters of one or two, joined by bonds which may or may not have interactions associated with them. Then we treat those clusters as single spins, and we carry out the Metropolis algorithm on them, for a few sweeps of the lattice.

But this is not the end. Now we do the whole procedure again, treating the clusters as spins, and joining them into bigger clusters of either one or two elements each, using exactly the same rules as before. (Note that the lattice of clusters is not a regular lattice, as the original system was, but this does not stop us from carrying out the procedure just as before.) Then we do a few Metropolis sweeps of this coarser lattice too. And we keep repeating the whole thing until the size of the blocks reaches the size of the whole lattice. In this way, we get to flip blocks of spins of all sizes from single spins right up to the size of the entire system. Then we start taking the blocks apart again into the blocks that made them up, and so forth until we get back to the lattice of single spins. In fact, Kandel and co-workers used a scheme where at each level in the blocking procedure they either went towards bigger blocks ("coarsening") or smaller ones ("uncoarsening") according to the following rule. At any particular level of the procedure we look back and see what we did the previous times we got to this level. If we coarsened the lattice the previous two times we got to this point, then on the third time only, we uncoarsen. This choice has the effect of biasing the algorithm towards working more at the long length-scales (bigger, coarser blocks).

Well, perhaps you can see why the complexity of this algorithm has put people off using it. The proof that the algorithm satisfies detailed balance is quite involved, and, since you're probably not dying to hear about it right now, we'll refer you to the original paper for the details. In the same paper it is demonstrated that the dynamic exponent for the algorithm is in the region of 0.2 for the two-dimensional Ising model—a value similar to, though not markedly better than, the Wolff algorithm. Lest you dismiss the multigrid method out of hand, however, let us just point out that the simulations do indicate that its performance is superior to cluster algorithms for large systems. These days, with increasing computer power, people are pushing simulations towards larger and larger lattices, and there may well come a point at which using a multigrid method could win us a factor of ten or more in the speed of our simulation.

### 4.4.4 The invaded cluster algorithm

Finally, in our round-up of Monte Carlo algorithms for the Ising model, we come to an unusual algorithm proposed by Jonathan Machta and co-workers called the **invaded cluster algorithm** (Machta *et al.* 1995, 1996). The thing which sets this algorithm apart from the others we have looked

at so far is that it is not a general purpose algorithm for simulating the Ising model at any temperature. In fact, the invaded cluster algorithm can only be used to simulate the model at the critical point; it does not work at any other temperature. What's more, for a system at the critical point in the thermodynamic limit the algorithm is equivalent to the Swendsen–Wang cluster algorithm of Section 4.4.1. So what's the point of the algorithm? Well, there are two points. First, the invaded cluster algorithm can find the critical point all on its own—we don't need to know what the critical temperature is beforehand in order to use the algorithm. Starting with a system at any temperature (for example $T = 0$ at which all the spins are pointing in the same direction) the algorithm will adjust its simulation temperature until it finds the critical value $T_c$, and then it will stay there. This makes the algorithm very useful for actually measuring the critical temperature, something which otherwise demands quite sophisticated techniques such as the finite-size scaling or Monte Carlo RG techniques of Chapter 8. Second, the invaded cluster algorithm equilibrates extremely fast. Although the algorithm is equivalent to the Swendsen–Wang algorithm once it reaches equilibrium at $T_c$, its behaviour while getting there is very different, and in a direct comparison of equilibration times between the two, say for systems starting at $T = 0$, there is really no contest. For large system sizes, the invaded cluster algorithm can reach the critical point as much as a hundred times faster than the Swendsen–Wang algorithm. (A comparison with the Wolff algorithm yields similar results—the performance of the Wolff and Swendsen–Wang algorithms is comparable.)

So, how does the invaded cluster algorithm work? Basically, it is just a variation of the Swendsen–Wang algorithm in which the temperature is continually adjusted to look for the critical point. From the discussions of Section 4.4.1 we know that below the critical temperature the Swendsen–Wang algorithm generates a percolating cluster of either up- or down-pointing spins which stretches across the entire lattice. The algorithm looks for the temperature at which this percolating cluster first appears, and takes that to be the critical temperature. Here's how it goes:

1. First we choose some starting configuration of the spins. It doesn't matter what choice we make, but it could, for example, be the $T = 0$ state.

2. We perform a single step of the Swendsen–Wang algorithm in which we consider in turn every pair of neighbouring spins which are pointing in the same direction and make a bond between them with probability $P_{\text{add}} = 1 - \mathrm{e}^{-2\beta J}$. But the trick is, we adjust the temperature (or the inverse temperature $\beta = (kT)^{-1}$) to find the point at which one of the clusters produced in this way just starts to percolate. For example, if we start off with all the spins pointing the same way, then every

single pair of neighbouring spins on the lattice are aligned, and are therefore candidates for linking with probability $P_{\text{add}}$. In this case, it is known that for the square lattice in two dimensions for instance, percolation will set in when the probability of making a link between two spins is $P_{\text{add}} = \frac{1}{2}$. Solving for the temperature, this is equivalent to $\beta J = \frac{1}{2}\log 2 = 0.346\ldots$, so this is the temperature at which the first step should be conducted.

3. Once the links are made, we flip each cluster separately with probability $\frac{1}{2}$, just as we do in the normal Swendsen–Wang algorithm. Then the whole procedure is repeated from step 2 again.

So what's the point here? Why does the algorithm work? Well, consider what happens if the system is below the critical temperature. In that case, the spins will be more likely to be aligned with their neighbours than they are at the critical temperature, and therefore there will be more neighbouring pairs of aligned spins which are candidates for making links. Thus, in order to make enough links to get one cluster to percolate across the entire lattice we don't need as high a linking probability $P_{\text{add}}$ as we would at $T_c$. But a lower value of $P_{\text{add}}$ corresponds to a higher value $T > T_c$ of the corresponding temperature. In other words, when the system is below the critical temperature, the algorithm automatically chooses a temperature $T > T_c$ for its Swendsen–Wang procedure. Conversely, if the system is at a temperature above the critical point, then neighbouring spins will be less likely to be aligned with one another than they are at $T_c$. As a result, there will be fewer places that we can put links on the lattice, and $P_{\text{add}}$ will need to be higher than it would be at $T_c$ in order to make one of the clusters percolate and fill the entire lattice.[10] The higher value of $P_{\text{add}}$ corresponds to a lower value of the temperature $T < T_c$, and so, for a state of the system above the critical temperature, the algorithm will automatically choose a temperature $T < T_c$ for its Swendsen–Wang procedure.

The algorithm therefore has a kind of negative feedback built into it, which always drives the system towards the critical point, and when it finally reaches $T_c$ it will just stay there, performing Swendsen–Wang Monte Carlo steps at the critical temperature for the rest of the simulation. We do not need to know what the critical temperature is for the algorithm to work. It finds $T_c$ all on its own, and for that reason the algorithm is a good way of measuring $T_c$. Notice also that when the temperature of the system is lower than the critical temperature, the algorithm performs Monte Carlo steps with $T > T_c$, and *vice versa*. Thus, it seems plausible that the algorithm would drive itself towards the critical point quicker than simply performing

[10] In fact, for sufficiently high temperatures it may not be possible to produce a percolating cluster at all. In this case the algorithm will perform a Monte Carlo step at $T = 0$.

a string of Swendsen–Wang Monte Carlo steps exactly at $T_c$. As discussed below, this is indeed the case.

Before we look at the results from the invaded cluster algorithm, let us discuss briefly how it is implemented. There are a couple of questions that need to be answered. What is the most efficient way of varying the temperature $T$ to find the value of $P_{\rm add}$ at which a percolating cluster first appears on the lattice? And how do we identify the percolating cluster? There is a very elegant answer to the first question: at the beginning of each step of the algorithm, we go over the entire lattice and for every pair of neighbouring spins which are pointing in the same direction, we generate a random real number between zero and one. We can imagine writing down these numbers on the bonds that they belong to. Then we go through those random numbers in order, starting with the smallest of them and working up, making links between the corresponding pairs of spins one by one. At some point, we will have added enough links to make one of the clusters on the lattice percolate.[11] At that point, we stop, and work out what fraction of all the possible links on the lattice we have made. This fraction is just equal to the probability $P_{\rm add}$ we would have to use to produce the percolating cluster. Rearranging our expression for $P_{\rm add}$, Equation (4.13), we can then calculate the corresponding temperature of the Monte Carlo step:

$$T = -\frac{2J}{\log(1-P_{\rm add})}. \tag{4.36}$$

Our other question—how we know when we have a percolating cluster—does not have such an elegant solution. Machta *et al.* (1995), who invented the algorithm, suggest two ways of resolving the problem. One is to measure the dimensions of each of the clusters along the axes of the lattice. When a cluster has a length along one of the axes which is equal to the dimension $L$ of the lattice, it is declared to be percolating and no more links are added to the system. The other suggestion is that, given that the system has periodic boundary conditions, you wait until one of the clusters has gone off one side of the lattice and come back on the other and joined up with itself once more.[12] Machta *et al.* find that these two criteria yield very similar results when used in actual simulations, implying that the algorithm is not particularly sensitive to the exact method used.

The results for the invaded cluster algorithm are impressive. Machta *et al.* found equilibration times 20 or more times faster than the for the Swendsen–Wang algorithm at $T_c$ for the two- and three-dimensional Ising systems they

[11] Again, this may not be true at high temperatures, where it is sometimes not possible to produce a percolating cluster for any value of $P_{\rm add}$. In this case we stop when we have added links between all pairs of aligned spins.

[12] An efficient way of implementing this second criterion using a tree data structure is described by Barkema and Newman (1998b).

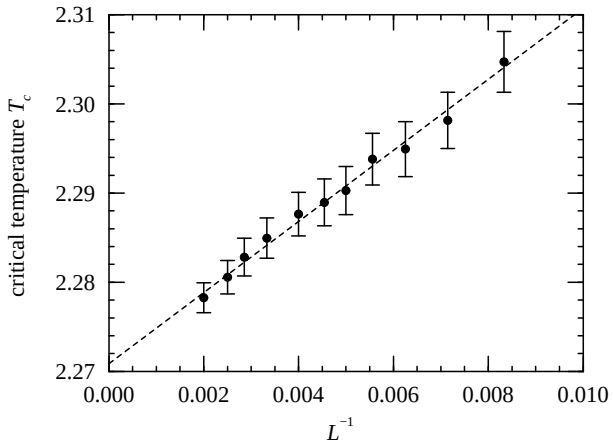


FIGURE 4.9 The critical temperature of the two-dimensional Ising model measured using the invaded cluster algorithm for systems of a variety of sizes from $L = 120$ up to $L = 500$. Here they are plotted against $L^{-1}$ and the extrapolation to $L = \infty$ at the vertical axis gives an estimate of $T_c = 2.271 \pm 0.002$ for the critical temperature in the thermodynamic limit. The data are taken from Machta *et al.* (1995).

examined, and of course, the invaded cluster algorithm allowed them to measure the value of $T_c$, which is not directly possible with the normal Swendsen–Wang algorithm. Figure 4.9 shows some of their results for the critical temperature of two-dimensional Ising systems of different sizes. On the finite lattice, the average estimate of $T_c$ which the algorithm makes is a little high.[13] However, if, as has been done here, one measures $T_c$ for a variety of system sizes, one can extrapolate to the limit $L = \infty$ to get an estimate of the true critical temperature in the thermodynamic limit. In this case we do this by plotting the measured $T_c$ as a function of $L^{-1}$ and extrapolating to the $L^{-1} = 0$ axis. The result is $T_c = 2.271 \pm 0.002$, which compares favourably with the known exact result of $T_c = 2.269$, especially given the small amount of CPU taken by the simulation. (The runs were 10 000 Monte Carlo steps for each system size.)

The invaded cluster algorithm can also be used to measure other quantities at the critical temperature—magnetization for example, or internal energy. However, a word of caution is in order here. For lattices of finite size, which of course include all the lattices in our Monte Carlo simulations,

[13]This behaviour depends on the criterion used to judge when percolation takes place. For some criteria the estimate of $T_c$ approaches the infinite system result from below rather than above.

the invaded cluster algorithm does not sample the Boltzmann distribution exactly. In particular, the fluctuations in quantities measured using the algorithm are different from those you would get in the Boltzmann distribution. To see this, consider what happens once the algorithm has equilibrated at the critical temperature. At this point, as we argued before, it should stop changing the temperature and just become equivalent to the Swendsen–Wang algorithm at $T_c$, which certainly samples the Boltzmann distribution correctly. However, in actual fact, because the lattice is finite, the innate randomness of the Monte Carlo method will give rise to variations in the temperature $T$ of successive steps in the simulation. The negative feedback effect that we described above will ensure that $T$ always remains close to $T_c$, but the size of fluctuations is very sensitive to small changes in temperature near to $T_c$ and as a result the measured fluctuations are not a good approximation to those of the true Boltzmann distribution.[14] Thus the invaded cluster algorithm is not well suited to estimating, for instance, the specific heat $c$ of the Ising model at $T_c$, which is determined by measuring fluctuations. On the other hand, one could use the algorithm to determine the value of $T_c$, and then use the normal Wolff or Swendsen–Wang algorithm to perform a simulation at that temperature to measure $c$. Determining $T_c$ in this way could also be a useful preliminary to determining the dynamic exponent of another algorithm, as we did earlier in the chapter for both the Metropolis and Wolff algorithms. Those calculations demanded a knowledge of the value of $T_c$, which is known exactly for the 2D Ising model, but not for many other models (including the 3D Ising model).

## 4.5 Other spin models

The Ising model, which we have spent the last two chapters examining, is the best studied spin model in statistical mechanics. There are however many others, some of which are of great importance. In the next few chapters we will look at some variations on the Ising model theme, including the conserved-order-parameter Ising model, in which the total magnetization of the lattice is held constant, Ising spin glasses, in which the strengths of the bonds between spins vary randomly over the lattice, and the random-field Ising model, in which each spin is subjected to its own local magnetic field which has a randomly chosen value. These models are interesting because, for a variety of reasons, their simulation demands Monte Carlo algorithms which are fundamentally different from the ones we have seen so far.

[14] As the size of the system gets larger however, the algorithm samples the Boltzmann distribution more accurately. This is the justification for our earlier claim that, in the thermodynamic limit, the invaded cluster algorithm is identical to the Swendsen–Wang algorithm at $T_c$.

In this section, we extend the Ising model in a different way. We look at a few of the many models which consider more general types of spins on the vertices of the lattice, spins which can take more than two discrete values, or spins which take a continuum of values. For the most part, these models can be simulated with techniques similar to the ones we have already seen, which is why we are considering them now. First we will look at Potts models.

### 4.5.1 Potts models

**Potts models** are a class of models which are similar to the Ising model except that the spins $s_i$ on each lattice site can take more than two different discrete values. Usually these values are represented by positive integers starting at 1, and a $q$-state Potts model is one in which each spin can have integer values $s_i = 1 \dots q$. Any two neighbouring spins then contribute an amount $-J$ to the Hamiltonian if they have the same value, or zero otherwise. The Hamiltonian can thus be written:

$$H = -J \sum_{\langle ij \rangle} \delta_{s_i s_j}, \tag{4.37}$$

where $\delta_{ij}$ is the Kronecker $\delta$-symbol, which is 1 when $i = j$ and zero otherwise.

For the case $q = 2$, the Potts model is equivalent to the Ising model, up to an additive constant in the Hamiltonian. To see this we note that Equation (4.37) can be rewritten as

$$H = -\tfrac{1}{2} J \sum_{\langle ij \rangle} 2\big(\delta_{s_i s_j} - \tfrac{1}{2}\big) - \sum_{\langle ij \rangle} \tfrac{1}{2} J. \tag{4.38}$$

But $2(\delta_{s_i s_j} - \frac{1}{2})$ is $+1$ when $s_i = s_j$ and $-1$ when they are different, so this expression is indeed equivalent to the Ising Hamiltonian except for the constant term $-\sum_{\langle ij \rangle} \frac{1}{2} J$. (Note though that the interaction energy is changed by a factor of two $J \to \frac{1}{2} J$ from Equation (3.1).)

For higher values of $q$ the Potts model behaves similarly in some ways to the Ising model. For $J > 0$ (the ferromagnetic case) it has $q$ equivalent ground states in which all the spins have the same value, and as the temperature is increased it undergoes a phase transition to a state in which each of the $q$ spin states occurs with equal frequency across the lattice. There are differences with the Ising model however. In particular, the entropy of a Potts model with $q > 2$ is higher than that of the Ising model at an equivalent temperature, because the density of states of the system as a function of energy is higher. For example, at temperatures just above $T = 0$, almost all of the spins will be in one state—say $q = 1$—but there will be a few isolated excitations on the lattice, just as there were in the Ising model. Unlike the

Ising model however, each excitation can take any of the possible spin values other than 1, which may be very many if $q$ is large, and this gives rise to many more low-lying excitation states than in the Ising case.

Monte Carlo simulation of Potts models is quite similar to the simulation of the Ising model. The simplest thing one can do is apply the single-spin-flip Metropolis algorithm, which would go like this. First we pick a spin $i$ from the lattice at random. It will have some value $s_i$. We choose at random a new value $s_i' \neq s_i$ from the $q-1$ available possibilities. Then we calculate the change in energy $\Delta E$ that would result if we were to make this change to this spin, and either accept or reject the move, with acceptance probability

$$A = \begin{cases} \mathrm{e}^{-\beta\Delta E} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.39}$$

(See Equation (3.7).) This algorithm satisfies the conditions of both ergodicity and detailed balance, for the same reasons that it did in the case of the Ising model. However, even for a single-spin-flip algorithm, the Metropolis algorithm is not a very good one to use for Potts models, especially when $q$ becomes large. To see this, let us consider an extreme case: the $q = 100$ Potts model on a square lattice in two dimensions.

At high temperatures, the acceptance ratio (4.39) is always either 1 or close to it because $\beta$ is small, so the algorithm is reasonably efficient. However, at lower temperatures this ceases to be the case. As the temperature decreases, more and more spins will take on the same values as their neighbours, forming ferromagnetic domains, just as in the Ising model. Consider the case of a spin which has four neighbours which all have different values. The four states of this spin in which it is aligned with one of its neighbours have lower energy, and therefore higher Boltzmann weight, than the other 96 possible states. If our spin is in one of the 96 higher energy states, how long will it take to find one of the four aligned states? Well, until we find one of them, all the states that the system passes through will have the same energy, so the Metropolis acceptance ratio will be 1. On average therefore, it will take about $100/4 = 25$ steps to find one of the four desirable states. As $q$ increases this number will get larger; it will take longer and longer for the system to find the low-lying energy states, despite the fact that the acceptance ratio for all moves is 1, simply because there are so many states to sort through.

Conversely, if the spin is in one of the four low-lying states, then there is an energy cost associated with excitation to one of the other 96, which means that the acceptance ratio will be less than one, and possibly very small (if the temperature is low). Thus it could be that nearly 96 out of every 100 attempted moves is rejected, giving an overall acceptance ratio little better than 4%.

One way to get around these problems is to use the **heat-bath algo-**

**rithm**. The heat-bath algorithm is also a single-spin-flip algorithm, but one which is more efficient at finding the energetically desirable states of the spins. The algorithm goes like this. First we choose a spin $k$ at random from the lattice. Then, regardless of its current value, we choose a new value $s_k$ for the spin in proportion to the Boltzmann weights for the different values—the values are drawn from a so-called "heat bath". In other words, we give the spin a value $n$ lying between 1 and $q$ with a probability

$$p_n = \frac{\mathrm{e}^{-\beta E_n}}{\sum_{m=1}^{q} \mathrm{e}^{-\beta E_m}} \tag{4.40}$$

where $E_n$ is the energy of the system when $s_k = n$. Note that we have normalized the probabilities to add up to one, so *some* value of the spin gets chosen on every step, even if it is just the same as the old value (though this will become increasingly unlikely as $q$ becomes large). Clearly this algorithm satisfies ergodicity, since on a lattice of $N$ spins the appropriate string of single-spin moves like this can take us from any state to any other in $N$ moves or fewer. And since we are choosing states with probabilities proportional to their Boltzmann weights, it should not come as a great surprise that this algorithm also satisfies the condition of detailed balance, though just to be thorough let's prove it. The probability of making the transition from a state in which $s_k = n$ to one in which $s_k = n'$ is just $p_{n'}$, and the probability of going back again is $p_n$. (Note that these probabilities do not depend on the initial state, only on the final one, unlike the Metropolis algorithm.) Then, using Equation (4.40), the ratio of forward and backward transition rates is

$$\frac{P(n \to n')}{P(n' \to n)} = \frac{p_{n'}}{p_n} = \frac{\mathrm{e}^{-\beta E_{n'}}}{\sum_{m=1}^{q} \mathrm{e}^{-\beta E_m}} \times \frac{\sum_{m=1}^{q} \mathrm{e}^{-\beta E_m}}{\mathrm{e}^{-\beta E_n}} = \mathrm{e}^{-\beta(E_{n'} - E_n)}, \tag{4.41}$$

exactly as detailed balance demands.

This algorithm is much more efficient than the Metropolis algorithm for large values of $q$, because it will choose the states with the highest Boltzmann weights most often, and can find them in one move, rather than wandering through large numbers of unfavourable states at random before finding them. In practice how is it implemented? First, we observe that once we have chosen which spin $k$ we are going to change, we can split the Hamiltonian, Equation (4.37), into the terms which involve $s_k$, and those which don't:

$$H = -J \sum_{\langle ij \rangle \ (i,j \neq k)} \delta_{s_i s_j} - J \sum_{i \text{ n.n. to } k} \delta_{s_i s_k}. \tag{4.42}$$

The first sum here is the same for all $q$ possible values of $s_k$, and therefore cancels out of the expression (4.40). The second sum has only $z$ terms, where $z$ is the lattice coordination number (which is four in the square

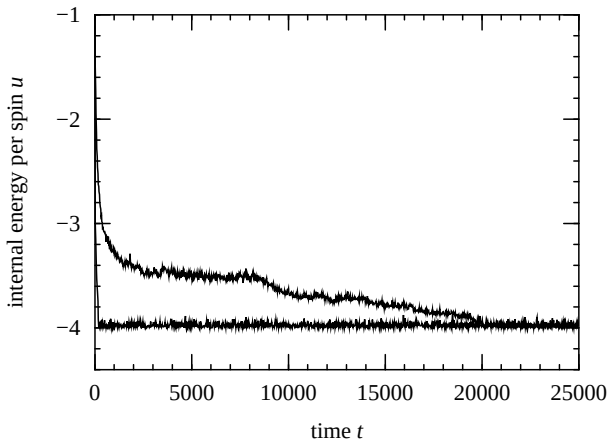


FIGURE 4.10 The internal energy of a $q = 10$ Potts model on a $20 \times 20$ square lattice at a temperature $T = 0.5J$, simulated using the Metropolis algorithm (upper line) and the heat-bath algorithm (lower line).

lattice case we have been considering). Just calculating these terms, rather than evaluating the entire Hamiltonian, makes evaluating $p_n$ much quicker.

Second, we notice that there are at most $z$ states of the spin $s_k$ in which it is the same as at least one of its neighbours. These are the only states in which the spin makes a contribution to the Hamiltonian. In all the other states it makes no contribution to the Hamiltonian. Thus, evaluating the Boltzmann factors $\mathrm{e}^{-\beta E_n}$ appearing in Equation (4.40) requires us to calculate the values of, at most, $z$ exponentials, all the other terms being just equal to 1.

Finally, we note that there is only a small spectrum of possible values for each Boltzmann factor, since the second term in Equation (4.42) can only take values which are multiples of $-J$, from zero up to $-zJ$. Thus we can, as we did for the Ising model, calculate the values of all the exponentials we are going to need at the very beginning of our simulation, and thereafter simply look up those values whenever we need to know one of them. The calculation of exponentials being a costly process in terms of CPU time, this trick considerably improves the efficiency of the algorithm.

In Figure 4.10 we compare the internal energy of a $q = 10$ Potts model on a $20 \times 20$ square lattice at low temperature simulated using the heat-bath and Metropolis algorithms. In this case, the heat-bath algorithm takes about 200 sweeps of the lattice to come to equilibrium, by contrast with the Metropolis algorithm which takes about 20 000 sweeps. This is an impressive gain in speed, although as usual we need to be careful about our claims. The heat-

bath algorithm is a more complex algorithm than the Metropolis algorithm, and each Monte Carlo step takes longer in the heat-bath case than in the Metropolis case. In this particular simulation, there was about a factor of three difference between the CPU time taken per step in the two algorithms. Even allowing for this however, the heat-bath algorithm is still more than thirty times faster at coming to equilibrium than the Metropolis algorithm. And this factor will increase as $q$ gets larger, making the heat-bath algorithm very definitely the algorithm of choice for large $q$ Potts models.

Before we move on, here is one more question about the heat-bath algorithm: what does the algorithm look like for the normal Ising model? The answer is easy enough to work out. In the Ising case the algorithm would involve choosing a spin at random and then, regardless of its current state, setting it either up or down with probabilities $p_{\rm up}$ and $p_{\rm down}$. From Equation (4.40) these two probabilities are:

$$p_{\rm up} = \frac{{\rm e}^{-\beta E_{\rm up}}}{{\rm e}^{-\beta E_{\rm up}} + {\rm e}^{-\beta E_{\rm down}}}, \qquad p_{\rm down} = \frac{{\rm e}^{-\beta E_{\rm down}}}{{\rm e}^{-\beta E_{\rm up}} + {\rm e}^{-\beta E_{\rm down}}}. \tag{4.43}$$

Now suppose that the initial state of the spin is down. What acceptance ratio $A$ do these equations represent for the move that flips the spin to the up state? Well, clearly the acceptance ratio is simply $A = p_{\rm up}$. If we multiply numerator and denominator by ${\rm e}^{\frac{1}{2}\beta(E_{\rm up}+E_{\rm down})}$, we can write it in terms of the change in energy $\Delta E = E_{\rm up} - E_{\rm down}$:

$$A = \frac{{\rm e}^{-\frac{1}{2}\beta\Delta E}}{{\rm e}^{-\frac{1}{2}\beta\Delta E} + {\rm e}^{\frac{1}{2}\beta\Delta E}}. \tag{4.44}$$

It is easy to show that the same expression applies for the move which flips the spin from up to down. Thus for the Ising model, the heat-bath algorithm can be regarded as a single-spin-flip algorithm with an acceptance ratio which is a function of the energy change $\Delta E$, just as it is in the Metropolis algorithm, although the exact functional dependence of $A$ on $\Delta E$ is different from that of the Metropolis algorithm. In Figure 4.11 we show $A$ as a function of $\Delta E$ for the two algorithms. For all values of $\Delta E$, the heat-bath acceptance ratio is lower than the corresponding value for the Metropolis algorithm, Equation (3.7), so it is always more efficient to use the Metropolis algorithm. This should come as no surprise, since, as we pointed out in the last chapter, the Metropolis algorithm is the most efficient possible single-spin-flip algorithm for simulating the Ising model. However, it is worth being familiar with Equation (4.44), because, for reasons which are not completely clear to the authors, some people still use it for simulating the Ising model, despite its relative inefficiency.

We have shown then that if we are going to simulate a Potts model with a single-spin-flip algorithm, the algorithm that we choose should depend

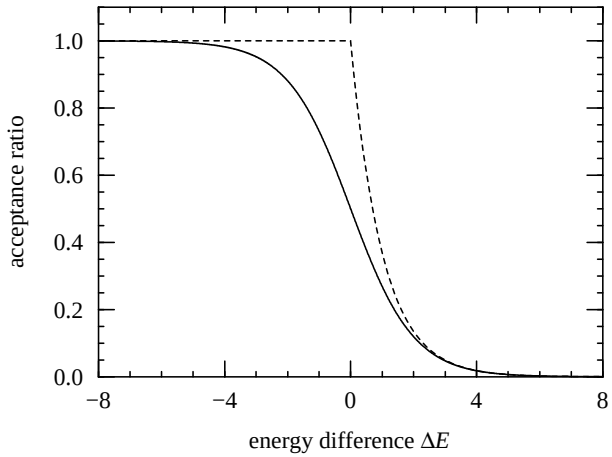


FIGURE 4.11 The acceptance ratio of the heat-bath algorithm for the Ising model (solid line) as a function of the energy difference between states $\Delta E$. For comparison, the acceptance ratio of the Metropolis algorithm is also shown (dashed line).

on the value of $q$. For small $q$ (for example, the Ising model, which is equivalent to $q = 2$) the Metropolis algorithm is the most efficient algorithm, but as $q$ gets larger there comes a point at which it is advantageous to switch over to the heat-bath algorithm. Where exactly the cross-over occurs is unfortunately dependent on what temperature we are interested in. At the low temperatures used for the simulations in Figure 4.10, clearly the cross-over was well below $q = 10$. However, at high temperatures where the probabilities of occurrence of the states of any particular spin become increasingly independent of the states of the spin's neighbours, there is less and less difference between the performance of the two algorithms, and we have to go to higher values of $q$ to make the extra complexity of the heat-bath algorithm worthwhile. In practice, for any serious study of a Potts model, one should conduct short investigatory simulations with both algorithms to decide which is more suited to the problem in hand.

### 4.5.2 Cluster algorithms for Potts models

In the last section we studied single-spin-flip algorithms for Potts models, and demonstrated that both Metropolis and heat-bath algorithms are efficient ways to perform simulations for certain ranges of temperature and $q$. Near the phase transition, however, these algorithms suffer from exactly the same problems as they did in the case of the Ising model: the correlation length diverges and the time taken to flip over the large correlated clusters that

appear gets longer and longer as we approach $T_c$. As with the Ising model, we can define a dynamic exponent $z$, and show that the correlation time $\tau$ at the critical point, measured in sweeps of the lattice, increases as $\tau \sim L^z$ with the size $L$ of the system simulated. If $z$ takes a large value, this can place severe limitations on the size of the system we can work with. As it turns out $z$, is indeed quite large. For the case of the $q = 3$ Potts model, for example, Schülke and Zheng (1995) have measured a value of $2.198 \pm 0.008$ for the dynamic exponent of the Metropolis algorithm in two dimensions, which is comparable to the value for the Ising model, and it is believed that $z$ has similar values for larger $q$ also (Landau *et al.* 1988). The solution to this problem is exactly the same as it was in the Ising model case: to try and flip the large correlated clusters all in one go using a cluster-flipping algorithm. In fact, the Wolff algorithm which we described in Section 4.2 generalizes to the Potts case in a very straightforward manner. The correct generalization is as follows.

1. Choose a seed spin at random from the lattice.

2. Look in turn at each of the neighbours of that spin. If they have the same value as the seed spin, add them to the cluster with probability $P_{\rm add} = 1 - {\rm e}^{-\beta J}$. (Note that the 2 has vanished from the exponent. This is the same factor of two that appeared in the interaction constant when we compared the Ising model to the $q = 2$ Potts model.)

3. For each spin that was added in the last step, examine each of *its* neighbours to find which ones, if any, have the same value and add each of them to the cluster with the same probability $P_{\rm add}$. (As with the Ising model, we notice that some of the neighbours may already be members of the cluster, in which case you don't have to consider adding them again. Also, some of the spins may have been considered for addition before, as neighbours of other spins in the cluster, but rejected. In this case, they get another chance to be added to the cluster on this step.) This step is repeated as many times as necessary until there are no spins left in the cluster whose neighbours have not been considered for inclusion in the cluster.

4. Choose at random a new value for the spins in the cluster, different from the present value, and set all the spins to that new value.

The proof that this algorithm satisfies detailed balance is exactly the same as it was for the Ising model. If we consider two states $\mu$ and $\nu$ of the system which differ by the changing of just one cluster, then the ratio $g(\mu \to \nu)/g(\nu \to \mu)$ of the selection probabilities for the moves between these states depends only on the number of bonds broken $m$ and the number made $n$ around the edges of the cluster. This gives us an equation of detailed

balance which reads

$$\frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)} = (1 - P_{\text{add}})^{m-n} \frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \mathrm{e}^{-\beta(E_\nu - E_\mu)}, \tag{4.45}$$

just as in the Ising case. The change in energy is also given by the same expression as before, except for a factor of two:

$$E_\nu - E_\mu = J(m - n), \tag{4.46}$$

and so the ratio of the acceptance ratios $A(\mu \to \nu)$ and $A(\nu \to \mu)$ for the two moves is

$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = [\mathrm{e}^{\beta J}(1 - P_{\text{add}})]^{n-m}. \tag{4.47}$$

For the choice of $P_{\text{add}}$ given above, this is just equal to 1. This equation is satisfied by making the acceptance ratios also equal to 1, and so, with this choice, the algorithm satisfies detailed balance.

The algorithm also satisfies the condition of ergodicity for the same reason that it did for the Ising model. Any spin can find itself the sole member of a cluster of one, and flipping the appropriate succession of such single-spin clusters can take us from any state to any other on a finite lattice in a finite number of steps.

As in the case of the Ising model, the Wolff algorithm gives an impressive improvement in performance near to the critical temperature. Coddington and Baillie (1991) have measured a dynamic exponent of $z = 0.60 \pm 0.02$ for the algorithm in the $q = 3$ case in two dimensions—a considerable improvement over the $z = 2.2$ of the Metropolis algorithm. As before, the single-spin-flip algorithms come into their own well away from the critical point because critical slowing down ceases to be a problem and the relative simplicity of these algorithms over the Wolff algorithm tends to give them the edge. One's choice of algorithm should therefore (as always) depend on exactly which properties of the model one wants to investigate, but the Wolff algorithm is definitely a good choice for examining critical properties.

The Swendsen–Wang algorithm can also be generalized for use with Potts models in a straightforward fashion, as can all of the other algorithms described in Section 4.4. In general however, the three algorithms we have described here—Metropolis, heat-bath and Wolff—should be adequate for most purposes.

### 4.5.3 Continuous spin models

A different generalization of the Ising model gives us the **continuous spin models**, in which the spins on the lattice have a continuous range of values, rather than a discrete spectrum like the Ising and Potts models. The two most widely studied such models are the **XY model** and the **Heisenberg**

**model**. In the XY model the spins are two-component vectors of unit length, which can point in any direction on a two-dimensional plane. Thus the spins can be represented by their components $s_x$, $s_y$, with the constraint that $s^2 \equiv s_x^2 + s_y^2 = 1$, or they can be represented by a single angle variable $\theta$, which records the direction that the spin points in. Note that although the spins are two-dimensional there is no reason why the lattice need be two-dimensional. For example, we can put XY spins on a cubic lattice in three dimensions if we want to.

The Heisenberg model follows the same idea, but the spins are three-dimensional unit vectors. (Again, the dimensionality of the spins and that of the lattice are independent. We can put Heisenberg spins on a two-dimensional lattice for instance.) Heisenberg spins can be represented either as three-component vectors with $s^2 \equiv s_x^2 + s_y^2 + s_z^2 = 1$, or by two angle variables, such as the $\theta$ and $\phi$ of spherical coordinates.

The Hamiltonian for each of these models is the obvious generalization of the Ising Hamiltonian:

$$H = -J \sum_{\langle ij \rangle} \mathbf{s}_i \cdot \mathbf{s}_j, \tag{4.48}$$

where $\mathbf{s}_i$ and $\mathbf{s}_j$ are the vectors representing the spins on sites $i$ and $j$. When $J > 0$ the spins can lower their energy by lining up with one another and the model is ferromagnetic, just as in the Ising case. When $J < 0$, the model is anti-ferromagnetic.

We can also define similar models with higher-dimensional spins—unit vectors in four or more dimensions—and there are many other generalizations of the idea to continuous-valued spins which have symmetries other than the simple spherical symmetry of these models. In this section we concentrate on the XY/Heisenberg class of models as examples of how Monte Carlo algorithms can be constructed for continuous spin models.

The first thing we notice about these continuous spin models is that they have a continuous spectrum of possible states, with energies which can take any real value between the ground state energy and the energy of the highest-lying state. The appropriate generalization of the Boltzmann distribution to such a model is that the probability of finding the system in a state with energy between $E$ and $E + \mathrm{d}E$ is

$$p(E)\,\mathrm{d}E = \frac{\mathrm{e}^{-\beta E}\rho(E)\,\mathrm{d}E}{Z}, \tag{4.49}$$

where $\rho(E)$ is the **density of states**, defined such that $\rho(E)\,\mathrm{d}E$ is the number of states in the interval $E$ to $E + \mathrm{d}E$. You may well ask what exactly it means to talk about the number of states in an energy interval when we have a continuous spectrum of states. This was one of the toughest problems which plagued the nineteenth century study of statistical mechanics,

and in fact it cannot be answered within the realm of the classical systems we are studying here. Only by studying quantum mechanical systems and then taking their classical limit can the question be answered properly. However, for the purposes of Monte Carlo simulation, we don't actually need to know what the density of states in the system is, just as we didn't need to know the complete spectrum of energies in the discrete case—the Monte Carlo method ensures that, provided we wait long enough for the system to equilibrate, all states will appear with their correct equilibrium probabilities in the simulation.[15]

The partition function $Z$ for a system with a continuous energy spectrum is the obvious generalization of the discrete case:

$$Z = \int \mathrm{e}^{-\beta E} \rho(E)\, \mathrm{d}E, \tag{4.50}$$

so that the probabilities in Equation (4.49) are properly normalized.

Note also that these continuous spin models have a continuum of ground states. For the ferromagnetic XY model, for example, the ground state is one in which all the spins are lined up pointing in the same direction. However, that direction can equally well be any of the allowed directions of the spins. The system therefore has one degree of freedom—rotation of all the spins by the same amount—which costs it nothing in energy, and this result extends to all the higher energy states of the lattice also. (Field theorists refer to this as a **Goldstone mode**.)

So, how do we go about simulating a continuous spin system? Let us take the XY model as our example. The simplest Monte Carlo algorithm for this model is the single-spin-flip Metropolis algorithm. The algorithm works in much the same way as it did for the Ising and Potts models. The only question we need to answer is what is the equivalent of "flipping" an Ising spin for a model in which the spins have continuous values? Clearly if we simply reversed the direction of a spin on each move we would not get an ergodic algorithm—each spin could then only point in the direction it started in or the opposite direction and there would be no way for it to reach any of the other possible directions that it is allowed to point in. So instead, rather than reversing the spin, we choose a new direction for it at random. Then the algorithm would look like this:

1. Choose a spin at random from the lattice, and choose a new direction for it by generating a random number between 0 and $2\pi$ to represent its new angle.[16]

[15] However, if the reader is interested in the derivation of the quantum mechanical density of states and its classical limit, we can recommend the excellent treatment by Pathria (1972).

[16] In the case of the Heisenberg model, whose spins are three-dimensional, choosing

2. Calculate the change in energy $\Delta E$ if we were to make this move.
3. Accept or reject the move with an acceptance ratio $A$ thus:

$$A = \begin{cases} \mathrm{e}^{-\beta\Delta E} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.51}$$

This algorithm now satisfies ergodicity, since we have a finite chance of accepting any move, and can therefore in theory get from any state of the lattice to any other in $N$ moves or fewer, where $N$ is the number of spins on the lattice. It also satisfies detailed balance, since the selection probabilities $g(\mu \to \nu)$ and $g(\nu \to \mu)$ for a move to go from state $\mu$ to state $\nu$ or back again are the same, and the ratio of the acceptance probabilities is just $\exp(-\beta\Delta E)$, as it should be.

This form of the Metropolis algorithm is the most widely used single-spin-flip algorithm for the XY model. However, we should point out that it is not the only form the algorithm can take, nor is it known whether this is the most efficient form. The thing is, there is quite a lot of flexibility about the choice of the new state for the spin we are proposing to change. The only restriction that the algorithm places on it is that the selection probabilities $g(\mu \to \nu)$ and $g(\nu \to \mu)$ for the forward and reverse moves be equal. This in turn only requires that the probability of picking a new direction $\theta'$ for the spin be a function solely of the angle $|\theta' - \theta|$ between the two states. Any function will do however, not just the uniform one we used above. If we choose a function which is biased towards small changes in the direction of our spin, then we will increase the acceptance ratio of the algorithm because the energy cost of any move will be small. However, the change of state will be small too, so the algorithm may take a lot of moves to reach equilibrium. (For example, many small changes in the direction of single spins would be necessary to cool the system from an initial $T = \infty$ state in which the directions of the spins are all random to a low-temperature state in which they all point in approximately the same direction.) Conversely, an algorithm in which we preferentially propose larger changes in direction $|\theta' - \theta|$ for our Monte Carlo moves would take fewer accepted moves to get from one state to another on average, but would tend to suffer from a lower mean acceptance ratio, because with larger changes in spin direction moves can have a larger energy cost.

As we mentioned, no one has, to our knowledge, investigated in any detail the question of how best to compromise between these two cases—all the large Metropolis studies of models like the XY and Heisenberg models have been carried out with the uniform choice of directions described above. It seems likely that the algorithm would benefit from a choice of smaller

a new direction at random isn't quite so simple. The problem of choosing a random spherically symmetric direction is discussed in Section 16.2.1.

angles at low temperatures, where the acceptance ratio for higher energy excitations becomes extremely small, and larger angles at high temperatures, where the important point is to cover the space of possible states quickly, the acceptance ratio being close to unity regardless of the moves proposed. A detailed investigation of this point would make an interesting study.[17]

Many continuous spin models exhibit a phase transition similar to that of the Ising model between a ferromagnetic and a paramagnetic phase, with the accompanying problem of critical slowing down. As with the Ising model, cluster algorithms can reduce these problems. A version of the Wolff algorithm which is suitable for the XY and Heisenberg models was given by Wolff in his original paper in 1989. The basic idea is that we choose at random both a seed spin from which to start building our cluster and a random direction, which we will denote by a unit vector $\hat{\mathbf{n}}$. Then we treat the components $\hat{\mathbf{n}} \cdot \mathbf{s}_i$ of the spins in that direction roughly in the same way as we did the spins in the Ising model. A neighbour of the seed spin whose component in this direction has the same sign as that of the seed spin can be added to the cluster by making a link between it and the seed spin with some probability $P_{\text{add}}$. If the components point in opposite directions then the spin is not added to the cluster. When the complete cluster has been built, it is "flipped" by reflecting all the spins in the plane perpendicular to $\hat{\mathbf{n}}$.

The only complicating factor is that, in order to satisfy detailed balance, the expression for $P_{\text{add}}$ has to depend on the exact values of the spins which are joined by links thus:

$$P_{\text{add}}(\mathbf{s}_i, \mathbf{s}_j) = 1 - \exp[-2\beta(\hat{\mathbf{n}} \cdot \mathbf{s}_i)(\hat{\mathbf{n}} \cdot \mathbf{s}_j)]. \tag{4.52}$$

Readers might like to demonstrate for themselves that, with this choice, the ratio of the selection probabilities $g(\mu \to \nu)$ and $g(\nu \to \mu)$ is equal to $\mathrm{e}^{-\beta\Delta E}$, where $\Delta E$ is the change in energy in going from a state $\mu$ to a state $\nu$ by flipping a single cluster. Thus, detailed balance is obeyed, as in the Ising case, by an algorithm for which the acceptance probability for the cluster flip is 1. This algorithm has been used, for example, by Gottlob and Hasenbusch (1993) to perform extensive studies of the critical properties of the XY model in three dimensions.

Similar generalizations to continuous spins are possible for all the algorithms we discussed in Section 4.4. However, the Metropolis algorithm and the Wolff algorithm are probably adequate to cover most situations.

---

[17] In a system where the energy varies quadratically about its minimum as we change the values of the spins, one can achieve a roughly constant acceptance ratio by making changes of direction whose typical size varies as $\sqrt{T}$. This might also be a reasonable approach for systems such as the XY and Heisenberg models when we are at low temperatures because, to leading order, the energies of these systems are quadratic about their minima.

## Problems

**4.1** An Ising system can be considered as being composed of regions of predominantly up- or down-pointing spins separated by domain walls. Assuming that in equilibrium the walls roughly speaking perform a random walk about the system, show that the dynamic exponent of the model (in any number of dimensions) should be about $z = 2$.

**4.2** Suggest a simple modification of the Wolff algorithm that would allow us to simulate Ising systems in a non-zero magnetic field $B$.

**4.3** What is the equivalent of Equation (4.24) for the Swendsen–Wang algorithm?

**4.4** Modify the Metropolis program given in Appendix B to carry out the heat-bath algorithm for the two-dimensional Ising model (see Equation (4.44)).

**4.5** Write a program to implement the Wolff algorithm for a $q$-state Potts model. (Hint: you might want to use the Wolff algorithm program for the Ising model given in Appendix B as a starting point.)