Lattice Simulations of the ϕ^4 Theory and Related Systems

Tadeusz Pudlik '09

Advisor: Professor William Loinaz
April 20th, 2009

Submitted to the Department of Physics of Amherst College
in partial fulfillment of the requirements for the degree of
Bachelor of Arts with Honors

Abstract

We introduce a sequence of increasingly sophisticated Monte Carlo methods to treat a set of problems involving random walks, the Ising model and the ϕ^4 quantum field theory. We begin with a series of exercises in which we try to predict and measure the behavior of simple, non-reversing and self-avoiding random walks. In the process, we prove a few analytic results, one of which we were unable to locate in the literature. We then study the two dimensional Ising model in some detail, discussing the analytical difficulties and comparing exact results with Monte Carlo estimates. Throughout this part of the thesis, we focus on the techniques which will later be employed to more frontier problems, including sampling schemes (simple, importance, Markov-chain) and statistical analysis of Monte Carlo data.

In the latter part of the thesis, we apply the tools learned thus far to the ϕ^4 quantum field theory. We begin with a short introduction to QFT through the path integral formalism, but quickly point out the equivalence of field-theoretic and statistical problems and from then on focus on the Landau-Ginzburg model. We corroborate the general result of Schaich (2006) in two dimensions, the nonlinear dependence of the coupling constant $[\lambda/\mu^2]$ on λ, but point out that his implicit acceptance of finite size scaling assumptions may have led him to underestimate the magnitude of the effect.

Acknowledgements

Like any extended project, this work could not have been completed without the kind support of numerous individuals. First of these is my advisor, Professor William Loinaz, who offered me his patience, advice and contagious good spirits throughout the last ten months. I would also like to acknowledge Professor Amy Wagaman, to whom I turned repeatedly with statistics questions, and Mr. Andrew Anderson, who explained the workings of the Amherst Computing Cluster to me. I am grateful to the Office of the Dean of the Faculty for supporting me financially throughout the early stages of my work, in the Summer of 2008.

More broadly, I would like to thank my Mother and the family back home for letting me go four years ago and for staying close ever since. Thanks are also due to my small Amherst universe that has sustained me throughout college.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is the travel log of a journey through the applications of Monte Carlo methods in physics. The original aim of this journey was the verification of the work of Charng (2001), who placed an upper bound on the probability of multiparticle production in the four-dimensional ϕ^4 quantum field theory. The outline of the path leading there was to start by learning some programming, perhaps through the implementation of random walk programs, move on to the Ising model for a couple months, corroborate the general thrust of the findings of Schaich (2006), and finally reproduce the work of Charng.

As we should have perhaps expected, there were quite a few detours on the way. The supposedly brief adventure with random walks grew into the first full chapter, which constitutes an introduction to the simplest Monte Carlo methods. It begins with an explanation of how to calculate π using a compass, a ruler and a bucket of rice, follows up with a discussion of the solved problems of the simple and non-reversing random walks, and culminates with our first encounter with the so-called classical n-vector model, in the guise of the self-avoiding walk (the $n = 0$ case). The n-vector model, which will prove to be a recurring theme, defeats us squarely in this chapter, demonstrating the severe limitations of the simple sampling approach used thus far.

Chapter 3 is devoted to developing the dynamic Monte Carlo method. This method allows us to deal with distributions that are impossible to sample directly, often because just writing them out would take an amount of time on the order of the age of the universe. Such distributions turn out to be prevalent and incredibly important: they include the Boltzmann distributions governing the behavior of all realistic models in statistical me-

chanics. To treat them, we develop sampling algorithms which mimic the dynamics of a system in thermal equilibrium, including the famous Metropolis algorithm. Most of this development is done in the context of the Ising model, one of the simplest models exhibiting a phase transition. We quote some parts of its (very difficult) exact solution and compare them with the results of our simulations. The dynamic algorithms perform very well, thereby settling our score against the n-vector model, of which Ising is the $n = 1$ case.

By the end of Chapter 3, we are already halfway through our tour of Monte Carlo (in terms of pages, at least), so it's only fitting that we should finally discuss quantum field theory. We offer only a very brief glimpse of this grand topic: we introduce the path integral formulation of quantum mechanics in a successful attempt to save the Klein-Gordon equation, run into the ϕ^4 theory (Klein-Gordon's more interesting cousin) and immediately jump back to the familiar ground of statistical mechanics by exploiting the formal equivalence between a path integral and a partition function. We show that the ϕ^4 is equivalent to the Landau-Ginzburg model, which is in turn an approximation to the Ising model. It takes us a full section to iron out the details of how to transplant the Landau-Ginzburg on a lattice, but when we succeed, the entire machinery of Chapter 3 is at our disposal.

We proceed to use this machinery, and in particular a technique known as finite size scaling, to explore the principal subject of Loinaz and Willey (1998), Bednarzyk (2001) and Schaich (2006): the critical coupling of the ϕ^4 theory in two dimensions, $[\lambda/\mu]_{\text{crit}}$. In the process, we make a leap of faith by extending finite size scaling beyond its proper domain of validity; this does reproduce the results of Schaich, but leaves us wondering whether they're correct.

After all the detours, we turn to the work of Charng very late—too late to make sense of how, exactly, he proposes to use data from lattice simulations to calculate the quantities of interest. We struggle with the problem for a while, but in the absence of references (the "forthcoming" *Physical Review* article on the topic expounding on the 1996 *PRL* by Mawhinney and Willey was never published) are forced to capitulate.

The lemmas postulated in Chapters 2–4 are proven in Appendix A. Most of the code used in our simulations, including the shell scripts for submitting jobs to the computing cluster, can be found in Appendix B. Finally, Appendix C is a guide to the literature I found helpful in completing this project.

Chapter 2

Static Monte Carlo

In this chapter, we present the results of a number of simple computer experiments performed as exercises in the summer of 2008. All of them are examples of what are known as static Monte Carlo methods: we generate samples directly from a known distribution and use sample averages as estimators of population averages. By the end of this chapter, both the power and the limitations of this technique should become apparent.

2.1 Simple Example: Estimating π

One of the simplest (but still instructive) applications of the Monte Carlo method is estimating the value of π using only a compass, a ruler and a large number of rice grains.[1] The procedure is as follows:

1. Draw a circle inscribed in a square. Call the circle's radius a. These shapes must be sufficiently large so that the rice grains can be considered essentially point-like.

2. Distribute rice grains uniformly over the surface of the square.

3. Count how many rice grains you threw, and how many of them fell within the circle.

Since the rice grains were distributed uniformly over the square, the ratio of the number that fell within the circle to the total should be equal to the

[1] This section is largely inspired by the opening section of Krauth (2006). A similar experiment was also carried out by Bednarzyk (2001).

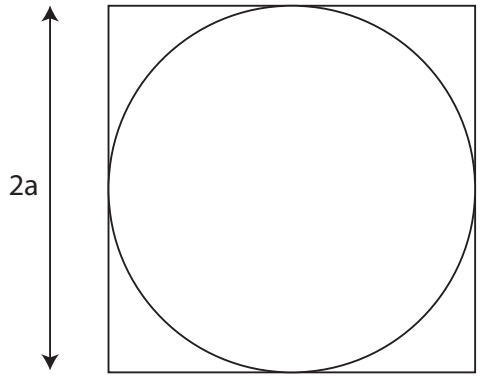


Figure 2.1: Monte Carlo estimation of π.

ratio of the circle's area to that of the square:

$$\frac{N_{\text{circle}}}{N_{\text{square}}} = \frac{A_{\text{circle}}}{A_{\text{square}}} = \frac{\pi a^2}{(2a)^2} = \frac{\pi}{4}. \tag{2.1}$$

Consequently,

$$\pi \approx 4\frac{N_{\text{circle}}}{N_{\text{square}}}. \tag{2.2}$$

While elegant in principle, this procedure quickly becomes rather tedious to carry out, even for relatively small numbers of rice grains. Fortunately, it is easy to implement it on a computer, which will then only take seconds to "throw" millions of "grains" by generating random numbers uniformly distributed on the square.[2] A sample implantation is shown in Appendix B.2.1.

We intuitively expect that the quality of the estimate (call the estimate η) depends on the number of rice grains thrown (call it x), but how quickly does it improve? Every time a rice grain is thrown, it has a certain probability (say θ) of falling into the circle. Let the two possible outcomes be $\xi = 1$ if the grain does fall into the circle, and $\xi = 0$ otherwise. Then the variance of ξ is

$$\text{Var}(\xi) = \langle \xi^2 \rangle - \langle \xi \rangle^2 = \left(\theta \cdot 1^2 + (1-\theta)\cdot 0^2\right) - (\theta \cdot 1 + (1-\theta)\cdot 0)^2 \tag{2.3}$$

$$= \theta(1-\theta). \tag{2.4}$$

[2]Random number generation is briefly discussed in Appendix B.1.2

Since the variance of a sum of independent variables is equal to the sum of their variances, the variance of $N_{\text{circle}} = \Sigma\xi$ is $x\text{Var}(\xi) = x\theta(1-\theta)$. The variance of our estimate, η, is therefore

$$\text{Var}(\eta) = \text{Var}\left(4\frac{N_{\text{circle}}}{N_{\text{square}}}\right) = 16\frac{x\theta(1-\theta)}{x^2} = 16\frac{\theta(1-\theta)}{x} \tag{2.5}$$

(We used the facts that $\text{Var}(cv) = c^2\text{Var}(v)$ for any constant c, and that $x = N_{\text{square}}$.) This might not look like much progress, since we don't know the value of θ. However, since θ is a probability, it lies in the closed interval $[0, 1]$; and on this interval, $\theta(1-\theta) < 1/4$.[3] Therefore,

$$\text{Var}(\eta) < \frac{4}{x}. \tag{2.6}$$

In some sense, this answers the question posed at the head of the paragraph. A more informative answer, however, can be obtained using Chebyshev's inequality,[4] which in our case states that

$$\Pr(|\eta - \pi| \geq \epsilon) \leq \frac{\text{Var}(\eta)}{\epsilon^2}, \tag{2.7}$$

where "Pr" stands for probability. It follows that the 68% confidence band is *no wider* than $\sqrt{\text{Var}(\eta)}/0.32$. This is the smallest rigorous upper bound that can be placed on the size of the confidence interval using only the knowledge of the mean and variance of ξ.[5]

We perform a series of experiments for different numbers of iterations. For each experiment, we report (in Table 2.1) the average of 100 trials as our best estimate, and uncertainty calculated using Equation 2.7. Since we're using an average as the estimate, its variance is

$$\text{Var}(\bar{\eta}) = \text{Var}\left(\frac{\eta_1 + \eta_2 + \cdots + \eta_{100}}{100}\right) = \frac{\text{Var}(\eta)}{100} < \frac{1}{25x}. \tag{2.8}$$

The uncertainties quoted are the 68% confidence bands in the sense of Equation 2.7, but with the variance of the sample mean, $\text{Var}(\bar{\eta})$, used instead of the variance of a single estimate, $\text{Var}(\eta)$.

[3] Of course, we're just "pretending" not to know θ: since the actual value is $\pi/4$, $\theta(1-\theta) \approx 0.17$.

[4] Chebyshev's inequality is proven in every probability course; see, for instance, DeGroot and Schervish (2002, section 4.8).

[5] In this case, we actually know more than the variance of ξ: we know that its distribution is binomial, and could calculate the confidence bounds directly from our knowledge of the distribution. However, such knowledge is very rarely available in Monte Carlo simulations, so we will not pursue this technique.

Table 2.1: Estimates of π using the program. The best estimate is the average of the results of 100 trials; the 68% confidence interval given by Chebyshev's inequality is given as an estimate of the uncertainty. For reference, the actual value is $\pi = 3.14159265\ldots$

Iterations (x)	Estimate	Run Time (s)
10^2	$3.16(4)$	2
10^3	$3.15(2)$	3
10^4	$3.142(4)$	2
10^5	$3.141(2)$	4
10^6	$3.1414(4)$	14
10^7	$3.1416(2)$	48

A disappointing feature of this method is that to improve the precision of the estimate by a factor of 10, we need to increase x by a factor of 100. Even though the running time is only linear in x (as the computer scientists would say, it's $O(x)$: of order x), this is still a hopelessly slow method of estimating π. For comparison, using Wallis' formula (see Sondow and Weisstein 2008):

$$\frac{\pi}{2} = \frac{2 \cdot 2}{1 \cdot 3} \cdot \frac{4 \cdot 4}{3 \cdot 5} \cdot \frac{6 \cdot 6}{5 \cdot 7} \cdots \tag{2.9}$$

gives a result accurate to within 10^{-4} after about $3 \cdot 10^5$ multiplications; achieving similar precision using our algorithm required more than $2 \cdot 10^9$ multiplications—*not including* the operations performed in generating random numbers![6]

Nonetheless, the Monte Carlo method is widely used (though not for estimating π). There are problems for which it offers better performance than any traditional numerical technique, and even some for which it's the only feasible approach. The most famous example of a problem that becomes the former is evaluating integrals in many dimensions. Our focus in this thesis, however, will be on the latter. The first example of such a problem is the estimation of certain properties of self-avoiding random walks (SAWs), which will be discussed in the next section.

[6] Averaging 100 trials, each consisting of 10^7 iterations, in each of which we need to square the two coordinates of a point to find whether it fell within the circle or not.

2.2 Random Walks

Random walks are used in many areas of applied mathematics, from financial modeling to the study of polymer structure (see Sethna, 2006, Exercises to Chapter 2 for an introduction to a broad array of applications). They are also of theoretical interest, as some of the simplest systems exhibiting critical phenomena. In the context of Monte Carlo work, it's worthy of note that every sequence of estimates we generate (such as the estimates of π from individual trials in the previous section) constitutes a random walk.

In this section, we will consider only random walks on a discrete *rectangular lattice*, a grid of points in a d-dimensional Cartesian space invariant under translation by a distance of $r = 1$ in the direction of any of the basis vectors. This restriction makes both numerical simulations and the analytical treatment much simpler, but at the apparent price of making the models inapplicable to most situations of practical interest: Brownian motion, foraging ant dynamics, polymer folding and other phenomena modeled as random walks generally don't take place on a rectangular lattice. There is some truth in such an accusation, but there is also less to it than meets the eye. For one thing, random walks on a lattice are still of inherent interest as a mathematical puzzle, relatively simple systems whose behavior (even in terms of statistical aggregates) is not always possible to predict. More importantly, however, random walks can be divided into so-called *universality classes*; in the vicinity of the critical point, the behavior of all systems belonging to one universality class is the same. For random walks, the critical point corresponds to the asymptotic limit as the number of steps becomes infinite. A random walk's membership in a universality class depends on dimensionality, symmetries and other general properties, but, somewhat surprisingly, is independent of whether the space in which the walk takes place is discretized. This implies that lattice walks may be good—in some respects perfect—models of apparently much more complicated systems.

We will not discuss the concept of universality in greater detail in this work, but the interested reader is referred to Sokal (1994) for a brief discussion in the context of random walks, or Sethna (2006, Chapter 12) for a general introduction. Because of the intimate connection between universality and critical behavior, it is a major theme of books such as Binney et al. (1992). In the rest of this section, we discuss three types of random walks: the simple random walk, the non-reversing random walk and the self-avoiding random walk.

2.2.1 Simple Random Walk

The most natural question to ask about a random walk is, how does its expected squared end-to-end distance, $\langle R_N^2 \rangle$, vary with the size of the walk, N? For the conceptually simplest of them, the unbiased, unconstrained random walk, this question is easily answered analytically—even in the d dimensional case![7] Notice that the length, R_N, of a N-step walk is given by

$$R_N = \sum_{i=1}^{N} \boldsymbol{s_i}, \tag{2.10}$$

where $\boldsymbol{s_i}$ is the ith step, which we will take to have length 1 and have the same probability of pointing in each of the $2d$ possible directions. It follows that

$$\begin{aligned} \langle R_N^2 \rangle &= \langle (\boldsymbol{R_{N-1}} + \boldsymbol{s_N})^2 \rangle \\ &= \langle R_{N-1}^2 \rangle + 2\langle \boldsymbol{R_{N-1}} \cdot \boldsymbol{s_N} \rangle + \langle s_N^2 \rangle \\ &= \langle R_{N-1}^2 \rangle + 1 \end{aligned} \tag{2.11}$$

The transition to the last line deserves some comment. Clearly, $s_i^2 = 1$ regardless of which of the possible values $\boldsymbol{s_i}$ takes, so $\langle s_N^2 \rangle = 1$. However, because the walk is unbiased, $\boldsymbol{R_{N-1}} \cdot \boldsymbol{s_N}$ takes on nonzero values of R_{N-1} and $-R_{N-1}$ with equal probabilities; consequently, its expectation is 0. Since $R_1^2 = s_i^2 = 1$, it follows that

$$\langle R_N^2 \rangle = N. \tag{2.12}$$

Note the curious result (which does not generalize to constrained walks) that the number of dimensions doesn't affect the expected distance from the origin to the walk's endpoint.

As an exercise, one can write a simple simulation verifying this result. The most direct approach is through *enumeration*: writing down every possible walk up to a given length. Because the number of walks increases exponentially with length (there are $4d^N$ walks of given N), this approach can only be used for relatively short walks. Nonetheless, as one would expect, the results of the simulation were in perfect agreement with prediction.

Another approach is a Monte Carlo method, in which a representative[8] sample is used instead of the entire distribution. We generated 1,000,000

[7] The argument given here is along the lines of Sethna (2006, chapter 2).

[8] By representative, we mean that every possible walk of a particular length was equally likely to be generated.

Table 2.2: Regression results for the unconstrained random walk.

Walk	Observed $\langle R_N^2 \rangle$	Expected $\langle R_N^2 \rangle$
$d = 1$	$1.0000(1)N$	N
$d = 2$	$0.99983(8)N$	N
$d = 3$	$1.00003(6)N$	N

walks of each length between 1 and 150 steps in one, two and three dimensions, and regressed R_N^2 on N.[9] A subtlety arose here: is it advisable to use ordinary least squares, or should we take the variances of the data points into account and run generalized least squares? An induction argument similar to the one used in deriving R_N^2 can be used to show that the variance of a random walk of length N is

$$\sigma_N^2 = \sigma_{N-1}^2 + 4(N-1). \tag{2.13}$$

Since $\sigma_1^2 = 0$,

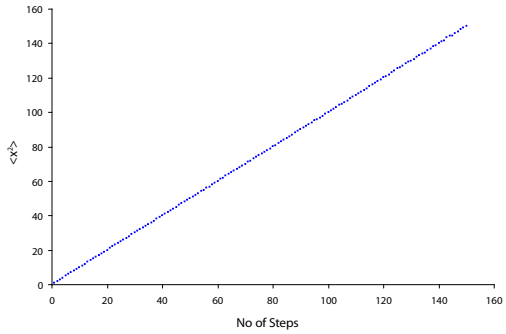
$$\begin{aligned} \sigma_N^2 &= 4(N-1) + 4(N-2) + 4(N-3) + \ldots + 4(N-(N-1)) \\ &= 4(1+2+3+\ldots+N-1) = 2N(N-1). \end{aligned} \tag{2.14}$$

The data points are averages, so their variances are $\sigma^2 = \sigma_N^2/N = 2N-2$. In the light of this fact, we weighted each data point by $1/\sigma^2$ in the regression. The plots of our data are shown in Figure 2.2; the regression results are in Table 2.2. The agreement between experiment and prediction is perfect.[10]

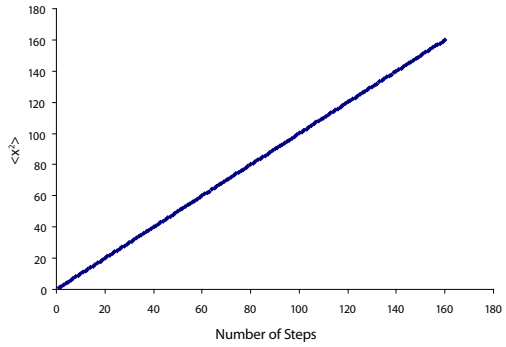
The simple random walk has many fascinating properties: one could ask, how likely is it that a walk will return to its origin? How long, on average, would one have to wait for this to take place? What is the relationship between the walk and the diffusion equation? These questions, however, are not related to our major theme of computer simulation, and the interested reader is referred to analytical treatments such as that of Spitzer (2001).

[9] The simulation code can be found in Appendix B.2.2. All regressions were carried out in *Mathematica*, using the commands LinearModelFit or NonlinearModelFit.

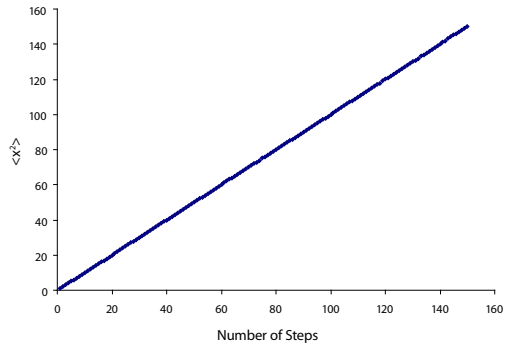
[10] What about the value for $d = 2$ not being within uncertainty of the prediction? This is actually expected: our measure of uncertainty is the standard deviation of the estimate, so about one in three results should fall outside of this range, as is indeed the case.



(a) $d = 1$



(b) $d = 2$



(c) $d = 3$

Figure 2.2: $\langle x^2 \rangle$ as a function of N for the random walk in one, two and three dimensions. Each data point is the average over a sample of 1,000,000 walks.

2.2.2 Non-Reversing Random Walk

The first constrained random walk we will consider is the non-reversing random walk (NRRW). It is identical to the simple random walk of the previous section, except for the fact that it never reverses its last step ($\boldsymbol{s_i} \neq -\boldsymbol{s_{i-1}}$). While less common in applications than the simple and self-avoiding varieties,[11] simulating the NRRW was a natural step towards more complicated constrained walks. (One can think of both the NRRW and the self-avoiding walk as walks that avoids crossing m of their last steps, with $m = 1$ for the NRRW and $m = \infty$ for the SAW.) Additionally, the non-reversing random walk has been studied in earlier theses on lattice simulations (Bednarzyk, 2001; Schaich, 2006).

Analytical Treatment The behavior of this walk in one dimension is not particularly interesting: $\langle R_N^2 \rangle = N^2$, since the walk must continue in the initially chosen direction. As the dimensionality of the walk increases, we would expect the difference in end-to-end distance between a NRRW and the simple walk to decrease, because the set of excluded moves (direction reversals) becomes an ever-smaller subset of all the possible moves. To derive the exact prediction, we will proceed by induction, like in Section 2.2.1.[12] For arbitrary N,

$$\langle R_N^2 \rangle = \langle (\boldsymbol{R_{N-1}} + \boldsymbol{s_N})^2 \rangle, \tag{2.15}$$

$$= \langle R_{N-1}^2 \rangle + 2\langle \boldsymbol{R_{N-1}} \cdot \boldsymbol{s_N} \rangle + \langle s_N^2 \rangle. \tag{2.16}$$

Since $\langle s_N^2 \rangle = 1$,

$$\langle R_N^2 \rangle = \langle R_{N-1}^2 \rangle + 2\langle \boldsymbol{R_{N-1}} \cdot \boldsymbol{s_N} \rangle + 1 \tag{2.17}$$

$$= \langle R_{N-1}^2 \rangle + 2\langle (\boldsymbol{R_{N-2}} + \boldsymbol{s_{N-1}}) \cdot \boldsymbol{s_N} \rangle + 1 \tag{2.18}$$

$$= \langle R_{N-1}^2 \rangle + 2\langle \boldsymbol{R_{N-2}} \cdot \boldsymbol{s_N} + \boldsymbol{s_{N-1}} \cdot \boldsymbol{s_N} \rangle + 1 \tag{2.19}$$

$$= \langle R_{N-1}^2 \rangle + 2\langle \boldsymbol{R_{N-2}} \cdot \boldsymbol{s_N} \rangle + 2\langle \boldsymbol{s_{N-1}} \cdot \boldsymbol{s_N} \rangle + 1 \tag{2.20}$$

Note the similarity between Equations 2.17 and 2.20: we stepped down the index on the $\boldsymbol{R_i}$ in exchange for a term of the form $2\langle \boldsymbol{s_N} \cdot \boldsymbol{s_i} \rangle$. This process

[11] One of the few applications we are aware of is to the dynamics of foraging ants (see Walker et al., 2006).

[12] We could not find a reference containing the expression of interest; the derivation that follows is our own.

can be continued until we're left only with $\boldsymbol{R_1} = \boldsymbol{s_1}$:

$$\langle R_N^2 \rangle = \langle R_{N-1}^2 \rangle + 2\langle \boldsymbol{s_{N-1}} \cdot \boldsymbol{s_N} \rangle + 2\langle \boldsymbol{s_{N-2}} \cdot \boldsymbol{s_N} \rangle + \cdots + 2\langle \boldsymbol{s_1} \cdot \boldsymbol{s_N} \rangle + 1, \tag{2.21}$$

$$= \langle R_{N-1}^2 \rangle + 1 + 2\sum_{i=1}^{N-1} \langle \boldsymbol{s_{N-i}} \cdot \boldsymbol{s_N} \rangle. \tag{2.22}$$

To proceed, we need an expression for $\langle \boldsymbol{s_{N-i}} \cdot \boldsymbol{s_N} \rangle$ in terms of d, i and N. Its derivation is somewhat involved, so it was relegated to Appendix A.1; here, we only quote the result.

Lemma 1. *For a non-reversing random walk on a rectangular, d dimensional lattice,*

$$\forall i \in \mathbb{N} < N \quad \langle \boldsymbol{s_{N-i}} \cdot \boldsymbol{s_N} \rangle = \left(\frac{1}{2d-1} \right)^i \tag{2.23}$$

Using Lemma 1, we rewrite Equation 2.22 as

$$\langle R_N^2 \rangle = \langle R_{N-1}^2 \rangle + 1 + 2\sum_{i=1}^{N-1} \left(\frac{1}{2d-1} \right)^i. \tag{2.24}$$

Keeping in mind that $\boldsymbol{R_1} = \boldsymbol{s_1} \Rightarrow \langle R_1^2 \rangle = 1$, we perform an induction using the preceding equation, and obtain

$$\langle R_N^2 \rangle = N + 2\sum_{i=1}^{N-1} i \left(\frac{1}{2d-1} \right)^{N-i}. \tag{2.25}$$

Finite sums are generally troublesome, but for this one *Mathematica* finds a concise form:

$$\langle R_N^2 \rangle = N - \frac{2N + 2d - 2Nd - 1 + (1-2d)\left(1/(2d-1)\right)^N}{2(d-1)^2}. \tag{2.26}$$

The formula is more complicated than that for the simple random walk, but it *is* an exact closed-form expression (something that remains elusive for the SAW, say), and it reduces nicely in the $d = 2$ and $d = 3$ cases which are of primary interest:

$$\langle R_N^2 \rangle = \begin{cases} 2N + \frac{3^{1-N}}{2} - \frac{3}{2} & \text{for } d = 2, \\ \frac{3}{2}N + \frac{5^{1-N}}{8} - \frac{5}{8} & \text{for } d = 3. \end{cases} \tag{2.27}$$

Table 2.3: Regression results for the non-reversing random walk.

Walk	Observed $\langle R_N^2 \rangle$	Expected $\langle R_N^2 \rangle$
$d = 2$	$2.000(1)N + 0.5009(2) \cdot 3^{1-N} - 1.501(1)$	$2N + 0.5 \cdot 3^{1-N} - 1.5$
$d = 3$	$1.5000(8)N + 0.1245(1) \cdot 5^{1-N} - 0.6246(8)$	$\frac{3}{2}N + 0.125 \cdot 5^{1-N} - 0.625$

Simulations We performed Monte Carlo simulations of the $d = 2$ and $d = 3$ NRRW, similar to the ones carried out for the simple random walk. For each size from 1 to 150 steps, 1,000,000 walks were generated and their average end-to-end distance squared was recorded.[13] In fitting to our model, Equation 2.27, we would like to use a weighted regression, since we expect the variance of the walk lengths to increase with N; however, the exact functional form of the variance is likely difficult to derive.[14] Therefore, we will use the sample variance as an estimate of the population variance. The regression results are in Table 2.3; they are in good agreement with predicted values.

The reader may be curious as to how the exact model of Equation 2.27 compares with the heuristic model of Schaich (2006, p. 12):

$$\langle R_N^2 \rangle \approx N \left(1 + \sum_i \frac{1}{d^i} \right). \tag{2.28}$$

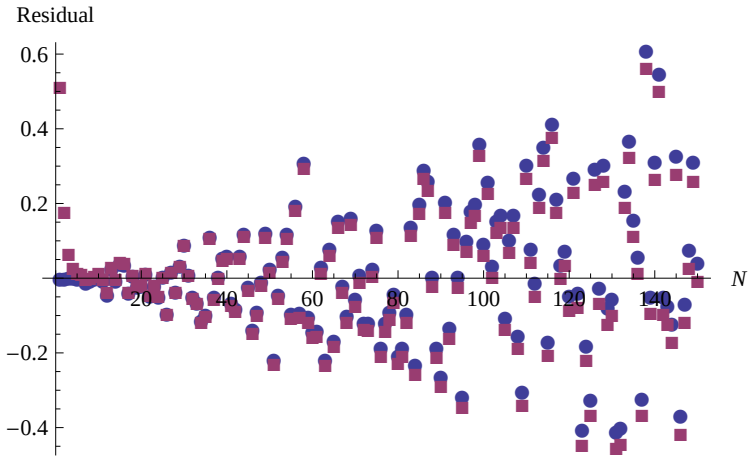
To this end, we regressed our data on both models and plotted the regression residuals (see Figure 2.3). The differences are noticeable for both the shortest ($N < 5$) and the longest ($N > 80$) walks; there's a trend in the residuals. The problem is exacerbated if we weigh the data points by their uncertainties, since the neglected a^{N-1} term is most important for short walks, which also have the smallest spread of lengths.

2.2.3 Self-Avoiding Walk

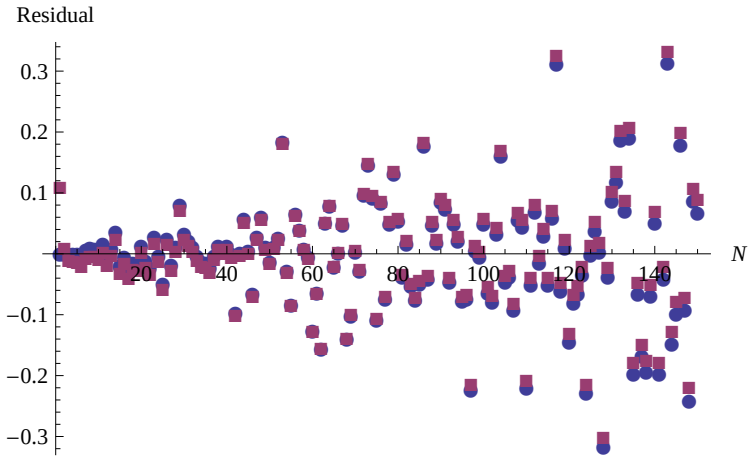
The self-avoiding random walk has been extensively studied, primarily as a model for linear polymers in a good solvent, such as DNA or proteins.

[13] Part of the simulation code can be found in Appendix B.2.3.

[14] There is one more nagging problem: the variance of $\langle R_1^1 \rangle$ equals zero identically, so its inverse cannot be used as a weight. We approximated this variance as 0.001 for the purposes of the regression, two orders of magnitude below the smallest nonzero variance in the data set.



(a) $d = 2$



(b) $d = 3$

Figure 2.3: Regression residuals for the two (a) and three (b) dimensional non-reversing random walk. The blue circles are the exact model and the violet squares—Schaich's heuristic approximation. As expected, the heuristic model performs poorly for short walks. (A weighted regression was used to estimate the exact model, but an unweighted one was used for the heuristic one, for reasons explained in the text.)

It turns out that these substances belong to the same universality class as the SAW, implying that some of their critical (long-chain limit) properties should be similar: share the same functional form and leading asymptotic behavior. These properties include the mean squared end-to-end distance $\langle R^2 \rangle$, but also the mean square distance of a monomer from the endpoints and certain more exotic quantities (interpenetration ratios, gyration radii, etc—see Section 2 of Sokal 1994).

Unfortunately, it turns out that the self-avoiding walk is much more difficult to treat analytically than either of the random walks we have considered thus far. In fact, exact expressions aren't known for any of the quantities referred to in the previous paragraph. This is not for want of effort (see Madras and Slade, 1996, for a review of the mathematical literature on the topic). Consequently, numerical methods have been used to provide approximations. Foremost among these is Monte Carlo simulation:[15] an algorithm is used to generate a representative sample of all SAWs of a given length, and the quantity of interest is estimated using the sample average.

Monte Carlo Algorithms for the SAW The easiest way to generate a sample of self-avoiding walks is by *simple sampling*: generate a sample of unconstrained random walks (those of Section 2.2.1) by building each step by step, from the origin, and reject all those which intersect themselves on the way. This method is both conceptually clean and very easy to implement. The only difficulty with it is the attrition rate of the walks: the longer our simple random walk, the more likely it is that it intersects itself somewhere. In fact, it takes roughly $\mathrm{e}^{\lambda N}$ attempts to generate a self-avoiding walk of length N using this method, where $\lambda \approx 0.25$ in three dimensions.[16] We can lower λ for $d = 3$ to ≈ 0.07 by using non-reversing walks instead of simple ones, but that's essentially the limit of this method: it still takes about 1,000 attempts to generate a single walk of $N = 100$!

An appealing idea is to use *inversely restricted sampling*: after each step, check which steps would lead to an intersection on the next one; then, choose the next step only from among those which do not result in an intersection. This was the method used by Schaich (2006, Section 3.2.2 and Code Snippet C.3) in his analysis. Unfortunately, the inversely restricted

[15] Although series expansions and renormalization group methods are also used.

[16] This estimate is from Sokal (1994, p. 20); his review article is the primary source for this section. Both the expression and the λ value are only approximate because of the analytical intractability of the SAW.

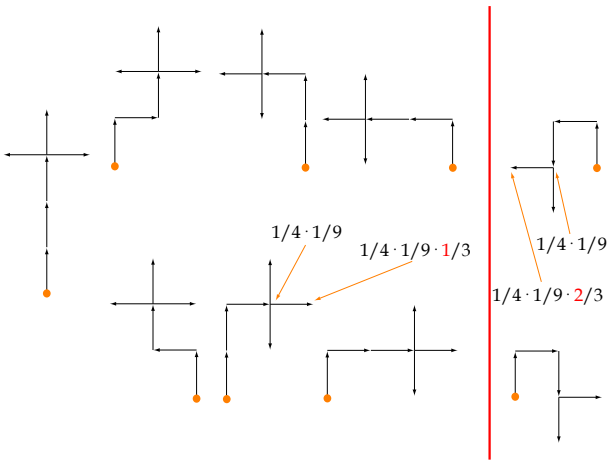


Figure 2.4: This figure depicts all of the possible two-dimensional SAWs of length 4 which begin with a step up. Notice that the Rosenbluth-Rosenbluth Algorithm produces the four walks to the right of the red line with higher probability $(2/54)$ than the walks to the left of the red line $(1/54)$. Adapted from Rosenbluth and Rosenbluth (1955).

sampling algorithm suffers from a complication that Schaich overlooked—it does not generate a uniform sample of all possible self-avoiding random walks. In particular, tightly wound walks are overrepresented in the sample, as explained in Figure 2.4. If one chooses to use this algorithm, the deviation from a uniform sample has to be addressed by weighting the walks when calculating quantities of interest. These weights are somewhat difficult to keep track of; worse yet, their sizes vary widely, so that the estimate ends up being based mostly on a few walks with large weights. The result is a rather large uncertainty. In the end, this method turns out to be less efficient than simple sampling (Sokal, 1994, p. 24).

A variety of more efficient algorithms for the simulation of SAWs exist. Dimerization algorithms concatenate two shorter SAWs to produce a long one; enrichment algorithms make multiple copies of every sufficiently long walk and continue on from there; and the many dynamic algorithms modify fragments of an existing walk to generate new ones. We have not implemented any of these more sophisticated algorithms, since our examination of random walks was only a preliminary exercise. Nonetheless, we encourage the interested reader to explore them further—the ingenuity of some of these methods makes them fascinating.

Table 2.4: Regression results for the self-avoiding random walk.

Dimension	A	ν
$d = 2$	$0.828(6)$	$0.7436(8)$
$d = 3$	$1.108(7)$	$0.5947(7)$

Simulation We studied the two- and three-dimensional self-avoiding walk, using simple sampling of non-reversing random walks. The mean end-to-end distance of a walk has the asymptotic form (Sokal, 1994):

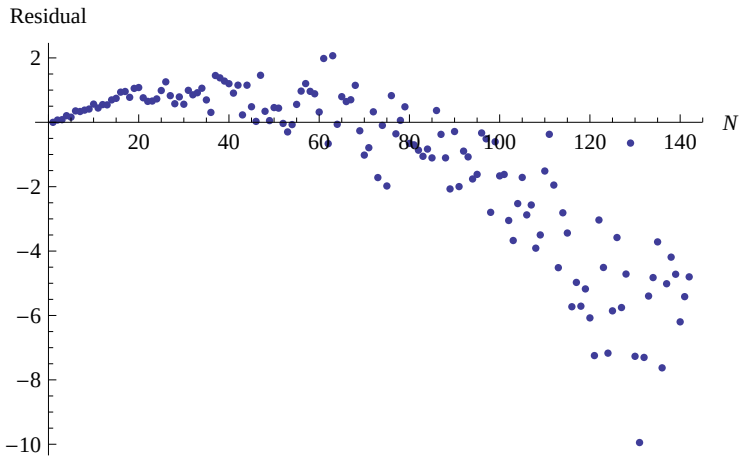
$$\langle R_N^2 \rangle = AN^{2\nu} \left(1 + O(N^{-\Delta})\right) \tag{2.29}$$

where ν and Δ are critical exponents. We estimated $\langle R_N^2 \rangle$ for walks of lengths up to 150 steps by averaging over 15,000 walks of each length. We then fit the results to Equation 2.29, neglecting the correction-to-scaling of $O(N^{-\Delta})$.[17] Unlike in earlier regressions, we had some qualms about whether the data points ought to be weighted by their estimated uncertainties. Equation 2.29 is only valid in the long-chain limit, so we would like to rely primarily on long walks in estimating its parameters; but it is precisely the long walks which have a large spread of $\langle R_N^2 \rangle$ values, and consequently a large uncertainty. When we eventually performed both procedures, our concerns turned out to be justified: the trend in the weighted regression residuals is evident (see Figure 2.5(a)). Consequently, the results reported in Table 2.4 are based on ordinary least squares. Based on the trend in the regression residuals, we expect our reported value in $d = 2$ to be an underestimate, and that in $d = 3$ an overestimate.

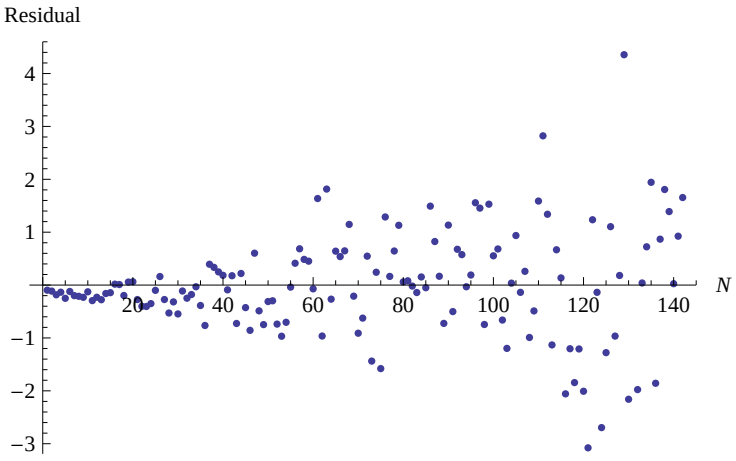
How do our results compare with literature values? The factor A is not universal, and as such is rarely reported (not in any of the references cited below). In two dimensions, it is believed that $\nu = 3/4$ (Madras and Slade, 1996, p. 19). In three dimensions, ν is variously estimated to be $\nu = 0.57(1)$ (de Forcrand et al., 1987), $\nu \approx 0.588$ (Sokal, 1994) or $\nu = 0.5877(6)$ (Li et al., 1995).[18] Our values are slightly biased in the expected directions, probably due to the inclusion of shorter walks for which correction-to-scaling factors cannot be neglected. Nevertheless, their agreement with literature values is much better than Schaich's, who reported $\nu = 0.6768(8)$ in two dimensions and $\nu = 0.5228(5)$ in three.

[17] The correction-to-scaling exponent Δ we ignore is estimated to be $\Delta = 0.56(3)$ (Li et al., 1995), and so significant only for short walks.

[18] Numerous other estimates exist—see Table II in Butera and Comi (1997) for seven more—but they are not significantly different from the aforementioned.



(a) Weighted



(b) Unweighted

Figure 2.5: Regression residuals for the three dimensional self-avoiding random walk: weighted (a) and unweighted (b). Ideally, residuals should spread out in a cone, like those in Figure 2.3. The trend in (a) is indicative of the omission of correction-to-scaling terms significant at low N. Note that this trend suggests that our estimate of $\langle R_N^2 \rangle$ (and hence ν) from the unweighted regression will be biased upwards: after all, the confounding low-N terms are still present. The plots for the two-dimensional walk are similar, but reflected about the x-axis, suggesting a downwards bias in ν.

2.3 Summary

Throughout this chapter, we have tried to tackle various problems using the static Monte Carlo method. At the heart of this method lies the concept of direct sampling: we were able to produce statistically independent outcomes (rice grains on a square, random walks on a lattice) in accordance with their true distribution. In other words, the probability of observing an outcome in the sample was equal to the probability of observing it in the population. Consequently, we were justified in using very simple statistical techniques—the sample mean served as an estimator of the population mean, and the variance of the sample as an estimator of the population variance. The success of this method is guaranteed by the weak law of large numbers: as the size of the sample increases, our estimators converge to the true value.

Direct sampling is "pure gold" (Krauth, 2006, p. 8), but as we saw in the case of the self-avoiding walk, it will sometimes break down. The failure of the simple and inversely restricted sampling techniques is worth pondering. What makes the self-avoiding walk problem so much more difficult, from the point of view of the Monte Carlo method, than the simple and non-reversing walks? The difficulty was our lack of knowledge of the true distribution of the SAWs. Instead of generating samples from that distribution, we were forced to sample a related one, and apply weights to calculate quantities of interest. In simple sampling, the weights were either 0 (if the walk intersected) or 1 (if it didn't); in inversely restricted sampling, they took on a more complicated form. In both cases, however, we were forced to discard as irrelevant an ever-increasing fraction of the information contained in the sample.

Unfortunately, many distributions of interest can't be sampled directly. This includes the Boltzmann distribution,

$$p(i) = \frac{e^{-\beta E_i}}{\sum_i^N e^{-\beta E_i}}. \tag{2.30}$$

If the number of states, N, is large, even just evaluating the denominator of Equation 2.30 becomes a serious problem. Trying to generate states at random and assigning appropriate weights to them leads to the same problem that thwarted the inversely restricted sampling algorithm: the vast majority of the weights will be close to zero, a few data points will dominate the sample, and the uncertainty of any estimate will be prohibitively large.

It turns out that there are clever techniques for overcoming these problems. They will be the topic of the next chapter.

Chapter 3

Dynamic Monte Carlo and the Ising Model

In this chapter, we discuss dynamic Monte Carlo methods and their application to the Ising model. While superficially unrelated to our ultimate goal of studying quantum field theory, these techniques will be integral to simulating the ϕ^4 model. In fact, it will turn out that the ϕ^4 theory is closely related to the Ising model. We consider the Ising first, because it has a more intuitive interpretation and has been well studied in the literature, allowing us to verify the performance of our programs.

While all of the simulations described in this chapter are of course our own, the theory is not. Virtually every factual assertion below is based either on the sources cited or on the additional references discussed in Appendix C.

3.1 The Ising Model

The Ising model is traditionally interpreted as a simple model of a ferromagnet. With each point on a rectangular lattice we associate a spin, s_i, which points "up" ($s_i = 1$) or "down" ($s_i = -1$). The energy of the system is given by the Hamiltonian expression

$$E = -J \sum_{\langle i\, j \rangle} s_i s_j - B \sum_i s_i, \tag{3.1}$$

where B is the external magnetic field, J is a constant (known as the coupling constant) and the notation $\sum_{\langle i\, j \rangle}$ is meant to indicate a sum over all

pairs of neighboring spins. For simplicity, we will only consider the case $J = 1$, $B = 0$, which already exhibits the phenomena we are interested in studying.

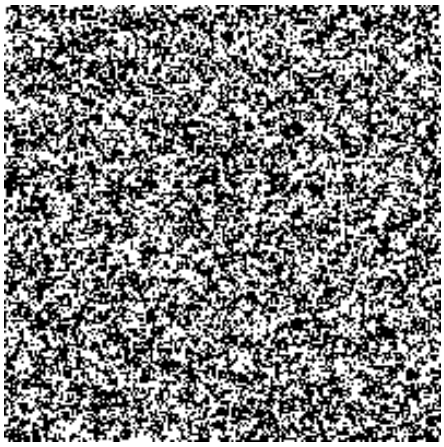
What happens at the boundary of the lattice? We could assume that boundary spins have fewer neighbors than mid-lattice ones, but that introduces an asymmetry we would rather avoid. Real magnets are assumed to be made up of a very large number of atoms carrying a magnetic moment, only a tiny fraction of which will inhabit the boundary; effects in the interior are much more important. To deal with this difficulty, we wrap the lattice around: if it's one dimensional, we make it into a loop (necklace), if it's two dimensional, we place it on a torus. Spins at the top of a lattice become neighbors of those at the bottom, spins on the left are aligned with those on the right, etc.[1]

The Ising model owes its enduring popularity to the fact that despite its simple structure it exhibits a phase transition: its behavior undergoes a qualitative change at a well-defined critical temperature, T_c. This transition is analogous to the one that takes place in ferromagnetic materials at the Curie temperature. At high temperatures, the model is in the symmetric phase, in which the spins are randomly aligned. (It's called symmetric because the large-scale properties of the system are unchanged under spin reversal—compare Figures 3.1(a) and 3.1(b).) At low temperatures, spontaneous magnetization takes place: the symmetry is broken and essentially all of the model's spins are parallel. At the critical temperature ($T_c = 2J/\ln(1+\sqrt{2}) \approx 2.269J$ in two dimensions), a switch from one regime to the other takes place. This change is gradual, both in the model and in real ferromagnets, so the transition is second-order.[2] The magnitude of the magnetization captures the degree to which the spins are aligned; it is zero on one side of the phase transition and nonzero on the other. By virtue of this property it is called the order parameter of the transition.

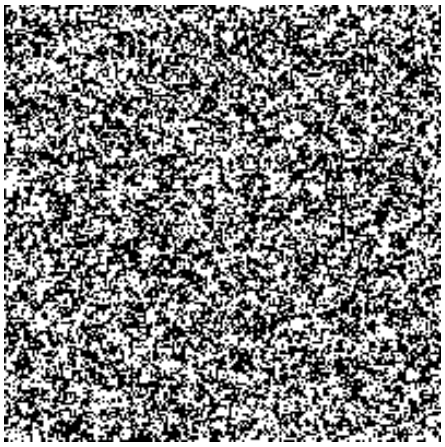
The Ising model is a classical system which can be treated using the tools of statistical mechanics. Its observable properties are Boltzmann-

[1] This procedure can be *performed* in any number of dimensions, of course; it just cannot be visualized in dimensions higher than two. Note that these so-called "periodic boundary conditions" aren't the only way to wrap up a lattice. Another popular choice are helical boundary conditions, which don't have as neat a geometrical interpretation, but offer some programming advantages. In this work we will use periodic boundary conditions exclusively, but see Chapter 13 of Newman and Barkema (1999) for an overview of other options.

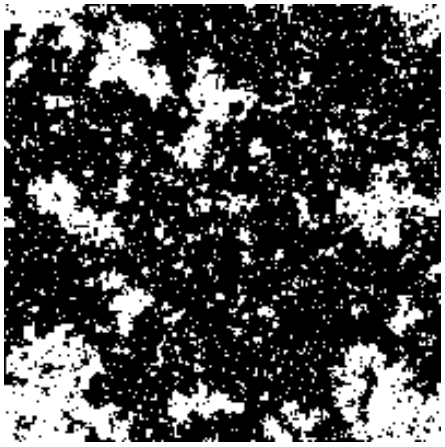
[2] In a first-order phase transition, such as that between liquids and gases, the energy suffers a discontinuity. Consequently, first-order transitions involve latent heat.



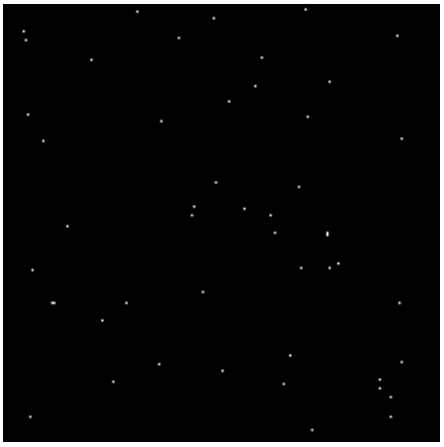
(a) $T = 4.8 > T_c$



(b) Inverted lattice at $T = 4.8 > T_c$



(c) $T = 2.269 \approx T_c$



(d) $T = 1.2 < T_c$

Figure 3.1: A 300×300 Ising model in the high, critical, and low temperature regimes. Black corresponds to spin up and white to spin down, except in Figure (b) where the lattice has been inverted.

weighted averages of values corresponding to different configurations:

$$\langle Q \rangle = \frac{1}{Z} \sum_{\mu} Q_{\mu} \mathrm{e}^{-\beta E_{\mu}}, \tag{3.2}$$

where the sum is over the possible configurations of the system and $\beta \equiv 1/kT$ (k is the Boltzmann constant). In the case of the Ising model, the configuration is just the arrangement of the spins. The observable could be magnetization per spin, say, or internal energy. The partition function is defined the usual way,

$$Z = \sum_{\mu} \mathrm{e}^{-\beta E_{\mu}}. \tag{3.3}$$

In principle, many quantities can be derived directly from the partition function, as well as using Equation 3.2; for instance, the internal energy,

$$U = \langle E \rangle = \frac{1}{Z} \sum_{\mu} E_{\mu} \mathrm{e}^{-\beta E_{\mu}} = \frac{1}{Z} \frac{\partial Z}{\partial \beta} = -\frac{\partial \log Z}{\partial \beta}. \tag{3.4}$$

The partition function, therefore, contains a wealth of information about the model, and is of interest in and of itself. If we could somehow obtain a workable expression for it, we would have effectively solved the Ising model.

Can we obtain such an expression? Only in some cases and with great ingenuity. We will limit ourselves to a historical sketch and refer the reader to other sources for the mathematical arguments, as they are too complicated to be presented here.[3] An exception could be made for the one-dimensional model solved by Ising himself, in his 1925 doctoral dissertation. Unfortunately, the $d = 1$ case is not interesting—it does not exhibit the phase transition. This was the disappointing conclusion of Ising's dissertation which led him to abandon the model. However, in 1936, Rudolf Peierls proved that in the two-dimensional model a phase transition was guaranteed to exist at some temperature; five years later, Hendrick Kramers and Gregory Wannier showed what this temperature must be, assuming that it is unique. Finally, in 1944, Lars Onsager found an analytic expression for the free energy per lattice site, $F = \lim_{N \to \infty} Z/N$, in two dimensions, thereby solving the model in this regime. His solution leaves something to be desired: it's infamously difficult to follow, and even the most elegant

[3] An introduction to the Ising model as a mathematical problem, including the proofs of Peierls and Kramers-Wannier discussed below, can be found in Cipra (1987).

of its later simplifications "would take at least a chapter of rather technical and unilluminating manipulations to duplicate" (Sethna, 2006, p. 166). The three dimensional Ising model presents an even grimmer picture: it remains unsolved to this day, as do more complicated two-dimensional variations (the $B \neq 0$ case, and all models assuming interactions between spins which aren't nearest neighbors). Some variants of the three-dimensional and long-range interaction models have been shown to be NP-complete (Istrail, 2000), suggesting that no simple analytical solution to these problems exists.

The situation facing us, then, is similar to the case of the random walk: the very basic variants can be treated analytically, but the difficulty of the problem increases rapidly as we consider seemingly slight complications.[4] We can again imagine two numerical approaches to the problem: an enumeration of all the possible states of the system (analogous to enumerating all random walks up to a certain length) or sampling the Boltzmann probability distribution using a Monte Carlo technique. Enumeration, however, is almost as hopeless as the analytical approach: even a 16×16 system, rather small if we're interested in probing the thermodynamic limit, has $2^{256} \approx 10^{77}$ states! Monte Carlo simulation, therefore, is the only way to go.[5]

3.2 Markov Chain Monte Carlo

In general, after a Monte Carlo simulation of a statistical system, we calculate quantities of interest from the formula

$$\langle Q \rangle = \frac{\sum_i^N Q_i \, p_i^{-1} \mathrm{e}^{-\beta E_i}}{\sum_i^N p_i^{-1} \mathrm{e}^{-\beta E_i}}, \tag{3.5}$$

where p_i is the probability of a state being generated in the simulation and N is the size of the sample. In simple sampling, we let $p_i = 1$ for all i. But statistical systems in equilibrium spend most of their time in a small subset of all available states (this is especially obvious at low temperatures, where

[4] The similarity of the two models is not entirely coincidental, as they turn out to be closely related: they are the $n = 0$ (SAW) and $n = 1$ (Ising) cases of the so-called classical n-vector spin model (according to Gaspari and Rudnick 1986, this relationship was first noticed in de Gennes 1972).

[5] There is actually another large family of approximate approaches to the Ising model: perturbative series expansions of the partition function, such as high-temperature expansions or expansions in $4 - \epsilon$ dimensions. We will not be concerned with them in this work.

just *one* state, the ground state, becomes dominant). As the reader may recall, we concluded the previous chapter with the unhappy observation that simple sampling Monte Carlo techniques break down precisely in this case, namely when the probability distribution to be evaluated is dominated by a few large terms that will be poorly represented in the sample. How then can we use a Monte Carlo method in this case? What we would like to do is make our simulation mimic the dynamics of a system in equilibrium and sample the *important* states, so that $p_i = \exp(-\beta E_i)/Z$ and

$$\langle Q \rangle = \frac{1}{N} \sum_{i}^{N} Q_i. \tag{3.6}$$

Very appropriately, this approach is known as importance sampling. To carry it out, we will introduce a new technique—Markov chain sampling.

A Markov process is a mechanism for producing a new state ν from an old one, μ, with a probability $P(\mu \to \nu)$ depending only on the states ν and μ.[6] Mathematically, it is specified by a transition matrix containing the probabilities of getting from any state to another:

$$\mathsf{T} = \begin{pmatrix} P(1 \to 1) & P(1 \to 2) & \cdots & P(1 \to n) \\ P(2 \to 1) & P(2 \to 2) & \cdots & P(2 \to n) \\ \vdots & \vdots & \ddots & \vdots \\ P(n \to 1) & P(n \to 2) & \cdots & P(n \to n) \end{pmatrix}. \tag{3.7}$$

Since the entries, $P(\mu \to \nu)$, are probabilities, each row sums to unity. A chain of states is generated by repeatedly applying this matrix to a vector of probabilities, $\boldsymbol{x}$. If a Markov process is to generate a chain of states appearing with Boltzmann probabilities, it must satisfy two conditions.

Firstly, it must be *ergodic*: it must have a nonzero probability of generating each possible state of the system, since the Boltzmann probabilities for each state are strictly positive. This doesn't mean that all entries of T must be nonzero, but there must be a way to transition from any state μ to any other, possibly in multiple steps.

Secondly, the Markov process must satisfy the condition of *detailed balance*. This condition is intended to eliminate limit cycles and guarantee that the process has an equilibrium (stationary) probability distribution, i.e. a probability vector $\boldsymbol{x}(t)$ that obeys

$$\boldsymbol{x}(\infty) = \mathsf{T} \cdot \boldsymbol{x}(\infty). \tag{3.8}$$

[6] A more formal introduction to Markov chains, including rigorous proofs of the results used here, can be found in Morningstar (2007).

In contrast, a limit cycle corresponds to

$$\boldsymbol{x}(\infty) = \mathsf{T}^n \cdot \boldsymbol{x}(\infty) \tag{3.9}$$

for some $n \neq 1$. The condition of detailed balance is that

$$p_\mu P(\mu \to \nu) = p_\nu P(\nu \to \mu) \quad \forall \mu, \nu. \tag{3.10}$$

This condition states that, on average, the we are equally likely to observe a transition from μ to ν as its reverse. This cannot be true of all states in a limit cycle, where the expected occupancy of states changes with time. Summing both sides of Equation 3.10 over ν, we get

$$\sum_\nu p_\mu P(\mu \to \nu) = \sum_\nu p_\nu P(\nu \to \mu). \tag{3.11}$$

This equation states that the rates at which the system exits (LHS) and enters (RHS) the state μ are equal, so the system is at equilibrium. Since the rows of a transition matrix sum to unity, we can rewrite this equation as

$$p_\mu = \sum_\nu p_\nu P(\nu \to \mu). \tag{3.12}$$

The probability distribution p_μ for which this equation holds will be the equilibrium probability distribution of the Markov process.

As long as these two conditions are satisfied, our Markov process will have the desired equilibrium distribution, and will approach it exponentially with the number of iterations.[7] Just how quickly it converges to this distribution is a valid question to which we will return when discussing specific algorithms.

3.2.1 The Metropolis Algorithm

By far the most famous algorithm for the Ising model, which we have used in the majority of our simulations of this system, is the Metropolis algorithm. It is a single spin flip algorithm: a new state is created from an old one through the flipping of only one spin on the lattice. To derive it, we proceed directly from the detailed balance condition, Equation 3.10, rewritten thus:

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{p_\nu}{p_\mu} = \mathrm{e}^{-\beta(E_\nu - E_\mu)}, \tag{3.13}$$

[7] Proving exponential convergence would take us too far afield, but the interested reader is referred to Newman and Barkema (1999), section 3.3.2, or Krauth (2006), section 1.1.4. Their treatment was the inspiration for much of ours.

where the last equality follows from the fact that we want to generate states with their Boltzmann probabilities. We now break down the transition probability $P(\cdot)$ into the probability of a move being selected, $g(\cdot)$, and being accepted, $A(\cdot)$:

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{g(\mu \to \nu)\, A(\mu \to \nu)}{g(\nu \to \mu)\, A(\nu \to \mu)}. \tag{3.14}$$

We will select the spin to flip at random (with uniform probability), so that the probability of a move being selected is the same for both the forward and the backward move. We are left with the condition

$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \mathrm{e}^{-\beta(E_\nu - E_\mu)}. \tag{3.15}$$

To make the algorithm efficient, we would like to accept the transition with as high a probability as possible—otherwise, we would be wasting moves, selecting spins only to discard them. Therefore, we set,

$$A(\mu \to \nu) = \begin{cases} 1 & \text{if } E_\nu \leq E_\mu, \\ \mathrm{e}^{-\beta(E_\nu - E_\mu)} & \text{if } E_\nu > E_\mu. \end{cases} \tag{3.16}$$

This choice of the acceptance probability is what defines the Metropolis algorithm. Since it's a single spin flip algorithm, and the probability of flipping a spin is always strictly positive, the Metropolis satisfies ergodicity: any configuration can be generated with nonzero probability through a sequence of spin flips. We derived the acceptance probability from the detailed balance condition, which therefore must be satisfied as well. And the ratio of the transition probabilities was set to the ratio of Boltzmann factors, guaranteeing that the stationary distribution is the desired one.

Evaluating exponentials on a computer takes a relatively long time, so an important practical consideration is that $\Delta E = E_\nu - E_\mu$ has only $2d + 1$ possible values, which need be evaluated only once, at the beginning of the simulation. If all of the selected spin's neighbors are aligned opposite to it, $\Delta E = -4Jd$ (since there are $2d$ neighbors, and for each of them the interaction energy, $-J \sum_{\langle i\, j \rangle} s_i s_j$, would fall by $2J$); for every spin that's aligned in the same direction, a factor of $2J$ should be added to ΔE.

3.2.2 Equilibration and Autocorrelation

The Metropolis algorithm may have the desired stationary probability distribution, but two questions immediately arise. Firstly, what state should

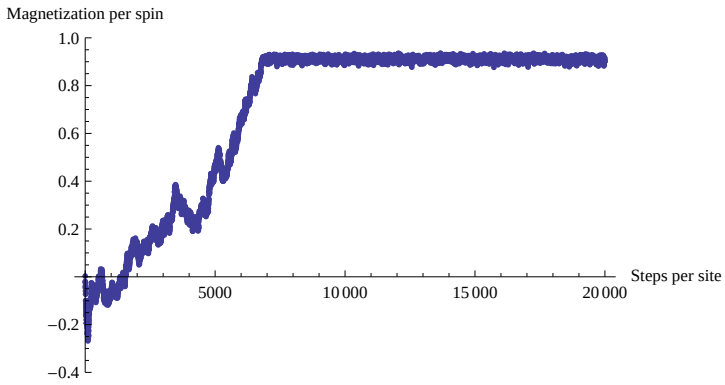


Figure 3.2: The magnetization of a 100×100 Ising lattice evolves in the course of a Monte Carlo simulation conducted at $T = 2.0 < T_c$. The simulation equilibrates after about 7,000 Metropolis steps per site.

we start with, and how long will the algorithm take to equilibrate (reach the stationary distribution)? Secondly, how many iterations do we have to wait to get two truly independent samples? The great advantage of the methods of Chapter 2 was that each sample we generated was independent of the others—now, two subsequent samples generated by the Metropolis algorithm differ by only one spin flip, and can hardly be considered such.

Concerning the first question, it doesn't matter very much what state we start with, though starting with a state typical of the temperature we would like to conduct our simulation at will result in faster equilibration. One trick is to start a new simulation with the final state of an old one, conducted at a similar temperature. We will know that a simulation has equilibrated once the observables (magnetization, energy, etc) start varying about a set value with a relatively small amplitude (see Figure 3.2). For our simulations on a 100×100 Ising lattice, equilibration never took more than 10,000 steps per site. Rarely, at low temperatures, the model may find itself in a metastable state, an example of which is shown in Figure 3.3. The energy of this state may be low, but other observables will generally deviate far from their stable-state values. Such a state will decay eventually, though in practice it may be necessary to rerun the simulation. Fortunately, the distinctive appearance of the lattice makes it easy to sift out the simulations which became trapped in such a state.

Once the simulation has equilibrated, how long do we have to wait to get two independent samples? One way to answer this question is to use

Figure 3.3: A 100×100 Ising model in a metastable state.

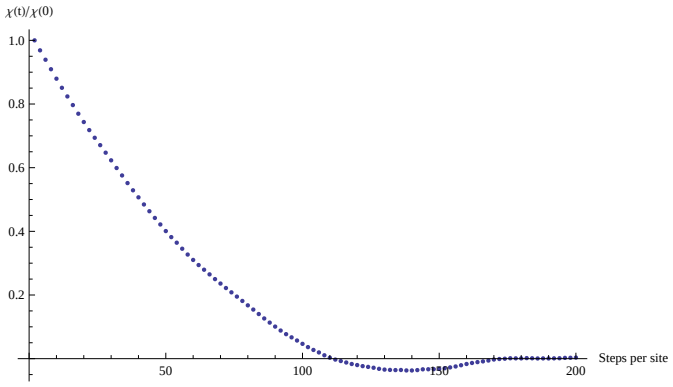


Figure 3.4: The magnetization autocorrelation function of a 100×100 Ising lattice simulated using a Metropolis algorithm at $T = 2.4 > T_c$. The autocorrelation time is about 50 steps per site.

the autocorrelation function of the magnetization,[8] defined as

$$\chi_{\mathrm{ac}}(t) = \int [m(t') - \langle m \rangle][m(t'+t) - \langle m \rangle]\, dt'. \tag{3.17}$$

$\chi_{\mathrm{ac}}(t)$ measures the degree to which magnetization at times t and t' is correlated; at short times, we expect the correlation to be significant, since we only flip spins one at a time, but at long times it should average out to zero. It turns out that for a Markov chain this function is characterized by an exponential decay, the characteristic time scale of which is given by the correlation time τ:

$$\chi_{\mathrm{ac}}(t) \propto \mathrm{e}^{-t/\tau}. \tag{3.18}$$

A plot of the autocorrelation function from one of our simulations is shown in Figure 3.4—the exponential decay is evident. Various techniques could be employed to derive an estimate of the correlation time from the data used to produce this figure, but for our purposes a rough gauge will be sufficient. Following Newman and Barkema (1999, p. 61), we will assume that the number of independent samples obtained about is $n \approx t_{\max}/2\tau$. It's important that this number be large enough for the intended statistical procedures, and that no fewer than n measurements be taken in the course of the simulation.

3.2.3 Data Analysis

Best Values We can generate estimates of a quantity of interest using Equation 3.6: averaging values recorded in the course of the simulation. This prescription is straightforwardly applied to the magnetization or energy, but how to estimate quantities not measured directly in the simulation? Two which are of interest are the magnetic susceptibility per spin and the specific heat per spin,

$$\chi = \frac{\partial \langle m \rangle}{\partial B} \quad \text{and} \quad c = \frac{1}{N} \frac{\partial \langle E \rangle}{\partial T}, \tag{3.19}$$

respectively, where N is the size of the lattice and m its magnetization per spin. Using a statistical mechanics trick,[9] it's possible to re-express these quantities as

$$\chi = \beta N \left(\langle m^2 \rangle - \langle m \rangle^2 \right) \quad \text{and} \quad c = \frac{k\beta^2}{N} \left(\langle E^2 \rangle - \langle E \rangle^2 \right). \tag{3.20}$$

[8] Analogous functions can be defined for other observables.

[9] The trick is to introduce appropriately coupled terms to the Hamiltonian and then set them to zero: see Newman and Barkema (1999), section 1.2.1.

The expectations in the formulas above are easily computed from Equation 3.6.

Bootstrap In calculating the best values, we needn't be concerned whether our samples are independent or not: we just average all of the data points. However, questions of independence become important if we want to use the classical statistical approach of Chapter 2 to estimate the uncertainty of our measurements. The variance of an average of n independent, identically distributed measurements is

$$\mathrm{Var}(\bar{\eta}) = \frac{\mathrm{Var}(\eta)}{n} = \frac{1}{n}\left(\langle\eta^2\rangle - \langle\eta\rangle^2\right). \tag{3.21}$$

It turns out that the appropriate generalization to our case is to set $n = t_{\max}/2\tau$, the ratio of the simulation length to the autocorrelation time. This classical approach, however, suffers from two limitations. Firstly, it relies on the generally imprecise estimate of the autocorrelation time. Secondly, and more importantly, it cannot be easily applied to quantities such as the specific heat and magnetization. The problem is that our estimates of pairs of quantities such as $\langle E\rangle$ and $\langle E^2\rangle$ are derived from the same set of measured energy values, and are therefore correlated; we would be underestimating the uncertainty if we used the usual uncertainty-propagation rules to calculate the variance of their difference. There are elaborate techniques designed to overcome these difficulties, but it's easier to abandon the classical approach altogether and use so-called nonparametric statistics. The procedure relevant to our problem is the bootstrap.

The idea behind the bootstrap is the following: say we computed some statistic $\hat{\rho}$ based on a data set of n points drawn from a distribution, F. The standard error of our statistic is a function of the distribution F:

$$\sigma(F) = \sqrt{\mathrm{Var}_F\left(\hat{\rho}(X_1, X_2, \ldots, X_n)\right)}, \tag{3.22}$$

where the X_i's are n random variables drawn from the distribution. We don't know F, but we can approximate it by the sample distribution, $\hat{F}$, and so approximate $\sigma(F)$ using $\sigma(\hat{F})$. Unfortunately, we generally don't know the expression for $\mathrm{Var}_F(\hat{\rho})$—except for the simplest distributions, it doesn't even exist in closed form. The solution is to estimate it using the following procedure:

1. Draw a bootstrap sample from $\hat{F}$: pick n data points at random, with replacement, from the data set.

2. Calculate the bootstrap replication, $\hat{\rho}_j$: compute the value of $\hat{\rho}$ from the bootstrap sample.

3. Repeat the previous two steps a large number B times—"1000 would not be excessive" (Newman and Barkema, 1999, p. 74)—obtaining bootstrap replications $\hat{\rho}_j$ for $j = 1, 2, 3, \ldots, B$.

The estimate of the statistic's error is then given by the standard deviation of the bootstrap replications,

$$\hat{\sigma}(F) = \sqrt{\frac{1}{B-1}\sum_{j=1}^{B}\left(\hat{\rho}_j - \frac{1}{B}\sum_{j=1}^{B}\hat{\rho}_j\right)}. \tag{3.23}$$

From our perspective, this technique offers two significant advantages over a more traditional treatment. Firstly, we don't need to modify it in any way on account of our data points being correlated. Secondly, we don't have to worry about the propagation of uncertainty: we let our statistic $\hat{\rho}$ be whatever final quantity we want to compute, such as χ or c. It's only disadvantage is that we need to record all of the values of energy (magnetization) produced in the course of the simulation, but this turns out not be a practical problem.

Bootstrap has become rather popular since the seminal article by Efron (1979), and there are many references available for those seeking a fuller explanation of the method. Our treatment is based on Efron and Gong (1983); a brief introduction in the context of Monte Carlo simulation of statistical systems can be found in Newman and Barkema (1999, section 3.4.3). More references and a longer discussion of the method are in Sprent and Smeeton (2007, section 14.3). A very friendly, applications-oriented text is Efron and Tibshirani (1993).

3.3 Ising Metropolis: Simulation Results

We used an implementation of the Metropolis algorithm (see Appendix B.2.4 for the code) to perform a simulation of the 100×100 Ising model at twenty-five equally spaced temperatures, $kT = 0.2, 0.4, \ldots, 5.0$. Our algorithm carried out 20,000 Metropolis steps per spin ("sweeps") at each temperature, measuring magnetization per spin and energy per spin every other sweep. We used the simulation results to estimate a number of quantities, shown in Figures 3.5–3.7 and discussed below. All error bars are 1σ bands derived from 1,000 bootstrapped samples.

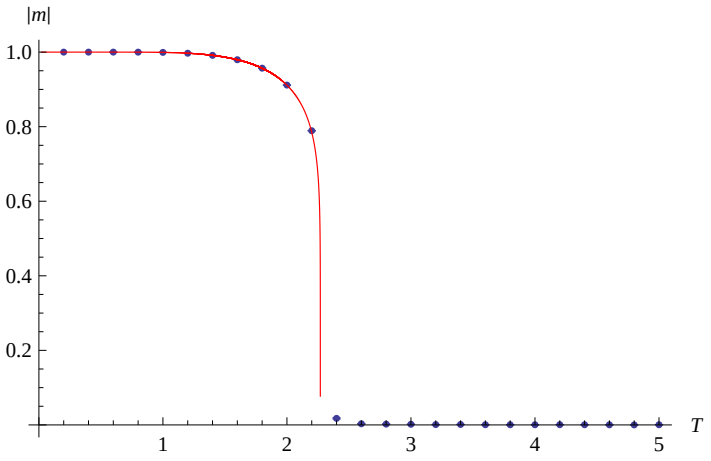


Figure 3.5: The magnetization per spin of a 100^2 Ising model as a function of temperature. The points are our measurements (error bars essentially invisible) and the solid line is the exact solution for a system in the thermodynamic limit.

Magnetization As expected, the absolute value of the magnetization—shown in Figure 3.5—is close to unity at low temperatures and close to zero at high ones, with a rather sharp transition around the critical temperature. It's worthy of note that the curve is not symmetric about the critical point: the magnetization is close to (though not quite) zero at $T > T_c$, but approaches unity rather slowly for $T < T_c$. The approach to unity to the left of the critical point is slow because the Ising model's phase transition is a continuous one. This mirrors the nature of the phase transition in ferromagnetic materials, which gradually lose their spontaneous magnetization as they are warmed up to the Curie temperature. However, the spontaneous magnetization of a ferromagnetic material is exactly zero at $T > T_c$, yet the magnetization in Figure 3.5 is noticeably above zero at $T = 2.4 > T_c$. This is a so-called finite size effect: it is only observed because we're simulating the model on a finite lattice. In Onsager's exact solution, assuming $J = k = 1$, the magnetization is given by

$$|m| = \left(1 - \sinh^{-4}\left(\frac{2}{T}\right)\right)^{1/8} \tag{3.24}$$

and falls to zero at the critical temperature (Pathria, 1996, section 12.3). Since we're usually interested in the thermodynamic limit of an infinite lattice, finite size effects amount to systematic error in Monte Carlo simu-

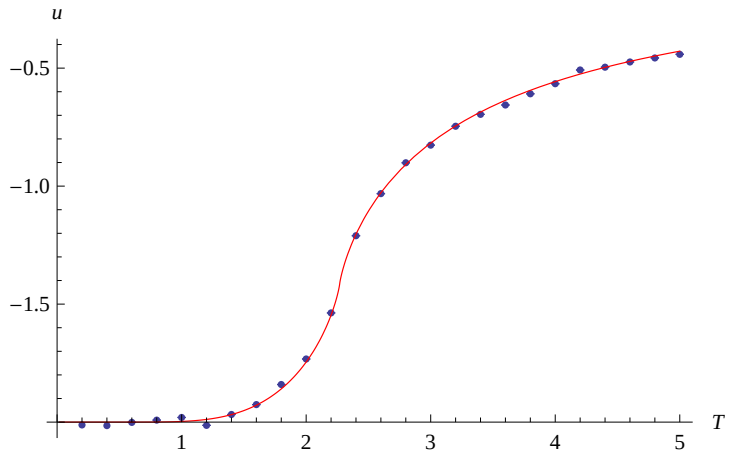


Figure 3.6: The energy per spin of a 100^2 Ising model as a function of temperature. The points are our measurements and the solid line is the exact solution for a system in the thermodynamic limit.

lations.

Energy The plot of internal energy per spin (Figure 3.6) shows predictable behavior: at low temperatures, all of the spins are aligned, and each contributes $-2J$ to the energy. As the temperature increases, so does the energy, finally approaching zero in the high-temperature limit. There's no discontinuity in the energy at the critical temperature, although there is an inflection point, suggesting that the first derivative (the specific heat) may diverge. The agreement with the exact solution is excellent, even better than in the case of the magnetization, as no finite-size effects are recognizable. For completeness, we give the functional form of the exact solution (Plischke and Bergersen, 1994, section 5.1.4):

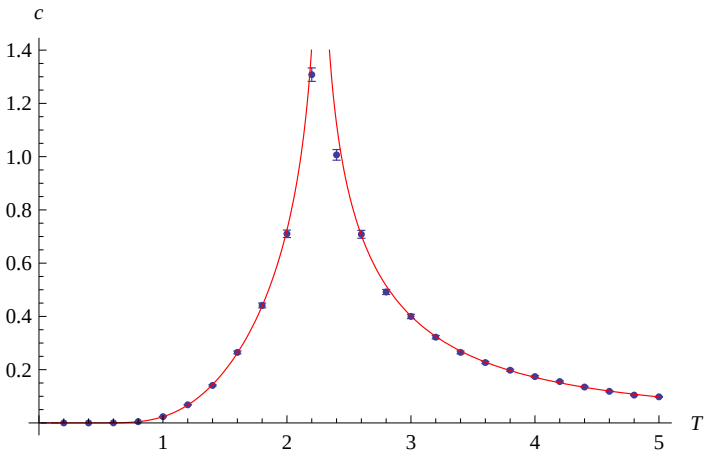
$$u(T) = -\coth\left(\frac{2}{T}\right)\left(1 + \frac{2}{\pi}(2\tanh^2(2/T) - 1)K(q^2)\right), \tag{3.25}$$

where $K(m)$ is the elliptic integral of the first kind,

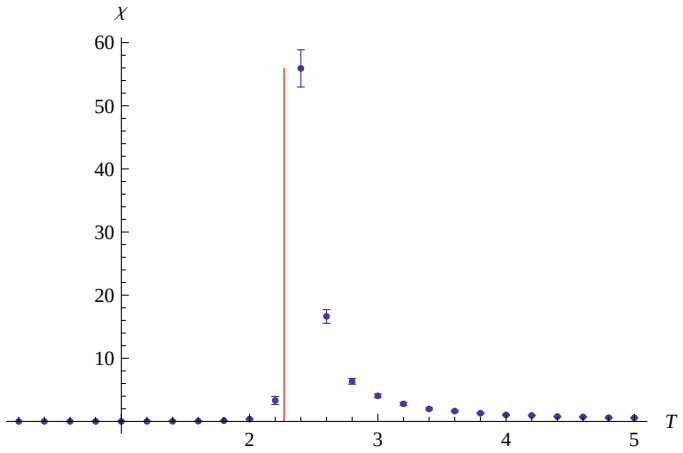
$$K(m) = \int_0^{\pi/2} \frac{d\phi}{\sqrt{1 - m\sin^2\phi}}, \tag{3.26}$$

and q is defined as

$$q(T) = \frac{2\sinh(2/T)}{\cosh^2(2/T)}. \tag{3.27}$$



(a) Specific heat



(b) Susceptibility

Figure 3.7: The specific heat and susceptibility of a 100^2 Ising model as a function of temperature. The solid line in (a) is the exact solution in the thermodynamic limit; the solid line in (b) marks the critical temperature.

Specific heat and susceptibility The specific heat and magnetic susceptibility (Figure 3.7) diverge at the critical temperature. Like the behavior of the magnetization, this is unsurprising from a physical perspective: as the temperature approaches T_c, the size of the spin clusters making up the system increases. The flipping of these clusters results in large fluctua-

tions in the energy and magnetization of the system. Since the specific heat and magnetic susceptibility are the standard deviation of the energy and the magnetization, respectively, they capture this phenomenon. The divergence in the susceptibility is sharper than that in the specific heat; this behavior is a feature of the exact solution to the model as well. Much like the asymptotic behavior of the SAW as $N \to \infty$ is described by Equation 2.29,

$$\langle R_N^2 \rangle = AN^{2\nu} \left(1 + O(N^{-\Delta})\right),$$

the asymptotic behavior of the specific heat and susceptibility of the Ising model are described by

$$\chi \propto |t|^{-\gamma} \tag{3.28}$$

$$c \propto |t|^{-\alpha}, \tag{3.29}$$

where $t = (T - T_c)/T_c$ while γ and α, like ν, are critical exponents characteristic of an entire universality class.[10] From the exact solution, we know that $\gamma = 7/4$ and $\alpha = 0$ (Pathria, 1996, p. 388), *i.e.* the susceptibility diverges exponentially and the specific heat—logarithmically. In fact, the full expression for the specific heat, used to generate Figure 3.7(a), is (again Plischke and Bergersen, 1994, section 5.1.4):

$$c(T) = \frac{4}{\pi}\left(\frac{1}{T}\coth\frac{1}{T}\right)^2 \left\{K\left(q^2\right) - E_1\left(q^2\right) - \left(1 - \tanh^2\frac{2}{T}\right)\times \right.$$
$$\left. \times \left[\frac{\pi}{2} - \left(1 - 2\tanh^2\frac{2}{T}\right)K\left(q^2\right)\right]\right\} \tag{3.30}$$

where q and $K(m)$ are defined as in the internal energy expression, Equation 3.25, and $E_1(m)$ is the complete elliptic integral of the second kind,

$$E_1(m) = \int_0^{\pi/2} \sqrt{1 - m\sin^2\phi}\, d\phi. \tag{3.31}$$

A closed form expression for the magnetic susceptibility is apparently not known.

The error bars on our estimates are generally rather small. Notice, however, that the error bars on the susceptibility and specific heat plots increase

[10] Models with different values of J, and even models on lattices with different topologies (triangular or hexagonal, rather than rectangular) will share the same values of these critical exponents.

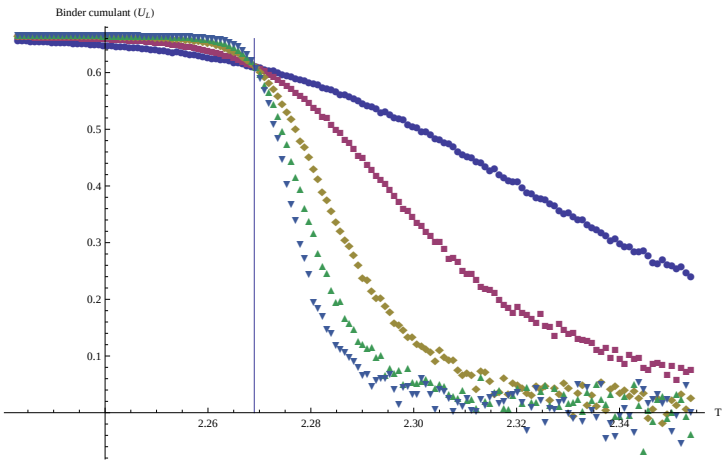


Figure 3.8: The BCL cumulant, U_L, of the Ising model as a function of temperature for a variety of lattice sizes. The solid line marks the critical temperature. The lattices are of linear dimension $L = 16, 32, 64, 128, 256$, with steeper curves corresponding to larger lattice sizes.

as the critical temperature is approached. This is due partially to the critical fluctuations described in the previous section, and therefore a feature of the model itself, but partially to an unfortunate phenomenon known as critical slowing-down that afflicts the Metropolis algorithm. It turns out that as we approach the critical point, the correlation time τ (see Equation 3.18) tends to diverge, following a power law similar to Equation 3.28. It goes as

$$\tau \propto \xi^z, \tag{3.32}$$

where ξ is the correlation length, the size of a typical cluster of correlated spins. On a finite lattice, the correlation length can never exceed the lattice linear dimension L, so we have $\tau \propto L^z$ as $T \to T_c$. For the Metropolis algorithm, $z = 2.1665 \pm 0.0012$ (Newman and Barkema, 1999, p. 91). As the correlation time increases, we get fewer and fewer independent measurements, and our confidence in the estimates suffers. Critical slowing-down is a major obstacle to the study of the critical properties of the Ising model; similar problems arise in other models, including the ϕ^4 theory. We will devote the next section to algorithms designed to overcome this difficulty.

The BCL Cumulant This fourth-order cumulant, due to Challa et al. (1986) and sometimes simply known as the Binder cumulant, is defined as

$$U_L = 1 - \frac{\langle m^4 \rangle}{3\langle m^2 \rangle^2}. \tag{3.33}$$

While not physically meaningful, this quantity has two interesting features. Firstly, it can be used to distinguish between a first- and second-order phase transition (Bhanot and Sanielevici, 1989). This may be a somewhat puzzling advantage: by definition, the internal energy is discontinuous at a first-order phase transition but continuous at a second-order one. Why do we need another indicator? The problem is that the discontinuity in the energy is only observed in the thermodynamic limit of an infinite lattice, $L \to \infty$. On finite lattices of our numerical simulations, all phase transitions will appear to be second-order, at least judging from the behavior of their internal energy. It turns out, however, that the BCL cumulant behaves differently near first- and second-order transitions, even on finite lattices. If the transition is second-order, the cumulant smoothly varies from its asymptotic value of $\approx 2/3$ at $T \ll T_c$ to ≈ 0 at $T \gg T_c$. If the transition is first-order, however, the cumulant has a minimum at the critical temperature which becomes more pronounced as the size of the lattice is increased.[11] As we mentioned in Section 3.1, the Ising model has a second-order phase transition. We used a mix of the Metropolis and Wolff algorithms (discussed in Section 3.4) to evaluate the cumulant at 600 temperatures near the critical point—the results, shown in Figure 3.8, are as expected of a second-order transition. An example of a BCL cumulant plot for a first-order transition can be found in Challa et al. (1986), Figure 16.

The second interesting feature of the cumulant is observed only in certain second-order transitions. The U_L curves for different lattice sizes intersect at a unique point, which corresponds to the critical temperature of the infinite system. In the Ising model, finite-size scaling arguments can be used to show that this phenomenon must take place—the interested reader is referred to Binder and Heermann (2002, p. 46f.). Similar behavior is observed in a variety of related models, and has been exploited in some studies of their phase transitions. Our results in Figure 3.8 exhibit it as well; however, as we will see in the next chapter, the curves don't quite intersect at one point, resulting in a slight systematic effect if we use them for the estimation of T_c.

[11] Observing how the properties of a simulated model vary with the lattice size is known as finite size scaling. It is an important technique in Monte Carlo studies of statistical systems; we will discuss it more broadly in Section 3.5.

3.4 Wolff Cluster Algorithm for the Ising Model

As we have seen, the Metropolis algorithm performs rather poorly in the vicinity of the phase transition. The cause of this are the domains of spins pointing in the same direction that form on a lattice near the phase transition (recall Figure 3.1(c)). These domains are difficult to flip one spin at a time: except at the boundary, the probability of one of their spins being flipped, should it be selected, is only

$$\mathrm{e}^{-8J/T_c} \approx \mathrm{e}^{-3.53} \approx 0.03. \tag{3.34}$$

The same acceptance probability is observed at low temperatures. In that regime, however, a few sweeps of the lattice suffice to produce a diametrically different state, as most of the misaligned spins are selected and turned parallel to their neighbors, and a small group of new outliers is generated. At the critical temperature, flipping a few spins in the interior of a domain does not produce a new, statistically independent state: for that, entire domains need to be reshaped, one unlikely spin flip at a time.

There are numerous algorithms for the Ising model designed to overcome this problem, but all of them adopt the same basic approach. Instead of flipping spins one at a time, these algorithms invert entire clusters of them in each iteration. Here we describe one of the most famous cluster algorithms, the Wolff algorithm, a variant of which we use in studying the ϕ^4 theory.

The idea behind the Wolff Algorithm is to build a cluster of spins by starting with a single randomly selected spin and adding aligned ones to it with a probability ρ that depends on the temperature. A single iteration of the algorithm is as follows:

1. Choose a seed spin on the lattice and add it to the cluster.

2. Consider all the nearest neighbors of the seed spin. If a neighbor points in the same direction as the seed spin, add it to the cluster with probability ρ.

3. If any spins were added to the cluster in the previous step, consider the nearest neighbors of each of them, and add them to the cluster with probability ρ if they're parallel to the seed spin.

4. Repeat step 3 until no new spins are added in an iteration.

5. Flip all spins in the cluster.

Note that in step 3 a spin that is the nearest neighbor of multiple cluster spins should be considered for addition multiple times.

To be a proper Monte Carlo algorithm, the Wolff must satisfy two conditions: ergodicity and detailed balance. In the Ising model, the first of these is satisfied as long as the probability of adding a spin to the cluster is always less than one; for then, we may get from one arbitrary state to another by choosing one "seed" spin at a time, adding none to the cluster, and flipping. As in the case of the Metropolis algorithm, the detailed balance condition is satisfied for a certain choice of ρ. In Appendix A.2, we argue the following Lemma.

Lemma 2. *The Wolff algorithm for the Ising model satisfies the detailed balance condition if the probability of adding a spin to the cluster, ρ, is given by*

$$\rho = 1 - \mathrm{e}^{-2\beta J}. \tag{3.35}$$

The Wolff algorithm mimics the behavior of the Metropolis at high temperatures, where the cluster will usually number only one spin. At low temperatures, the cluster will span the lattice, so that an independent configuration is generated in each iteration. In both of these environments, its performance is comparable to that of the Metropolis algorithm, although it is generally slowed down by the greater overhead. Near the critical point, the autocorrelation time for lattice configurations generated from either algorithms diverges as

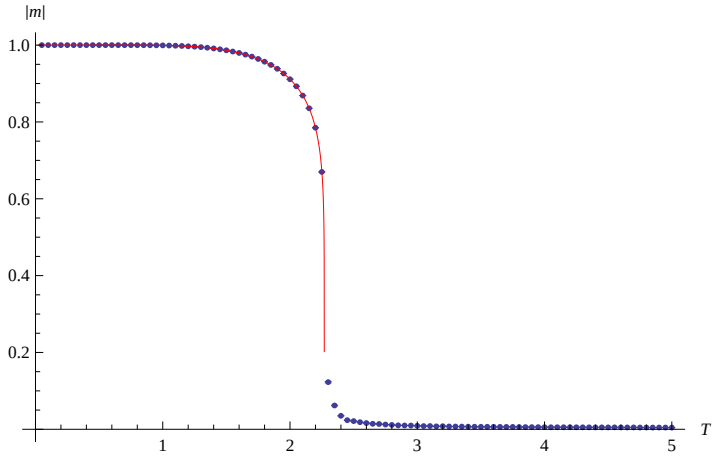
$$\tau \sim \xi^z. \tag{3.36}$$

However, $z = 2.17$ for the Metropolis algorithm and $z = 0.25 \pm 0.01$ for the Wolff—a marked improvement (Newman and Barkema, 1999, p. 101).

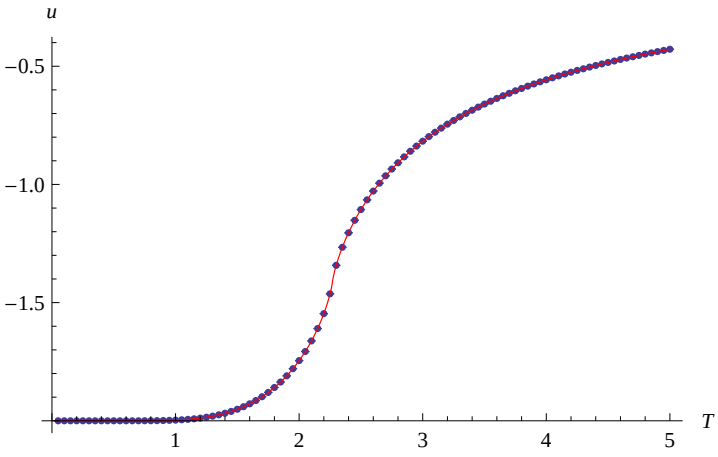
To test our implementation of this algorithm (contained in Appendix B.2.6), we simulated a 300×300 Ising model at 100 evenly-spaced temperatures $kT = 0.05,\ 0.1, \ldots, 5.0$. The runs consisted of 5,000 iterations, each comprising 4 Metropolis sweeps and one Wolff flip.[12] The magnetization, energy, specific heat and magnetic susceptibility plots obtained are shown in Figures 3.9 and 3.10.

A more extensive discussion of this and other algorithms for the Ising model can be found in Newman and Barkema (1999, Chapter 4) and Landau and Binder (2005, Chapter 5).

[12] This impressive numerical feat took Amherst's Computing Cluster less than a minute.

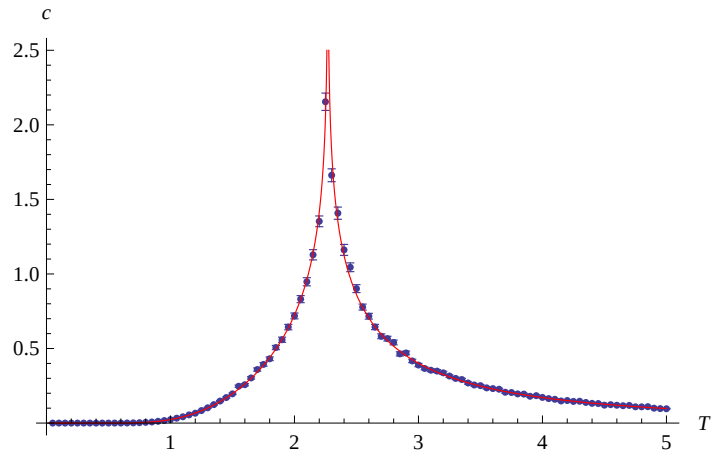


(a) Magnetization per spin

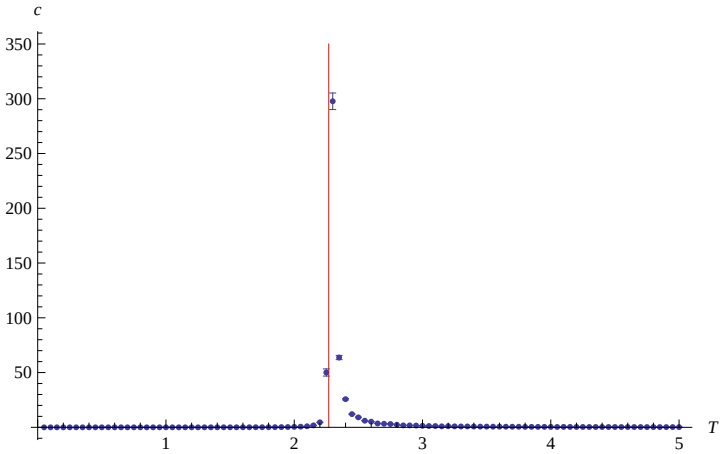


(b) Energy per spin

Figure 3.9: The magnetization and energy of a 300^2 Ising model as a function of temperature. The solid lines are the exact solutions in the thermodynamic limit.



(a) Specific heat



(b) Susceptibility

Figure 3.10: The specific heat and susceptibility of a 300^2 Ising model as a function of temperature. The solid line in (a) is the exact solution in the thermodynamic limit; the solid line in (b) marks the critical temperature.

3.5 Finite Size Scaling

As we have seen, the specific heat and magnetic susceptibility of the Ising model exhibit characteristic behavior near the critical temperature. It turns out that this behavior is systematically affected by the size of the lattice, in ways that can be exploited to obtain estimates of quantities such as the critical temperature and the critical exponents in the infinite-lattice limit. One of the standard procedures for obtaining these estimates is finite size scaling, to which we now briefly turn.

The idea behind the finite size scaling method is to exploit the existence of a system's critical exponents.[13] Critical exponents describe the singular behavior of various quantities near the critical point. We have seen some of them already:

$$c \propto |t|^{-\alpha}, \tag{3.37}$$

$$\chi \propto |t|^{-\gamma}, \tag{3.38}$$

$$\xi \propto |t|^{-\nu}, \tag{3.39}$$

where we define $t = (T - T_c)/T_c$, as in Equation 3.28. Now, we can express the divergence of the magnetic susceptibility (say) in terms of the correlation length,

$$\chi \propto \xi^{\gamma/\nu}. \tag{3.40}$$

The behavior of the susceptibility in a finite system differs from that in the thermodynamic limit because the correlation length cannot exceed L, the size of the lattice. For a finite system, then

$$\chi_L = \xi^{\gamma/\nu} \chi_0(L/\xi), \tag{3.41}$$

where $\chi_0(\cdot)$ is a function with the properties

$$\begin{aligned} \chi_0(x) &= \text{constant} && \text{for } x \gg 1, \\ \chi_0(x) &\propto x^{\gamma/\nu} && \text{for } x \to 0. \end{aligned}$$

It is convenient to eliminate ξ from Equation 3.41 in favor of t by defining the so-called scaling function of the susceptibility,

$$\tilde{\chi}(x) = x^{-\gamma} \chi_0(x^{\nu}). \tag{3.42}$$

[13] Only continuous phase transitions are characterized by critical exponents; consequently, finite size scaling can't be applied to first-order transitions. Fortunately, all of the transitions we consider in this work are second-order. We have shown this for the Ising model already, using the BCL cumulant, in Section 3.3. We will use the same method to show that it's the case for the ϕ^4.

Substituting it into Equation 3.41 and using Equation 3.39 to eliminate ξ, we get

$$\chi_L = L^{\gamma/\nu}\tilde{\chi}(L^{1/\nu}t). \tag{3.43}$$

All of the L-dependence of the susceptibility scaling function is shown explicitly in the previous equation. Therefore, if one plots

$$\tilde{\chi}(L^{1/\nu}t) = \chi_L L^{-\gamma/\nu} \tag{3.44}$$

for a number of different lattice sizes and temperatures close to T_c, one will obtain a single solid curve—as long as the values of ν, γ and T_c were chosen correctly. Even relatively small deviations from the true values will result in the curves separating. This point is illustrated Figure 3.11.

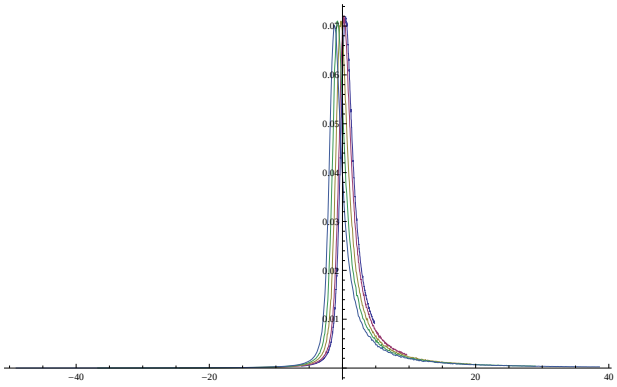
The finite size scaling method as presented, while shrewd, leaves some questions open. In particular, how should we determine which values of the parameters ν, γ and T_c result in the curves overlapping most closely? In Figure 3.11, we judged the quality of the collapse by eye. A more quantitative approach is to choose the values of the parameters which minimize the variance of the set of curves, defined as

$$\sigma^2 = \frac{1}{x_{\max} - x_{\min}} \int_{x_{\min}}^{x_{\max}} \sum_L \tilde{\chi}_L^2(x) - \left(\sum_L \tilde{\chi}_L(x) \right)^2 dx. \tag{3.45}$$

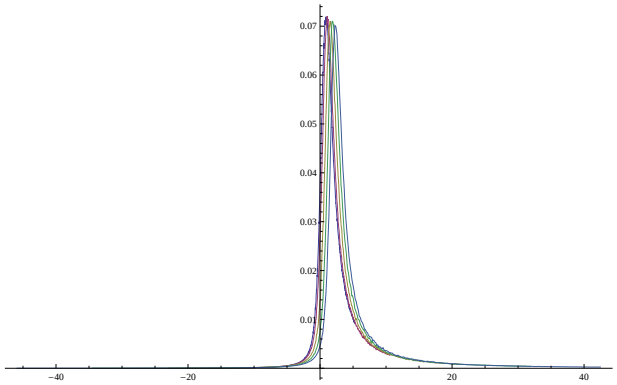
Evaluating this function is problematic, however. The finite size scaling relations given by Equations 3.37–3.39 hold only in the vicinity of the critical point, so if we choose too large an interval $[x_{\min}, x_{\max}]$, we will obtain erroneous results. A somewhat radical response to this challenge is to focus on the maxima of the scaling functions, which correspond to the maxima of the susceptibilities. Say that the scaling function is maximized at $x_0 = L^{1/\nu}t_0$. Rewriting t_0 in terms of the temperature and T_c, we obtain the following condition:

$$T_0 = T_c(1 + x_0 L^{-1/\nu}), \tag{3.46}$$

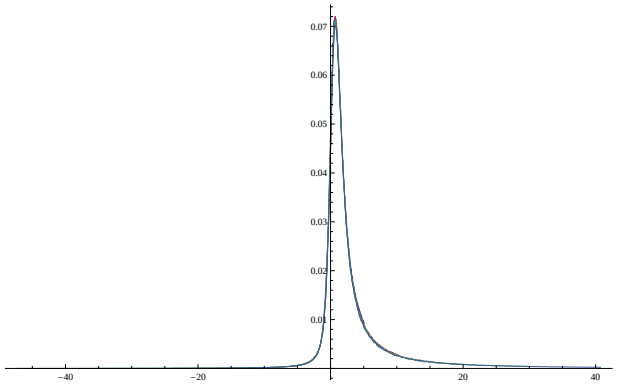
where T_0 is the temperature at which the susceptibility is maximized for a given L. Thus, it's in principle possible to estimate T_c and ν (but not γ) by regressing T_0 on L. Of course, because we're only using the maxima of the scaling functions, rather than entire curves, this method will only work if the quality of our Monte Carlo data is high. Since we're probing the critical region, where quantities such as the susceptibility and the specific heat fluctuate wildly (recall the large error bars in that region of Figures 3.7(a) and 3.7(b)), obtaining satisfactory statistics may require long runs or looking at many lattice sizes. An alternative is to minimize Equation 3.45 for a



(a) $T = 2.28$



(b) $T = 2.26$



(c) $T = 2.269 \approx T_c$

Figure 3.11: Finite size scaling applied to the Ising model: the scaling function of the magnetization plotted against $L^{1/\nu}t$. The critical exponent values were set to their exact values, $\nu = 1$ and $\gamma = 7/4$, in all of the above plots. The critical temperature was varied as shown, dramatically affecting the quality of the collapse. The lattices are $L = 64, 128, 256, 512$, and the data was generated using a mixed Metropolis-Wolff algorithm.

sequence of narrowing intervals $[x_{\min}, x_{\max}]$, and extrapolate to the limit of zero width.

The solution to these dilemmas must depend on the system being studied. We will not discuss them further here, because we did not perform any finite size scaling analysis of the Ising model beyond what was necessary to plot Figure 3.11. However, the technique is used in the next chapter to analyze data from ϕ^4 theory simulations; we cover the statistical questions in more detail there.

The explanation of finite size scaling in this section drew heavily on Newman and Barkema (1999, section 8.3.2). The interested reader may also want to consult Pathria (1996, section 13.5).

3.6 Summary

Despite its simple mathematical formulation, the Ising model has proven to be a challenging and rewarding problem to study. Coming to grips with it required the development of a new set of tools.

First, we have seen that a Markov process satisfying the conditions of ergodicity and detailed balance can be used to sample the Boltzmann distribution, and that we don't need to evaluate the partition function to design one. In some sense, the opposite is the case: we are able to evaluate certain derivatives of the partition function, such as the expected energy, using the Markov process. This fact will be critical in the next chapter, where we investigate the ϕ^4 quantum field theory by studying the partition function of an equivalent statistical mechanics system, the Landau-Ginzburg model.

Second, we have seen that a variety of characteristic phenomena take place in the vicinity of the phase transition. Some of them can be used as indicators of the phase transition, either directly (BCL cumulant) or after a procedure such as finite size scaling (susceptibility, specific heat). We are even able to distinguish between a first- and second-order phase transition using quantities measured in the course of a simulation.

We have promised in the title to study quantum field theory; we are now prepared to fulfill this promise. The next chapter is devoted to the ϕ^4 model.

Chapter 4

ϕ^4 Theory

In the last two chapters we have gradually introduced the Monte Carlo techniques that we now wish to apply to our primary research problem: the ϕ^4 quantum field theory. We begin with a discussion of the QFT motivation of our study; however, it will quickly become clear that the questions we are interested in can be rephrased as questions about a statistical mechanics system similar to the Ising model of the previous chapter. As we will see in due time, one of these questions concerns the critical coupling of the theory, $[\lambda/\mu^2]_{\text{crit}}$.

4.1 A Brief Tour of QFT

What is QFT? In the words of Zee (2003), "Quantum field theory arose out of our need to describe the ephemeral nature of life." In ordinary quantum mechanics, we represent individual particles as normalized wavefunctions: we set the probability of finding each particle *somewhere* to unity. But if our particles are always certain to be somewhere, how can they be created or destroyed? This is an important question, because particles are born and die all the time; for instance, an atomic transition from an excited to a lower-energy state will often involve the creation of a photon:

$$A^* \to A + \gamma. \tag{4.1}$$

If we try to make sense of this in ordinary quantum mechanics, we end up treating the electromagnetic field classically. This approach does produce some correct results, but is not very satisfactory, because it puts particles of matter and particles of light on an unequal footing. As a result, it cannot be extended to explain slightly more exotic but still ubiquitous events, such as

muon decay or electron-positron annihilation. Quantum field theory gives a quantitative account of such phenomena.

We will begin our short tour of QFT with a discussion of the Klein-Gordon equation, a reasonable attempt at generalizing the Schrödinger equation that goes terribly awry. To make sense of this, we introduce the path integral formulation of quantum mechanics, which allows for a particularly smooth transition from the quantum mechanics of particles to that of fields. Next, we show how the Klein-Gordon equation can be saved by interpreting it as the equation of a quantum field, and introduce the ϕ^4 model as its more interesting cousin. At the end of the section we part ways with quantum field theory. Instead of pursuing the usual perturbation theory approach which would lead us to Feynman diagram calculations, we demonstrate that the ϕ^4 theory is equivalent to a statistical mechanics system which can be studied using the methods of the previous chapters.

All of the results we discuss in this section are well-established, and the techniques used to derive them are not central to our work. Consequently, our treatment of the topic will be rather informal. The reader interested in a more thorough introduction may want to consult some of the QFT literature discussed in Appendix C.3.

4.1.1 The Klein-Gordon Equation

One of the problems of ordinary statistical mechanics is that it's nonrelativistic. We can motivate the Schrödinger equation for a free particle by starting from the classical relationship between (kinetic) energy and momentum,

$$E = \frac{p^2}{2m}, \tag{4.2}$$

and replacing E and p with the corresponding quantum mechanical differential operators,

$$E \to \imath\frac{\partial}{\partial t}, \quad \text{and} \quad p \to -\imath\hbar\boldsymbol{\nabla}. \tag{4.3}$$

We obtain

$$\imath\frac{\partial}{\partial t} = -\frac{\hbar^2}{2m}\boldsymbol{\nabla}, \tag{4.4}$$

as promised. In special relativity, the analog of Equation 4.2 is given by

$$p^2 = \frac{E^2}{c^2} - \boldsymbol{p}\cdot\boldsymbol{p} = m^2c^2, \tag{4.5}$$

where by p^2 we denote the magnitude of the momentum 4-vector, and by $\boldsymbol{p}$ the vector of momenta ($\boldsymbol{p} = p_x\hat{\boldsymbol{\imath}} + p_y\hat{\boldsymbol{\jmath}} + p_z\hat{\boldsymbol{k}}$). If we use the substitutions of Equation 4.3 to transform this relativistic relation into a differential equation, we obtain

$$\left(\frac{1}{c^2}\frac{\partial^2}{\partial t^2} - \nabla^2\right) + \frac{m^2c^2}{\hbar^2} = 0. \tag{4.6}$$

Denoting the solution of this equation by ϕ and setting $\hbar = c = 1$,[1]

$$\left(\frac{\partial^2}{\partial t^2} - \nabla^2\right)\phi + m^2\phi = 0. \tag{4.7}$$

This is the Klein-Gordon equation. Unfortunately, we quickly run into difficulties if we try to interpret it in direct analogy to the Schrödinger equation. Once we solve the latter, the probability density function is given by

$$\rho = \phi^*\phi, \tag{4.8}$$

while the probability current[2] is,

$$\boldsymbol{j} = -\frac{\imath\hbar}{2m}(\phi^*\boldsymbol{\nabla}\phi - \phi\boldsymbol{\nabla}\phi^*). \tag{4.9}$$

For the solutions of the Klein-Gordon equation to be properly relativistic, we want ρ to be the time component of the 4-vector of which $\boldsymbol{j}$ is the space component. But then,

$$\rho = \frac{\imath\hbar}{2m}\left(\phi^*\frac{\partial\phi}{\partial t} - \phi\frac{\partial\phi^*}{\partial t}\right). \tag{4.10}$$

This latter expression is no longer strictly nonnegative! Since the Klein-Gordon equation is second order in the time derivatives, we are free to specify ϕ and $\partial\phi/\partial t$ so as to generate states with $\rho < 0$. But probability densities can't be negative; this indicates that something's wrong, either with the Klein-Gordon equation or with our interpretation of it. As we will soon see, it's our interpretation that's at fault: the equation describes not individual particles, but fields on which they live.[3]

We now turn to a brief review of the path integral formulation of quantum mechanics. It will allow us to make sense of the Klein-Gordon equation and, more importantly, see the connection between QFT and statistical mechanics.

[1] This convention, known as natural units, is ubiquitous in QFT. We will use it from now on.

[2] See Griffiths (2005, Problem 1.14).

[3] This discussion of the Klein-Gordon equation was based on Ryder (1996, section 2.2).

4.1.2 Path Integrals and Quantum Fields

Consider the standard double-slit experiment: a particle is emitted from a source S at $t = 0$, passes through a screen with two holes (A_1 and A_2) and is detected on the other side by a detector O at time $t = T$. The amplitude for detection is equal to the sum of the amplitude for the particle to propagate from S through A_1 to O and the amplitude for the particle to propagate $S \to A_2 \to O$. In other words,

$$\mathcal{A}(\text{detection at } O) = \mathcal{A}(S \to A_1 \to O) + \mathcal{A}(S \to A_2 \to O). \tag{4.11}$$

As we drill more holes in the screen, the amplitude is still the sum over all holes:

$$\mathcal{A}(\text{detection at } O) = \sum_i \mathcal{A}(S \to A_i \to O). \tag{4.12}$$

Of course, an analogous thing happens if we add another screen B between A and O, with holes at B_i. The total amplitude is just a sum over the possible paths. But now, let's continue this process by placing infinitely many screens between S and O, and then drilling infinitely many holes in each of them, so that the screens disappear. Extending the idea we've used thus far, we arrive at the conclusion that

$$\mathcal{A}(\text{detection at } O) = \sum_{\text{paths}} \mathcal{A}(S \to O \text{ in time } T \text{ following a particular path}). \tag{4.13}$$

As the distances between the screens become infinitesimal, we replace the sum with an integral, thus arriving at the path integral.

This is a neat idea, but how to use it to make calculations? The first thing to note is that the amplitude $\mathcal{A}(S \to O \text{ in time } T \text{ following a particular path})$ is just a product of the amplitudes for traversing each subsequent segment of the path. Secondly, recall that the amplitude for a particle to propagate from point q_I to point q_F in time T is given by

$$\langle q_I | \, \mathrm{e}^{-\imath H T} \, | q_F \rangle \,, \tag{4.14}$$

where H is the Hamiltonian and we use Dirac notation to represent the states.[4] Making further progress is not difficult, but the notation can get a

[4] Dirac notation is very convenient and commonly used among physicists. Unfortunately, the otherwise excellent text of Griffiths (2005) usually used to teach intermediate QM at Amherst mentions it only in passing. If the subsequent manipulations (particularly those in Appendix A.3) look mysterious, we strongly recommend a glance at the first few chapters of Townsend (2000).

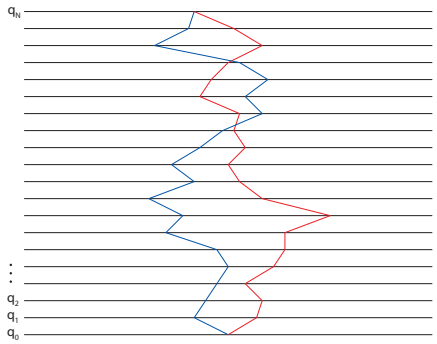


Figure 4.1: Two possible paths from an initial to a final point, with knots at $q_0, q_1, \ldots, q_N$.

bit out of hand. To prevent the mathematical details (important and interesting as they are) from distracting us at this point, we relegate the rest of the argument to Appendix A.3, and here merely state the result as a lemma:

Lemma 3. *The amplitude for a particle to transition from an initial state $|q_I\rangle$ to a final state $|q_F\rangle$ is given by*

$$\langle q_I| \, \mathrm{e}^{-\imath HT} \, |q_F\rangle = \int \mathcal{D}q(t) \, \mathrm{e}^{\imath S(q)}, \tag{4.15}$$

where $S(q)$ is the action (time integral of the Lagrangian):

$$S(q) = \int_0^T L(q, \dot{q}) \, dt = \int_0^T \frac{1}{2} m\dot{q}^2 - V(q) \, dt, \tag{4.16}$$

and the symbol $\int \mathcal{D}q(t)$ is shorthand for

$$\int \mathcal{D}q(t) \equiv \lim_{N\to\infty} \left(\frac{-\imath 2\pi m}{\delta t} \right)^{N/2} \prod_{j=0}^{N-1} \int dq_j. \tag{4.17}$$

The formidably looking expression for $\int \mathcal{D}q(t)$ is nothing but a precise statement of what we mean by "sum over all paths": the q_j's are the successive knots through which our path passes (see Figure 4.1); we let their number go to infinity, and integrate over their possible positions. Note the structural similarity between Equation 4.15 and the partition function (*e.g.* Equation 3.3). As we will show in the next section (4.1.3), this is more than

just an accidental resemblance. Emphasizing the connection, the amplitude for transitions from the ground state back to the ground state, which is of particular interest, is traditionally denoted Z:

$$Z \equiv \langle 0| \, \mathrm{e}^{-\imath HT} \, |0\rangle = \int \mathcal{D}q(t) \mathrm{e}^{\imath \int_0^T L(q,\dot{q}) \, dt}. \tag{4.18}$$

The preceding discussion was concerned with a single particle, but it is readily generalized to the case of many particles: we simply replace the single particle Lagrangian with its multiparticle equivalent, so that the action becomes

$$S(q) = \int_0^T dt \left[-V(q_1, q_2, \ldots, q_N) + \sum_a^N \frac{1}{2} m_a \dot{q}_a^2 \right]. \tag{4.19}$$

What if we want to describe a continuous field? We define its Lagrangian as the integral of a Lagrangian density, $\mathcal{L}$, over all space; the Lagrangian density depends on the value ϕ of the field at the point at which it's evaluated:

$$S(\phi) = \int_0^T dt \int \mathcal{L}(\phi) \, d^{n-1}x, \tag{4.20}$$

where $n-1$ is the number of space dimensions (generally 3, but sometimes fewer). We will want to integrate over all time as well, and write

$$S(\phi) = \int \mathcal{L}(\phi) \, d^n x, \tag{4.21}$$

where time is now denoted as x_0, the first of the spacetime coordinates.

But how does all this formalism relate to particle interactions? Where are the particles we purport to describe? They're actually not here yet, because our Lagrangian describes the unperturbed vacuum. Particles would consist of propagating excitations in the field, analogous to waves on a lake. But to observe a wave on a lake, we must disturb its surface somehow—similarly, to create and annihilate particles in a quantum field theory, we need to introduce their sources and sinks. These are added by tacking on a term $J(x)\phi$ to the Lagrangian density. All in all, we obtain the following expression:

$$Z = \int \mathcal{D}\phi \, \mathrm{e}^{\imath \int \mathcal{L} + J\phi \, d^n x}. \tag{4.22}$$

Doing quantum field theory amounts to evaluating the integral above. For each source-sink pair, we get a particle; consequently, if we expand Z in

terms of J, each expansion coefficient will be the amplitude for a process involving a number of particles equal to the corresponding power of J.

There are some restrictions on the form the Lagrangian density may take, arising from the requirement of Lorentz invariance. The simplest choice is

$$\mathcal{L}(\phi) = \frac{1}{2}\left(\frac{\partial\phi}{\partial t}\right)^2 - \frac{1}{2}\left(\nabla\phi\right)^2 - \frac{1}{2}m^2\phi^2, \tag{4.23}$$

known as the free-field theory. In fact, it's the only allowed Lagrangian for which the integral in Equation 4.22 can be evaluated exactly. Interestingly, it turns out that the Euler-Lagrange equation derived from this Lagrangian is

$$\left(\frac{\partial^2}{\partial t^2} - \nabla^2\right)\phi + m^2\phi = 0. \tag{4.24}$$

This is the Klein-Gordon equation of Section 4.1.1. The problems with the Klein-Gordon equation that we encountered in that Section arose from our incorrect interpretation. The equation does not describe a single particle, but rather an entire field. On the other hand, our stumbling upon this equation explains why this theory is known as the free field theory: we originally obtained the Klein-Gordon equation from an energy-momentum relation that included no potential energy term. The particles living on the Klein-Gordon field do not interact with each other.

If we would like our particles to interact, we need to introduce additional terms into the Lagrangian. One choice is to set

$$\mathcal{L} = \frac{1}{2}\left(\frac{\partial\phi}{\partial t}\right)^2 - \frac{1}{2}\left(\nabla\phi\right)^2 - \frac{1}{2}\mu^2\phi^2 - \frac{\lambda}{4}\phi^4. \tag{4.25}$$

This expression for the Lagrangian defines the ϕ^4 quantum field theory. Equation 4.22 cannot be evaluated exactly for this model. The usual approach to this problem is to expand Z in powers of both J (number of particles) and λ (the strength of the coupling) and use perturbative methods—work only to low orders in λ, often as low as first order. The famous Feynman diagrams are in essence just a mnemonic device for keeping track of terms in this double expansion.

As we announced at the head of this chapter, our line of attack will be different. In the next section, we will show that the theory can be recast as a statistical mechanics problem. The rest of the chapter will be devoted to tackling this statistical system using the Monte Carlo methods with which we are by now familiar. Before we end our tour of QFT, however, we need

to acknowledge and address one more issue: the regularization and renormalization of ϕ^4 theory.

We have mentioned that Feynman diagrams are a way of keeping track of the double expansion of Z in terms of λ and J, but we have not looked at any of them. This is all for the better, because it saved us from a nasty shock: the contributions to Z from some of the diagrams are infinite! These diagrams are known as divergent. We have phrased our discussion in terms of a sum over paths in position space, but Feynman diagrams are usually set up so as to involve momentum-space integrals. A divergent diagram is one which corresponds to an integral that blows up as we try to extend the integration all the way to infinite momentum. This is a symptom of the fact that QFT, like all physical theories, has a limited range of applicability: it breaks down in sufficiently extreme conditions. One time-honored way to deal with this problem is to postulate a cutoff momentum, Λ: we integrate over momenta up to Λ, and no higher. This method, and others like it, are called regularization.

Postulating a momentum cutoff may seem like a misguided thing to do: won't our predicted amplitudes depend on the value of Λ? Yes, they will, but they also depend on the parameters λ and μ: we let λ and μ be functions of Λ, so that the dependencies of the predicted amplitude on these parameters cancel out, and the amplitude does not depend on our choice of Λ. What if an experimentalist were to measure μ or λ directly? They would actually record values μ_R and λ_R, known as the renormalized (or simply "physical"—Zee 2003, p. 148) parameters, which are functions of the theoretical parameters λ, μ and Λ. If we used these parameters in computing our Feynman diagrams in the first place, we wouldn't observe the divergences. The renormalized parameters, rather than the theoretical ones, are of physical interest. To contrast them with the renormalized parameters, the theoretical ones (technically called "bare") are usually subscripted, thus: μ_0, λ_0.

As it turns out, λ is unaffected when we renormalize the ϕ^4 theory, so we may write $\lambda_R = \lambda_0 = \lambda$. However, μ is affected. One way to renormalize it, which is convenient for our purposes for reasons too complicated to go into here, is given in Loinaz and Willey (1998). The authors show that the renormalized parameter is defined implicitly as

$$\mu_R^2 = \mu_0^2 + 3\lambda \int_0^\infty \mathrm{e}^{-\mu_R^2 t} \left[\mathrm{e}^{-2t} I_0(2t)\right]^2 dt, \tag{4.26}$$

where $I_0(z)$ is the modified Bessel function of the first kind, equal to the

sum of the series,

$$I_0(z) = \sum_{k=0}^{\infty} \frac{(z^2/4)^k}{(k!)^2}. \tag{4.27}$$

Any results we obtain in our lattice simulations will pertain to the bare parameter μ_0, rather than the renormalized parameter μ_R. Since it's μ_R that's of physical significance, however, we will eventually find ourselves solving Equation 4.26 numerically to obtain it.

With these considerations in the back of our minds, we end our tour of QFT. In the next section, we will show how the ϕ^4 model can be studied using the tools of statistical mechanics.

4.1.3 The ϕ^4 as a Statistical System

Let's recall the path integral for a three-dimensional[5] ϕ^4 theory:

$$Z = \int \mathcal{D}\phi \mathrm{e}^{\imath S\, d^4x}, \tag{4.28}$$

where

$$S = \int \frac{1}{2}\left(\frac{\partial\phi}{\partial t}\right)^2 - \frac{1}{2}\left(\nabla\phi\right)^2 - \frac{1}{2}\mu_0^2\phi^2 - \frac{\lambda}{4}\phi^4\, d^4x. \tag{4.29}$$

The expression for Z looks almost like a partition function for a system with a continuum of states, except for the factor of $\imath$ and the appearance of a Lagrangian instead of a Hamiltonian. The latter two differences are geometric: they're due to the fact that quantum field theories live in relativistic spacetime (Minkowski space), while statistical mechanics models are defined in the more familiar Euclidean space. These two spaces differ by their metric—the way distance is defined. In Euclidean space,

$$ds^2 = dx^2 + dy^2 + dz^2 + \cdots. \tag{4.30}$$

In contrast, in Minkowski space, the natural measure of distance is the spacetime interval:[6]

$$ds^2 = dt^2 - dx^2 - dy^2 - dz^2. \tag{4.31}$$

[5] When we say three-dimensional, we mean three *spatial* dimensions. There's an additional time dimension, which is why we have d^4x, rather than d^3x in the expression for Z.

[6] We're still using natural units, in which $c = 1$; otherwise, there would be a factor of c^2 preceding the dt^2.

The embarrassingly direct solution to this problem, which turns out to be the correct one, is to perform a substitution $t \to \imath t$ in our Lagrangian. This procedure, known by the name of Wick rotation in complex analysis, converts from Minkowski to Euclidean space. Our differential expression becomes,

$$\frac{1}{2}\left(\frac{\partial}{\partial t}\right)^2 - \frac{1}{2}(\nabla)^2 \to -\frac{1}{2}\left(\frac{\partial}{\partial t}\right)^2 - \frac{1}{2}(\nabla)^2 = -(\nabla_4)^2, \tag{4.32}$$

where the symbol ∇_4 denotes the gradient in a four-dimensional Euclidean space. At the same time, we also transform

$$d^4x = dt\, d^3x \to -\imath d^4x_E. \tag{4.33}$$

All in all, our Lagrangian density becomes

$$\mathcal{L} = -(\nabla_4\phi)^2 - \frac{1}{2}\mu_0^2\phi^2 - \frac{\lambda}{4}\phi^4 = -\mathcal{H}_E, \tag{4.34}$$

where $\mathcal{H}_E$ is a Hamiltonian density:

$$\mathcal{H}_E = (\nabla_4\phi)^2 + \frac{1}{2}\mu_0^2\phi^2 + \frac{\lambda}{4}\phi^4. \tag{4.35}$$

As a result, our path integral has been transformed into

$$Z_E = \int \mathcal{D}\phi \exp\left(\imath \int (-\imath d^4x_E)(-\mathcal{H}_E)\right) = \int \mathcal{D}\phi\, \mathrm{e}^{-\int \mathcal{H}_E\, d^4x_E}, \tag{4.36}$$

the partition function of a four-dimensional statistical system with a continuum of states, known in statistical mechanics as the Landau-Ginzburg model. The integral of the Hamiltonian density over all (four-dimensional) space is just the energy of a particular field configuration, and the notation $\int \mathcal{D}\phi$ means a sum over all possible configurations of the field. It turns out that the Landau-Ginzburg model is an approximation to the Ising model of the same number of dimensions, and can be shown to belong to the same universality class (Binney et al., 1992, Chapter 7 and Appendix K). In the next section, we will see how to model it on a lattice.

4.2 The ϕ^4 on a Lattice

While the Landau-Ginzburg model is a statistical mechanics system, we're not quite in familiar territory yet. The Ising model we've studied in Chapter 3 was different in two important respects: it was defined on a lattice,

rather than a continuous space, and its spins could only take on two values (1 and -1), while the Landau-Ginzburg "spin" ϕ can take on a continuum of values. We will deal with the former problem below by discretizing the space on which the model is defined. The latter issue, the continuity of the spin variable, can be resolved by slightly modifying our algorithms; we will tackle it in Section 4.2.2.[7]

4.2.1 Discretization

To discretize the continuum theory specified by Equations 4.35 and 4.36, we must do three things: replace the continuous field with one defined on a lattice, deal with the derivatives and get rid of the integral over all space. The first task is easy. Instead of defining the field at every point in space, we define it only at discrete lattice sites. The derivatives are a bit more tricky. The simplest way to deal with them is to define them in terms of nearest neighbors on the lattice,

$$\frac{\partial \phi}{\partial x} \to \frac{\phi(x + \frac{a}{2}) - \phi(x - \frac{a}{2})}{a}, \tag{4.37}$$

where a is the lattice spacing. The discretization of the integral over the Hamiltonian density is similarly straightforward: it is converted into a sum over lattice sites. Since our discretization is identical to that of Loinaz and Willey (1998) and Schaich (2006), we will not strain the reader's patience with more algebra, and merely assert the result. The partition function of the system is the usual sum over all possible states,

$$Z = \sum_{\mu} \mathrm{e}^{-\beta E_\mu},$$

where

$$\beta E = \frac{1}{2} \sum_{\langle i,\, j \rangle} (\phi_i - \phi_j)^2 + \sum_i \left(\frac{1}{2} \mu_0^2 \phi_i^2 + \frac{\lambda}{4} \phi_i^4 \right). \tag{4.38}$$

The first sum is over all pairs of neighboring lattice sites, while the second is over all lattice sites. We are assuming a rectangular lattice, so $i = 1, 2, \ldots, L^2$.

[7] This section is based on Schaich (2006, sections 7.2–3).

4.2.2 Algorithms for the ϕ^4

How can we simulate the ϕ^4 theory on a lattice? We call the reader's attention to the similarity between Equation 4.38 and Equation 3.1, repeated here for convenience:

$$E = -J \sum_{\langle i\, j \rangle} s_i s_j - B \sum_i s_i.$$

In both cases, we have a nearest-neighbor interaction term and terms proportional to the magnitudes of the spins. We will exploit this similarity and use modifications of the algorithms of Chapter 3.

Metropolis In its general outline, our Metropolis algorithm for the ϕ^4 model is much like the Ising variant:

1. Choose a spin on the lattice with uniform probability.
2. Assign a new value to the spin chosen.
3. Accept the new assignment with probability

$$\rho = \begin{cases} \exp(\beta E_{\text{new}} - \beta E_{\text{old}}) & \text{for } \beta E_{\text{new}} > \beta E_{\text{old}}, \\ 1 & \text{for } \beta E_{\text{new}} \leq \beta E_{\text{old}}. \end{cases} \tag{4.39}$$

The major difference is that ϕ, unlike the spins of the Ising model, can take on a continuum of values. Therefore, if our algorithm is to be ergodic, we can't just flip them ($\phi \to -\phi$). Instead, in point 2 we add to the spin a value drawn from some distribution: $\phi \to \phi + X$, $X \sim p_X(x)$. There are few restrictions on our choice of $p_X(x)$, since the detailed balance condition can always be satisfied by an appropriate choice of ρ. We picked a uniform probability distribution on a symmetric interval,

$$p_X(x) = \begin{cases} \frac{1}{2a} & \text{for } x \in [-a, a], \\ 0 & \text{otherwise.} \end{cases} \tag{4.40}$$

Our main motivation was that uniform deviates are generated faster than values drawn from other distributions. We obeyed the "time honored rule of thumb" (Krauth, 2006, p. 7) and chose the width of the interval, $2a$, with a view towards achieving an acceptance rate around 0.5; after a number of trial simulations, we settled on $a = 2.5$.

Wolff Our implementation of the Wolff algorithm is the naïve but standard one (Brower and Tamayo, 1989; Charng, 2001; Schaich, 2006). The outline is again similar to the Ising implementation:

1. Choose a cluster seed spin on the lattice.

2. Add neighboring spins to the cluster with probability ρ.

3. Invert the cluster: $\phi \to -\phi$.

As in the case of the Ising model, we will add spins to the cluster only if they are properly aligned, *i.e.* have the same sign. The derivation of a value of ρ that guarantees detailed balance has been relegated to Appendix A.4, and here we only quote the result:

Lemma 4. *The Wolff algorithm for the ϕ^4 theory with the energy given by Equation 4.38 satisfies the detailed balance condition if the probability of adding a spin to the cluster, ρ, is given by*

$$\rho = 1 - \mathrm{e}^{-2\phi(o)\phi(c)}, \tag{4.41}$$

where $\phi(c)$ is a cluster spin and $\phi(o)$ is its neighbor being considered for addition to the cluster.

It is immediately obvious from the outline of our Wolff algorithm that it cannot be ergodic, for it doesn't change the lengths of the spins. Therefore, in our simulations, we have to intersperse Wolff flips with Metropolis sweeps to ensure ergodicity. Choosing the ratio of Metropolis to Wolff steps is an important but difficult problem. It is important because the equilibration and autocorrelation times, as well as the degree of critical slowing down, most likely depend on the Metropolis-Wolff ratio. It is difficult, however, because the nature of this dependence will likely change as one varies μ_0^2 and λ. To the best of our knowledge, no systematic study of the effect of the Metropolis-Wolff ratio has been carried out. Different values have been favored in the literature, from $1:1$ (Brower and Tamayo, 1989; Loinaz and Willey, 1998; Charng, 2001) to $5:1$ (Schaich, 2006). In our runs, we performed four Metropolis sweeps (steps per site) for each Wolff flip.

Our implementation of the mixed Metropolis-Wolff algorithm can be found in Appendix B.2.7.

4.3 Phase Transition Indicators

In the previous chapter, we have mentioned two approaches to finding the critical point of a model: finding the intersection of the BCL cumulant curves and performing a finite size scaling procedure on the susceptibility, specific heat or other quantity that obeys a scaling law in the vicinity of the critical point. In this section, we will describe our attempts at applying these techniques to the Landau-Ginzburg model. We focus exclusively on the two dimensional model from now on.

Our goal in the search for the phase transition of the Landau-Ginzburg model is the corroboration of the surprising results of Schaich (2006), who argued that the critical coupling $[\lambda/\mu^2]_{\mathrm{crit}}$ depends on λ in a nonlinear fashion.

Our efforts throughout the section will be facilitated by the fact that the two-dimensional Landau-Ginzburg model is in the same universality class as the two-dimensional Ising model. This implies that their critical exponents are the same; in particular, $\nu = 1$, $\gamma = 7/4$ (*e.g.*, Newman and Barkema, 1999, p. 235) and $\beta = 1/8$ (Pathria, 1996, p. 387).

4.3.1 The BCL Cumulant

Deterred by the technical difficulties inherent in finite size scaling, we initially hoped to base our estimates on the intersection point of the BCL cumulant curves. We define it in analogy to Equation 3.33:

$$U_L = 1 - \frac{\langle \phi^4 \rangle}{3\langle \phi^2 \rangle^2}. \tag{4.42}$$

The most obvious way to proceed is to obtain BCL cumulant data for a variety of lattices and find the point at which the curves are closest together. For two curves, say f and g, this is equivalent to minimizing the loss function

$$LF = |f(x) - g(x)|. \tag{4.43}$$

For three curves (f, g, h), the appropriate formula would be

$$LF = |f(x) - g(x)| + |f(x) - h(x)| + |h(x) - g(x)|. \tag{4.44}$$

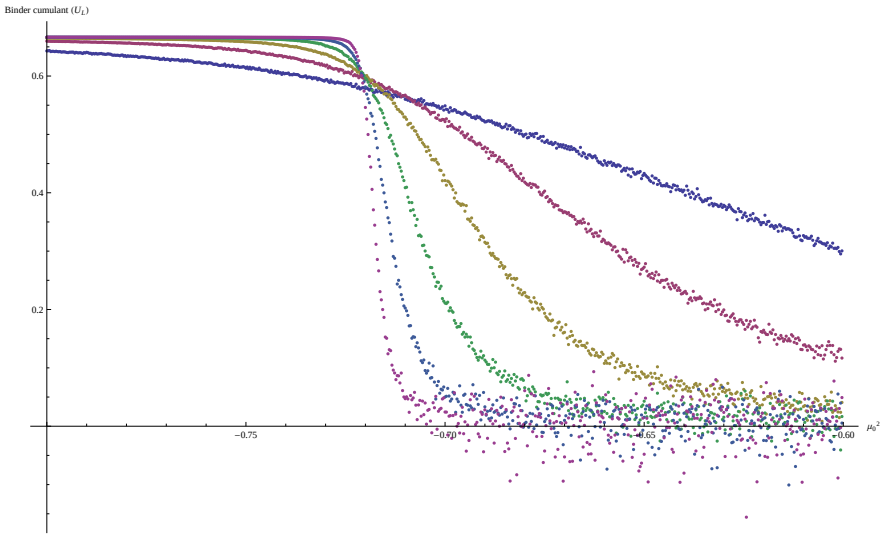


Figure 4.2: BCL cumulant curves for the ϕ_2^4 theory, $\lambda = 0.5$, with $L = 16, 32, 64, 128, 256, 512$. The sharper curves correspond to larger lattice sizes.

In general, the analogous loss function for n curves will have $n!/2(n-2)!$ elements:[8]

$$LF = \sum_{j>i+1}^{n} \sum_{i=1}^{n} |f_i(x) - f_j(x)|. \tag{4.45}$$

Minimizing this function numerically is an easily programmed task, especially given that our functions are only defined at a relatively small number of points. We proceeded to generate some data, and observed the disturbing phenomenon depicted in Figure 4.2: the curves fail to intersect at a unique point! The effect is clearly not due to random fluctuations: as a result of extensive averaging, there is very little scatter in the points, particularly in the vicinity of the intersection point. While the curves do seem to intersect closer and closer to one another as the lattice size increases, there is a systematic leftward drift in the position of the intersection point. We investigated other regions of the parameter space ($\lambda = 1.0, 0.1$) with similar resolution and observed the same effect.

A similar phenomenon was predicted by Binder and Heermann (2002, p. 46), though the authors believed it to consist of "scatter," discernible only

[8] Because a list of n distinct objects has $n!/(n-r)!$ permutations of length r, and we treat permutations containing the same objects as identical.

when working with "very small linear dimension." What we observe could be "Binder scatter," though there is a more disturbing possibility: the finite size scaling arguments used to argue for the existence of a unique intersection point may not hold as well in our case. When we discussed finite size scaling in Section 3.5, we always considered varying the temperature; in our study of the ϕ^4 model, we are varying one of the parameters. In general, the two procedures are not equivalent, though the similarity between figures such as 4.2 and 3.8 (page 44) suggests the difference may not matter much. We will return to this issue in the next section, as we discuss a more direct application of finite size scaling to our system.

The presence of the systematic effect discussed above highlighted the need to develop a more robust estimation approach than finding the data point at which the distance between the cumulant curves is minimized. There is an evident trend in the U_L curves: they become sharper and their inflection point shifts leftwards as the lattice size is increased. We searched for a technique that would allow us to quantitatively estimate the $L \to \infty$ limit of a sequence of such transformations, but to no avail. There appears to be no established statistical method for estimating such a limit without the knowledge of the functional form of the BCL curves. Therefore, we fiddled with the $\lambda = 0.5$ data a bit, in the hope of finding some simple empirical regularity. The general shape of the U_L curves is sigmoidal; consequently, a very good fit can be obtained using a hyperbolic tangent function, of the form

$$U_L = a \tanh\left(b(\mu_0^2 - c)\right) + d, \tag{4.46}$$

where a, b, c and d are parameters. A sample fitted curve is depicted in Figure 4.3, and the fit parameters are in Table 4.1.

Encouragingly, the parameter b is linear in L. A positive correlation between b and L was expected, given that the limit $b \to \infty$ of a sequence of hyperbolic tangent curves is a step function; the BCL curves appear to behave similarly. The important parameter, however, is c, which determines the location of the inflection point: as the lattice size increases, we expect c to approach the position of the critical point. Inspired by the finite size scaling relationship for the magnetic susceptibility, we attempted regressing c on $L^{-1/\nu}$. Unfortunately, the results are not as clear-cut as one might have hoped: there is a systematic variation in c that isn't accounted for by the fit (see Figure 4.4). There may be a relationship between this variation and the lack of a unique intersection point of the BCL curves, for in both cases using larger lattices always results in a more negative estimate of the critical point, with the effect gradually fading as the lattice size increases.

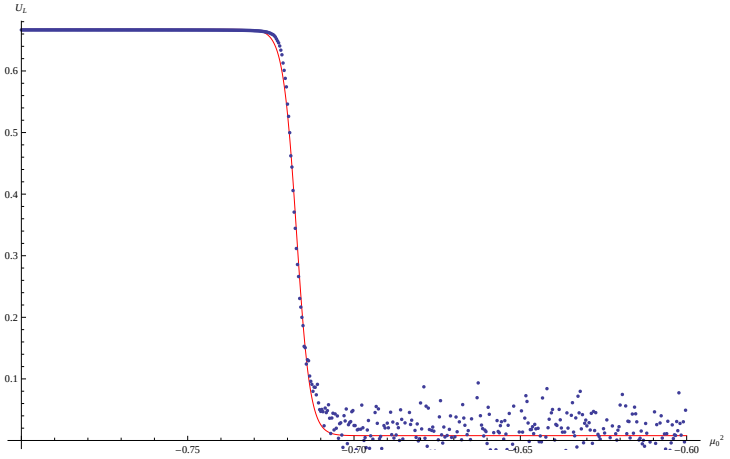


Figure 4.3: BCL cumulant curve for the ϕ_2^4 theory, $\lambda = 0.5$, with $L = 512$. The solid red line is a fit to Equation 4.46.

Table 4.1: The best-fit parameters obtained when BCL cumulant data for $\lambda = 0.5$ was fitted to Equation 4.46. The uncertainty estimates are 1σ bands, under the assumption of normally distributed errors.

L	a	b	c	d
16	0.252(2)	10.5(1)	0.6432(8)	0.410(2)
32	0.2662(8)	19.4(1)	0.6717(2)	0.3811(6)
64	0.3118(7)	24.6(2)	0.6928(1)	0.3585(5)
128	0.3233(8)	64.2(8)	0.7061(1)	0.3464(7)
256	0.3285(9)	123(2)	0.71340(9)	0.3401(9)
512	0.330(1)	246(9)	0.71729(9)	0.338(1)

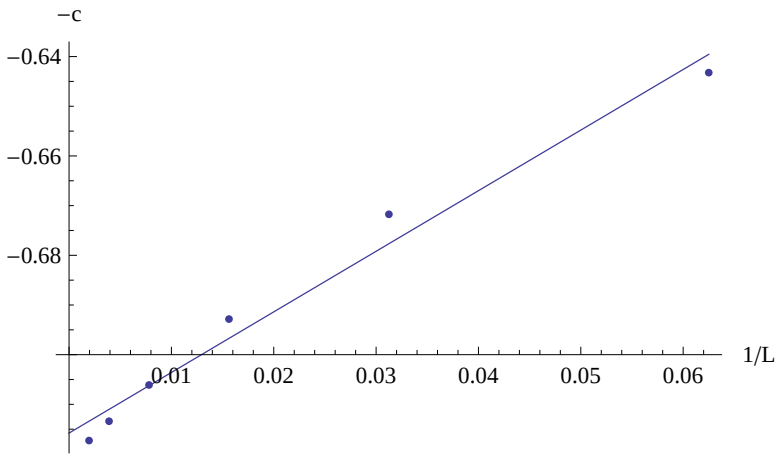


Figure 4.4: The parameter c from Table 4.1 regressed on L^{-1}. Note that we plot $-c$, rather than c, since the critical point corresponds to $\lim_{L\to\infty} -c$.

The apparent systematic effect in the results and the need to fit elaborate functional forms made us reluctant to use the BCL cumulant as a phase transition indicator. Consequently, we employed only the susceptibility in our analysis.

4.3.2 Susceptibility

We can define the "magnetic" susceptibility per spin of the ϕ^4 model analogously to that of the Ising:

$$\chi = \frac{1}{L^2}\left(\langle\phi^2\rangle - \langle\phi\rangle^2\right). \tag{4.47}$$

We expect it to diverge at the critical point, obeying the same scaling relation as the Ising susceptibility (Equation 3.38):

$$\chi \propto |t|^{-\gamma}.$$

There is an obvious difficulty here, one we brushed aside when discussing the BCL cumulant but must now confront head-on: temperature is not a well-defined property of our model. In our "energy" expression, Equation 4.38, there is already a β on the left hand side! We respond to this challenge in the standard way (Loinaz and Willey, 1998; Schaich, 2006): we fix λ and vary μ_0^2 as if it were the temperature. Justifying this procedure

rigorously is difficult, but at the very least, it seems like a reasonable thing to try. We will let the results speak for themselves; their testimony is more favorable than one might initially expect.

As we have remarked in Section 3.5, there are two ways to extract estimates of the critical point from susceptibility data using finite size scaling: collapsing the susceptibility scaling function curves for different lattice sizes or collapsing only their maxima. Using the entire curves is preferable, but requires that we know the value of the susceptibility at every point in an interval about the critical point; our data, of course, consists of samples at discrete points. To obtain the missing parts of the curve, we need to engage in interpolation. The state-of-the-art technique for doing so with Monte Carlo data is the multiple histogram method, which exploits features of the Boltzmann distribution to produce an excellent interpolation, even from a small number of data points. Unfortunately, the method is computationally intensive, difficult to implement, and would require us to store all of the data generated in the course of our simulations (rather than just the run averages).

As an alternative, we considered using optimally smoothed splines, which are reputed to produce very good interpolations (Hastie et al. 2001, section 5.4, Fujioka and Kano 2006). Unfortunately, existing implementations of this technique in languages such as R could not be used in a minimization procedure designed to find the best estimate of the critical point. We considered writing our own implementation, but failed to find sufficiently specific references.

Faced with these problems, we abandoned the idea of using entire scaling functions and instead used only their maxima. Consider Equation 3.44, repeated here for convenience:

$$\tilde{\chi}(L^{1/\nu}t) = \chi_L L^{-\gamma/\nu}.$$

Recall that $\tilde{\chi}(\cdot)$ is the susceptibility scaling function and χ_L is the magnetic susceptibility per spin. It follows from this equation that for a given L the maximum of the scaling function must coincide with the maximum of the susceptibility. If we find the maxima of the susceptibility for different lattice sizes, we can use an analog of Equation 3.46 to find the critical point using a simple regression:

$$\mu_m = \mu_c(1 + x_m L^{-1/\nu}), \tag{4.48}$$

where μ_m is the value of μ_0^2 at which the susceptibility is maximized, μ_c is the position of the critical point, x_m is the position of the maximum of the susceptibility scaling function (independent of L or μ_0^2) and $\nu = 1$ is

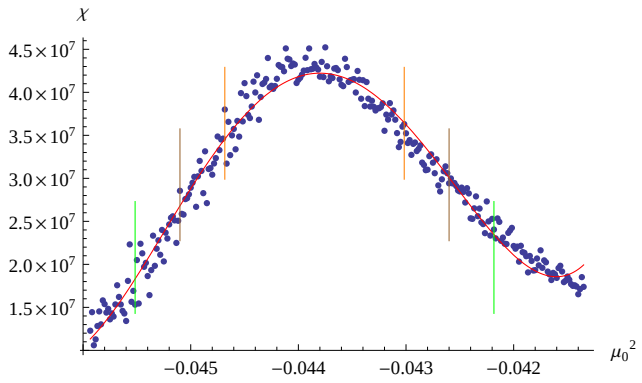


Figure 4.5: Estimating the susceptibility maxima using a polynomial fit. The robustness of the method is tested by fitting the polynomial to four different regions. The first fit was over the entire range shown; the other three ranges are indicated by pairs of vertical lines. See Table 4.2 for the results.

the critical exponent. To find the maximum of the susceptibility, we resort to the simple technique of polynomial fitting: we fit a general fourth-order polynomial,

$$f(x) = a + bx^2 + cx^3 + dx^4, \tag{4.49}$$

to a section of the susceptibility curve between the inflection points, and use the polynomial's maximum as an estimate of the susceptibility maximum. The polynomial is an approximation to the Taylor expansion of the function, so the technique works well as long as the section of the curve we're considering is small enough.[9] The uncertainty in the estimate of the maximum is evaluated using the bootstrap method with 250 resamples, which ought to give an estimate of the error accurate to about 5% (Efron, 1979)—perfectly satisfactory, since we will follow the standard practice and quote only the uncertainty on the last digit.

One could object to the polynomial fitting technique described in the last paragraph by pointing out that the result obtained depends on the range of μ_0^2 about the maximum to which we fit the polynomial. This is a legitimate concern, although sensitivity to the choice of range will show up in the uncertainty estimate: if the result depends on which points were included in the sample, the bootstrap estimates obtained from different re-

[9] In the rare case when the number of data points in the vicinity of the maximum was insufficient (fewer than twenty), we observed Runge's phenomenon: the polynomial exhibited spurious curvature. In those cases, we fit a third-order polynomial instead.

Table 4.2: Estimates of the critical point obtained from data in Figure 4.5 by considering different ranges of points. Note that the estimate of the critical point is insensitive to the size of the interval considered.

μ_0^2 Lower Bound	μ_0^2 Upper Bound	Critical point
-0.0459333	-0.0413500	$-0.0438(2)$
-0.0455167	-0.0421833	$-0.0439(2)$
-0.0451000	-0.0426000	$-0.0439(2)$
-0.0446833	-0.0430167	$-0.0439(6)$

samples will be widely disparate, and the reported error of the result will be large.[10] We also performed a small experiment verifying the robustness of the method: we fit fourth-order polynomials to different sections of the susceptibility curve for $\lambda = 0.02$, $L = 192$. The curve itself and the regions of it used in our fits are shown in Figure 4.5; the estimates of the critical point from the fits are reported in Table 4.2. The estimates are in very good agreement. The increase in uncertainty for the smallest range shouldn't cause alarm: a glance at Figure 4.5 reveals that the amplitude of the noise is almost as large as the curvature, so a bigger uncertainty is expected.

Having obtained the susceptibility maxima, we use a weighted linear regression to determine μ_c from Equation 4.48. As in Section 2.2.2, we use the estimated sample variance (the uncertainty estimates from the bootstrap) to determine the weights. The estimate of the intercept from the regression is our best estimate of the critical point in the thermodynamic limit. The natural estimate of the uncertainty in this value is the regression standard error, but that's not the estimate we use. The standard error will give the correct estimates only if the underlying model is indeed linear—in other words, if Equation 4.48 is valid and $\nu = 1$. Otherwise, the standard error will be an underestimate of the uncertainty. To guard against this possibility, we used the jackknife method to measure the uncertainties. The jackknife is similar to the bootstrap, but instead of a large number of random resamples, uses "leave-one-out" resamples by removing one data point. Its only advantage over the bootstrap is computational: we need only produce as many resamples as there are points in the data set, rather than upwards of 250. Otherwise the two techniques are essentially the same. For a brief introduction through examples, the reader is referred to Efron (1979).

[10] The bootstrap method was discussed in some detail in Section 3.2.3, page 38.

Because we are concerned about the validity of Equation 4.48, we perform an additional set of regressions based on it, in which not only the intercept but also the critical exponent ν are estimated. This is carried out by regressing the position of the susceptibility maximum on $L^{-1/\nu}$ instead of L^{-1}, as in the previous fit. We continue to use the jackknife technique to estimate the errors. Since the true value of this exponent is known to be 1, deviations from that value would be a sign that Equation 4.48 is false. We will refer to this regression as Regression 2, to distinguish it from the linear regression of the previous paragraph.

4.4 The Simulation

We used a mixture of the Metropolis and Wolff algorithms described in Section 4.2.2 to obtain BCL cumulant and susceptibility measurements for a variety of λ values. In this section, we describe the data we've collected and the results of its analysis.

Since we are interested in making a statement about the critical coupling, which is related the limit of the transition line as $\lambda \to 0$, we performed simulations at $\lambda = 0.02, 0.03, 0.04, 0.05, 0.1, 0.5, 0.7, 1.0$. We initially intended to perform each of these on lattices of seven sizes, $L = 64, 128, 192, 256, 384, 512$ and 600. However, a bug in a program corrupted some of our results, so that results for all seven lattices were available only for $\lambda = 0.02, 0.03, 0.04, 0.1, 1.0$. For the remaining values of λ, results were available for four lattices, $L = 64, 128, 256$ and 512. The μ_0^2 ranges of the simulations were chosen with a view towards sampling the susceptibility peaks.

We performed an analysis of the susceptibility data as described in the previous section. The results of both regressions are shown in Table 4.3. The estimated positions of the critical points are in agreement for $\lambda \geq 0.04$; the higher uncertainties in Regression 2 reveal a tendency to overfit the data, not surprising given that three parameters (rather than two, as in Regression 1) are determined from a rather small data set. Below $\lambda = 0.04$, however, the two critical point estimates diverge. This is coupled with a decrease in the estimate of $1/\nu$, which thus far was within uncertainty of unity. To see what's going on, we plot Regression 1 for $\lambda = 0.02$ (see Figure 4.6). The data points all fall within uncertainty of the regression line, but they don't seem to lie on a straight line. Instead, they arrange themselves on a concave curve, roughly resembling the plot of $f(x) = \sqrt{x}$. This is reflected in the estimate of $1/\nu$ from Regression 2: a better fit is achieved

Table 4.3: Best estimates of the critical points in ϕ^4 theory from the susceptibility curves. In Regression 1, it was assumed that $\nu = 1$, while in Regression 2, ν was a free parameter. In both, the susceptibility maximum was regressed on $L^{-1/\nu}$.

λ	Regression 1	Regression 2	
	μ_0^2	μ_0^2	$1/\nu$
1.0	$-1.2725(1)$	$-1.2728(4)$	$0.96(7)$
0.7	$-0.9515(2)$	$-0.9511(3)$	$1.04(6)$
0.5	$-0.7211(2)$	$-0.7209(9)$	$1.0(2)$
0.1	$-0.18432(3)$	$-0.1845(2)$	$0.86(9)$
0.05	$-0.10070(3)$	$-0.1008(2)$	$0.86(15)$
0.04	$-0.0826(1)$	$-0.083(5)$	$0.7(6)$
0.03	$-0.064096(4)$	$-0.06453(1)$	$0.63(8)$
0.02	$-0.04465(8)$	$-0.04510(2)$	$0.6(2)$

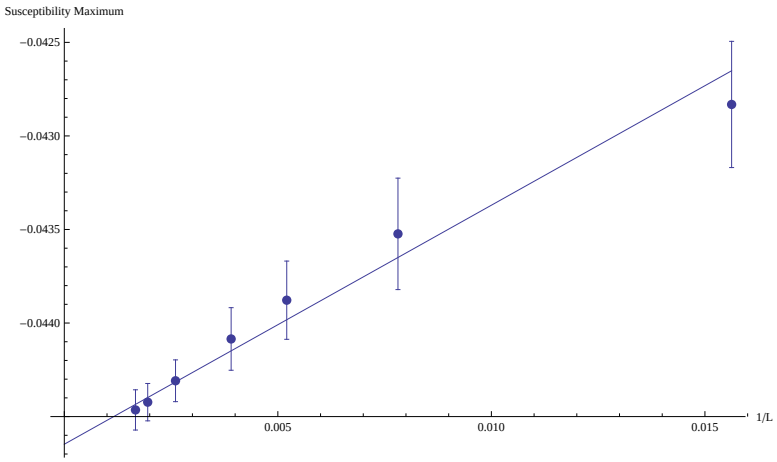


Figure 4.6: Regression 1 for $\lambda = 0.02$. Notice that the data does not fall on a straight line, but rather on a concave curve.

if $1/L$ is raised to a fractional power. A comparison with Figure 4.4 suggests that we're observing the same phenomenon we ran into when looking at the BCL cumulant data: the critical point "drifts" towards more negative μ_0^2 values as the lattice size increases.

One apparent difference is that Figure 4.4 presents data for $\lambda = 0.5$, while Figure 4.6 presents data $\lambda = 0.02$. The susceptibility fits for $\lambda = 0.5, 0.7, 1.0$ reveal no "drift," as the estimates of $1/\nu$ from Regression 2 indicate (they're all consistent with $\nu = 1$). This difference, however, does not imply that the effect is more pronounced for the BCL cumulant than for the susceptibility. It can be explained by noting that Figure 4.4 includes data for small lattices, $L = 16$ and $L = 32$, which were not used in susceptibility simulations due to concerns about corrections to scaling; as can be seen in Figure 4.6, smaller lattices contribute more to the curvature than large ones, so their exclusion from the susceptibility fits is responsible for the impression that the "drift" is less pronounced there.

We are forced to conclude that the "drift" of the critical point is probably an actual feature of the system, rather than an artifact of the indicators as was initially hoped. At worst, this may mean that our extension of finite size scaling from variations in temperature to variations in μ_0^2 was illegitimate; in that case, the ν that appears in our regression is not the universal correlation length exponent of the Ising universality class, but just a parameter that may depend on L. It's possible, however, that finite size scaling is applicable, and the deviations from linear behavior are just particularly noticeable corrections to scaling. This optimistic interpretation is consistent with the observation that the effect is particularly pronounced for small lattices.

Regardless of the correct interpretation of the deviations from finite size scaling, it is worthwhile to calculate an estimate of the critical coupling $[\lambda/\mu]_{\text{crit}}$. The reader may recall from Section 4.1.2 that the critical values of μ_0^2 we obtained are the "bare," unphysical parameters. We use Equation 4.26

$$\mu_R^2 = \mu_0^2 + 3\lambda \int_0^\infty \mathrm{e}^{-\mu_R^2 t} \left[\mathrm{e}^{-2t} I_0(2t)\right]^2 dt,$$

to obtain their renormalized counterparts, listed in Table 4.4. The uncertainties are obtained by solving the preceding equation for the upper and lower bound on each parameter. From these estimates, we can calculate the coupling constant f,

$$f = \frac{\lambda}{\mu_R^2}. \tag{4.50}$$

Table 4.4: Best estimates of the critical points in ϕ^4 theory from the susceptibility curves—renormalized values.

λ	μ_R^2 (Regression 1)	μ_R^2 (Regression 2)
1.0	10.275(3)	10.28(1)
0.7	10.224(8)	10.21(1)
0.5	10.23(1)	10.21(5)
0.1	10.55(2)	10.65(28)
0.05	10.27(25)	10.90(64)
0.04	10.63(8)	11(4)
0.03	10.71(4)	11.19(11)
0.02	10.68(13)	11.43(3)

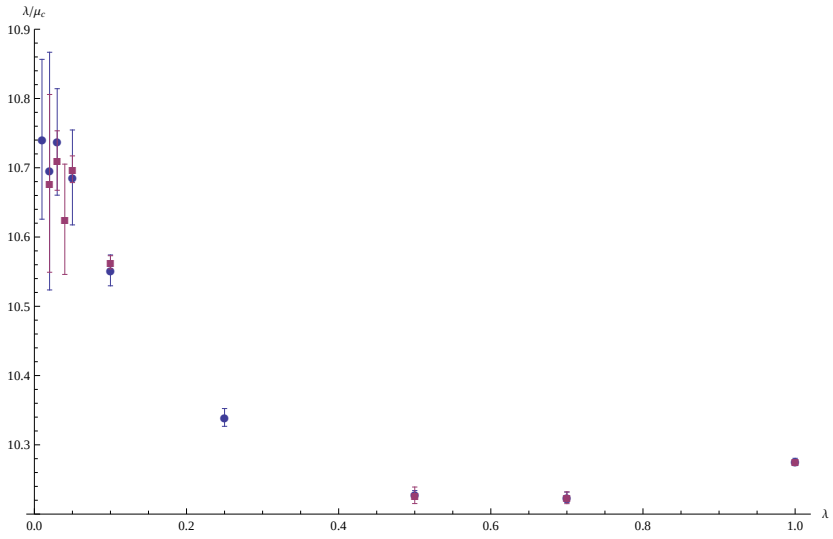
We plot f as a function of λ in Figure 4.4 using data from both regressions and contrast our results with those of Schaich (2006). As expected, the results from Regression 1 are more akin to Schaich's, since he also performed a linear fit. They also benefit from lower uncertainties, which can be traced back to the lack of overfitting. However, Regression 2 shows a more robust trend; this is somewhat surprising, because if fitting ν were spurious, we'd expect the larger uncertainties to be accompanied by a larger scatter in the observation points. It's possible that treating ν as a fit parameter accounts for the "drift" better than a linear fit. Of course, it may also be that such a regression systematically biases μ_R^2 downwards, and the robustness of the trend is a reflection of the robustness of the bias. Resolving this issue is relegated to future work. The important point is that the results of Regression 2 are not in conflict with Schaich's original finding that the $\lambda - \mu$ regression line deviates from the straight line once postulated by Loinaz and Willey (1998); on the contrary, they serve to show that a linear model derived from finite size scaling may underestimate this deviation.

Since data from Regression 1 is so similar to that obtained by Schaich, it is well fitted by his models,

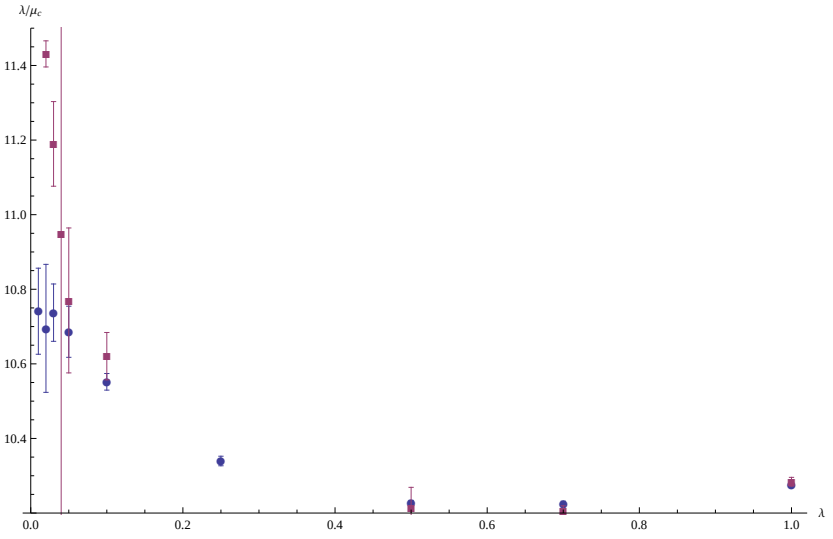
$$f(\lambda) = a + b\lambda + c\lambda\log(\lambda) \tag{4.51}$$

$$f(\lambda) = a + b\lambda + c\lambda\log(\lambda) + d\lambda^2\log(\lambda). \tag{4.52}$$

The best estimates of the critical coupling (*i.e.* the coupling in the limit $\lambda \to 0$) from these models is $10.69(1)$ and $10.7(2)$, respectively. These values are somewhat below his ($10.774(31)$ and $10.874(17)$) due to the outlier at $\lambda = 0.05$; if the outlier is dropped, the estimates become $10.78(2)$



(a) Regression 1



(b) Regression 2

Figure 4.7: Plots of f as a function of λ, using data from Regressions 1 (a) and 2 (b). In both cases, our results (red squares) are compared with Schaich's (blue disks). The three data sets are indistinguishable for $\lambda \geq 0.5$, but differ for $\lambda \leq 0.1$.

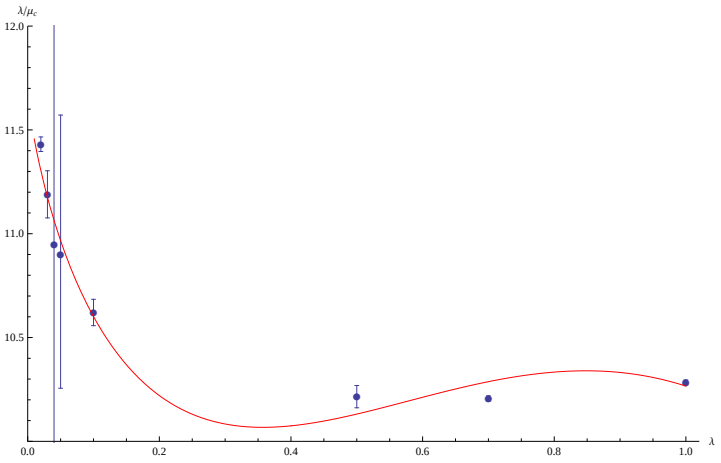


Figure 4.8: Data from Regression 2 fitted to Equation 4.52.

and $10.76(4)$, which are in the same range. Data from Regression 2, is not fitted particularly well by these logarithmic expressions: Equation 4.51 is not steep enough, and Equation 4.52 results in spurious curvature (see Figure 4.8). The best estimate of the critical coupling from this fit is $11.7(1)$, significantly higher than the estimate from Regression 1.

Chapter 5

Conclusion

In this work, we have studied a broad selection of classical problems which can be treated using Monte Carlo methods, from random walks, through the Ising model to the ϕ^4 quantum field theory. We corroborated the surprising discovery of Schaich (2006) concerning the critical coupling of the ϕ^4 model. We also used a more flexible regression model to argue that the critical coupling may be even higher than Schaich was able to show. This is so because the finite size scaling assumptions motivating his linear model are violated for the lattice sizes under consideration, especially near the $\lambda \to 0$ point that is of interest.

On our way to consider the ϕ^4 model, we devoted more attention than is usual to the subject of random walks, and independently derived a pleasing minor result: the exact expression for the end-to-end distance of the non-reversing random walk.

We particularly regret not having been able to pursue two avenues of further research. Firstly, we considered (but never implemented) an alternative method for determining the position of a critical point in ϕ^4 theory: real-space renormalization, a technique based on the Monte Carlo renormalization group (see Newman and Barkema, 1999, Section 8.4). It would be interesting to compare the estimates obtained from a renormalization algorithm with the susceptibility and BCL cumulant estimates. The similar behavior of the susceptibility and the BCL cumulant is in retrospect not very surprising, as both indicators depend on finite size scaling. Since real-space renormalization does not, it may provide an independent perspective on the phase transition line of the ϕ^4 theory.

Another project we were unable to pursue was the verification of Schaich's results for the three dimensional ϕ^4 theory. Schaich followed exactly the

same procedure in his analysis of the statistical mechanical problem, which we believe may have been a mistake. The three dimensional ϕ^4 theory is in the same universality class as the four dimensional Ising model, implying that $\nu = 1/2$ (Binney et al., 1992, p. 176), rather than $\nu = 1$. It may have been more appropriate, therefore, to regress the estimates of the critical points from finite lattices on L^{-2}, rather than L^{-1}.

Appendix A

Proofs

This chapter contains the proofs of lemmas postulated in the text.

A.1 Proof of Lemma 1

Lemma 1 states that for a non-reversing random walk on a rectangular, d dimensional lattice,

$$\langle \boldsymbol{s_{N-i}} \cdot \boldsymbol{s_N} \rangle = \left(\frac{1}{2d-1} \right)^i \quad \forall i \in \mathbb{N} < N, \tag{A.1}$$

where $\boldsymbol{s_i}$ is a vector representing the ith step of the walk. We will proceed by induction.

The base case is $\langle \boldsymbol{s_{N-1}} \cdot \boldsymbol{s_N} \rangle$. Given a step $\boldsymbol{s_{N-1}}$ on a d dimensional rectangular lattice, the step $\boldsymbol{s_N}$ can be taken in one of $2d-1$ directions ("up" or "down" each of the dimensions, but with the restriction $\boldsymbol{s_N} \neq -\boldsymbol{s_{N-1}}$). Of these possible steps, $2d-2$ will be orthogonal to $\boldsymbol{s_{N-1}}$, and one will be parallel to it ($\boldsymbol{s_N} = \boldsymbol{s_{N-1}}$). The orthogonal $2d$ steps result in $\boldsymbol{s_{N-1}} \cdot \boldsymbol{s_N} = 0$, while the unique parallel step yields $\boldsymbol{s_{N-1}} \cdot \boldsymbol{s_N} = 1$. Therefore,

$$\langle \boldsymbol{s_{N-1}} \cdot \boldsymbol{s_N} \rangle = \frac{1}{2d-1}. \tag{A.2}$$

The induction assumption is that there exists a $j \in \mathbb{N}$ such that

$$\langle \boldsymbol{s_{N-j}} \cdot \boldsymbol{s_N} \rangle = \left(\frac{1}{2d-1} \right)^j. \tag{A.3}$$

We will show that from this assumption it follows that

$$\langle \boldsymbol{s_{N-(j+1)}} \cdot \boldsymbol{s_N} \rangle = \left(\frac{1}{2d-1} \right)^{j+1}. \tag{A.4}$$

In the following discussion, $\boldsymbol{\omega}$ denotes a non-reversing random walk, and $\boldsymbol{\omega_k}$ its kth step. Consider the following three sets of non-reversing random walks. The set A_i is the set of all NRRWs of i steps terminating with step $\boldsymbol{s_N}$ (i.e., $\boldsymbol{\omega_i} = \boldsymbol{s_N}$); it has a_i elements. The set B_i contains all NRRWs of i steps terminating with step $\boldsymbol{s_N}$, such that their first step is parallel to $\boldsymbol{s_N}$:

$$B_i = \{\boldsymbol{\omega} \in A_i \mid \boldsymbol{\omega_1} \cdot \boldsymbol{\omega_i} = 1\}. \tag{A.5}$$

The set B_i numbers b_i elements. Finally, the set C_i contains all NRRWs of i steps terminating with step $\boldsymbol{s_N}$, such that their first step is *anti*parallel to $\boldsymbol{s_N}$:

$$C_i = \{\boldsymbol{\omega} \in A_i \mid \boldsymbol{\omega_1} \cdot \boldsymbol{\omega_i} = -1\}. \tag{A.6}$$

This set has c_i elements.

The sets A_i, B_i, C_i are related to the sets A_{i+1}, B_{i+1}, C_{i+1} in the following way:

- All of the elements of A_{i+1} can be generated from the elements of A_i by adding a step to the beginning of each. On a d-dimensional rectangular lattice, there are $2d$ possible steps to be added, but one of these will violate the non-reversibility requirement and is forbidden. So, $a_{i+1} = (2d-1)a_i$.

- We can generate one element of B_{i+1} from each element of A_i, with the exception of those elements of A_i which also belong to C_i. This is because a vector parallel to $\boldsymbol{s_N}$ can be added to the beginning of each walk in A_i except for those for which $\boldsymbol{\omega_1} \cdot \boldsymbol{s_N} = -1$ (since we would violate the requirement of non-reversibility). Therefore, $b_{i+1} = a_i - c_i$.

- Similarly, we can generate an element of C_{i+1} from each element of A_i with the exception of those elements which also belong to B_i. Hence, $c_{i+1} = a_i - b_i$.

Now, the expectation value of $\langle \boldsymbol{s_{N-j}} \cdot \boldsymbol{s_N} \rangle$ is just the weighted sum of all the possible values, with the weights being the probabilities of observing particular values:

$$\langle \boldsymbol{s_{N-j}} \cdot \boldsymbol{s_N} \rangle = (1) \cdot \frac{b_j}{a_j} + (-1) \cdot \frac{c_j}{a_j} + (0) \cdot \frac{a_j - b_j - c_j}{a_j} = \frac{b_j - c_j}{a_j}. \tag{A.7}$$

Similarly,

$$\langle \boldsymbol{s_{N-(j+1)}} \cdot \boldsymbol{s_N} \rangle = \frac{b_{(j+1)} - c_{(j+1)}}{a_{(j+1)}} = \frac{(a_j - c_j) - (a_j - b_j)}{(2d-1)a_j} \tag{A.8}$$

$$= \frac{1}{2d-1}\frac{b_j - c_j}{a_j} = \frac{1}{2d-1}\langle \boldsymbol{s_{N-j}} \cdot \boldsymbol{s_N} \rangle. \tag{A.9}$$

Hence, by induction on $\mathbb{N}$,

$$\langle \boldsymbol{s_{N-i}} \cdot \boldsymbol{s_N} \rangle = \left(\frac{1}{2d-1}\right)^i. \tag{A.10}$$

□

A.2 Proof of Lemma 2

Lemma 2 states that the Wolff algorithm for the Ising model satisfies the detailed balance condition if the probability of adding a spin to the cluster, ρ, is given by

$$\rho = 1 - \mathrm{e}^{-2\beta J}.$$

The argument offered below, not very rigorous but hopefully compelling, is based on that of Newman and Barkema (1999, Section 4.2).

The detailed balance condition is given by Equation 3.10, repeated here for convenience:

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

Like in the case of the Metropolis algorithm, we break the transition probability into the selection and acceptance ratios, obtaining

$$\frac{g(\mu \to \nu)\, A(\mu \to \nu)}{g(\nu \to \mu)\, A(\nu \to \mu)} = e^{-\beta(E_\nu - E_\mu)}. \tag{A.11}$$

Let's consider two states μ and ν which differ only by the flip of one cluster.

We will assume that the probability ρ of adding a spin to the cluster takes the same values for all spins aligned with the seed (and zero for all antialigned spins). Then, the probability of adding all the spins making up the cluster to a seed is the same for the cluster that transforms $\mu \to \nu$ and the one that transforms $\nu \to \mu$. The difference lies in the spins which are *not* added, even though they are aligned with the seed. If the cluster formed in state μ has m "bonds" between cluster spins and aligned neighboring

spins, the probability of the cluster forming is proportional to $(1-\rho)^m$. In contrast, the cluster formed in state ν will have n "bonds" between eligible neighboring spins and cluster spins (with $m \neq n$, generally), and the probability of it forming and being flipped will be proportional to $(1-\rho)^n$. So,

$$\frac{g(\mu \to \nu)}{g(\nu \to \mu)} = (1-\rho)^{m-n}, \tag{A.12}$$

and we require

$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = (1-\rho)^{n-m}\, e^{-\beta(E_\nu - E_\mu)}. \tag{A.13}$$

Conveniently, the energy change depends on m and n as well. Every bond between an aligned neighbor and a cluster spin (of which there are m) corresponds to a bond broken in the transition $\mu \to \nu$, and to a consequent energy increase of $\Delta E = +2J$. At the same time, every antialigned neighbor-cluster spin pair (of which there are n) corresponds to a fall in the energy of $\Delta E = -2J$. Therefore,

$$E_\nu - E_\mu = 2J(m-n) \tag{A.14}$$

and

$$\frac{A(\mu \to \nu)}{A(\nu \to \mu)} = \left[e^{2\beta J}\,(1-\rho)\right]^{n-m}. \tag{A.15}$$

At this point, we notice the "delightful fact" (Newman and Barkema, 1999, p. 95) that choosing

$$\rho = 1 - \mathrm{e}^{-2\beta J} \tag{A.16}$$

results in the right hand side of Equation A.15 equaling one. This means that we can construct the cluster by adding aligned neighboring spins with this probability, and once we have attempted to add every such spin, we are allowed to flip the cluster without any further worry. □

A.3 Proof of Lemma 3

Lemma 3 states that the amplitude for a particle to transition from an initial state $|q_I\rangle$ to a final state $|q_F\rangle$ is given by

$$\langle q_I|\, \mathrm{e}^{-\imath HT}\, |q_F\rangle = \int \mathcal{D}q(t)\, \mathrm{e}^{\imath S(q)}, \tag{A.17}$$

where $S(q)$ is the action (time integral of the Lagrangian):

$$S(q) = \int_0^T L(q, \dot{q})\, dt = \int_0^T \frac{1}{2} m\dot{q}^2 - V(q)\, dt, \tag{A.18}$$

and the symbol $\int \mathcal{D}q(t)$ stands for

$$\int \mathcal{D}q(t) \equiv \lim_{N\to\infty} \left(\frac{-\imath 2\pi m}{\delta t}\right)^{N/2} \prod_{j=0}^{N-1} \int dq_j. \tag{A.19}$$

We begin with the expression for the amplitude familiar from the standard formulation of quantum mechanics, $\langle q_I|\, \mathrm{e}^{-\imath HT}\, |q_F\rangle$. We can break up the time T into a large number N of segments of length $\delta t = T/N$, and write

$$\langle q_I|\, \mathrm{e}^{-\imath HT}\, |q_F\rangle = \langle q_I|\, \mathrm{e}^{-\imath H\delta t} \mathrm{e}^{-\imath H\delta t} \cdots \mathrm{e}^{-\imath H\delta t}\, |q_F\rangle\,. \tag{A.20}$$

Since $|q\rangle$ forms a complete set of states, we have $\int\, dq\, |q\rangle\, \langle q| = 1$. We can therefore insert a 1 between every two factors of $\exp(-\imath H\delta t)$, obtaining

$$\langle q_I|\, \mathrm{e}^{-\imath HT}\, |q_F\rangle = \left(\prod_{j=1}^{N-1} \int\, dq_j\right) \langle q_I|\, \mathrm{e}^{-\imath H\delta t}\, |q_{N-1}\rangle\, \langle q_{N-1}|\, \mathrm{e}^{-\imath H\delta t}\, |q_{N-2}\rangle \cdots \langle q_1|\, \mathrm{e}^{-\imath H\delta t}\, |q_F\rangle\,. \tag{A.21}$$

Now, let's focus on one of the terms $\langle q_{j+1}|\, \mathrm{e}^{-\imath H\delta t}\, |q_j\rangle$. We will take the Hamiltonian to be

$$H = \frac{\hat{p}^2}{2m} + V(\hat{q}), \tag{A.22}$$

where $\hat{p}$ is the momentum operator and $\hat{q}$—the position operator. Plugging this back into our bracket,

$$\langle q_{j+1}|\, \mathrm{e}^{-\imath H\delta t}\, |q_j\rangle = \langle q_{j+1}|\, \mathrm{e}^{-\imath \hat{p}^2\delta t/2m}\, \mathrm{e}^{-\imath V(\hat{q})\delta t}\, |q_j\rangle\,. \tag{A.23}$$

We would like to get rid of the operators in the above expression. That's easy in the case of the potential: the ket $|q_j\rangle$ is an eigenket of the position operator, so we can immediately write

$$\langle q_{j+1}|\, \mathrm{e}^{-\imath H\delta t}\, |q_j\rangle = \mathrm{e}^{-\imath V(q_j)\delta t}\, \langle q_{j+1}|\, \mathrm{e}^{-\imath \hat{p}^2\delta t/2m}\, |q_j\rangle\,. \tag{A.24}$$

To do the same with the momentum operator, we will enter a complete set of momentum eigenstates, $\int\, dp\, |p\rangle\, \langle p| = 1$:

$$\langle q_{j+1}|\, \mathrm{e}^{-\imath H\delta t}\, |q_j\rangle = \int\, dp\, \mathrm{e}^{-\imath V(q_j)\delta t}\, \langle q_{j+1}|\, \mathrm{e}^{-\imath \hat{p}^2\delta t/2m}\, |p\rangle\, \langle p|q_j\rangle \tag{A.25}$$

$$= \int\, dp\, \mathrm{e}^{-\imath V(q_j)\delta t}\, \mathrm{e}^{-\imath p^2\delta t/2m}\, \langle q_{j+1}|p\rangle\, \langle p|q_j\rangle\,. \tag{A.26}$$

Now, to get rid of all those kets, we will use the fact that a momentum eigenstate is a plane wave in position space:

$$\langle q|p\rangle = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{\imath pq}. \tag{A.27}$$

It follows that

$$\langle q_{j+1}|\,\mathrm{e}^{-\imath H\delta t}\,|q_j\rangle = \int dp\,\mathrm{e}^{-\imath V(q_j)\delta t}\,\mathrm{e}^{-\imath p^2\delta t/2m}\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{\imath pq_{j+1}}\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-\imath pq} \tag{A.28}$$

$$= \int \frac{dp}{2\pi}\,\mathrm{e}^{-\imath V(q_j)\delta t}\,\mathrm{e}^{-\imath p^2\delta t/2m}\,\mathrm{e}^{\imath p(q_{j+1}-q_j)} \tag{A.29}$$

$$= \mathrm{e}^{-\imath V(q_j)\delta t}\int \frac{dp}{2\pi}\,\mathrm{e}^{-\imath p^2\delta t/2m+\imath p(q_{j+1}-q_j)}. \tag{A.30}$$

If we do the integral and perform a few algebraic manipulations, we get

$$\langle q_{j+1}|\,\mathrm{e}^{-\imath H\delta t}\,|q_j\rangle = \sqrt{\frac{-\imath 2\pi m}{\delta t}}\mathrm{e}^{-\imath V(q_j)\delta t}\,\mathrm{e}^{\imath\delta t(m/2)[(q_{j+1}-q_j)/\delta t]^2}. \tag{A.31}$$

Plugging this into Equation A.21, we obtain

$$\langle q_I|\,\mathrm{e}^{-\imath HT}\,|q_F\rangle = \left(\frac{-\imath 2\pi m}{\delta t}\right)^{N/2}\prod_{j=0}^{N-1}\int dq_j \exp\left(\imath\delta t\frac{m}{2}\sum_{j=0}^{N-1}\left(\frac{q_{j+1}-q_j}{\delta t}\right)^2 - \imath V(q_j)\delta t\right), \tag{A.32}$$

where $q_0 \equiv q_I$ and $q_N \equiv q_F$. We will now take the continuum limit as $N\to\infty$ (which corresponds to $\delta t\to 0$). To do so, we replace $[(q_{j+1}-q_j)/\delta t]^2$ with $\dot{q}^2$ and $\delta t\sum_{j=0}^{N-1}$ with $\int_0^T dt$. The result is,

$$\langle q_I|\,\mathrm{e}^{-\imath HT}\,|q_F\rangle = \lim_{N\to\infty}\left(\frac{-\imath 2\pi m}{\delta t}\right)^{N/2}\prod_{j=0}^{N-1}\int dq_j \exp\left(\imath\int_0^T dt\,\frac{1}{2}m\dot{q}^2 - V(q)\right). \tag{A.33}$$

It is customary to denote the integral over paths using the following symbol:

$$\int \mathcal{D}q(t) \equiv \lim_{N\to\infty}\left(\frac{-\imath 2\pi m}{\delta t}\right)^{N/2}\prod_{j=0}^{N-1}\int dq_j. \tag{A.34}$$

This allows us to simplify Equation A.33 to

$$\langle q_I|\,\mathrm{e}^{-\imath HT}\,|q_F\rangle = \int \mathcal{D}q(t)\exp\left(\imath\int_0^T \frac{1}{2}m\dot{q}^2 - V(q)\,dt\right). \tag{A.35}$$

An interesting feature of this equation is that the term in the exponent is just the classical action—the time integral of the Lagrangian. It turns out that this holds in general, and we may write,

$$\langle q_I | \, \mathrm{e}^{-\imath HT} \, | q_F \rangle = \int \mathcal{D}q(t) \mathrm{e}^{\imath \int_0^T L(q,\dot{q}),dt} = \int \mathcal{D}q(t) \mathrm{e}^{\imath S(q)}, \tag{A.36}$$

where $L(q, \dot{q})$ is the Lagrangian of the particle. □

The foregoing treatment was based on Zee (2003, Chapter I.2) and Townsend (2000, Chapter 8).

A.4 Proof of Lemma 4

Lemma 4 concerns the ϕ^4 theory with the energy given by Equation 4.38, repeated here for convenience:

$$\beta E = \frac{1}{2} \sum_{\langle i,\, j \rangle} (\phi_i - \phi_j)^2 + \sum_i \left(\frac{1}{2} \mu_0^2 \phi_i^2 + \frac{\lambda}{4} \phi_i^4 \right)$$

Define $s_n = \phi_n / |\phi_n|$. The claim is that the naïve Wolff algorithm for a model with the preceding lattice action will satisfy detailed balance if the probability of adding a spin to the cluster, ρ, is given by

$$\rho = \begin{cases} 1 - \mathrm{e}^{-2\phi(o)\phi(c)} & \text{if } s_o = s_c, \\ 0 & \text{otherwise,} \end{cases}$$

where $\phi(c)$ is a cluster spin and $\phi(o)$ is its neighbor being added to the cluster. The argument we use to prove it below is inspired by that of Schaich (2006, section 7.3).

Recall from the previous proof the detailed balance condition, Equation A.11:

$$\frac{g(\mu \to \nu)\, A(\mu \to \nu)}{g(\nu \to \mu)\, A(\nu \to \mu)} = e^{-\beta(E_\nu - E_\mu)}. \tag{A.37}$$

Again, we assume that the states μ and ν differ only by the flip of one cluster. Say the cluster formed in state μ has m "bonds" between cluster spins and aligned neighbor spins (there are m spins which could have been added to the cluster with nonzero probability, but were not), and similarly, the cluster in state ν has n such bonds. The ratio of the selection probabilities is then

$$\frac{g(\mu \to \nu)}{g(\nu \to \mu)} = \frac{\prod_{i=1}^m (1 - \rho_i)}{\prod_{i=1}^n (1 - \rho_i)}. \tag{A.38}$$

In the discussion of the Ising Wolff algorithm, we were able to simplify this expression to Equation A.12 by assuming that ρ_i takes on the same value for every aligned neighboring spin. Unfortunately, this is not longer the case in the ϕ^4 model. To make progress, we need to expand the right hand side of Equation A.37. Since the lengths of all spins remain unchanged in the Wolff flip, the terms depending on the second and fourth power of ϕ do not contribute to the energy difference. The spins in the interior of the cluster do not contribute to the energy change at all, since both they and their neighbors are flipped, and $(a-b)^2 = (-a+b)^2$. Therefore, the energy change is entriely due to the $m+n$ bonds between spins on the edge of the cluster and their neighbors outside the cluster. Denote the spins constituting bond i as $\phi(c)_i$ (the cluster spin) $\phi(o)_i$ (the neighbor). The only difference between the states μ and ν comes from the flipping of the cluster, so that $\phi(c)_i \to -\phi(c)_i$. Therefore,

$$\beta E_\nu - \beta E_m u = \frac{1}{2}\sum_{i=1}^{n+m}\left[(\phi(o)_i + \phi(c)_i)^2 - (\phi(o)_i - \phi(c)_i)^2\right] \tag{A.39}$$

$$= \frac{1}{2}\sum_{i=1}^{n+m} \phi(o)_i^2 + 2\phi(o)_i\phi(c)_i + \phi(c)_i^2 - \phi(o)_i^2 + 2\phi(o)_i\phi(c)_i - \phi(c)_i^2 \tag{A.40}$$

$$= \sum_{i=1}^{n+m} 2\phi(o)_i\phi(c)_i. \tag{A.41}$$

We plug this result and Equation A.38 into Equation A.37 to obtain

$$\frac{A(\mu\to\nu)}{A(\nu\to\mu)} = \frac{\prod_{i=1}^{n}(1-\rho_i)}{\prod_{i=1}^{m}(1-\rho_i)} \exp\left(-\sum_{i=1}^{n+m} 2\phi(o)_i\phi(c)_i\right) \tag{A.42}$$

$$= \frac{\prod_{i=1}^{n}(1-\rho_i)}{\prod_{i=1}^{m}(1-\rho_i)} \prod_{i=1}^{n+m} \exp\left(-2\phi(o)_i\phi(c)_i\right). \tag{A.43}$$

But now, notice that the product $\phi(o)_i\phi(c)_i$ is always positive for the m bonds broken in the transition $\mu \to \nu$, and always negative for the n bonds created. We may therefore write,

$$\frac{A(\mu\to\nu)}{A(\nu\to\mu)} = \frac{\prod_{i=1}^{n}(1-\rho_i)}{\prod_{j=1}^{m}(1-\rho_j)} \left(\prod_{i=1}^{n} \exp 2\phi(o)_i\phi(c)_i\right)\left(\prod_{j=1}^{m} \exp -2\phi(o)_j\phi(c)_j\right). \tag{A.44}$$

As in the case of the Ising model, it is convenient to set the acceptance ratio to unity. We can then rewrite the previous equation as

$$\prod_{j=1}^{m}(1-\rho_j)\exp(2\phi(o)_j\phi(c)_j) = \prod_{i=1}^{n}(1-\rho_i)\exp(2\phi(o)_i\phi(c)_i). \tag{A.45}$$

It is easy to see that this condition is satisfied if we choose

$$\rho_i = 1 - \exp\left(-2\phi(o)_i\phi(c)_i\right) \quad \forall\, i. \tag{A.46}$$

□

Appendix B

The Works

This chapter contains technical details of the computations carried out in the course of this thesis.

B.1 Programming Languages and Libraries

B.1.1 Programming Languages Used

At the onset of this project, I knew the basics of Mathematica, but had no programming experience; therefore, I had the freedom to choose the language and assorted tools best suited to my needs. The two previous Amherst theses in this area used Java (Bednarzyk, 2001) and C++ (Schaich, 2006), but both authors confessed to have been influenced by previous familiarity in making their choice. I wrote some early programs in Python, C and C++, but settled on the last one due to its popularity and widespread reputation for (machine, if not programming) speed. Popularity was important, because it meant that many good tutorials and books devoted to teaching C++ were easy to find, as were numerous helpful function libraries such as GSL or Fog's RandomC; furthermore, some general scientific computing books, such as Press et al. (2007) and Karniadakis and Kirby (2003), use C++ as the language of their examples. Speed mattered, too, since I were to eventually perform large simulations. Consequently, all of the code used in the course of this thesis was written either in C++ or in Mathematica.

My choice of programming aids was strongly influenced by Schaich's *Appendix B: Efficient Programming*, which I wholeheartedly recommend to anyone planning work in computer simulations. Debuggers, code profil-

ers and automated builds (Makefiles) made coding the necessary programs significantly easier. I departed from his recommendations in two respects, however:

1. I *did not* use an automated version control system. Such programs, which keep track and attempt to reconcile changed made to a piece of code by different programmers over time, could indeed be invaluable for a larger project, involving multiple researchers. My own work involved relatively small programs and no collaboration; therefore, automated version control seemed superfluous.

2. I *did* take advantage of an Integrated Development Environment (IDE). In this, I followed the recommendation of the original source of much of Schaich's advice, Wilson (2006). This choice was entirely a matter of convenience: I found learning how to use an IDE easier than mastering the traditional ("ancient," in Wilson's words) alternatives of emacs and gdb. The IDE I used for most of my work was Code::Blocks 8.02,[1] a dedicated C++ environment. I also experimented with Eclipse, a cross-platform IDE popular with Java developers. However, the documentation for its C++ functionality was too scant for a beginner.

B.1.2 Random Number Generators

The generation of random numbers is a "mature" field, meaning that few people who need high-quality random numbers will write their own generators. Neither have I. In early programs, I used the simple but solid Ran2 generator described in Press et al. (2007, p. 342). All of the ϕ_2^4 theory simulations, however, were carried out using Agner Fog's implementation[2] of the Mersenne Twister algorithm. The algorithm was designed by Makoto Matsumoto and Takuji Nishimura specifically for Monte Carlo applications; it's characterized by a very long period and good randomness (see Matsamuto and Nishimura 1998 for a detailed description).

The reader interested in the topic of random number generation is referred to the excellent introduction in Press et al. (2007, chapter 7).

[1] As of this writing, available at `http://www.codeblocks.org/`.

[2] The implementation and its documentation can be found on the author's website, `http://www.agner.org/random/`.

B.2 Code Snippets

B.2.1 Estimating π

The following C++ program will produce an estimate of π using a simple Monte Carlo technique discussed in Section 2.1:

```
   ,
#include<iostream> // Necessary for producing input, output

using namespace std;

int main() {

  Ran random(12356);
  unsigned int InCircle = 0;
  unsigned int OutOfCircle = 0;

  cout << "Number of iterations?" << endl;
  long unsigned int Iterations;
  cin >> Iterations;

  for(long unsigned int i = 0; i < Iterations; ++i) {
    double x = random.doub();
    double y = random.doub();
    if( x*x + y*y < 1 ) {
      ++InCircle;
    } else {
      ++OutOfCircle;
    }
  }

  double Pi = 4 * InCircle / (OutOfCircle + InCircle);

  cout << "Best estimate after " << Iterations << " iterations is "
       << Pi << endl;

  return 0;
}
```

B.2.2 Simple Random Walk

The program below was used to simulate simple random walks in three dimensions. Analogous programs were used for one- and two-dimensional walks.

```
   ,
// Tadeusz Pudlik
```

```
// 07/13/08
// 3D random walk to file.

#include "nr3.h"   // For Ran2, the Numerical Recipes random
#include "ran.h"   // number generator.
#include<iostream> // to get N, P, Trials
#include<iomanip>  // for output formatting
#include<fstream>  // to produce output file
#include<cmath>    // for sqrt()

using namespace std;

unsigned long int GetTrials(); unsigned int GetStepNo();

int main() {
  unsigned int N = GetStepNo();
  unsigned long int Trials = GetTrials();

  // Initialize random number generator
  Ran random(73881); // Arbitrarily chosen seed

  // Output file initialization
  ofstream fout("3DWalk.txt");
  fout << "Step number " << " Avg Disp Sq \n" << flush;

  double direction; // to be randomized
  long int x; // displacement of walker in the x direction
  long int y; // displacement of walker in the y direction
  long int z; // displacement of walker in the z direction
  long int AvgSq; // average displacement squared

  for(unsigned int k = 1; k <= N; ++k) { // Master loop
    AvgSq = 0;
    for (unsigned long int j = 0; j < Trials; ++j) {// run trials...
      x = 0; y = 0; z = 0;
      for (unsigned int i = 0; i < k; ++i) {// walk the walk...
        direction = random.doub();
        if      (direction < 1.0/6) ++x;
        else if (direction < 1.0/3) --x;
        else if (direction < 0.5) ++y;
        else if (direction < 2.0/3) --y;
        else if (direction < 5.0/6) ++z;
        else --z;
      }
      AvgSq += x*x + y*y + z*z;
    }
    double fAvgSq = AvgSq*1.0/Trials;

    //Output
```

```
    fout << setw(11) << k
         << setw(13) << fAvgSq
         << "\n" << flush;
  }

  cout << "Done!" << endl;

  return 0;
}

unsigned long int GetTrials() {
  unsigned long int t;
  cout << "How many trials per walk length you wish to run? " << flush;;
  cin >> t;
  return t;
}

unsigned int GetStepNo() {
  unsigned int n;
  cout << "How long would you like the longest walk to be? " << flush;
  cin >> n;
  return n;
}
```

B.2.3 Non-Reversing Random Walk

The code below was used to generate samples of NRRWs in $d = 3$. The code for $d = 2$ was a simple modification of the one shown below.

```
   ,
/*
Ted Pudlik
03/26/09

This program generates a sample of non-reversing random walks
in a range of lengths.  Its output is the mean end-to-end
distance squared, <R_N^2>, and the standard deviation of
this mean.  It's an improvement on my early NRRW programs,
written in the summer, which didn't record the standard
deviation.
*/

#include<iostream> // to get N, P, Trials
#include<iomanip> // for output formatting
#include<fstream> // to produce output file
#include<cmath> // for sqrt()
#include "randomc.h"
```

```
using namespace std;

unsigned long int GetTrials();
unsigned int GetStepNo();

int main() {
  int N = GetStepNo();              // Largest length of a walk
  long int Trials = GetTrials();    // Trials per length

  // Initialize random number generator
  CRandomMersenne RanGen(time(NULL));

  // Output file initialization
  ofstream fout("NRRW.txt");
  fout << "Step No " << " Avg Disp Sq " << " Var \n" << flush;

  int prevdir = 7;
  int dir;    // to be randomized
  long int x; // displacement of walker in the x direction
  long int y; // displacement of walker in the y direction
  long int z; // displacement of walker in the z direction
  double Product; // to speed up the calculation of the two subsequent vars
  double AvgSq; // average displacement squared
  double AvgSqSq; // the square of average disp sq (for calculating sigma)

  // Run simulation
  for(int k = 1; k <= N; ++k) { // Master loop
    AvgSq = 0;
    AvgSqSq = 0;
    for (long int j = 0; j < Trials; ++j) {// run trials...
      x = 0; y = 0; z = 0;                 // Clear variables
      for (int i = 0; i < k; ++i) {        // walk the walk...
            dir = RanGen.IRandom(0,5);
            while(prevdir == dir){dir = RanGen.IRandom(0,5);}
            switch(dir){
            case 0: ++x; break;
            case 1: ++y; break;
            case 2: ++z; break;
            case 3: --x; break;
            case 4: --y; break;
            case 5: --z; break;
            default: cerr << "Central direction switch failure!" << endl;
            }
            prevdir = (dir+3)%6; // reversing previous step forbidden
      } // end of walk loop
      Product = x*x + y*y + z*z;
      AvgSq += Product;
      AvgSqSq += Product*Product;
    } // end of trials loop
```

```
    AvgSq /= Trials;
    AvgSqSq /= Trials;
    // Write output to file:
    fout << setw(7) << k
         << setw(11) << AvgSq
         << setw(10) << sqrt(AvgSqSq - AvgSq*AvgSq)
         << "\n" << flush;
  } // end of master loop

  cout << "Done!" << endl;
  return 0;
}

unsigned long int GetTrials() {
  unsigned long int t;
  cout << "How many trials per walk do you wish to run? " << flush;;
  cin >> t;
  return t;
}

unsigned int GetStepNo() {
  unsigned int n;
  cout << "How long would you like the longest walk to be? " << flush;
  cin >> n;
  return n;
}
```

B.2.4 Ising Model: Metropolis Algorithm

The code below was used in our first simulation of the Ising model, described in Section 3.3.

```
   ,
/*
 * Metropolis.cpp
 *
 *  Created on: Oct 12th, 2008
 *      Author: Tadeusz Pudlik
 *
 * This program is intended to run the Metropolis algorithm,
 * as described in Schaich or Newman.  The output
 * is the complete state of the model after time T,
 * produced as a text file "Metro_Raster.txt", as well as
 * 10000 evenly-spaced measurements of the energy and
 * magnetization, "Metro_EM.txt.". External mag field is
 * assumed to be zero.
 */
```

```
#include "nr3.h" // for random # generator
#include "ran.h" // for random # generator
#include<iostream> // to get N, P, Trials
#include<iomanip> // for output formatting
#include<fstream> // to produce output file
#include<cmath> // for sqrt()

using namespace std;

long int GetTime(int x);
double GetBeta();
bool GetInitialConditions();
int GetSeed();

int main() {
    cout << "Welcome to Metropolis2!" << endl;
    int J = 1;
    double beta = GetBeta();
    int N = 200;
    int LatticeSize = N*N;
    long int T = GetTime(LatticeSize);
    bool Initial = GetInitialConditions();
    int Seed = GetSeed();
    double E = 0; // Energy
    double M = 0; // Magnetization

    // Initialize random number generator
    Ran random(Seed); // Arbitrarily chosen seed

    // Output file initialization
    ofstream fout("Metro_Raster.txt");
    ofstream fout2("Metro_EM.txt");

    // Lattice initialization
    int history[N][N];
    if (Initial == true){// Randomize starting spin configuration...
        for(int i = 0; i < N; ++i) {
            for(int j = 0; j < N; ++j) {
                if(random.doub() > 0.5){
                    history[i][j] = 1;
                    M += 1;
                } else {
                    history[i][j] = -1;
                    M -= 1;
                }
            }
        }
    } else {// ... or set all spins to spin-up, depending on user's choice
        for (int i = 0; i < N; ++i) {
```

```
        for(int j = 0; j < N; ++j) {
            history[i][j] = 1;
        }
    }
    M = LatticeSize;
}

// Prepare for running algorithm
int xn; int xp; int yn; int yp; // the positions of the
                                   surrounding sites
int n; // number of neighboring spins with the same orientation
double p; // the probability of the spin being flipped
double t; // random numer to compare p with
long int RecordingInterval = T/10000;
// Get the exponentials:
double expm2 = exp(-beta*4*J);
double expm4 = exp(-beta*8*J);

// Calculate energy and energy squared
for(int i = 0; i < N; ++i){
    for(int j = 0; j < N; ++j){
        //Enforce boundary conditions:
        if(i+1 == N){xn = 0;} else {xn = i+1;}
        if(i == 0){xp = N-1;} else {xp = i-1;}
        if(j+1 == N){yn = 0;} else {yn = j+1;}
        if(j == 0){yp = N-1;} else {yp = j-1;}

        //Get the energy--n used for clarity
        n = history[xn][j] + history[i][yn]
          + history[xp][j] + history[i][yp];
        E += -J*history[i][j]*n;
    }
}

//Run the algorithm
for(long int i = 0; i < T; ++i) {
    n = 0;
    long int x = random.int64()%N;
    long int y = random.int64()%N;

    // Enforce periodic boundary conditions
    if(x+1 == N){xn = 0;} else {xn = x+1;}
    if(x == 0){xp = N-1;} else {xp = x-1;}
    if(y+1 == N){yn = 0;} else {yn = y+1;}
    if(y == 0){yp = N-1;} else {yp = y-1;}

    // Calculate m
    if(history[xn][y] == history[x][y]){n++;}
    if(history[xp][y] == history[x][y]){n++;}
```

```
        if(history[x][yn] == history[x][y]){n++;}
        if(history[x][yp] == history[x][y]){n++;}

        // Use m to get the probability of a transition
        if(n <= 2){p = 1;} else
        if(n == 3){p = expm2;} else
        if(n == 4){p = expm4;}


        // Do the transition with specified probability
        t = random.doub();
        if(p >= t){
            E += 4*J*(n-2);
            history[x][y] = -history[x][y];
                if(history[x][y] == 1){
                    M += 2;
                } else if(history[x][y] == -1){
                    M -= 2;
                }
        }

        /* Write output to file (since Excel couldn't deal with
        too large a dataset anyway, I'll only record 10000 values*/

        if(i\% RecordingInterval == 0){
                fout2 << setw(10) << i/LatticeSize
                      << setw(10) << E/LatticeSize
                      << setw(10) << M/LatticeSize
                      << "\n" << flush;
        }
    } // End of Metropolis loop
    fout2.close(); // Close data output file

    // Write the output to file
    for(int i = 0; i < N; ++i){
        for(int j = 0; j < N; ++j){
                    fout << setw(3) << history[i][j] << flush;
        }
        fout << "\n" << flush;
    }
    fout.close(); // Close raster output file

    return 0;
    }

long int GetTime(int x) {
    long int t;
    cout << "How many iterations per lattice site "
         << "would you like the algorithm to perform? " << endl;
```

```
    cin >> t;
    return t*x;
}

double GetBeta() {
    double t;
    cout << "How high would you like the temperature to be? "
         << flush;
    cin >> t;
    if(t != 0){return 1/t;} else {return 1E37;}
}

bool GetInitialConditions() {
    char x;
    cout << "Do you want the initial lattice to be randomized (R) "
         << "or all spin-up (S)? "
         << flush;
    cin >> x;
    if(x == 'R'){return true;} else if(x == 'S'){return false;} else {
        cerr << "Improper input! Should be either R or S!" << flush;
        return 1;
    }
}

int GetSeed(){
    int s;
    cout << "Please set the seed for the "
         << "random number generator: " << flush;
    cin >> s;
    return s;
}
```

B.2.5 Linked List Class

The linked list class consists of a header file and a cpp file. It's a basic data structure: its advantages are ease of coding and fast deletion, but it's time-consuming to search. This implementation is based on Parlante (2001).

The header file ,

```
/*
This program implements a simple linked-list class.

Ted Pudlik
10/26/08
 */
```

```
#include<iostream>
#include<sstream>

using namespace std;

struct Node {
  int XValue;
  int YValue;
  struct Node* Next;
};


class FIFO {
public:
  FIFO();
  Node* Head;

  void Push(int x, int y);
  int GetTopX();
  int GetTopY();
  void ScrapTopNode();
  bool Search(int x, int y);
  int Count();
private:
  int Size;
};
```

The cpp file ,

```
#include "LL.h"

FIFO::FIFO()
: Size(0), Head(NULL)
{
}

int FIFO::Count(){
    struct Node* current = Head;
    int count = 0;

    while (current != NULL) {
        count++;
        current = current->Next;
    }
    return count;
}

void FIFO::Push(int x, int y){
```

```
    struct Node *Sample = new Node;
    Sample->XValue = x;
    Sample->YValue = y;
    Sample->Next = Head;
    Head = Sample;
    Size++;
}

int FIFO::GetTopX(){
    return Head->XValue;
}

int FIFO::GetTopY(){
    return Head->YValue;
}

void FIFO::ScrapTopNode(){
    struct Node *Temp = Head->Next;
    delete Head;
    Head = Temp;
}

bool FIFO::Search(int x, int y){
    struct Node* current = Head;
    while (current != NULL) {
        if(current->XValue == x && current->YValue == y){return true;}
        current = current->Next;
    }
    return false;
}
```

B.2.6 Ising Model: Wolff Algorithm

This program is a cluster version of the Ising model program that uses a mix of the Metropolis and Wolff algorithms. It takes advantage of the linked list class introduced in Section B.2.5. It was used to produce the data for Figures 3.9(a) through 3.10(b).

```
   ,
/*
 *  Created on: Apr 5th, 2008
 *      Author: Tadeusz Pudlik
 */

#include<iostream>
#include<iomanip> // for output formatting
#include<fstream> // to produce output file
```

```
#include<cmath> // for sqrt(), fabs()
#include<ctime> // for seeding the random number generator
#include "randomc.h" // for random number generator
#include "LL.h"

using namespace std;

int main(unsigned int argc, char** const argv) {
    //****************************************
    // COMMAND-LINE PARAMETERS
    if(argc != 6){cerr << "Wrong number of command line parameters"
                       << endl; return 1;}
    double beta = atof(argv[1])/(atof(argv[2]) +atof(argv[3])); // inv temp
    int N = atof(argv[4]);        // Height and width of the lattice
    long int T = atof(argv[5]);   // Number of sweeps to be performed

    // MODEL PARAMETERS
    int MetroWolffRatio = 4;
    int LatticeSize = N*N;
    int J = 1;
    double E = 0; // Energy
    double M = 0; // Magnetization
    //****************************************

    // Initialize random number generator
    CRandomMersenne RanGen(time(NULL));

    // Lattice initialization
    int Lattice[N][N];
    if (true){// Randomize starting spin configuration...
        for(int i = 0; i < N; ++i) {
            for(int j = 0; j < N; ++j) {
                if(RanGen.Random() > 0.5){
                        Lattice[i][j] = 1;
                        M += 1;
                } else {
                        Lattice[i][j] = -1;
                        M -= 1;
                }
            }
        }
    }

    // Prepare for running algorithm
    int xn; int xp; int yn; int yp; // the positions of the surrounding sites
    int n; // number of neighboring spins with the same orientation
    double p = 0; // the probability of the spin being flipped
    double pwolff = 1 - exp(-2*beta); // the probability of a spin being added
                                      // to the cluster
```

```
 double t; // random numer to compare p with
// Get the exponentials:
double expm2 = exp(-beta*4*J);
double expm4 = exp(-beta*8*J);
FIFO Pocket;
FIFO Cluster;
bool ClusterArray[N][N];
 for (int i = 0; i < N; ++i) {
     for(int j = 0; j < N; ++j) {
         ClusterArray[i][j] = false;
     }
 }

 //Run the algorithm
 for(int i = 0; i < T; ++i) {

 // Do the appropriate number of Metropolis sweeps
 for(int counter = 0; counter < MetroWolffRatio*LatticeSize; ++counter){

         n = 0;
         long int x = RanGen.IRandom(0,N-1);
         long int y = RanGen.IRandom(0,N-1);

         // Enforce periodic boundary conditions
         if(x+1 == N){xn = 0;} else {xn = x+1;}
         if(x == 0){xp = N-1;} else {xp = x-1;}
         if(y+1 == N){yn = 0;} else {yn = y+1;}
         if(y == 0){yp = N-1;} else {yp = y-1;}

         // Calculate m
         if(Lattice[xn][y] == Lattice[x][y]){n++;}
         if(Lattice[xp][y] == Lattice[x][y]){n++;}
         if(Lattice[x][yn] == Lattice[x][y]){n++;}
         if(Lattice[x][yp] == Lattice[x][y]){n++;}

         // Use m to get the probability of a transition
         if(n <= 2){p = 1;} else
         if(n == 3){p = expm2;} else
         if(n == 4){p = expm4;}

         // Do the transition with specified probability
         t = RanGen.Random();
         if(p >= t){
             E += 4*J*(n-2);
             Lattice[x][y] = -Lattice[x][y];
             if(Lattice[x][y] == 1){
                 M += 2;
             } else if(Lattice[x][y] == -1){
                 M -= 2;
```

```
            }
        }
}

 // Do a Wolff cluster flip
    n = 0;
    long int x = RanGen.IRandom(0,N-1);
    long int y = RanGen.IRandom(0,N-1);
    ClusterArray[x][y] = true;
    Cluster.Push(x,y);
    Pocket.Push(x,y);

    // Build cluster
    for(int PocketSize = 1; PocketSize > 0; --PocketSize) {
        /*Here, I'd like to withdraw spins one by one and examine their
          neighbours, if they're not in the cluster already*/
        x = Pocket.GetTopX();
        y = Pocket.GetTopY();
        Pocket.ScrapTopNode();

        // Enforce boundary conditions
        if(x+1 == N){xn = 0;} else {xn = x+1;}
        if(x == 0){xp = N-1;} else {xp = x-1;}
        if(y+1 == N){yn = 0;} else {yn = y+1;}
        if(y == 0){yp = N-1;} else {yp = y-1;}

        // Examine and potentially add neighbors
        if(   Lattice[xn][y]==Lattice[x][y]
           && ClusterArray[xn][y] == false
           && RanGen.Random() < pwolff)
          {Pocket.Push(xn,y); ++PocketSize;
           Cluster.Push(xn,y); ClusterArray[xn][y] = true;}
        if(   Lattice[xp][y]==Lattice[x][y]
           && ClusterArray[xp][y] == false
           && RanGen.Random() < pwolff)
          {Pocket.Push(xp,y); ++PocketSize;
           Cluster.Push(xp,y); ClusterArray[xp][y] = true;}
        if(   Lattice[x][yp]==Lattice[x][y]
           && ClusterArray[x][yp] == false
           && RanGen.Random() < pwolff)
          {Pocket.Push(x,yp); ++PocketSize;
           Cluster.Push(x,yp); ClusterArray[x][yp] = true;}
        if(   Lattice[x][yn]==Lattice[x][y]
           && ClusterArray[x][yn] == false
           && RanGen.Random() < pwolff)
          {Pocket.Push(x,yn); ++PocketSize;
           Cluster.Push(x,yn); ClusterArray[x][yn] = true;}
    }
```

```
        // Flip the cluster (and calculate magnetization on the run)
        for(int j = Cluster.Count(); j > 0; --j){
            x = Cluster.GetTopX();
            y = Cluster.GetTopY();
            Cluster.ScrapTopNode();
            ClusterArray[x][y] = false;

            Lattice[x][y] = -Lattice[x][y];
            M += 2*Lattice[x][y];
        }

        E = 0;
        // Calculate the energy
        // (difficult to do on the run because of Wolff flip)
        for(int k = 0; k < N; ++k){
            for(int j = 0; j < N; ++j){
             //Enforce boundary conditions:
             if(k+1 == N){xn = 0;} else {xn = k+1;}
             if(k == 0){xp = N-1;} else {xp = k-1;}
             if(j+1 == N){yn = 0;} else {yn = j+1;}
             if(j == 0){yp = N-1;} else {yp = j-1;}

             //Get the energy--n used for clarity
             n = Lattice[xn][j] + Lattice[k][yn]
               + Lattice[xp][j] + Lattice[k][yp];
             E += -J*Lattice[k][j]*n/2;
            }
        }

        // Output results
        if(i >= T/2){
            cout << setw(25) << i << setw(25) << 1/beta
                 << setw(25) << E << setw(25) << M/LatticeSize << endl;
        }
}


        return 0;
}
```

B.2.7 ϕ^4 Theory: Metropolis-Wolff

This program is very similar to that of Appendix B.2.6, except for the substitution of a continuous spin for the discrete one. It uses the linked list class introduced in Section B.2.5.

```
   ,
/*
```

```
Phi 4 MetroWolff 2 Cout main file

Author: Tadeusz Pudlik
Date: February 9th, 2009

This program is designed to run the mixed
Wolff-Metropolis algorithm for the Phi 4 model
on the Amherst cluster.  It uses Cout as its
only output command.
*/


#include<iostream>
#include<iomanip> // for output formatting
#include<fstream> // to produce output file
#include<cmath> // for sqrt(), fabs()
#include<ctime> // for seeding the random number generator
#include "LL.h" // for linked list class
#include "randomc.h" // for random number generator

using namespace std;

void EnforceBoundaryConditions(int N, int x, int y,
                               int & xn, int & yn, int & xp, int & yp);

int main(unsigned int argc, char** const argv) {
    //*************************************************************************
    // USER-EDITABLE PARAMETERS
    int MetroWolffRatio = 4;    // Number of Metropolis lattice sweeps
                               // per Wolff flip
    double StepRange = 2.5;     // Largest possible Metropolis step

    // COMMAND-LINE PARAMETERS
    if(argc != 7){cerr << "Wrong number of command line parameters"
                       << endl; return 1;}
    double musquared = atof(argv[1])/atof(argv[2]) - atof(argv[3]);
    double lambda = atof(argv[4]);      // lambda
    int N = atof(argv[5]);              // Height and width of the lattice
    long int T = atof(argv[6]);          // Number of iterations
                                        // to be performed
//*************************************************************************

    // Initialize lattice
    int LatticeSize = N*N;  // Numer of sites in the lattice
    double E = 0;           // Energy
    double M = 0;           // Magnetization
    double SqAvgM = 0;      // Square of the average of the magnetization
                            //   (for susceptibility measurements)
    double AvgSqM = 0;      // Average of the square of the magnetization
```

```
                            //   (for susceptibility measurements)
    double Avg4M = 0;       // Average of the fourth power of the magnetization
                            //   (for susceptibility measurements)
    double Lattice[N][N];
    for (int i = 0; i < N; ++i) {
        for(int j = 0; j < N; ++j) {
            Lattice[i][j] = 0;
        }
    }

    // Initialize random number generator
    CRandomMersenne RanGen(time(NULL));

    // Prepare for running algorithm---declare variables used throughout
    int xn; int xp; int yn; int yp; // the positions of the surrounding sites
    FIFO Pocket;                    // Linked list
    FIFO ClusterList;               // Wolff cluster will be stored here...
    bool Cluster[N][N];             // ... as well as here, to improve
                                    //searchability at little memory cost
    for (int i = 0; i < N; ++i) {
        for(int j = 0; j < N; ++j) {
            Cluster[i][j] = false;
        }
}
    double squared;                 // Used in computing energy changes
    double quartic;                 // Used in computing energy changes
    double interaction;             // Used in computing energy changes
    double p;                       // Represents probability of transition
                                    // in Metropolis

    //Run the algorithm
    for(int i = 0; i < T; ++i) {

        // Run an appropriate number of Metropolis sweeps
        for(int counter = 0; counter < MetroWolffRatio*LatticeSize; ++counter){
           long int x = RanGen.IRandom(0,N-1);
           long int y = RanGen.IRandom(0,N-1);
           double step = (RanGen.Random()-0.5)*2*StepRange;
           double OldSpin = Lattice[x][y];
           double NewSpin = OldSpin + step;

           EnforceBoundaryConditions(N,x,y,xn,yn,xp,yp);

           // Get the energy change, step-by-step for clarity
           squared = (2+0.5*musquared)*(NewSpin*NewSpin - OldSpin*OldSpin);
           quartic = 0.25*lambda*(pow(NewSpin,4)-pow(OldSpin,4));
           interaction = -step*(Lattice[x][yn] + Lattice[x][yp] +
                        Lattice[xn][y]+Lattice[xp][y]);
           double DeltaE = squared + quartic + interaction;
```

```
   // Use DeltaE to get the probability of a transition
   if(DeltaE < 0){p = 1;} else p = exp(-DeltaE);

   // Do the transition with specified probability
   if(p == 1 || p >= RanGen.Random()){
      Lattice[x][y] = NewSpin;
      M += NewSpin-OldSpin;
   }
}

// Do a Wolff cluster flip
long int x = RanGen.IRandom(0,N-1);
long int y = RanGen.IRandom(0,N-1);

EnforceBoundaryConditions(N,x,y,xn,yn,xp,yp);

// Examine and potentially add neighboring spins
if(   Lattice[xn][y]*Lattice[x][y] > 0
   && RanGen.Random() < 1 - exp(-2*Lattice[xn][y]*Lattice[x][y]) )
  {Pocket.Push(xn,y); ClusterList.Push(xn,y); Cluster[xn][y] = true;}
if(   Lattice[xp][y]*Lattice[x][y] > 0
   && RanGen.Random() < 1 - exp(-2*Lattice[xp][y]*Lattice[x][y]) )
  {Pocket.Push(xp,y); ClusterList.Push(xp,y); Cluster[xp][y] = true;}
if(   Lattice[x][yp]*Lattice[x][y] > 0
   && RanGen.Random() < 1 - exp(-2*Lattice[x][yp]*Lattice[x][y]) )
  {Pocket.Push(x,yp); ClusterList.Push(x,yp); Cluster[x][yp] = true;}
if(   Lattice[x][yn]*Lattice[x][y] > 0
   && RanGen.Random() < 1 - exp(-2*Lattice[x][yn]*Lattice[x][y]) )
  {Pocket.Push(x,yn); ClusterList.Push(x,yn); Cluster[x][yn] = true;}

// Build cluster
for(int PocketSize = Pocket.Count(); PocketSize > 0; --PocketSize) {
    /*Here, I'd like to withdraw spins one by one and examine
      their neighbours, if they're not in the cluster already*/
    x = Pocket.GetTopX();
    y = Pocket.GetTopY();
    Pocket.ScrapTopNode();

    EnforceBoundaryConditions(N,x,y,xn,yn,xp,yp);

    // Examine and potentially add neighbors
    if(   Cluster[xn][y] == false
       && Lattice[xn][y]*Lattice[x][y] > 0
       && RanGen.Random() < 1 - exp(-2*Lattice[xn][y]*Lattice[x][y]))
      {Pocket.Push(xn,y); ++PocketSize; ClusterList.Push(xn,y);
       Cluster[xn][y] = true;}
    if(   Cluster[xp][y] == false
       && Lattice[xp][y]*Lattice[x][y] > 0
```

```
       && RanGen.Random() < 1 - exp(-2*Lattice[xp][y]*Lattice[x][y]))
      {Pocket.Push(xp,y); ++PocketSize; ClusterList.Push(xp,y);
       Cluster[xp][y] = true;}
    if(   Cluster[x][yp] == false
       && Lattice[x][yp]*Lattice[x][y] > 0
       && RanGen.Random() < 1 - exp(-2*Lattice[x][yp]*Lattice[x][y]))
      {Pocket.Push(x,yp); ++PocketSize; ClusterList.Push(x,yp);
       Cluster[x][yp] = true;}
    if(   Cluster[x][yn] == false
       && Lattice[x][yn]*Lattice[x][y] > 0
       && RanGen.Random() < 1 - exp(-2*Lattice[x][yn]*Lattice[x][y]))
      {Pocket.Push(x,yn); ++PocketSize; ClusterList.Push(x,yn);
       Cluster[x][yn] = true;}
}
// Flip the cluster (and calculate magnetization on the run)
for(int ClusterSize = ClusterList.Count(); ClusterSize > 0; --ClusterSize){
     x = ClusterList.GetTopX();
    y = ClusterList.GetTopY();
    ClusterList.ScrapTopNode();
    Cluster[x][y] = false;

    Lattice[x][y] = -Lattice[x][y];
    M += 2*Lattice[x][y];
}


// Calculate the lattice energy
E = 0;
for(int f = 0; f < N; ++f){
    for(int g = 0; g < N; ++g){
        //Enforce boundary conditions:
        if(f+1 == N){xn = 0;} else {xn = f+1;}
        if(f == 0){xp = N-1;} else {xp = f-1;}
        if(g+1 == N){yn = 0;} else {yn = g+1;}
        if(g == 0){yp = N-1;} else {yp = g-1;}

        //Get the energy
        squared = 0.5*musquared*Lattice[f][g]*Lattice[f][g];
        quartic = 0.25*lambda*Lattice[f][g]
                  *Lattice[f][g]*Lattice[f][g]*Lattice[f][g];
        interaction = 0.25*((Lattice[f][g]-Lattice[xn][g])
                     *(Lattice[f][g]-Lattice[xn][g])
                    + (Lattice[f][g]-Lattice[f][yn])
                     *(Lattice[f][g]-Lattice[f][yn])
                    + (Lattice[f][g]-Lattice[xp][g])
                     *(Lattice[f][g]-Lattice[xp][g])
                    + (Lattice[f][g]-Lattice[f][yp])
                     *(Lattice[f][g]-Lattice[f][yp]));
        E += interaction + squared + quartic;
```

```
            }
        }

        // Calculate quantities needed for susceptibility & Binder cumulant,
        // if the simulation has equilibrated
        if(i>200){
            AvgSqM += M*M/(T-200);
            SqAvgM += fabs(M/(T-200));
            Avg4M += 2*M*M*M*M/(T-200);
        }
} // End of Metropolis and Wolff loop

    // Record results
    cout << setw(20) << lambda << setw(20) << musquared
         << setw(20) << AvgSqM - SqAvgM*SqAvgM            // Susceptibility
         << setw(20) << 1 - (Avg4M/(3*AvgSqM*AvgSqM))
         << endl;    // BCL cumulant

    return 0;
}

void EnforceBoundaryConditions(int N, int x, int y,
                               int & xn, int & yn, int & xp, int & yp){
    if(x+1 == N){xn = 0;} else {xn = x+1;}
    if(x == 0){xp = N-1;} else {xp = x-1;}
    if(y+1 == N){yn = 0;} else {yn = y+1;}
    if(y == 0){yp = N-1;} else {yp = y-1;}
    return;
}
```

B.2.8 Sample Cluster Script

This section contains two sample shell scripts for submitting jobs to the cluster. First, the master script Wolff-all.sh:

```
   ,
#! /bin/tcsh -f

# This is the script one must run to submit the job to the cluster.

set taskName = Wolff
set script = ${taskName}-one.sh
set baseDirectory = `pwd`
set baseResultsDirectory = ${baseDirectory}/Phi4Production/lambda005/production600

set commandPathname = ${baseDirectory}/${taskName}.cmd
set totalJobs = 600
```

```
# Create the results directory and make it world-writeable so that
# Condor doesn't complain.
mkdir --parents --mode=700 ${baseResultsDirectory}

# Emit the arguments needed for the condor script.  These are for all
# tasks in the job.
printf "## Global job properties\n\n" > ${commandPathname}
printf "universe =     vanilla\n" >> ${commandPathname}
printf "notification = never\n" >> ${commandPathname}
printf "getenv =       true\n" >> ${commandPathname}
printf "initialdir =   ${baseDirectory}\n" >> ${commandPathname}
printf "priority =     5\n" >> ${commandPathname}
printf "executable =   ${taskName}-one.sh\n" >> ${commandPathname}
printf 'requirements = (((Arch=="'INTEL'") || (Arch=="'x86_64'")) &&
                        (OpSys=="'LINUX'"))' >> ${commandPathname}

# Loop through the arguments to be passed for each task.
set i = 0
while (${i} < ${totalJobs})

    # Make the results directory for this specific job.
    set resultsDirectory = ${baseResultsDirectory}/${i}
    mkdir --mode=700 ${resultsDirectory}
    #Simulation parameters; the executible sets musquared = a/b - c
    set a = ${i}
    set b = 300000
    set c = 0.101
    set lambda = 0.05
    set N = 600
    set T = 100200

    # Emit the arguments for this task.
    printf "\n## Task properties\n" >> ${commandPathname}
    printf "log = ${resultsDirectory}/log.rtf\n" >> ${commandPathname}
    printf "output = ${resultsDirectory}/out.rtf\n" >> ${commandPathname}
    printf "error = ${resultsDirectory}/err.rtf\n" >> ${commandPathname}
    printf "arguments = ${a} ${b} ${c} ${lambda} ${N} ${T}\n" >> ${commandPathname}
    printf "queue\n" >> ${commandPathname}

    @ i = ${i} + 1

end

# Submit the job.
condor_submit ${commandPathname}

# Clean up.
rm ${commandPathname}
```

The script above invokes another, the smaller script Wolff-one.sh:

```
   ,
#! /bin/tcsh -f

if (${#argv} != 6) then

    echo "This program requires six command line parameters!"
    exit

endif

./WolffCondor.exe ${argv[1]} ${argv[2]} ${argv[3]} ${argv[4]} ${argv[5]} ${argv[6]}
```

Appendix C

Survey of useful literature

This chapter contains a review of the literature I found most useful in the course of my thesis work. Many of these sources, particularly those pertaining to the Ising model, have already been mentioned in the text, but I collect them here for convenience.

C.1 Programming

I learned C++ primarily from Cohoon (1997), which in retrospect may not have been the best choice. It did contain many useful exercises and taught me the basics, but in later chapters heavily relied on a visual library (EZ Windows) which is of no use in scientific computing. A tutorial found on *Physics Forums*[1] was a valuable introduction to some elementary concepts. Later in the course of my project I obtained Prata (2004); it served as a clear and comprehensive reference, and would have been a good book to learn the language from in the first place.

Gould and Tobochnik (1996) offer an inspiring introduction to scientific computing, but the example code is regrettably in True BASIC, no longer a widely used programming language. The third edition, published in 2006, has examples in Java instead, and may be valuable for those interested in using this language.

Finally, Press et al. (2007) is a comprehensive but accessibly written source on numerical algorithms. Often called a standard reference, it is more of a do-it-yourself guide to scientific computing, very much in the style of Horowitz and Hill (1989).

[1] As of this writing, located at `http://www.physicsforums.com/showthread.php?t=32703`.

C.2 Monte Carlo Methods & the Ising Model

Two books on Monte Carlo method have been particularly useful in the course of this work. An accessible and very well written introduction, through the lens of lattice models such as the Ising, is Newman and Barkema (1999), which has served as the major source for Chapter 3; it focuses specifically on statistical physics and goes on to discuss glasses and ice models. More general, but also more challenging, is Krauth (2006). Another standard is Binder and Heermann (2002), but it's very terse and rather poorly written compared to the previous two, so much so that it is frequently easier to understand the original papers it cites.[2] The very readable book by Binney et al. (1992) focuses on critical phenomena and fleshes out the connection between the Landau-Ginzburg model and quantum field theory, although it approaches the problem from the other side than we have here: the LG model is treated as a toy statistical mechanics system about which we'd like to learn more using field-theoretic methods such as Feynman diagrams.

A few works with narrower focus deserve mention. An excellent review of the Monte Carlo analysis of random walks, including Markov-chain sampling, can be found in Sokal (1994). As mentioned earlier in the text, the mathematics of Markov processes is discussed in considerable detail and rigor (but with an eye to a physics, rather than mathematics audience) in Morningstar (2007).

C.3 Quantum Field Theory

A brief but comprehensive review of sources which might be of interest to undergraduates interested in QFT can be found here: `http://fliptomato.wordpress.com/2006/12/30/`. It listed every book I stumbled upon before discovering it, and more.

Schaich (2006) discusses quantum field theory, and in particular the ϕ^4 theory, in considerably greater depth than I have here. In particular, he covers Feynman diagrams and performs the renormalization procedure. Griffiths (1987) is a well-regarded undergraduate text on particle physics that introduces the basic ideas of QFT. Most of what I've learned, however,

[2] If you do pick it up, be ready for sentences like the following, found on the cover (!) of the 1997 edition: "[This book] deals with the computer simulation of thermodynamic properties of many-body condensed-matter systems that use random numbers generated by a computer in physics and chemistry."

I learned from Zee (2003). A word of caution is in order here: Dirac notation is omnipresent in his book, so if you're not comfortable with it (I wansn't), you may want to take a look at the first few chapters of Townsend (2000).

Ryder (1996) is a useful, but somewhat dry introduction. While Zee introduced QFT through the path integral formalism of quantum mechanics, as I have here, Ryder (and Schaich) take the so-called canonical approach that begins by considering the field ϕ as a creation-annihilation operator.

Bibliography

Bednarzyk, C.: 2001, *Monte Carlo Studies of Some Quantum Field Theories*, Senior thesis at Amherst College

Bhanot, G. V. and Sanielevici, S.: 1989, *Physical Review D* **40(10)**, 3454

Binder, K. and Heermann, D.: 2002, *Monte Carlo Simulation in Statistical Physics: An Introduction*, Solid-State Sciences, Springer, 4th edition

Binney, J. J., Dowrick, N. J., Fisher, A. J., and Newman, M. E. J.: 1992, *The Theory of Critical Phenomena: An Introduction to the Renormalization Group*, Oxford Science Publications, Clarendon Press

Brower, R. C. and Tamayo, P.: 1989, *Physical Review Letters* **62(10)**, 1087

Butera, P. and Comi, M.: 1997, *Physical Review B* **56(13)**, 8212

Challa, M. S., Landau, D. P., and Binder, K.: 1986, *Physical Review B* **34(3)**, 1841

Charng, Y.-Y.: 2001, *Ph.D. thesis*, University of Pittsburgh, Pittsburgh, PA 15260

Cipra, B. A.: 1987, *The American Mathematical Monthly* **94(10)**, 937

Cohoon, J. P.: 1997, *C++ Program Design: an Introduction to Programming and Object-Oriented Design*, McGraw-Hill

de Forcrand, P., Koukiou, F., and Petritis, D.: 1987, *Journal of Statistical Physics* **49(1–2)**, 223

de Gennes, P. G.: 1972, *Physics Letters A* **38(5)**, 339

DeGroot, M. H. and Schervish, M. J.: 2002, *Probability and Statistics*, Addison-Wesley, 3rd edition

Efron, B.: 1979, *SIAM Review* **21(4)**, 460

Efron, B. and Gong, G.: 1983, *The American Statistician* **37(1)**, 36

Efron, B. and Tibshirani, R. J.: 1993, *An Introduction to the Bootstrap*, Vol. 57 of *Monographs on Statistics and Applied Probability*, Chapman & Hall

Fujioka, H. and Kano, H.: 2006, *International Journal of Statistics and Systems* **1(2)**, 111

Gaspari, G. and Rudnick, J.: 1986, *Phys. Rev. B* **33(5)**, 3295

Gould, H. and Tobochnik, J.: 1996, *An Introduction to Computer Simulation Methods (Applications to Physical Systems)*, Addison-Wesley Publishing Company, second edition

Griffiths, D.: 1987, *Introduction to Elementary Particles*, Wiley & Sons

Griffiths, D.: 2005, *Introduction to Quantum Mechanics*, Pearson Prentice Hall, Upper Saddle River, NJ 07458, 2nd edition

Hastie, T., Tibshirani, R., and Friedman, J.: 2001, *The Elements of Statistical Learning*, Springer Series in Statistics, Springer

Horowitz, P. and Hill, W.: 1989, *The Art of Electronics*, Cambridge University Press, 2nd edition

Istrail, S.: 2000, in *Proceedings of the 32nd ACM Symposium on the Theory of Computing*, pp 87–96, Association for Computing Machinery, ACM Press, Portland, Oregon

Karniadakis, G. E. and Kirby, R. M.: 2003, *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation*, Cambridge University Press

Krauth, W.: 2006, *Statistical Mechanics: Algorithms and Computations*, Oxford Master Series, Oxford University Press, Great Clarendon Street, Oxford OX2 6DP

Landau, D. P. and Binder, K.: 2005, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge University Press, 2nd edition

Li, B., Madras, N., and Sokal, A. D.: 1995, *Journal of Statistical Physics* **80(3–4)**, 661

Loinaz, W. and Willey, R. S.: 1998, *Physical Review D* **58(7)**, 076003

Madras, N. N. and Slade, G.: 1996, *The Self-Avoiding Walk*, Springer

Matsamuto, M. and Nishimura, T.: 1998, *ACM Trans. on Modeling and Computer Simulation* **8(1)**, 3

Mawhinney, R. D. and Willey, R. S.: 1995, *Phys. Rev. Lett.* **74(19)**, 3728

Morningstar, C.: 2007, *The Monte Carlo method in quantum field theory*, Last accessed January 25th, 2009 at `http://arxiv.org/abs/hep-lat/0702020v1http://arxiv.org/abs/hep-lat/0702020v1`.

Newman, M. and Barkema, G.: 1999, *Monte Carlo Methods in Statistical Physics*, Clarendon Press, Great Clarendon Street, Oxford OX2 6DP

Parlante, N.: 2001, *Linked List Basics*, Part of the Stanford CS Education Library, last accessed February 3rd, 2009

Pathria, R. K.: 1996, *Statistical Mechanics*, Butterworth-Heinemann, 2nd edition

Plischke, M. and Bergersen, B.: 1994, *Equilibrium Statistical Physics*, World Scientific, 2nd edition

Prata, S.: 2004, *C++ Primer Plus*, Sams, 5th edition

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P.: 2007, *Numerical Recipes. The Art of Scientific Computing*, Cambridge University Press, third edition

Rosenbluth, M. N. and Rosenbluth, A. W.: 1955, *Journal of Chemical Physics* **23**, 356

Ryder, L. H.: 1996, *Quantum Field Theory*, Cambridge University Press, second edition

Schaich, D.: 2006, *Lattice Simulations of Nonperturbative Quantum Field Theories*, Senior Thesis at Amherst College, Last accessed at `https://cms.amherst.edu/media/view/10268/original/schaich06.pdf` on August 29th, 2008.

Sethna, J. P.: 2006, *Statistical Mechanics: Entropy, Order Parameters and Complexity*, Oxford Master Series, Oxford University Press, Great Clarendon Street, Oxford OX2 6DP

Sokal, A. D.: 1994, *Monte Carlo Methods for the Self-Avoiding Walk*

Sondow, J. and Weisstein, E. W.: 2008, *Wallis Formula*, From MathWorld—a Wolfram Web Resource, Last accessed on July 13th, 2008, at `http://mathworld.wolfram.com/WallisFormula.html`

Spitzer, F.: 2001, *Principles of Random Walk*, Springer, 2nd edition

Sprent, P. and Smeeton, N. C.: 2007, *Applied Nonparametric Statistical Methods*, Texts in Statistical Science, Chapman & Hall/CRC, 4th edition

Townsend, J.: 2000, *A Modern Approach to Quantum Mechanics*, University Science Books, 2nd edition

Walker, J. I., Fetzner, R. P., and Baxter, G. W.: 2006, *APS Meeting Abstracts* pp C1126+, http://meetings.aps.org/link/BAPS.2006.MAR.C1.126

Wilson, G. V.: 2006, *American Scientist* **94(1)**, 5, Last accessed on March 24th, 2009, at `http://www.third-bit.com/articles/amsci-swc-2005.pdf`

Zee, A.: 2003, *Quantum Field Theory in a Nutshell*, Princeton University Press