A CURSORY STUDY OF THE THERMODYNAMIC AND MECHANICAL PROPERTIES OF MONTE-CARLO SIMULATIONS OF THE ISING MODEL

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Seng Kai Wong, B.Sc.

James A. Glazier, Director

Kathie E. Newman, Director

Graduate Program in Physics
Notre Dame, Indiana
April 2005

A CURSORY STUDY OF THE THERMODYNAMIC AND MECHANICAL PROPERTIES OF MONTE-CARLO SIMULATIONS OF THE ISING MODEL

Abstract

by

Seng Kai Wong

The Ising model has been very successful in simulating ferromagnetic and antiferromagnetic materials. It is, in fact, the pedagogical explanation for the behavior of magnetic materials. Researchers have adapted the Ising model to simulate materials as disparate as foams, cell aggregates and metallic crystals.

In this research, I used different modifications of Metropolis algorithms to simulate phase separation and Brownian motion in the Ising model. One of the algorithms (Algorithm Three in the text) is a choice popular with many researchers. My goal is to illuminate the differences in the dynamical and equilibrium properties of various algorithms and model parameters.

I found that the most popular choice is not always the right choice. It produces a non-Boltzmann equilibrium and its simulated droplets diffuse much slower than other near-Boltzmann algorithms. In fact, the non-Boltzmann algorithm does not have a critical point, while the others do.

In my phase separation simulations, I investigated a number of thermodynamical properties of the two-dimensional Ising model, including the surface energy, surface tension, partition function, free energy and entropy as a function of model parameters and algorithms.

To obtain a functional form for analyzing phase-separation, I developed a second-nearest neighbor *Solid-On-Solid* (*SOS*) model. I found that the SOS model agrees well with the Ising model up to about one-half the critical temperature. I also used heuristic arguments to create a modified SOS model and compared it to simulation results for up to fourth-nearest-neighbor interactions.

I discovered unexpected behavior when I used the model to simulate Brownian motion. For all the algorithms, droplets diffuse slower than predicted at low temperatures, which I explain by assuming that the underlying lattice is sticky.

One could devise more tests to further distinguish and delineate the limitations of the algorithms, like simulating Stoke's flow. When one modifies or add more terms to the Ising Hamiltonian to simulate different phenomena, one needs to modify the algorithm's acceptance probabilities accordingly in order to preserve detailed balance and Boltzmann equilibrium. I have presented a clear method to create algorithms that obey detailed balance and produce near-Boltzmann equilibria.

CONTENTS

FIGURES

TABLES

SYMBOLS

In order of appearance:

U	internal energy
Q	heat flow
W	work done, interface width
N	number of moleclues
μ	critical exponent, chemical potential, index of lattice states
T	temperature
S	entropy
P	pressure
V	volume
L	length, lattice dimension
σ	surface/line tension, spin of lattice point
F	Helmholtz free energy
k	Boltzmann's constant
Z	partition function
$\mathbf{f}$	force
A	area
T_c	critical temperature
z	coordination number, the number of neighbors within the interaction range
H	energy of a lattice, Hamiltonian of spin-spin interactions
J_{ij}	coupling constant between spins i and j

$\mathbb{Z}$	integers
μ, ν	states of a lattice
M	number of unique lattice states
$\omega_\mu(t)$	probability of a lattice being in state μ at time t
p_μ	probability of a lattice being in state μ at equilibrium
$P(\nu \to \mu)$	transition probability per unit time
$A(\nu \to \mu)$	acceptance probability
$\boldsymbol{T}$	transition matrix
h_i	height of interface from baseline at position i
r_i	step size, defined as the difference in height between adjacent positions
ε_r	energy of a step of size r
J_1	coupling constant between first-nearest neighbors
J_2	coupling constant between second-nearest neighbors
n_r	number of steps of height r pixels
E	total energy of an interface
$C^L_{\{n_r\}}$	number of permutations possible in arranging L steps
ζ_r	exponential of ε_r divided by the temperature
Ξ	the partition function of a unit horizontal length of interface in the SOS model
γ_1	exponential of J_1 divided by the temperature
γ_2	exponential of J_2 divided by the temperature
P_r	probability of occurrence of a step of height r pixels
K	ratio of J_{ij} to temperature
α	roughness exponent
T_r	roughening temperature
H_m	external magnetic field
τ	torque field

h	slope of an interface with respect to the baseline
G	Gibbs free energy
ϕ	lattice anisotropy
t_r	equilibration time constant
β	growth exponent
ε_a	fitting parameter for step energies
ε_b	fitting parameter for step energies
ε_c	fitting parameter for step energies
δ	fitting parameter for degeneracies in step energies
$\mathcal{A}$	fitting parameter for virtual steps
D_f	fractal dimension
D	diffusion coefficient
Ω	friction coefficient
η	dynamic viscosity
J_A	area constraint
$\mathcal{J}$	probability current
μ_m	mobility
λ	frequency of collisions
t_{mf}	mean free time
x_{mf}	mean free path

ACKNOWLEDGMENTS

I am grateful to Professor James A. Glazier who has extended me an extraordinary amount of leeway and patience. His unstinting faith in me and encouragement gave me the confidence to finish this research. His uncompromising criticism and guidance have helped elevate the standard, define and refine the course of this research. However, I must claim that any mistakes in this thesis are mine.

I am also grateful to Professor Kathie Newman whose countless gentle prods have time and again directed me towards a fruitful path and illuminated previously dark avenues. The significance of her suggestions, I have often realized too late. She poured her heart out for me while I rediscover Murphy's law at every corner.

To my parents who gave me unconditional support.

Two roads diverged in a wood, and I-
I took the one less traveled by,
And that has made all the difference.

-Robert Frost

Do not go where the path may lead
Go instead where there is no path and leave a trail.

-Ralph Waldo Emerson

CHAPTER 1

INTRODUCTION

What do polymers [99, 113], liquid surfaces [20, 94], the growth of crystal surfaces [56, 71], the Ising model at low temperatures [1, 42, 115], foams [34, 39, 46, 61] and cell membranes [30, 88] have in common? All of them involve surface/line phenomena where a line or a surface separates two or more phases. Although a polymer is not technically a phase-separating surface, we can describe its response under external forces and heat in much the same way we describe the surface of a liquid. All these phenomena are thermodynamically equivalent so the same formalism can therefore describe them. Statistical mechanics often groups these topics under the heading of cooperative phenomena.

Phase separation requires two competing forces. One tends to restore a given configuration, the other tends to disrupt it. The restoring force usually causes a surface to assume shapes that minimize the surface's energy or a polymer to minimize its length. The disrupting forces usually have the form of thermal energies or pressures, tend to reduce the cooperative interactions between molecules or monomers and cause the surface or polymer to assume higher-energy configurations. Although the detailed mechanisms at work may differ in each case, competition of forces and energy minimization are quite universal and produce surprisingly similar patterns in totally unrelated situations. Nature abounds with such examples [6, 57, 112].

Lenz first proposed the Ising model as a model for ferromagnetism [15, 62, 73].

His student Ising subsequently solved its phase transition exactly. Since the Ising model's invention, a rich variety of research fields have applied it in ways well beyond its original intended purpose. Even today, hundreds of papers appear annually on the application of the Ising model to various phenomena, including neural-network function, protein folding, flocking birds, beating heart cells, phase separation in binary alloys, biological membranes and social behaviors in human society [65].

Because of its derivation, all phenomena the Ising model or its derivatives (*e.g.* the Potts model [101, 102]) can describe relate thermodynamically to the properties of liquid surfaces.

This dissertation does not expound the similarities among the various phenomena I described above, nor survey how seemingly unrelated processes can generate similar patterns, but rather attempts to elucidate the behavior of the Ising model and other models based on the Ising model. Since the Ising model can simulate so many phenomena, I feel that systematic investigation and characterization of models and algorithms and determination of model behaviors and limitations in various circumstances are important. Blindly applying a model is at best wasteful, at worst misleading.

My review of studies of surface phenomena had led me to propose two methods to investigate the behavior of models: static and dynamic. I include surface roughness, surface free energy, surface tension, surface entropy, critical temperature and specific heat among static phenomena. I include the equilibration rate, diffusion and viscosity among dynamic phenomena.

I developed two simulations to investigate the static and dynamic properties of my models:

1. Simulating interfaces between two phases at various angles.
2. Simulating the Brownian motion of droplets of various sizes in a medium.

In the flat-interface simulations, I investigate the surface tension (a static property) as well as the rate at which the interface equilibrates (a dynamic property). In the Brownian motion simulations, I investigate the diffusion of droplets (dynamic) and the equilibrium shape of droplets (static). [1]

The remainder of this introduction briefly describes the thermodynamics of polymers, soap films and liquid and crystalline surfaces and presents a discretized model of surfaces which will later guide my modeling. Finally, I present the Ising model and the algorithms that I use to simulate phase-separation and my rationale for using them.

1.1 Polymers

A polymer consists of a chain of molecules called monomers. Left to its own devices, a polymer tends to curl up on itself and form a clump to minimize its free energy. A tensile force can stretch a polymer to several times its original length. Examples of polymers include natural rubbers and proteins.

The basis of all surface/line phenomena is the first law of thermodynamics:

$$dU = đQ + đW + \mu dN, \tag{1.1}$$

where dU is the change in internal energy of the polymer, $đQ$ is the differential heat flow into the polymer, $đW$ is the differential work done on the polymer, μ is the chemical potential and dN is the change in the number of molecules. The notation $đX$ means that changes in X depend on the details/path of the process, and the differential is *inexact*. Without a bar, changes only depend on the two end points, and the differential is *exact*. For a reversible process, we have from the second law

[1] All the simulations in this dissertation used two-dimensional lattices, therefore, a line instead of a surface separates pairs of phases. However, I will use the term "surface tension" since it is unambiguous.

of thermodynamics:

$$dS = \frac{đQ}{T}, \tag{1.2}$$

where T is the temperature and dS is the change in entropy of the polymer. The integrating factor $1/T$ converts the inexact differential $đQ$ into an exact differential. Combining both laws (equations 1.1, 1.2), we find:

$$dU = TdS + đW + \mu dN. \tag{1.3}$$

The work done on the polymer consists of two parts:

$$đW = -PdV + \sigma dL, \tag{1.4}$$

where the first term is the familiar pressure-volume term for an ideal gas, σ is the line tension of the polymer and L is the length of the polymer. Combining equations 1.3 and 1.4, and assuming that dV and dN are negligible as we stretch the polymer isothermally, we obtain:

$$dU = TdS + \sigma dL. \tag{1.5}$$

In general, σ is a function of T. By rearranging the terms in equation 1.5, we obtain:

$$\sigma = \left(\frac{\partial U}{\partial L}\right)_T - T\left(\frac{\partial S}{\partial L}\right)_T. \tag{1.6}$$

In equation 1.6, σ consists of two parts: an internal-energy contribution and an entropic contribution. The entropic contribution to the line tension in a polymer relates to the number of ways a polymer can fold upon itself. A fully stretched polymer has only one configuration, as a consequence, its entropy is zero. On the other hand, folded polymers can fold in many ways that result in a polymer of the same effective length; as a result, its entropy is non-zero for most lengths. In contrast, the surface tension of a hard solid comes mostly from its internal energy. We can illustrate the difference by observing that the atoms in a hard solid have

fixed positions; they are not free to move around like the molecules in a polymer. The entropic contribution to the surface tension in hard solids is therefore minimal. The surface tension in a hard solid comes mainly from molecular forces instead of molecular movements.

Equation 1.5 gives the internal energy U as a function of the entropy S and the length L:

$$U = U(S, L). \tag{1.7}$$

Since all of my simulations fix the temperature and length, working with a natural energy of state of the T-L canonical ensemble is more convenient. The familiar Helmholtz free energy is such a quantity:

$$\begin{aligned} F &= U - TS, \\ &= -kT \ln Z, \end{aligned} \tag{1.8}$$

where k is Boltzmann's constant and Z is the partition function of the T-L ensemble. Differentiating both sides of equation 1.8 and using equation 1.5, I obtain:

$$\begin{aligned} dF &= dU - TdS - SdT \\ &= \sigma dL - SdT. \end{aligned} \tag{1.9}$$

As expected, F is a function of T and L. The derivatives of F with respect to T and L yield:

$$\left(\frac{\partial F}{\partial L}\right)_T = \sigma, \tag{1.10}$$

$$\left(\frac{\partial F}{\partial T}\right)_L = -S. \tag{1.11}$$

Equation 1.10 shows that the line tension of the polymer is just its free energy per unit length at constant temperature. Given the definition of F (equation 1.8), equation 1.10 is consistent with equation 1.6. Because F has to be an exact differential,

we also have the following relation:

$$\left(\frac{\partial \sigma}{\partial L}\right)_T = -\left(\frac{\partial S}{\partial T}\right)_L. \tag{1.12}$$

The above thermodynamic arguments are model independent. In order to proceed further, I must make some assumptions about the states of molecules in the polymer chain. Meyer [79] proposed the first kinetic theory of elasticity. The many current models of molecular chains vary in sophistication; see references [38, 49, 59, 67, 80, 99].

The simplest approach models a polymer as a chain of N identical monomers. The length of each monomer is l. Each monomer can either point to the right or left. If N_+ monomers point to the right and N_- monomers point to the left, the total length of the polymer is:

$$L = (N_+ - N_-)l. \tag{1.13}$$

Even though the monomers are identical, we can distinguish them by their fixed positions in the chain. We can arrange the $N = N_+ + N_-$ monomers in:

$$\Omega = \frac{N!}{N_+!N_-!} \tag{1.14}$$

ways. The entropy of the chain is:

$$S = -k \ln \Omega. \tag{1.15}$$

Taking the thermodynamic limit($N \to \infty$) and using Stirling's approximation:

$$S = -k(N \ln N - N_+ \ln N_+ - N_- \ln N_-). \tag{1.16}$$

Since the joints can turn freely, the internal energy is independent of L. The tension arises entirely from the entropic term. Substituting equation 1.16 into equation 1.6, the tension of the chain is:

$$\sigma = \frac{kT}{2l} \ln \frac{1 + L/Nl}{1 - L/Nl}. \tag{1.17}$$

The tension of the chain increases with temperature, in contrast to the behavior of a steel spring. This linearity in temperature resembles the pressure's linear dependence on temperature in an ideal gas. In both cases, the proportionality depends less on internal energies than on the number of available configurations. Figure 1.1 plots the behavior of σ with respect to the extension L. For small extensions ($L \ll Nl$), σ is linear in L, which agrees with Hooke's law.

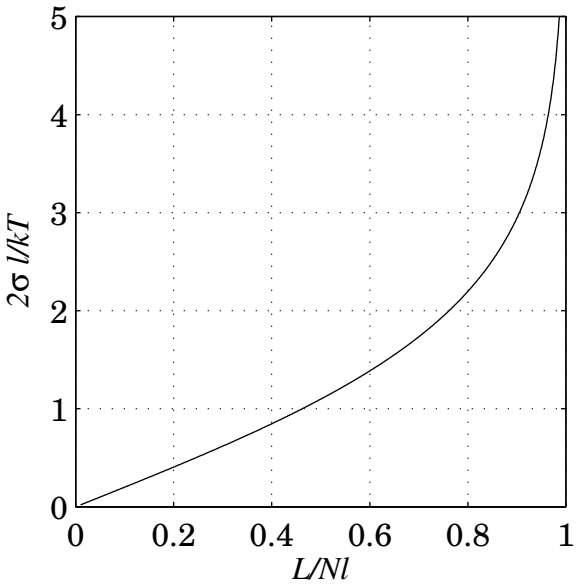


Figure 1.1. Plot of surface tension σ (equation 1.17) as a function of length L for a simple model of a polymer.

Many polymer models extend the one I have described. Molecules can point in all directions instead of just 0° and 180°. The model may be three-dimensional. The direction a molecule points may have an associated energy. I will not discuss these models here, since the references I have mentioned cover them. The purpose of this brief discussion is to compare the typical behavior of polymer tension to the surface tension of liquids.

1.2 Soap films

I next consider the surface tension of a soap film. Consider a soap film stretched inside a rectangular wire frame (figure 1.2), of which one side can move (AB).

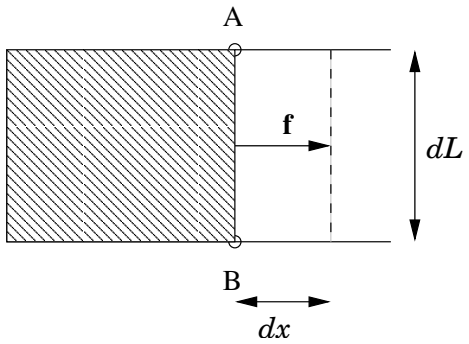


Figure 1.2. Soap film stretched inside a rectangular wire frame. I have exaggerated the magnitude of dx for clarity.

The surface tension of the soap film is normal to the line AB and points left. Without the external force $\mathbf{f}$ pulling AB to the right, the soap film would collapse to zero area. At equilibrium, the two competing forces are equal:

$$f = \sigma L\,. \tag{1.18}$$

Consider an infinitesimal extension of the soap film as we pull AB to the right. Assume that the force needed to pull the soap film an infinitesimal distance is constant. The work done on the soap film is just:

$$đW = f dx = \sigma L dx = \sigma dA. \tag{1.19}$$

As the soap film stretches, the internal energy spreads over a larger surface area, so the soap film cools. If the stretching is isothermal, the soap film must absorb heat energy in order to maintain its temperature. Therefore, stretching in general also involves heat flow. From the first law of thermodynamics:

$$dU = \sigma dA + T dS. \tag{1.20}$$

Dividing both sides of equation 1.20 by dA, we again find that the surface tension results from the internal energy and entropy (see equation 1.6). Introducing the Helmholtz free energy (equation 1.8), we obtain:

$$dF = \sigma dA - SdT. \tag{1.21}$$

Equation 1.21 is analogous to equation 1.9, except that F is the free energy of the T-A ensemble. Differentiating F with respect to its independent variables, A and T, we obtain the same set of equations as equations 1.10 to 1.12, but with A replacing L:

$$\left(\frac{\partial F}{\partial A}\right)_T = \sigma, \tag{1.22}$$

$$\left(\frac{\partial F}{\partial T}\right)_A = -S, \tag{1.23}$$

$$\left(\frac{\partial \sigma}{\partial T}\right)_A = -\left(\frac{\partial S}{\partial A}\right)_T. \tag{1.24}$$

Thus the surface tension is simply the Helmholtz free energy per unit area at constant temperature. If we substitute equation 1.24 into equation 1.20, we relate U to σ:

$$\left(\frac{\partial U}{\partial A}\right)_T = \sigma - T\left(\frac{\partial \sigma}{\partial T}\right)_A = -T^2\frac{\partial}{\partial T}\left(\frac{\sigma}{T}\right), \tag{1.25}$$

consistent with the familiar thermodynamic relations:

$$U = kT^2\frac{\partial}{\partial T}\ln Z = -T^2\frac{\partial}{\partial T}\left(\frac{F}{T}\right). \tag{1.26}$$

We can derive the specific heat capacity C_v of the soap film from U:

$$\begin{aligned} C_v &= \left(\frac{\partial U}{\partial T}\right)_V = \left(\frac{\partial U}{\partial T}\right)_A, \\ &= -\frac{\partial}{\partial T}\left[T^2\frac{\partial}{\partial T}\left(\frac{F}{T}\right)\right]_A. \end{aligned} \tag{1.27}$$

Empirically, the surface tension of a liquid in thermodynamic equilibrium with its vapor has the form:[2]

$$\sigma = \sigma_o \left(1 - \frac{T}{T_c}\right)^{\mu}. \tag{1.28}$$

At the critical temperature T_c, the surface tension is zero and liquid and vapor phases are indistinguishable. In other words, the meniscus that separates liquid from vapor no longer exists. Temperatures above T_c have only one phase: the vapor/gas phase. The value of μ is roughly 1.28 ± 0.06 and is believed to be a universal constant [119].

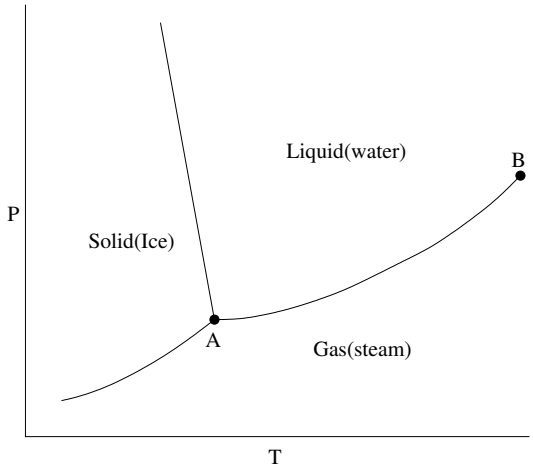


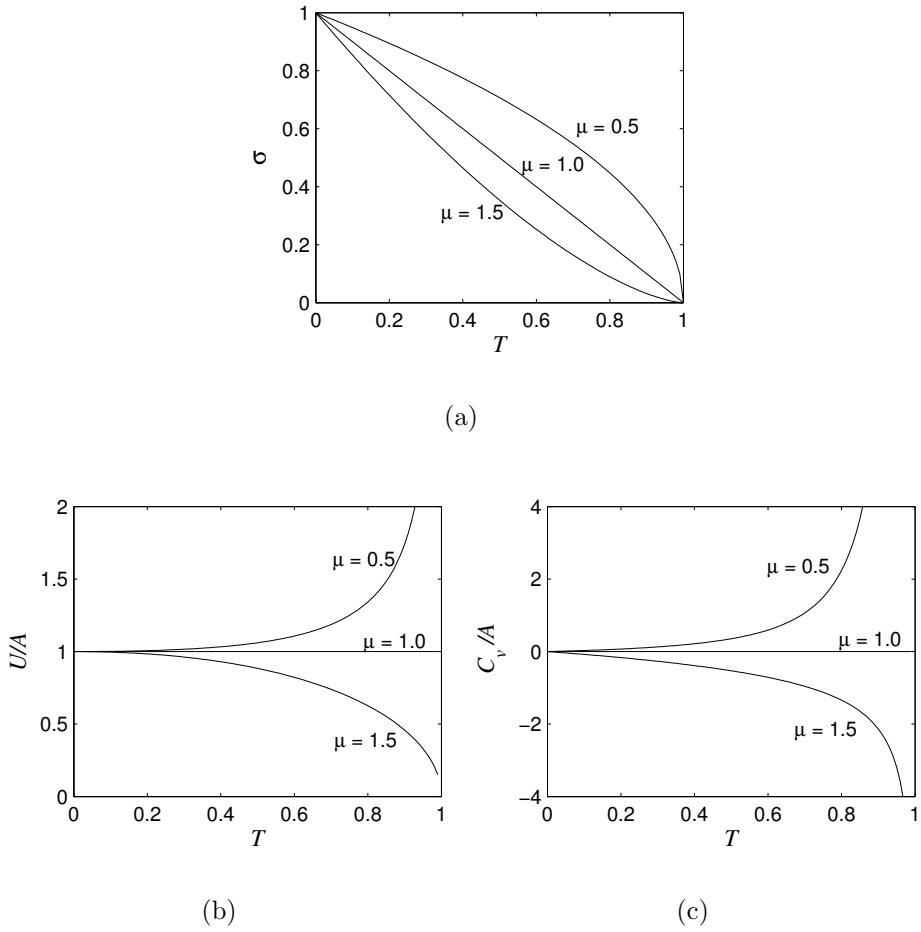
Figure 1.3. Phase diagram for water. A is the triple point. B is the critical point. The critical temperature of water is about 647°K. The curve AB traces the locus of boiling points.

The boiling point depends on the pressure. Even at boiling, the liquid phase and vapor phase remain well separated. The boiling point traces a curve on the P-T plot, which ends at the critical point. Figure 1.3 sketches the phase diagram of water [72].

Figure 1.4 shows the surface tension, σ (equation 1.28), the normalized surface internal energy, U/A (equation 1.25) and the normalized surface specific heat, C_v/A

[2]Following the convention in the literature I have cited, papers by Mon *et al.* ([81]-[84]) and Hasenbusch *et al.* ([51]-[55]), I denote the critical exponent μ, confusingly, the same as the symbol for the chemical potential.

(equation 1.27) versus temperature for three different values of μ and $\sigma = T_c = 1$. In deriving U/A and C_v/A, I have assumed $F = \sigma A$.



(a)

(b) (c)

Figure 1.4. Liquid-vapor equilibrium based on empirical equation 1.28. (a) Surface tension (equation 1.28), (b) normalized surface internal energy (equation 1.25) and (c) normalized surface specific heat (equation 1.27) as a function of temperature for three different values of μ and $\sigma = T_c = 1$.

Typically, the surface tension of liquids decreases with increasing temperature, as figure 1.4(a) shows. $\sigma = 0$ for $T \geqslant T_c$. We call such behavior *normal thermocapillarity*. In contrast, the line tension of polymer increases with temperature (figure 1.1).

The first derivative of F (or σ) with respect to T, U (figure 1.4(b)), is continuous

at T_c for $\mu > 1$, but discontinuous for $\mu \leqslant 1$. In both cases, the second derivative of F with respect to T, C_v (figure 1.4(c)), diverges at T_c. Therefore, for $\mu > 1$, the critical temperature associates with a second-order phase transition, while for $\mu \leqslant 1$, the critical temperature associates with a first-order phase transition with a latent heat [108], similar to what happens when ice melts or water boils.

1.3 Bulk Liquid Surfaces

As a third example, I consider the surface tension between two immiscible liquids in thermodynamic equilibrium. When we consider bulk liquids, we need to separate bulk from surface properties. For each liquid, we have from the first law of thermodynamics,

$$\begin{aligned} dU_a &= PdV_a + TdS_a + \mu dN_a, \\ dU_b &= PdV_b + TdS_b + \mu dN_b, \end{aligned} \tag{1.29}$$

where P is the pressure and μ the chemical potential of each liquid. At equilibrium, the two liquids have the same P, T, and μ. For the liquids including the surface, the general equation for dU is:

$$dU = PdV + TdS + \mu dN + \sigma dA. \tag{1.30}$$

We define the surface component of any variable to be:

$$X_s = X - X_a - X_b. \tag{1.31}$$

Since $V = V_a + V_b$, the surface internal energy is:

$$\begin{aligned} dU_s &= dU - dU_a - dU_b \\ &= TdS_s + \mu dN_s + \sigma dA. \end{aligned} \tag{1.32}$$

Using a suitably defined interface (the Gibbs surface[3]) eliminates the dN_s term:

$$dU_s = TdS_s + \sigma dA. \tag{1.33}$$

Thus, the surface and bulk properties separate. Again, as in equations 1.6 and 1.22, the surface tension arises from the surface internal energy and the surface entropy. We may then proceed as we did in the previous two examples. However, we may employ an important property of U and its dependent variables: V, S, A and N. The variables in this set including U are all extensive variables while P, T, σ, μ are all intensive, *i.e.* if we double V, S, A or N, we also double U:

$$U(\lambda V, \lambda S, \lambda A, \lambda N) = \lambda U(V, S, A, N). \tag{1.34}$$

Applying Euler's theorem (see Appendix A) to U_s, we obtain:

$$U_s = TS_s + \sigma A. \tag{1.35}$$

As usual, we define the surface Helmholtz free energy as:

$$F_s = U_s - TS_s = \sigma A. \tag{1.36}$$

Therefore, the surface tension between two immiscible liquids equals the surface Helmholtz free energy per unit area. My previous two examples were general, making no assumptions about the internal energy. Similarly, the extensivity of U applies to all the equations I derived previously.

1.4 Discrete Lattices

In the preceding sections, I discussed the general thermodynamics of surfaces and derived some general relations between the surface tension, Helmholtz free energy,

[3]Generally, the interface between two liquids is not sharp. The density of one liquid gradually decreases while the other gradually increases at the interface. Typically, the density profiles of both liquids are symmetrical and the Gibbs surface lies at the point where the density of each liquid is half its respective bulk value [20, 23].

internal energy, entropy and specific heat. I also compared the behaviors of the surface tensions of polymers and liquids. As I mentioned at the beginning of this chapter, the Ising model can simulate many situations. Since the Ising model is discrete in nature, I will next present a discrete version of surface tension in the *Solid-On-Solid* model, or *SOS* for short [19]. The results of the SOS model will serve as a basis for analysis of simulation results in chapter 4.

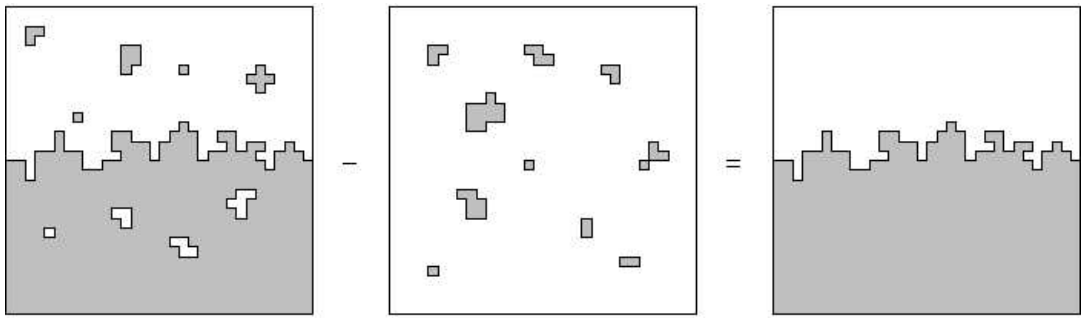
Onsager's seminal paper [95] was the first and only successful analytic derivation of the surface tension of the Ising model. Unfortunately, the result applied to a two-dimensional lattice with nearest-neighbor interactions. All subsequent attempts to derive the surface tension for higher-order interactions and/or higher dimensions have relied on approximations (*e.g.* series expansion) or computational methods (*e.g.* Monte-Carlo simulation).

Previous computational determinations of the surface tension [1, 42, 51, 54, 55, 82, 83] calculated the partition functions for two-phase states and one-phase states, then took the difference between the two (figure 1.5). This method works well for temperatures well below the critical temperature, T_c. For temperatures near T_c, other methods extract the interfacial free energy from the probability distribution of spontaneous magnetization in both the two-phase and one-phase states [9, 11].

We can force a two-phase state by fixing the top and bottom rows of the lattice to have opposite spins, while we achieve a one-phase state by fixing the top and bottom rows to have the same spin. The left and right sides of the lattice connect to impose a periodic boundary condition. The lattice therefore has a cylindrical shape.

Subtracting *bulk fluctuations*[4] from the lattice amounts to taking the ratio of the two-phase and one-phase partition functions, where the surface free energy is

[4]Bulk fluctuations are homogeneous spin domains which a region of differing spin surrounds and where the boundary between the two opposite spins does not span the entire lattice.



(a) two-phases (b) one-phase (c) two-phases - bulk fluctuations

Figure 1.5. Schematic showing the subtraction of bulk fluctuations from the lattice. In (a), the "white" phase and "grey" phase each occupy approximately 50% of the lattice, and a boundary line extends across the lattice at mid-figure. Additionally, small droplets of grey phase penetrate the white, and of white the grey. In (b), the primary white phase has small droplets of grey phase. In (c), I have suppressed the bulk fluctuations, leaving just an interface between the two phases.

simply the logarithm of the surface partition function:

$$F_s = kT \ln Z_s = kT \ln \frac{Z_2}{Z_1}. \tag{1.37}$$

As the temperature increases, we expect bulk fluctuations to increase both in size and density. When fluctuations typically occur within an interaction range of each other, they are no longer isolated and they stretch across the entire lattice. At that point, Z_2 becomes indistinguishable from Z_1. The free energy and surface tension go to zero. We expect such behavior for normal thermocapillarity.

My investigation, unlike previous attempts, applies fourth-nearest-neighbor interactions to the SOS model. I also modify the Metropolis algorithm specifically to inhibit bulk fluctuations. As far as I know, no one has previously investigated the effects of modified algorithms on surface dynamics. Previous SOS approximations have modeled the first nearest-neighbor Ising model in a non-zero magnetic field [86].

At low temperatures, when bulk fluctuations are small, fleeting and sparse, I expect their effects on the surface tension to be negligible, and my SOS results to

agree with previous work on the Ising model (chapter 3). What happens at higher temperatures? Do the free energy and surface tension approach zero in the absence of bulk fluctuations? The short answer is that they still go to zero. As it turns out, the surface itself fluctuates so wildly that the entropic term (TS) dominates the internal energy (U). In the liquid-vapor equilibrium analogy, we would arrive at the critical point without boiling the liquid or crossing the curve AB in figure 1.3. However, I do expect, the critical temperature to be higher than in the absence of bulk fluctuations.

CHAPTER 2

THE ISING MODEL AND MONTE-CARLO SIMULATIONS

Consider a two-dimensional lattice with square elements, figure 2.1:

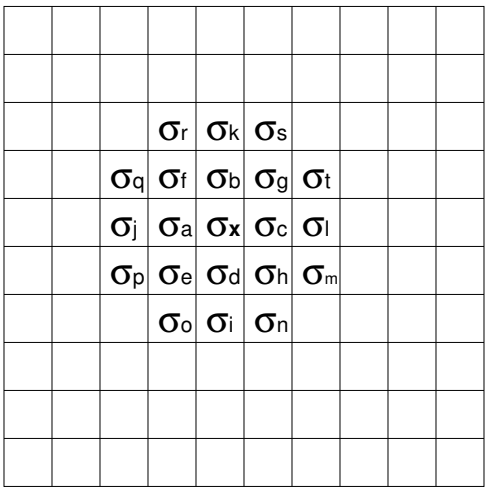


Figure 2.1. A 10×10 square lattice.

As I mentioned earlier, the Ising model is a model on a discrete lattice. Each lattice point, which I shall call a *pixel*, associates with a *spin*.[1] In the Ising model, the value of the spin is either 0 or 1 (or +1 and -1). Each pixel interacts with its neighbors according to the Ising-model Hamiltonian (equation 2.1, below). The Ising-model Hamiltonian defines the energies of interaction between spins or pixels. The form of the Hamiltonian is independent of the dimension and shape of the lattice. I can equally apply the Hamiltonian to one- or three-dimensional lattices or

[1] As is customary in the literature, I use a spin terminology borrowed from the Ising model's ferromagnetic origin. Spin does not carry the same meaning as in magnetic materials. I use it to identify different phases or droplets.

lattices with triangular or hexagonal elements. I use the square lattice because it is easy to implement and illustrates all the salient behaviors of the model.

Figure 2.1 shows a square lattice of 10×10 pixels. The symbol σ denotes the spin. Although I use the same symbol to denote the spin and the surface tension, the meaning of σ should be clear from the context.

Figure 2.1 labels some of the spins. Each spin interacts with its neighbors over some range. I express the range of interaction as "up to nth-nearest neighbors." Spins σ_a to σ_t mark the neighbors of pixel σ_x up to fourth-nearest neighbors. The coordination number z is the number of neighbors within the interaction range.[2]

If all interactions are of equal strength, for first-nearest-neighbor interactions, $z = 4$; for second-nearest-neighbor interactions, $z = 8$; for third-nearest-neighbor interactions, $z = 12$; for fourth-nearest-neighbor interactions, $z = 20$. Due to constraints on computer power and time, I stop at fourth-nearest-neighbor interactions. In this work, I only consider interactions of equal strength.

I define my implementation of the the Ising-model Hamiltonian is defined as follows: If two neighboring pixels (within the interaction range) differ in spin, then the mismatched link between them contributes to the energy of both pixels. Otherwise, its contribution is zero. Using figure 2.1 as an example, the energy of pixel x is:

$$H_x = \sum_{i=1}^{z} J_{xi}(1 - \delta_{\sigma_x,\sigma_i}), \tag{2.1}$$

where i ranges over all the neighbors of x within the range of interaction. When σ_x and σ_i are the same, the link between them does not contribution to H_x. When they differ, the link between them increases H_x by J_{xi}. J_{xi} is the coupling constant between spin x and i. Positive J_{xi} describes a ferromagnet, while negative J_{xi} describes an anti-ferromagnet.

[2]Strictly speaking, z = number of neighbors only if all the interactions are of equal strength. Otherwise, each neighbor contribute to z in proportion to its strength of interaction.

The total energy of the lattice is:

$$H = \frac{1}{2}\sum_{x} H_x. \tag{2.2}$$

I include the factor of 1/2 because I count each link twice and the coupling constant is symmetrical: $J_{ij} = J_{ji}$.

2.1 Lattice States

I define a *lattice state* as a particular assignment of spins. Suppose the lattice is in a state μ with energy H_μ. Two states differ if at least one spin in one assignment differs from the corresponding spin in the other. An $L \times L$ Ising lattice has $M = 2^{L \times L}$ possible states. Multiple states may have the same energy. For example, if I flip all the spins in an Ising lattice ($0 \to 1$, $1 \to 0$), its energy remains the same. The set of lattice states with the same fixed energy forms a *microcanonical ensemble*. The set of lattice states allowed when the lattice can exchange energy with a reservoir at fixed temperature forms a *canonical ensemble*.

Gibbs [44] showed that for a system in thermal equilibrium with a heat reservoir at temperature T, the equilibrium probability distribution is:

$$p_\mu = \frac{e^{-E_\mu/kT}}{Z}, \tag{2.3}$$

where the partition function Z is:

$$Z = \sum_{\mu}^{M} e^{-E_\mu/kT}. \tag{2.4}$$

Equation 2.3 is the familiar Boltzmann distribution. The expectation value for a quantity Q at thermal equilibrium is:

$$\langle Q \rangle = \sum_{\mu}^{M} p_\mu Q_\mu = \frac{\sum_{\mu}^{M} Q_\mu e^{-E_\mu/kT}}{Z}, \tag{2.5}$$

where Q_μ is the value of Q when the lattice is in state μ. Q can represent any of the thermodynamic properties we want to investigate, like the surface tension or the specific heat.

However, if we simply pick a state μ at random and accept or reject it with probability p_μ (equation 2.3), the number of states is so large that we would end up rejecting practically all candidate states since the probability of accepting each state will be of order $\frac{1}{M}$. Instead of generating states at random, we rely on a Markov process to generate state μ from state ν.

2.2 Markov Processes

A coin-toss experiment consists of a series of independent processes because each trial is independent of the previous results. A Markov process [31] differs from the coin-toss experiment because each trial depends on the preceding result. The probability that a state ν transforms into state μ depends only on the two states and not on any additional past history.

I define $P(\nu \to \mu)$ to be the transition probability from state ν to state μ per unit time. I also define $\omega_\mu(t)$ to be the probability that the lattice is in state μ at time t.

The *Master Equation* [90] describes how the lattice evolves from state ν to state μ:

$$\frac{\mathrm{d}\omega_\mu(t)}{\mathrm{d}t} = \sum_{\nu}^{M} \left[\omega_\nu(t)P(\nu \to \mu) - \omega_\mu(t)P(\mu \to \nu)\right]. \tag{2.6}$$

The first term on the right-hand side of equation 2.6 describes all possible transitions into state μ and the second term describes all transitions out of state μ. The transition of the state into itself ($\mu \to \mu$) cancels from both terms. Since the lattice must be in some state ν and must transform into some state μ:

$$\sum_{\nu}^{M} \omega_\nu(t) = 1, \tag{2.7}$$

$$\sum_{\mu}^{M} P(\nu \to \mu) = 1. \tag{2.8}$$

At equilibrium, $\omega_\mu(t)$ is constant and $\frac{\mathrm{d}\omega_\mu(t)}{\mathrm{d}t}$ vanishes:

$$\lim_{t\to\infty} \omega_\mu(t) = p_\mu. \tag{2.9}$$

The Master Equation 2.6 then reduces to:

$$\sum_{\nu}^{M} p_\nu P(\nu \to \mu) = \sum_{\nu}^{M} p_\mu P(\mu \to \nu). \tag{2.10}$$

Using the sum rule from equation 2.8, equation 2.10 further simplifies to:

$$\sum_{\nu}^{M} p_\nu P(\nu \to \mu) = p_\mu. \tag{2.11}$$

In expanded form, equation 2.11 becomes:

$$\begin{pmatrix} P(1\to 1) & P(2\to 1) & \dots & P(M\to 1) \\ P(1\to 2) & P(2\to 2) & \dots & P(M\to 2) \\ \vdots & \vdots & \ddots & \vdots \\ P(1\to M) & P(2\to M) & \dots & P(M\to M) \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_M \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_M \end{pmatrix}, \tag{2.12}$$

If we further define the transition matrix $\boldsymbol{T}$ with elements $T_{\mu\nu} = P(\nu \to \mu)$ and column vector $\boldsymbol{p}$ with elements p_μ, we can write equation 2.12 as:

$$\boldsymbol{T} \cdot \boldsymbol{p} = \boldsymbol{p}. \tag{2.13}$$

Hence the equilibrium probability distribution $\boldsymbol{p}$ is an eigenvector of $\boldsymbol{T}$ with eigenvalue 1.

2.3 Detailed Balance

Although the simple, static equilibrium distribution $\boldsymbol{p}$ solves equation 2.13, periodic solutions are also possible. In periodic solutions, $\boldsymbol{p}$ is not a constant, but cycles

through a periodic loop, a *limit cycle*. For a limit cycle of length n, $\boldsymbol{p}$ returns to its initial value after n iterations [90], *i.e.*:

$$\boldsymbol{T}^n \cdot \boldsymbol{p} = \boldsymbol{p}. \tag{2.14}$$

In order to prevent limit cycles, I impose the additional condition of *detailed balance* on the transition probabilities $P(\mu \to \nu)$. Detailed balance holds at equilibrium. Mathematically, detailed balance requires that:

$$p_\mu P(\mu \to \nu) = p_\nu P(\nu \to \mu). \tag{2.15}$$

Referring to equation 2.10, this condition (equation 2.15) simply states that both sides of equation 2.10 are equal, term by term. Detailed balance ensures that at equilibrium, the rate of transition into state μ equals the rate of transition out of state μ. Hence, at equilibrium, the probability distribution $\boldsymbol{p}$ is conserved.

2.4 Ergodicity

An $M \times M$ matrix $\boldsymbol{S}$ whose elements satisfy the conditions:

1. $0 \leq S_{ij} \leq 1, \qquad \forall \quad i, j = 1, 2, \ldots, M;$
2. $\sum_i S_{ij} = 1,$

is a *stochastic matrix*, *e.g.* the transition matrix, $\boldsymbol{T}$ which I defined in section 2.2, is a stochastic matrix.

If we can arrange the columns and rows of a stochastic matrix in the form:

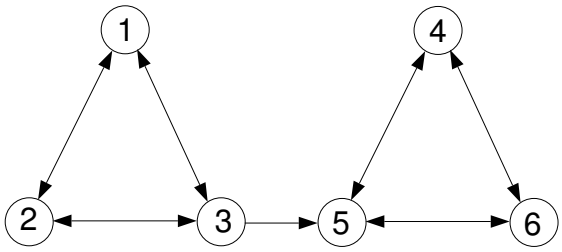
$$\boldsymbol{S} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{0} & \boldsymbol{C} \end{pmatrix} \tag{2.16}$$

where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ are square matrices and $\boldsymbol{0}$ is a zero matrix, the stochastic matrix is *reducible*. If $\boldsymbol{B} = \boldsymbol{0}$, it is *completely reducible* [5].

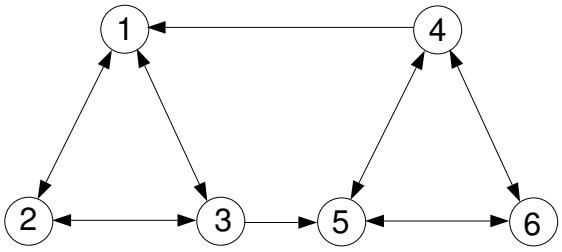
A completely reducible stochastic matrix is undesirable in Monte-Carlo simulation since the states the matrix describes fall into disconnected subsets. If an

initial state lies in subset $\boldsymbol{A}$, all subsequent states will remain within $\boldsymbol{A}$, which will obviously not represent the equilibrium distribution accurately. Representing the equilibrium distribution requires that the transition matrix $\boldsymbol{T}$ be ergodic.

A Markov process is *ergodic* if it connects any state to any other state in a finite number of steps. Since we rely on Markov chains to generate an ensemble of states, ergodicity guarantees that we will eventually reach the equilibrium distribution from any initial state. In practice, equilibration may take a very long time. Although some transition probabilities $P(\mu \rightarrow \nu)$ may be zero, ergodicity guarantees that other paths link state μ to state ν.



(a) Non-ergodic



(b) Ergodic

Figure 2.2. Examples of (a) non-ergodic and (b) ergodic state diagrams. Circles denote states, while arrows denote allowed transitions.

Figure 2.2(a) shows a non-ergodic state diagram where states 4, 5, and 6 cannot access states 1, 2 and 3. Figure 2.2(b) shows an ergodic state diagram. While not every potential transition exists, any state connects to any other state in a finite number of steps.

One simple way of ensuring that a transition matrix is irreducible and ergodic is to make sure that it is positive definite. If all elements are positive definite, no arrangement of the rows and columns forms a square, zero submatrix.

By definition, if $\boldsymbol{S}$ is a stochastic matrix, then $\boldsymbol{S}^n$ is also a stochastic matrix. For example, the sum of each column of $\boldsymbol{S}^2$ is:

$$\begin{aligned} \sum_i \sum_k S_{ik} S_{kj} &= \sum_k S_{kj} \sum_i S_{ik} \\ &= \sum_k S_{kj} \\ &= 1. \end{aligned} \tag{2.17}$$

By extension, the same result follows for any power of $\boldsymbol{S}$. I can thus interpret $(\boldsymbol{T}^n)_{\mu\nu}$ as the probability of a transition from state ν to state μ within n time steps. Due to ergodicity, we expect that $(\boldsymbol{T}^n)_{\mu\nu}$ will be independent of n and ν in the limit $n \to \infty$. Regardless of the initial conditions, the probability of a transition to state μ after equilibration (if it happens), must equal the probability of finding the system in state μ:

$$\lim_{n\to\infty} (\boldsymbol{T}^n)_{\mu\nu} = p_\mu. \tag{2.18}$$

Equation 2.18 implies that every element of $\boldsymbol{T}^n$ in the same row has the same value:

$$\lim_{n\to\infty} \boldsymbol{T}^n = \begin{pmatrix} p_1 & p_1 & \cdots & p_1 \\ p_2 & p_2 & \cdots & p_2 \\ \vdots & \vdots & \ddots & \vdots \\ p_M & p_M & \cdots & p_M \end{pmatrix}. \tag{2.19}$$

The sum of each column equals 1, as expected for a stochastic matrix. Clearly, as $n \to \infty$,

$$\lim_{n\to\infty} \boldsymbol{T}^n \boldsymbol{\omega}(0) = \boldsymbol{p}, \qquad \forall \quad \boldsymbol{\omega}(0). \tag{2.20}$$

2.5 Spectra and Degeneracy of Stochastic Matrices

An $M \times M$ stochastic matrix has M eigenvalues. From equation 2.13, we know that 1 is an eigenvalue of the transition matrix. What is the significance of the other eigenvalues and eigenvectors? What if the unit eigenvalue is degenerate? Does degeneracy indicate coexistence of more than one equilibrium eigenstate?

The following theorems relate to present discussion [5, 7]:

1. The largest eigenvalue of a stochastic matrix is 1. The corresponding eigenvector is *positive*, *i.e.* all of its elements are positive.
2. Other eigenvectors with corresponding eigenvalues $|\lambda| < 1$ cannot be probability distributions (at least one of the elements in p_μ has to be negative, see proof in [5]).
3. A positive-definite stochastic matrix $\boldsymbol{S} > 0$, has only one real unit eigenvalue. In other words, it is not degenerate. It has only one equilibrium eigenstate (Perron, 1907 [100]). The equilibrium distribution p_μ is independent of the initial conditions.
4. If a stochastic matrix contains zero entries, $\boldsymbol{S} \geq 0$, but is still irreducible (ergodic), it always has a real unit eigenvalue. It may have other eigenvalues of unit modulus ($e^{i2n\pi/h}, n = 1, 2, \ldots, h$), but the eigenvalue one itself is non-degenerate. To the largest eigenvalue, one, corresponds a positive eigenvector (Frobenius, 1912 [40]). Only one physical equilibrium eigenstate exists, $\boldsymbol{p} \in \mathbb{R},\ 0 < \boldsymbol{p} \leq 1$.

Since all elements in $\boldsymbol{T}$ are positive definite, ergodicity ensures a unique equilibrium probability distribution. The following subsection 2.6 presents an algorithm for generating a Markov chain that satisfies the condition of detailed balance and ergodicity.

2.6 Metropolis Algorithm

Many algorithms can generate a Markov chain of Ising states [70, 90]. One of the simplest and most popular is the *Metropolis algorithm* [78]. Later, I will consider some modifications to the Metropolis algorithm and their effects on the conditions of detailed balance and ergodicity.

The Metropolis algorithm generates a new state from the current state by changing one of the current state's spins. For this reason, we call the Metropolis algorithm a *single-spin-flip* algorithm. Because the initial state and final state only differ by one spin, the typical energy difference is small.

In a Monte-Carlo simulation, we first randomly select a pixel from the lattice. Then, we flip the spin of the selected pixel. If its current spin is 1, we flip it to 0, and *vice versa*. If ν denotes the state with the flipped pixel, the transition probability is:

$$P(\mu \to \nu) = \frac{1}{L^2} A(\mu \to \nu), \tag{2.21}$$

where $A(\mu \to \nu)$ is the probability of accepting the new spin. A lattice of size $L \times L$ has $L \times L$ states that differ from the current state by one spin flip. Since the choice of pixel is random, all pixels have equal probability $(1/L^2)$ of being chosen. If we do not accept the new spin, the spin of the selected pixel reverts to its previous spin.

If the state ν differs from state μ by more than one spin flip, then $P(\mu \to \nu) = 0$. However, the algorithm is still ergodic because any state can transform into any other state in a finite series of spin flips. For example, if we refer to figure 2.2(b), although state 1 cannot transform into state 4 in a single transition, *i.e.* $P(1 \to 4) = 0$, it can still transform into state 4 via states 3 and 5. The Forbenius theorem in section 2.5 then implies that a unique equilibrium distribution of states, p_μ exists.

To determine the correct acceptance probabilities, $A(\mu \to \nu)$, I invoke detailed balance (equation 2.15):

$$\frac{p_\mu}{p_\nu} = \frac{P(\nu \to \mu)}{P(\mu \to \nu)} = \frac{A(\nu \to \mu)}{A(\mu \to \nu)}. \tag{2.22}$$

The factor $1/L^2$ cancels since it occurs in both the forward $(\nu \to \mu)$ and backward $(\mu \to \nu)$ transitions. If we want the state distribution to corresponds to the

Boltzmann probability distribution (equation 2.3), then the ratio must be:

$$\frac{A(\nu \to \mu)}{A(\mu \to \nu)} = e^{-(E_\mu - E_\nu)/kT}. \tag{2.23}$$

We are free to choose the values of $A(\nu \to \mu)$ as long as they satisfy equation 2.23. We can, of course, choose $A(\nu \to \mu) = p_\mu$ but doing so is the same as generating an arbitrary state and accepting it with probability p_μ. As I mentioned earlier, p_μ is of order $1/(2^{L\times L})$. Even a modest-sized lattice, say $L = 32$, will take a time longer than the age of the universe to reach equilibrium. The problem with this choice (p_μ) of acceptance probability is that it only depends on the final state. The point of the Metropolis algorithm is to accept the new state with a probability that depends on the energy difference between the initial and final states.

The Metropolis algorithm uses the acceptance probability:

$$A(\nu \to \mu) = \begin{cases} e^{-(E_\mu - E_\nu)/kT}, & \text{if } E_\mu - E_\nu > 0 \\ 1, & \text{if } E_\mu - E_\nu \leq 0 \end{cases}, \tag{2.24}$$

if states μ and ν differ by one spin flip. $A(\nu \to \mu) = 0$ if states μ and ν differ by more than one spin flip. From here onwards, I will drop the qualifier that $A(\nu \to \mu) = 0$ if states μ and ν differ by more than one spin flip, since all the algorithms I describe in this thesis are of the single-spin-flip type.

The Metropolis algorithm accepts all transitions that lower the lattice energy and accepts transitions that increase the energy with Boltzmann probability (figure 2.3). If the forward transition is $E_\mu - E_\nu > 0$, then the backward transition must be $E_\mu - E_\nu < 0$, and *vice versa*. Therefore, the ratio between the two acceptance probabilities in equation 2.24 satisfies detailed balance and the Boltzmann condition (equation 2.23).

In the zero-temperature limit, the acceptance probability becomes a step function. The Metropolis algorithm is the most efficient single-spin-flip algorithm. Other

multiple-spin-flip algorithms that flip a cluster of spins at a time may be more efficient but I will not discuss them here.

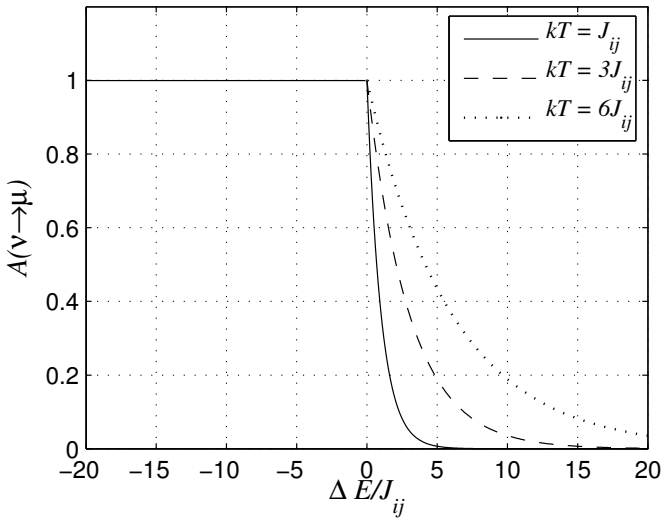


Figure 2.3. Metropolis acceptance probabilities at three different temperatures (in units of J_{ij}). The interaction range is up to fourth-nearest neighbor, $z = 20$.

Because of the definition of the Metropolis acceptance probabilities (equation 2.24), with J_{ij} positive, like spins tend to congregate together in order to minimize the number of mismatched links. The effect of the algorithm on a domain of uniform spin within another spin is thus similar to the effect of surface tension on an interface between two immiscible liquids. The Metropolis algorithm and the Ising model together contain the two basic components for simulating liquid interfaces: a restoring force (of energy scale J_{ij}) and a disrupting force (of energy scale kT).

The full Metropolis algorithm consists of the following steps:

1. Randomly select a pixel.
2. Flip the pixel's spin ($0 \rightarrow 1, 1 \rightarrow 0$).
3. Calculate the resultant change in energy.
4. Accept the spin flip with probability given by equation 2.24.
5. Repeat steps 1 to 4.

Randomness occurs in the Metropolis algorithm in steps 2 and 4. Every time we perform steps 2 and 4, we use a pseudo-random-number generator to randomly select a pixel and generate a random number between 0 and 1. If the random number is smaller than the probability calculated in step 4, we accept the new spin, otherwise, the pixel reverts to its previous spin. I use a Lagged Fibonacci pseudo-random-number generator to generate the random numbers for my simulations [64, 103]:

$$x_n = (x_{n-55} - x_{n-24}) \mod m, \tag{2.25}$$

where $m = 2^{30} - 1$. Two previous random numbers (x_{n-55}, x_{n-24}) generate each new random number (x_n). The value of x_n lies between 0 and $m - 1$. To produce a number between 0 and 1, I divide x_n by $m - 1$.

Each iteration of the Metropolis algorithm counts as a spin-flip *attempt*. The basic unit of time in Monte-Carlo simulations is the *Monte-Carlo step*. On a lattice of size $L \times L$, we say a Monte-Carlo step has elapsed after $L \times L$ spin-flip attempts. I express all simulation times as multiples or fractions of Monte-Carlo steps.

2.7 Generalized Perron and Frobenius Theorems

The elements of a stochastic matrix satisfy the sum rule:

$$\sum_{\mu}^{M} P(\nu \to \mu) = 1. \tag{2.26}$$

The Metropolis transition probabilities, however, do not add up to 1 (Since a lattice can transform into any one of $L \times L$ states in a single spin-flip attempt, $M = L^2$):

$$\begin{aligned} \sum_{\mu=1}^{L^2} P(\nu \to \mu) &= \frac{1}{L^2} \sum_{\mu=1}^{L^2} A(\nu \to \mu) \\ &\leq \frac{1}{L^2} \sum_{\mu=1}^{L^2} 1 \\ &\leq 1. \end{aligned} \tag{2.27}$$

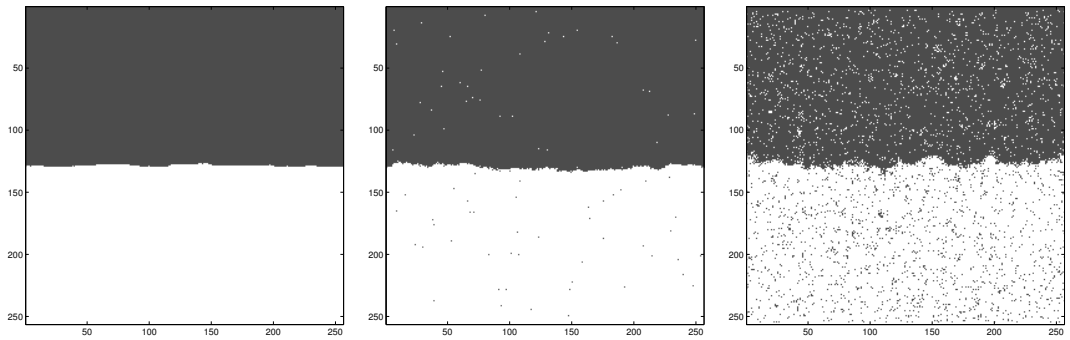
The reason for this discrepancy is that $P(\nu \to \mu)$ is only defined up to a ratio (equation 2.23). The transition matrix is therefore not a true stochastic matrix. Fortunately, all is not lost. All the preceding discussions about stochastic matrices still hold, because the Perron and Frobenius theorems hold for general non-negative and positive-definite matrices [5]. In general cases, the largest eigenvalue is not 1, but $\lambda_m > 0$. The theorems in section 2.5 also hold for general non-negative and positive matrices with the eigenvalue λ_m replacing the unit eigenvalue.

2.8 Modified Metropolis Algorithms

I have shown that achieving a Boltzmann distribution in Monte-Carlo simulations is not trivial. The two paramount requirements are detailed balance and ergodicity– both of which the Metropolis algorithm includes. Thus the Metropolis algorithm leads to a unique equilibrium distribution of lattice states, p_μ.

I use a modified version of the Metropolis algorithm to simulate interfaces and Brownian motion for two reasons.

For fourth-nearest-neighbor interactions, the coordination number is 20, therefore, the possible energy change due to a single spin flip ranges from $-20J_{ij}$ to $+20J_{ij}$. The two extreme cases correspond to the vanishing of a lone spin and the appearance of a lone spin in a domain of another spin. The appearance of a lone spin is also called *nucleation*. The two transitions are complimentary: if the forward transition is nucleation, the backward transition is disappearance, and *vice versa*. Nucleations cause bulk fluctuations within regions of homogeneous spins. At low temperatures, the probability of nucleations is low, but for temperatures near $6J_{ij}$, their probability is not negligible (figure 2.3). At low temperatures, nucleated domains are small, transient and far apart. At high temperatures, nucleated domains may have significant densities and grow to large sizes (figures 2.4).



(a) $kT = J_{ij}$ (b) $kT = 3J_{ij}$ (c) $kT = 6J_{ij}$

Figure 2.4. Bulk fluctuations at three different temperatures. The interaction range is up to fourth-nearest neighbors.

Nucleations are undesirable because they blur the line between the interface and the bulk and complicate the calculation of interfacial energy.

Nucleation is also undesirable in simulating Brownian motion because nucleation may cause the center of mass of a domain to jump a large distance after a single spin flip. Modifications of the Metropolis algorithm are thus necessary in order to eliminate nucleation. Are other Monte-Carlo algorithms better suited to this problem?

The Kawasaki algorithm is a *single-spin-exchange* algorithm as opposed to the Metropolis algorithm which is a single-spin-flip algorithm. In the Kawasaki algorithm, two pixels exchange spins. conserving the total spin of the lattice. The Kawasaki algorithm preserves detailed balance because neither true nucleation nor disappearance events can occur. If the two pixels that exchange spins are not *neighboring* (defined as within the interaction range) to each other, then "nucleation" exists (non-local spin exchange). Exchange of neighboring pixels' spins cannot cause nucleation (local spin exchange). In any case, a series of local spin-exchanges could result in a lone heterogeneous spin in a region of differing spin. Because the total spin is conserved, the lone spin will persist for much longer time than it would in the

Metropolis algorithm. Lone spins will cause errors in calculating the interface width and the determination of the center-of-mass of droplets. Therefore, the Kawasaki algorithm is not suitable for the study of surface tension and diffusion constants.

The Swendsen-Wang, Wolff, and Niedermayer algorithms are all *cluster-flip* algorithms [90]. These algorithms select a region of contiguous and homogeneous spins and flip the spin of the entire region. These algorithms are unsuitable because they may dramatically increase or decrease the size of a droplet which my Brownian-motion simulations must conserve.

I thus have chosen to study three different modifications of the Metropolis algorithm, all of which reduce nucleation. The three modifications suppress nucleation to varying degrees. I will investigate the effect of the degree of suppression on the surface tension and diffusion constant. I will also briefly discuss the consequences of suppressing nucleation on detailed balance and ergodicity.

2.8.1 Algorithm One

My first modification is the simplest. I simply set the acceptance probability of a nucleation event to zero:

$$A(\Delta E = +zJ_{ij}) = 0. \tag{2.28}$$

All other transition probabilities remain the same. Otherwise, I follow the Metropolis algorithm. Due to this modification, the nucleation and disappearance transitions no longer satisfy detailed balance:

$$\frac{A(\Delta E = +zJ_{ij})}{A(\Delta E = -zJ_{ij})} = \frac{0}{1} \neq e^{-zJ_{ij}/kT}. \tag{2.29}$$

All other transitions still satisfy detailed balance. Therefore, suppression of nucleation is tantamount to violation of detailed balance in the Metropolis algorithm.

All the transitions within an interaction range of the interface are necessarily not

nucleation events.[3] At low temperatures, when nucleations are small, transient and far from the interface, suppressing nucleation while leaving other transitions intact should not affect the interface. At high temperatures, however, bulk fluctuations (caused by nucleations) are big and close to the interface. If a bulk fluctuation comes to within an interaction range of the interface, it becomes part of the interface. Therefore, at high temperatures, I cannot ignore the effects of bulk fluctuations on the dynamics of the interface.

Next, I estimate the highest temperature at which I can ignore bulk fluctuations without affecting the equilibrium state of the interface. For a fourth-nearest-neighbor interaction, $z = 20$, and, at $kT = 6J_{ij}$, the probability of nucleation (when it is unsuppressed) is $\exp(-20/6) = 0.036$. However, the probability of bulk fluctuations is actually higher, because a *nucleus* (a lone heterogeneous spin) may induce its neighboring pixels to switch spin. These transitions are, by definition, not nucleations, but they contribute to bigger bulk fluctuations. I have calculated the actual density of heterogeneous spins at $kT = 6J_ij$ to be about 0.051 ± 0.001, so on average, a 10×10 region of uniform spin has roughly 5 heterogeneous spins. Assuming uniform distribution of nucleation, there is roughly 1 heterogeneous spin per 20 uniform spins, and thus the heterogeneous spins are beyond fourth-nearest neighbors of each other. At higher temperatures, on average, the bulk fluctuations will be within an interaction range of each other and the interface. Therefore, for fourth-nearest-neighbor interactions, I can ignore the effects of bulk fluctuations on the interface up to $kT \simeq 6J_{ij}$.

Since an interface can transform into any other interface via a series of spin flips, all interface states are accessible and the set of all interface states is ergodic. When I talk about equilibrium, I mean that the interface is at equilibrium. Henceforth, I

[3]By definition, nucleation means a spin flip with $\Delta E = zJ_{ij}$. Within an interaction range of the interface, ΔE is always less than zJ_{ij}.

refer to the interface equilibrium distribution as simply the *equilibrium distribution*.

2.8.2 Algorithm Two

My second method for suppressing nucleations is to account for the pixel's surroundings. In Algorithm One, except for nucleation, I allow the pixel to flip its spin regardless of its surrounding spins. In Algorithm Two, I only allow it to flip to spins chosen randomly from its neighbors. If the pixel and its surrounding spins are all the same, it will never flip, preventing nucleation. The transition probability (equation 2.21) then becomes:

$$P(\nu \to \mu) = \frac{1}{L^2}\frac{n}{z}A(\nu \to \mu), \tag{2.30}$$

where n is the number of neighboring pixels with different spin and z is the coordination number. The backward transition probability is therefore:

$$P(\mu \to \nu) = \frac{1}{L^2}\frac{z-n}{z}A(\mu \to \nu). \tag{2.31}$$

The energy change in the forward direction is:

$$\Delta E = (z - 2n)J_{ij}. \tag{2.32}$$

The energy change in the backward direction is $-\Delta E$ from equation 2.32. By writing n in terms of z and ΔE, the ratio of the two transition probabilities becomes:

$$\frac{P(\nu \to \mu)}{P(\mu \to \nu)} = \frac{z - \Delta E}{z + \Delta E}\frac{A(\nu \to \mu)}{A(\mu \to \nu)}. \tag{2.33}$$

Obviously, if I continue to use the Metropolis acceptance ratio of equation 2.23, equation 2.33 does not satisfy detailed balance (equation 2.22), so I do not reach a Boltzmann equilibrium distribution of states. In order to restore detailed balance (minus nucleation), I redefine $A(\nu \to \mu)$ to be:

$$A(\nu \to \mu) = \begin{cases} \frac{z+\Delta E}{z-\Delta E}e^{-\Delta E/kT}, & \text{if } \Delta E > 0 \\ 1, & \text{if } \Delta E \leq 0 \end{cases}. \tag{2.34}$$

With this compensation, the ratio in equation 2.33 is again $\exp(-(E_\mu - E_\nu)/kT)$. The case $\Delta E = zJ_{ij}$ never occurs because it corresponds to a nucleation event. Since I reject nucleation events, the next highest-energy transition is $\Delta E = 18J_{ij}$. Simple calculation shows that $A(\Delta E = 18) > 1$ when $kT > 6.11J_{ij}$. This temperature is close to the limit in section 2.8.1 where suppression of nucleations begins to affect the dynamics of the interface. For temperatures above $6.11J_{ij}$, I have to renormalize all $A(\nu \to \mu)$ so that the highest probability is equal to 1. Since I do not intend to simulate at temperatures higher than $6J_{ij}$, I will use equation 2.34 as it stands. Figure 2.5 shows the modified acceptance probabilities.

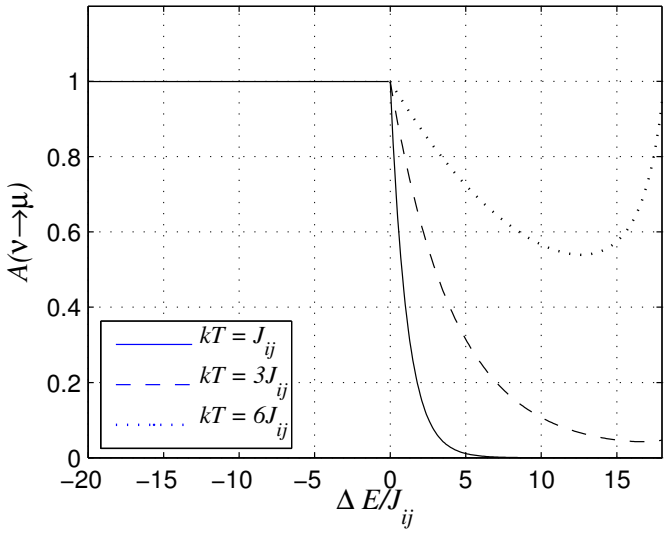


Figure 2.5. Acceptance probability for Algorithm Two (equation 2.34), with $z = 20$.

With this modification, I expect the equilibrium-interface configurations to be the same as for Algorithm One. The interface states are also ergodic, since any interface can transform into any other interface via a series of spin flips.

In summary, for Algorithm Two, we

1. Randomly select a pixel. Call this pixel the candidate pixel.
2. Randomly select a pixel from the candidate pixel's neighbor list. Call this the target pixel.[4]

[4]Some researchers, for example Glazier [45] use the opposite definitions for the target and candidate pixels.

3. Flip the candidate pixel's spin to the target pixel's spin.
4. Calculate the energy change.
5. Accept the spin flip with the probability given by equation 2.34.
6. Repeat steps 1 to 5.

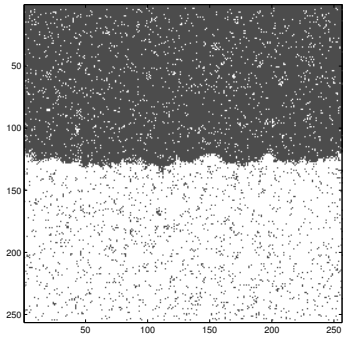
2.8.3 Algorithm Three

The previous two sections showed that we can suppress nucleations in multiple ways. Different methods violate detailed balance to varying degrees. Algorithms One and Two violate detailed balance just enough to suppress nucleations. For low temperatures, they do not affect the equilibrium distribution of interface configurations. For comparison, I propose a third modification that violates detailed balance to a greater degree.

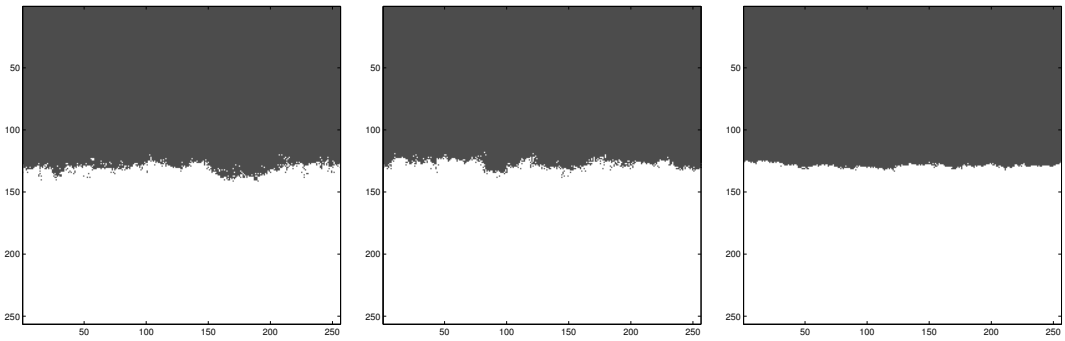
Algorithm Three is exactly the same as Algorithm Two except that I use the original Metropolis acceptance probability (equation 2.24). As I pointed out in the previous section, the resultant interface equilibrium is not a Boltzmann equilibrium.

Many investigators have used this algorithm in conjunction with the Potts model (where the spins can assume more than two possible values) to model cell sorting [47, 48], foams [46, 63, 74, 117], *etc.*. However, none investigated the consequences of ignoring detailed balance. Restoring detailed balance in the Potts model is harder than in the Ising model, although one researcher has considered this restoration briefly for the Potts model [121]. Thus I expect my findings on the algorithmic dependence of the Ising Model to interest those using Potts models to simulate biological cells, as I discuss in chapter 5.

Figure 2.6 shows snapshots of interfaces for the three modified algorithms and the Metropolis algorithm for the same temperature and interaction range. Even without nucleation, some isolated heterogeneous spins are still quite visible near the interface for Algorithms One and Two. These isolated spins do not result from



(a) Metropolis algorithm



(b) Algorithm One (c) Algorithm Two (d) Algorithm Three

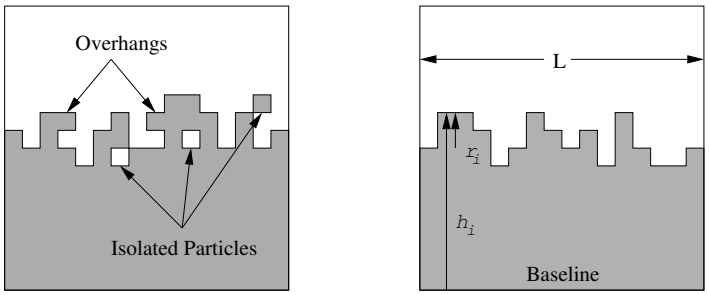
Figure 2.6. A comparison between interface shapes for different algorithms. The temperature is $6J_{ij}$. The interaction range is up to fourth-nearest neighbors.

nucleations, but occurr because all but one spin in an initial region of uniform spin have switched to a different spin, except for one in the middle. Unlike in the Kawasaki algorithm, an isolated spin is short lived in the Metropolis algorithm.

CHAPTER 3

SOLID-ON-SOLID MODEL

In the first chapter, I discussed the thermodynamics of surfaces. In the second chapter, I presented a method to simulate those surfaces. In this chapter, I present a model that I use to fit my simulation results. The model is the *Solid-On-Solid* model, or *SOS* for short. I will derive the interface energy of the SOS model at different temperatures and angles. The excellent reviews of Leamy *et al.* [71] and Burton *et al.* [18] further discuss the SOS model in the context of solid surfaces and crystal growth.



(a) (b)

Figure 3.1. (a) An interface with overhangs and isolated particles, and (b) an interface with no overhangs or isolated particles.

Like the Ising model, the SOS model is a discrete model of surfaces. The SOS model avoids overhangs and isolated particles on the interface (figure 3.1(a)). The SOS model can approximate the Ising model at low temperatures [19].

The heights of the interface from the baseline at each horizontal position: $h_i, i = 1, \ldots, L$ fully describe the profile of the interface. The absence of overhangs and isolated particles makes h_i a single-valued function. Each set of $\{h_i\}$ describes a unique interface profile. Instead of working with heights, however, I choose to work with step-sizes $r_i = h_i - h_{i-1}$, because the interface energy directly relates to r_i. Before I define the energy of an interface, I must define the energy of a step.

3.1 Step Energies

In the Ising model, each pixel has an energy. Here, I extend the definition to include the energy of a step. Suppose that the coupling constant between first-nearest neighbors is J_1 and between second-nearest neighbors is J_2. A pixel lying next to a horizontal interface has one first-nearest neighbor and two second-nearest neighbors that differ from it in spin (figure 3.2(a)). Therefore, its energy is $J_1 + 2J_2$.

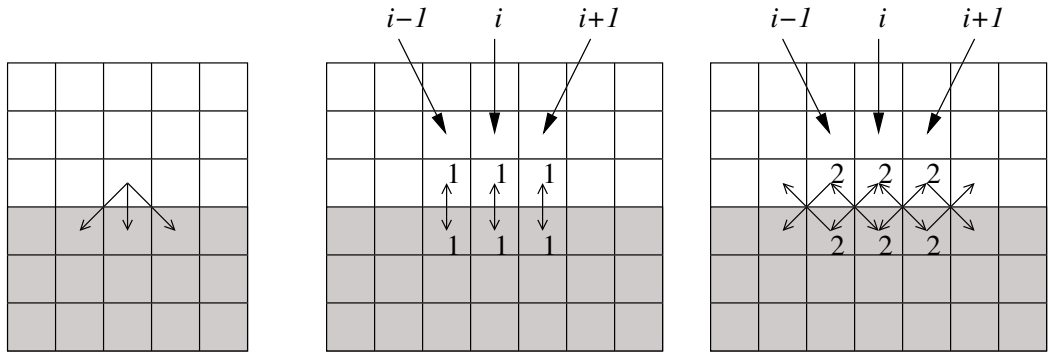


Figure 3.2. (a) First- and second-nearest neighbors (arrows). (b) Energies from first- (left) and second- (right) nearest-neighbor interactions. The energy of a pixel corresponds to the number of arrows pointing away from that pixel. I only show the neigbhbors of some surface pixels, that is why some arrows are single-headed, while others are double-headed.

Figures 3.2(b) show the energies of surface pixels for three successive positions:

$i-1$, i and $i+1$. The energy of a zero-height step, $\varepsilon_{r=0}$, is thus equal to $J_1 + 2J_2$ for either side of the interface or $2J_1 + 4J_2$ for both sides of the interface. Next, consider steps of height one, two or three pixels (figures 3.3):

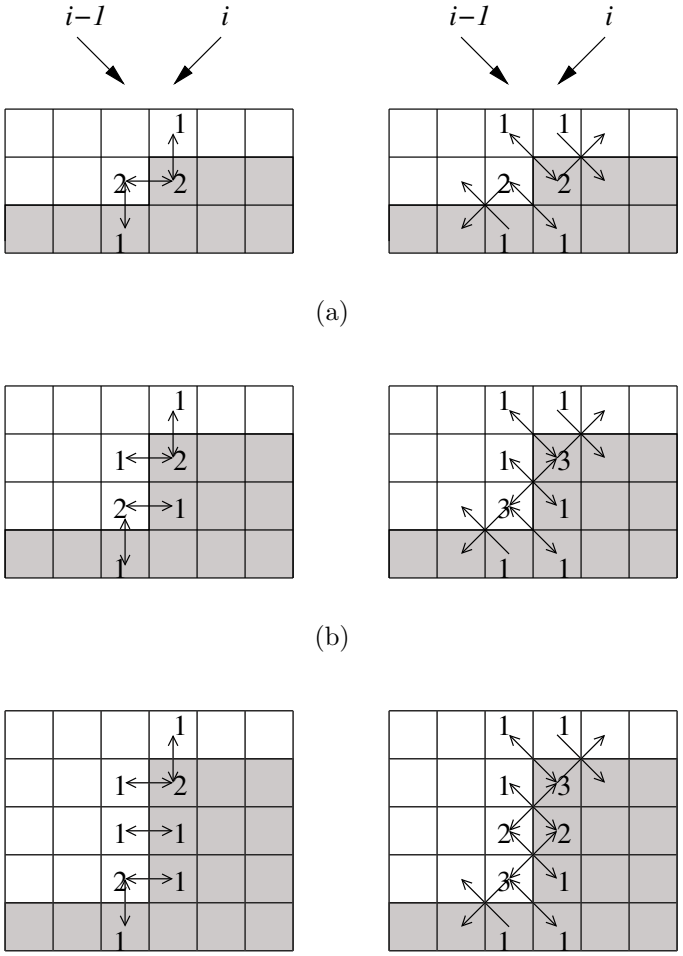


Figure 3.3. Diagrams depicting the energies of steps of height (a) one pixel ($r_i = 1$), (b) two pixels ($r_i = 2$) and (c) three pixels ($r_i = 3$). Left diagrams: energies from first-nearest-neighbor interactions. Right diagrams: energies from second-nearest-neighbor interactions. By definition, $r_{i-1} = 0$. Only neighbors of surface pixels nearest to the step are shown, that is why some arrows are single-headed and some are double-headed.

From here on, I will always give the energies from both sides of interface. The

steps depicted in figures 3.3 have energies:

$$\begin{aligned} \varepsilon_{r=0} + \varepsilon_{r=1} &= 6J_1 + 8J_2, \\ \varepsilon_{r=0} + \varepsilon_{r=2} &= 8J_1 + 12J_2, \\ \varepsilon_{r=0} + \varepsilon_{r=3} &= 10J_1 + 16J_2. \end{aligned} \tag{3.1}$$

Reflecting the interface along a normal, $r \to -r$, the energies do not change, so $\varepsilon_{+r} = \varepsilon_{-r}$. By using $\varepsilon_{r=0} = 2J_1 + 4J_2$, the diligent reader may prove to herself that the energies of the six smallest steps are those in table 3.1.

TABLE 3.1

STEP ENERGIES FOR DIFFERENT STEP SIZES FOR UP TO SECOND-NEAREST-NEIGHBOR INTERACTIONS.

r	ε_r
0	$2J_1 + 4J_2$
±1	$4J_1 + 4J_2$
±2	$6J_1 + 8J_2$
±3	$8J_1 + 12J_2$
±4	$10J_1 + 16J_2$
±5	$12J_1 + 20J_2$
±6	$14J_1 + 24J_2$

All $\varepsilon_{r\neq0}$ obey the same equation:

$$\varepsilon_{r\neq0} = (2 + 2|r|)J_1 + 4|r|J_2. \tag{3.2}$$

By defining:

$$C_r = \begin{cases} 1, & \text{if } r = 0 \\ 0, & \text{if } r \neq 0 \end{cases}, \tag{3.3}$$

the equation for step energy of any size becomes:

$$\varepsilon_r = (2 + 2|r|)J_1 + 4(|r| + C_r)J_2. \tag{3.4}$$

At this point, we may be tempted to extend the definition of step energies to longer-range interactions. Unfortunately, for third-nearest-neighbor interactions

or longer, the step energy depends not only on the height of a step but also on the height of neighboring steps. For example, if we consider up to third-nearest-neighbor interactions, a step sequence of $r = \{0, 0, 3, 2, 0\}$ gives a different energy from $r = \{0, 0, 3, -2, 0\}$. In the first case, we obtain $\varepsilon_{total} = 20J_1 + 32J_2 + 40J_3$, in the second case, $\varepsilon_{total} = 20J_1 + 32J_2 + 32J_3$. Long-range interactions depend on the local slope, *i.e.* neighboring steps. The first sequence has a greater slope than the second sequence. The greater the local slope, the greater the energy from long-range interactions.

Another way of looking at this problem is to consider interfaces $r = \{1, 1, 1, 1, \ldots\}$ and $r = \{1, -1, 1, -1, \ldots\}$. The mean slope of the former interface is 45°, of the latter 0°, but equation 3.4, which does not include third-nearest-neighbor interactions gives the same energy for both. I shall explain later how I adapt equation 3.4 to include longer-range interactions and to include the dependence on average slope.

3.2 Partition Function of the SOS Model

We can now calculate the partition function of an interface in the SOS model. Suppose:

$$n_r = \text{number of steps of height } r \text{ pixels}, \tag{3.5}$$

where $r = 0, \pm 1, \pm 2, \ldots$. Since the horizontal length of the lattice is L, the values of n_r range from 0 to L. If $n_r = 0$, then no steps have height r pixels. If $n_r = L$, all steps are r pixels high. The sum of all n_r must equal the lattice width:

$$\sum_{r=-\infty}^{\infty} n_r = L. \tag{3.6}$$

The total energy of the interface is:

$$E = \sum_{r=-\infty}^{\infty} \varepsilon_r n_r, \tag{3.7}$$

where equation 3.4 gives ε_r. The number of ways to arrange L steps, n_0 of which are flat, n_1 of which are of plus one pixel, n_{-1} of which are of minus one pixel, ..., is:

$$C^L_{\{n_r\}} = \frac{L!}{n_0!n_1!n_{-1}!n_2!n_{-2}!\ldots}. \tag{3.8}$$

Therefore, the partition function is:

$$Z = \sum_{\{n_r\}=0}^{L} C^L_{\{n_r\}} e^{-E(\{n_r\})/kT}, \tag{3.9}$$

where,

$$\sum_{\{n_r\}=0}^{L} \equiv \sum_{n_0=0}^{L}\sum_{n_1=0}^{L}\sum_{n_{-1}=0}^{L}\sum_{n_2=0}^{L}\sum_{n_{-2}=0}^{L}\ldots, \tag{3.10}$$

and equation 3.7 gives $E(\{n_r\})$. The partition function consists of a configuration term $C^L_{\{n_r\}}$ (similar to the one I discussed in section 1.1) and an internal energy term E. Therefore, the surface tension has an entropic contribution and an internal-energy contribution. If we further define:

$$\zeta_r \equiv e^{-\varepsilon_r/kT}, \tag{3.11}$$

we may write the partition function as:

$$Z = \sum_{\{n_r\}=0}^{L} \left(C^L_{\{n_r\}} \prod_{r=-\infty}^{\infty} \zeta_r^{n_r} \right). \tag{3.12}$$

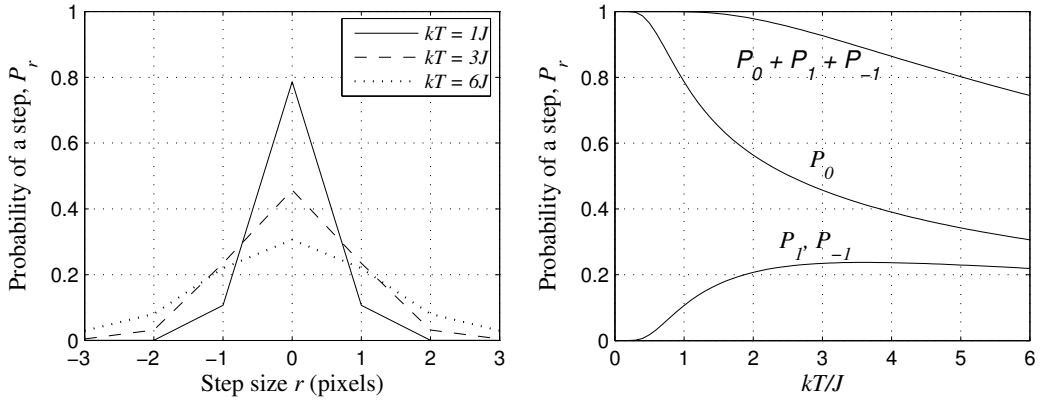
In view of the constraint on n_r (equation 3.6), equation 3.12 is just the multinomial expansion of ζ_r. Therefore,

$$\begin{aligned} Z &= (\zeta_0 + \zeta_1 + \zeta_{-1} + \zeta_2 + \zeta_{-2} + \ldots)^L \\ &= \Xi^L, \\ \Xi &\equiv \sum_{r=-\infty}^{\infty} \zeta_r, \end{aligned} \tag{3.13}$$

where Ξ is the partition function per unit horizontal length of the interface, *i.e.* the partition function of each step. The probability for a step r pixels high is:

$$P_r = \frac{\zeta_r}{\Xi}. \tag{3.14}$$

Because $\varepsilon_{+r} = \varepsilon_{-r}$, $P_{+r} = P_{-r}$ (figure 3.4(a)). Figure 3.4(b) shows the probabilities P_0, P_1, P_{-1} as a function of temperature. At zero temperature, the interface is in its ground state, *i.e.* horizontal. As the temperature increases, the interface begins to fluctuate, but because of the P_r symmetry, the interface remains horizontal on average.



(a) (b)

Figure 3.4. (a) Probability of an r-pixel-height step. (b) Probabilities of the three smallest steps as a function of temperature, for $J_1 = J_2 = J$. Equation 3.17 gives Ξ, the partition function.

An interface with $r_{45} = \{1, 1, 1, 1, \ldots\}$ has the same energy as one with $r_0 = \{1, -1, 1 - 1, \ldots\}$. However, only one combination of steps gives a 45° interface, while $\frac{L!}{2(L/2)!}$ combinations of steps can produce a 0° interface. Therefore, the density of states (equation 3.8) peaks sharply at zero angle.

Substituting equations 3.4 and 3.11 into equation 3.13 and noting that $\varepsilon_{+r} = \varepsilon_{-r}$, we find:

$$\Xi = e^{-(2J_1+4J_2)/kT} + 2\sum_{r=1}^{\infty} e^{-[(2+2r)J_1+4rJ_2]/kT}. \tag{3.15}$$

If we define:

$$\gamma_1 \equiv e^{-J_1/kT},$$
$$\gamma_2 \equiv e^{-J_2/kT}, \tag{3.16}$$

then Ξ simplifies to:

$$\begin{aligned}
\Xi &= \gamma_1^2\gamma_2^4 + 2\gamma_1^2 \sum_{r=1}^{\infty} \left(\gamma_1^2\gamma_2^4\right)^r \\
&= \gamma_1^2\gamma_2^4 + 2\gamma_1^2 \left(\frac{1}{1-\gamma_1^2\gamma_2^4} - 1\right) \\
&= \frac{\gamma_1^2\gamma_2^4(1-\gamma_1^2\gamma_2^4+2\gamma_1^2)}{1-\gamma_1^2\gamma_2^4}.
\end{aligned} \tag{3.17}$$

Equation 3.17 is the partition function for a single step in the SOS model on a square lattice with interactions up to second-nearest neighbors.

The free energy, surface tension, entropy, internal energy and the specific-heat capacity of the horizontal interface are:

$$\begin{aligned}
F &= -kT\ln Z = -LkT\ln\Xi, \\
\sigma &= F/L = -kT\ln\Xi \\
&= 2J_1 + 4J_2 - kT\ln\left(\frac{1-\gamma_1^2\gamma_2^4+2\gamma_1^2}{1-\gamma_1^2\gamma_2^4}\right), \\
U &= -T^2\frac{\partial(F/T)}{\partial T} = kT^2\frac{\partial}{\partial T}\ln Z = \langle E\rangle \\
&= L\left[2J_1 + 4J_2 + \frac{4J_1\gamma_1^2 + 8J_2\gamma_1^4\gamma_2^4}{(1-\gamma_1^2\gamma_2^4)(1-\gamma_1^2\gamma_2^4+2\gamma_1^2)}\right], \\
S &= -\frac{\partial F}{\partial T} \\
&= k\left(\ln Z + T\frac{\partial}{\partial T}\ln Z\right) = \frac{U-F}{T}, \\
C_v &= \frac{\partial U}{\partial T} \\
&= k\frac{\partial}{\partial T}\left(T^2\frac{\partial}{\partial T}\ln Z\right) = \frac{1}{kT^2}(\langle E^2\rangle - \langle E\rangle^2).
\end{aligned} \tag{3.18}$$

Since the energy of a step depends only on its size and not on the neighboring steps, the steps do not interact with each other. Just like ideal gas molecules in a closed container in thermal contact with a heat reservoir, the steps reach equilibrium with the heat reservoir independently. The ensemble of steps thus behaves like an ideal gas. However, unlike ideal gas molecules, the fixed horizontal position of steps

makes them distinguishable. Therefore, the partition function of the entire interface is just the product of the partition functions for each step (equation 3.13).

Because each bond or link connects a spin-up pixel to a spin-down pixel, the total energy of both sides of the interface is exactly twice the energy of either side. Therefore, the partition function for both spins is equal to the product of the partition functions for up spins and down spins:

$$Z_{up+down} = Z_{up} \times Z_{down} = Z_{up}^2 = Z_{down}^2. \tag{3.19}$$

3.3 Special Cases of the SOS Model

Comparing the SOS model with known results for the Ising model is instructive. For nearest-neighbor interactions, $J_1 = J, J_2 = 0$. Onsager and most investigators untill this day consider only this case. If we define $K \equiv \frac{J}{kT}$, equations 3.18 then simplify to:

$$\begin{aligned} \Xi &= \gamma_1^2 \coth K, \\ \sigma &= 2J + kT \ln(\tanh K), \\ S/L &= k[2K\,\mathrm{csch}\, 2K - \ln(\tanh K)], \\ U/L &= 2J(1 + \mathrm{csch}\, 2K), \\ C_v/L &= kK^2 \mathrm{csch}\, 2K \coth 2K. \end{aligned} \tag{3.20}$$

Figure 3.5 shows the dependencies of these quantities on temperature.

The results agree with the treatment in [75] which obtained the partition function by summing over r_i instead of n_r. A few lines of calculation show that $\sigma = 0$ at $kT_c = 2.269J$. Amazingly, the surface tension in equation 3.20 coincides with Onsager's exact result for $T \leqslant T_c$, supporting my earlier observation that at low temperatures, bulk fluctuations have negligible effect on the interface equilibrium

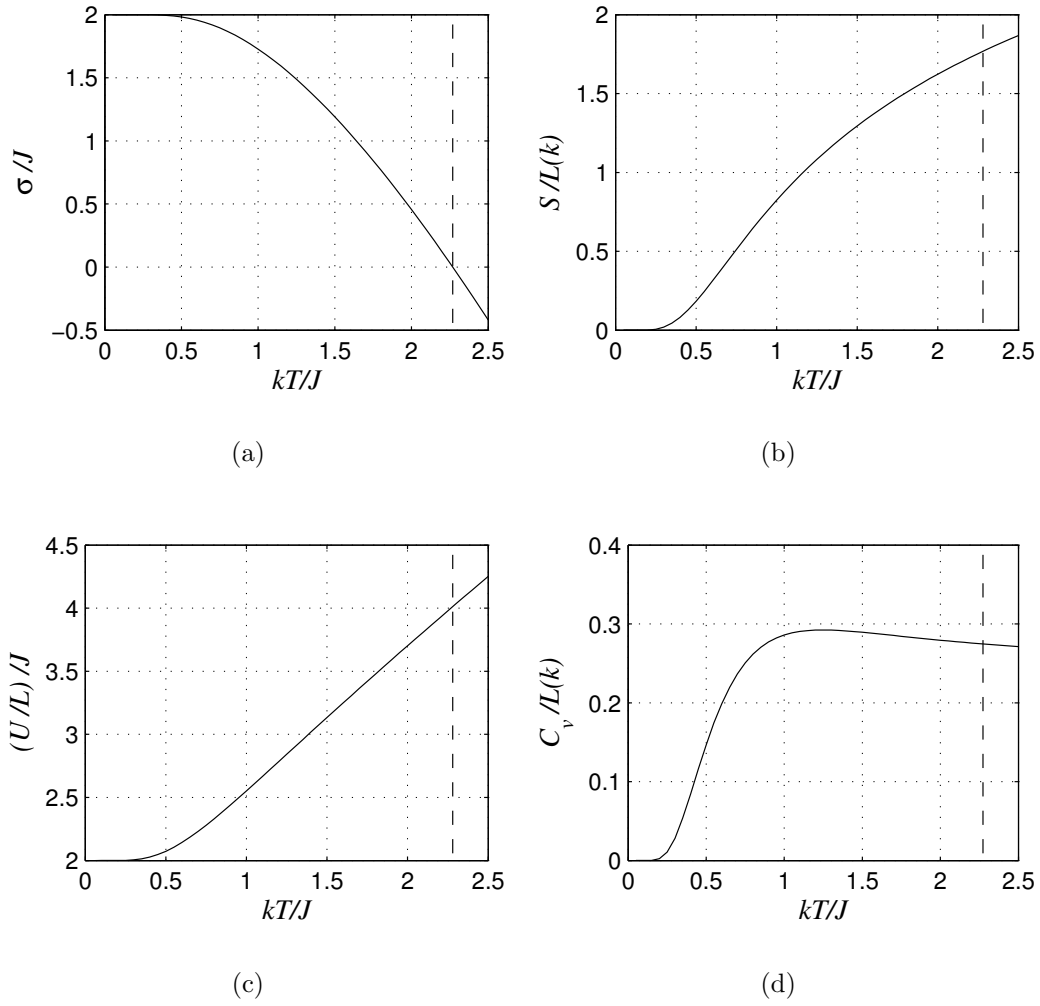


Figure 3.5. Temperature dependence of thermodynamic variables for the first-nearest neighbor SOS model. (a) Surface tension σ, (b) entropy per unit length S/L, (c) energy per unit length U/L and (d) heat capacity per unit length C_v/L for a horizontal interface. Division by the energy parameter J normalizes all energies to be dimensionless by. Dashed lines indicate the critical temperature kT_c.

(section 2.8.1). However, the similarity between the SOS model and the Ising model ends here.

In the Ising model, the surface tension is zero above the critical point $kT_c = 2.269J$, but from figure 3.5(a), the surface tension for the SOS model is negative for $kT > kT_c$, which is unphysical. The entropy per unit length, S/L, in figure 3.5(d), does not show any particular problem for $kT > kT_c$, but the energy per

unit length, U/L does, since in the Ising model, it can never exceed $4J$ (when all the first-nearest neighbors are of a different spin). In the SOS model, the step size has no limit, consequently, the energy per step does not have a maximum value and $U/L > 4J$ for $kT > kT_c$. Finally, the specific-heat capacity for the Ising model diverges at kT_c, indicating a second-order phase transition. The specific-heat capacity for the SOS model remains finite and analytic at kT_c, and does not even show a peak at kT_c. The SOS model shows no signature of a phase transition.

The similarity of the surface tensions in the two models (up to kT_c) contrasts with the differences in their other thermodynamic quantities. If we examine equations 3.18, surface tension is a function of the surface free energy and the length of the interface. All other quantities depend only on the surface free energy. The differences therefore lie in the definition of the length of the interface. The SOS model assumes that the interface length is equal to the lattice length. In the Ising model, the length increases with temperatures. The ratio between the free energy and the interface length for the SOS model fortuitously coincides with the derivative of the free energy with respect to the interface length for the Ising model.

The behavior of σ_{SOS} differs qualitatively from that for a liquid. If we compare figure 3.5(a) with figure 1.4(a), the surface tension for the SOS model (figure 3.5(a)) is a convex function, but the surface tension of liquids (figure 1.4(a), with the exponent $\mu \simeq 1.28$) is a concave function. μ is the exponent in equation 1.28.

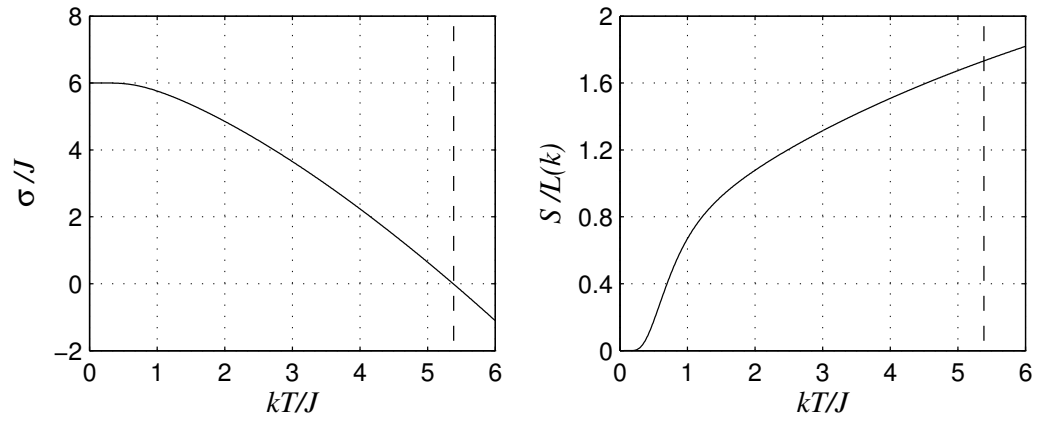
The internal energy per step (figure 3.5(c)), is similar in form to the internal energy of a chain of harmonic oscillators, which is not surprising, since the step energy ε_r (equation 3.4) is similar in form to the energy of the simple harmonic oscillator, $\varepsilon_n = (n + \frac{1}{2})\hbar\omega$, when $J_2 = 0$. In this respect, the SOS model is similar to a chain of harmonic oscillators.

We may argue a heuristically concerning the effect of nucleations on surface

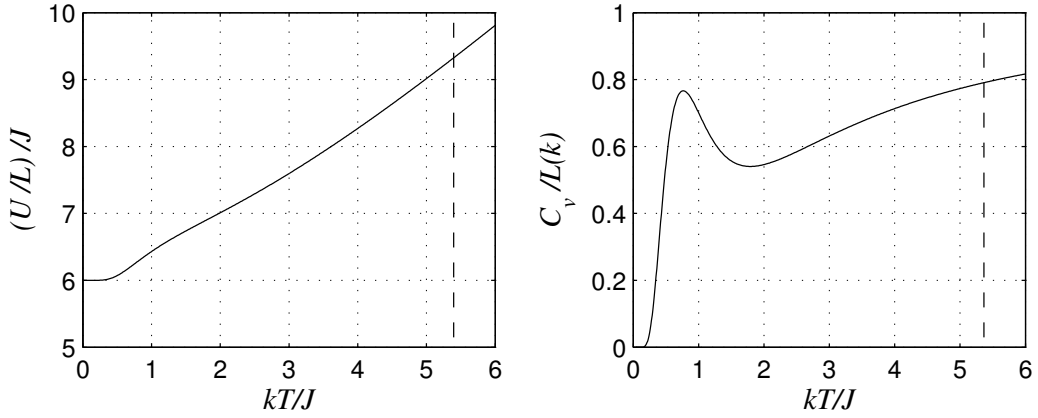
dynamics at low temperatures. In first-nearest-neighbor interactions, a nucleation event increases the energy by $4J$, and at $kT = 2.269J$, the probability of nucleation is 0.17, which is about 1 heterogeneous spin per 6 pixels. For first-nearest-neighbor interactions, each pixel has only 4 neighbors, therefore, on average, nucleation events are beyond an interaction range of each other and they do not interact. The SOS model ignores both nucleations and overhangs. Previously, Hartmann and Zittartz [86] observed that they could obtain the bulk free energy by taking into account only a subset of interface configurations (those without overhangs). The reason the results coincide is unknown [19]. I will show later (section 3.6) for non-zero-angle interfaces, that the surface tension differs from the one predicted from the Ising model.

Another interesting special case that of the second-nearest-neighbor interactions with equal coupling constants for all neighbors ($J_1 = J_2 = J$). Figure 3.6 shows the corresponding plots for the surface tension σ, entropy per unit length S/L, energy per unit length U/L and heat capacity per unit length C_v/L. Solving $\sigma = 0$ numerically, gives $kT_c = 5.379J$.

The behaviors of the SOS model for nearest-neighbor (figures 3.5) and the second-nearest-neighbor (figures 3.6) interactions share some common characteristics. In both cases, $\sigma \to 0$ as $T \to T_c$ (T_c is higher for second-nearest-neighbor interactions) and the specific heat of the interface does not diverge at T_c. The only notable difference between the two cases is the low-temperature peak in the specific heat at $kT \simeq 0.8J$ (figure 3.6(d)). Exploring whether this peak exists for longer-range interactions or other types of lattice would be interesting.



(a) (b)



(c) (d)

Figure 3.6. Temperature dependence of thermodynamic variables for the second-nearest neighbor SOS model ($J_1 = J_2 = J$). (a) Surface tension σ, (b) entropy per unit length S/L, (c) energy per unit length U/L and (d) heat capacity per unit length C_v/L for the horizontal interface. Division by the energy parameter J normalizes all energies to be dimensionless. Dashed lines indicate the critical temperature kT_c.

3.4 Dependence of T_c on the Ratio J_2/J_1

The only exact solution I can compare my results to is Onsager's. The last section (3.3) showed not only that the critical temperature of the SOS model is the same as the Curie temperature of Ising model (for first-nearest-neighbor interactions), but

also that the surface tension of the SOS model for a horizontal interface coincides with that of the Ising model for all temperatures up to T_c. For $J_2 \neq 0$, we cannot solve the Ising model exactly. Although most investigations of the Ising model have focused on nearest-neighbor interactions, a number have attempted to illuminate its critical behavior and phase diagram for longer-range interactions.

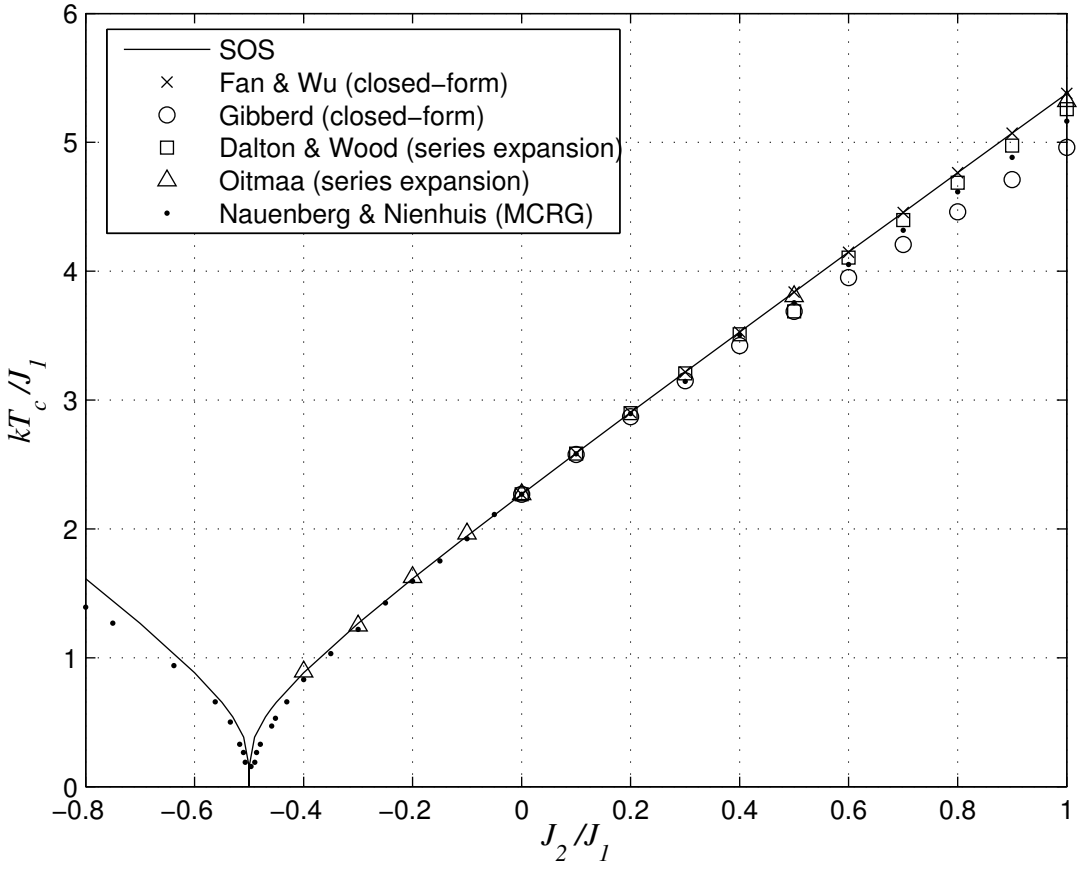


Figure 3.7. The dependence of kT_c on the relative strength of first- and second-neighbor interactions for the SOS model and various approximations. I obtained the SOS T_c by numerically solving for $\sigma = 0$ (equation 3.18) and setting $J_1 = 1$. The references are Fan & Wu [37], Gibberd [43], Dalton & Wood [25], Oitmaa [93] and Nauenberg & Nienhuis [87].

While I have not found any investigations of the surface tension of the Ising model as a function of both the temperature and the ratio J_2/J_1, there there were a number calculated approximately the critical temperature of the Ising model as a function of the ratio J_2/J_1. Figure 3.7 shows the variation of kT_c as a function of

the ratio J_2/J_1 for a number of approximations, including the SOS model.

Approximation methods include the closed-form approximation [37, 43], the *Bethe-Peierls-Weiss (BPW) approximation [24], series expansions [25, 93], Monte-Carlo renormalization-group method (MCRG)* [87, 110], standard numerical Monte-Carlo methods [12] and the SOS approximation [16]. Burkhardt's SOS results only considered the diagonal (45°) interface and used transfer matrices. His [16] results are agree with Dalton & Wood's.

Qualitatively, my results agree with all other approximations. Quantitatively, my results seem to suggest an upper limit for the critical temperature. All other methods seem to produce critical temperatures equal to or smaller than mine. In particular, my results coincide with Fan & Wu's for $0 \leqslant J_2/J_1 \leqslant 1$.

The SOS model correctly predicts the location of the cusp which occurs for $J_2/J_1 = -0.5$. For temperatures below T_c, the cusp marks the transition from either the antiferromagnetic ground state (AF, $J_1 < 1$) or ferromagnetic ground state (F, $J_1 > 1$) to the super-antiferromagnetic ground state (SAF) (figure 3.8). Above T_c, the lattice becomes paramagnetic.

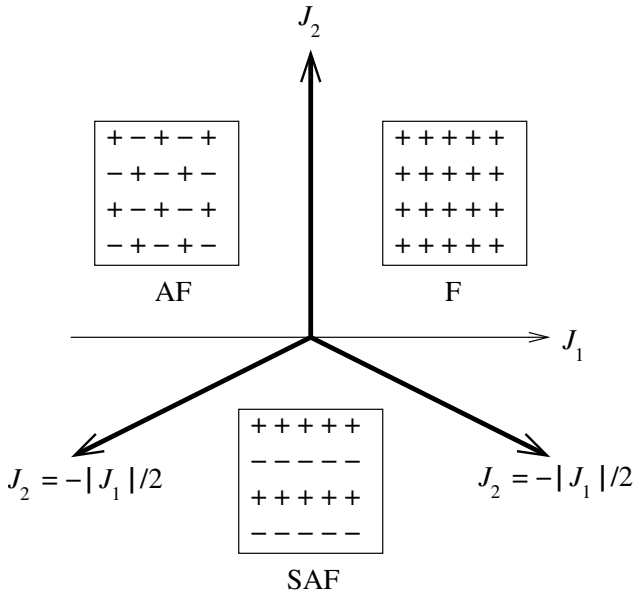


Figure 3.8. The phase diagram of the Ising model below T_c, for interactions up to second-nearest neighbors in zero magnetic field.

The ratio J_2/J_1 relates to the the coordination number z. The coordination number is the weighted number of neighbors within the range of interaction. If the interaction strengths of all neighbors are the same, then z equals the total number of neighbors within the interaction range. If the interaction weakens with increasing range, then z is smaller than the total number of neighbors within the range. Therefore, z measures of the effective range of interaction. For interactions up to second-nearest neighbors with J_1 and J_2 the first- and second-nearest-neighbor coupling constants, the coordination number is:

$$z = z_1 + z_2 \left(\frac{J_2}{J_1} \right), \tag{3.21}$$

where z_1 and z_2 are the number of first- and second-nearest neighbors respectively. For a two-dimensional square lattice, $z_1 = 4$, $z_2 = 4$ (figure 2.1).

Since the 1930s, a number of approximations have calculated the dependence of the critical temperature kT_c on the coordination number, starting with the mean-field theory of Bragg and Williams [14]. The Bragg-Williams theory is similar to the Weiss theory of ferromagnetism. The original Bragg-Williams paper dealt with the order-disorder transition in metallic alloys. Chang [21] cast these results as:

$$kT_c = \frac{2zV_B}{4}, \tag{3.22}$$

where V_B is equal to my $2J_1$. Using the definition $K_c = J_1/kT_c$, we obtain:

$$\begin{aligned} kT_c &= zJ_1, \\ zK_c &= 1. \end{aligned} \tag{3.23}$$

This result simply states that in the Bragg-Williams approximation, the critical temperature is the coordination number. The longer the effective range of interaction, the higher the critical temperature.

An approximation attributed to Bethe [10] only accounts for first-nearest-neighbor interactions:

$$e^{-2J/kT_c} = 1 - \frac{2}{z}. \tag{3.24}$$

Taking the log of both sides of equation 3.24:

$$zK_c = -\frac{z}{2}\ln\left(1 - \frac{2}{z}\right). \tag{3.25}$$

For large z, we expand the logarithm and keeping only the first two terms of the series expansion to obtain:

$$zK_c = 1 + \frac{1}{z}. \tag{3.26}$$

The Bethe approximation (equation 3.26), approaches the Bragg-Williams approximation (equation 3.23), when $z \gg 1$.

Domb and Potts [28] extended Bethe's approximation to include second-nearest-neighbor interactions:[1]

$$\left[\frac{1}{2}\left(1 + e^{-2J/kT_c}\right)\right]^{z/2} = \frac{1}{2}, \tag{3.27}$$

which I rewrite as:

$$zK_c = -\frac{z}{2}\ln[2^{(1-2/z)} - 1]. \tag{3.28}$$

Hiley and Joyce [58] related a previously unpublished result of Dalton:

$$zK_c = 1 + \frac{4.3}{z}, \tag{3.29}$$

while Domb and Dalton [27] found that in two dimensions,

$$zK_c = 1 + \frac{A}{z^\gamma} \qquad (0 < \gamma < 1), \tag{3.30}$$

but gave no values for A and γ.

Figures 3.9 plots the Bragg-Williams [14] (equation 3.23), Bethe [10] (equation 3.24), Domb-Potts [28] (equation 3.27) and Hiley-Joyce [58] (equation 3.29) results

[1] In Domb & Potts' original paper [28], the r.h.s. of equation 3.27 is 4 instead of 1/2, which is inconsistent with an earlier equation in their paper and also with figure 9 in their paper.

together with my results for the second-nearest-neighbor SOS model. All approximations approach the mean-field Bragg-Williams approximation as $z \to \infty$. This convergence is not surprising since, by definition, the mean-field method assumes infinite-range interaction. The results of Domb and Potts come closest to the SOS model.

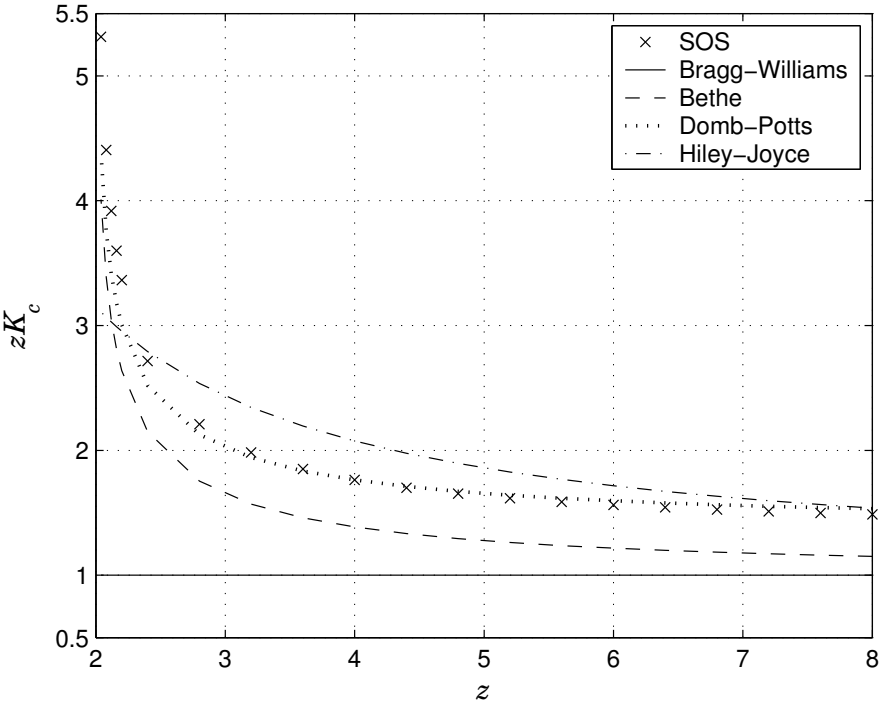


Figure 3.9. zK_c *vs.* number of neighbors z for the SOS model and various approximations to the Ising model. The references are Bragg-Williams [14], Bethe [10], Domb-Potts [28] and Hiley-Joyce [58]. The dimensionless quantity K_c is given by $K_c = J_1/kT_c$.

3.5 The Interface Width and the Roughness Exponent

Apart from thermodynamic properties, we can also investigate the geometrical properties of the interface, particularly, its width and roughness exponent.

In a two-dimensional lattice, I define the width w of the interface to be the root-mean-squared of the height difference between two points separated by a horizontal

distance of d pixels, *i.e.* the standard deviation of r_i:

$$w^2 = \langle (h(k+d) - h(k))^2 \rangle. \tag{3.31}$$

Since the baseline is arbitrary, I choose $h(1) = 0$, therefore,

$$\begin{aligned} h(k) &= \sum_{i=1}^{k} r_i, \\ (h(k+d) - h(k))^2 &= \left(\sum_{i=k}^{k+d} r_i \right)^2 \\ &= \sum_{i,j=k}^{k+d} r_i r_j \\ &= \sum_{i=k}^{k+d} r_i^2 + \sum_{\substack{i,j=k \\ i \neq j}}^{k+d}{}' r_i r_j, \end{aligned} \tag{3.32}$$

where $\sum'$ denotes a sum over terms with $i \neq j$. For distances much greater than the interaction range, we expect r_i to be independent of r_{i+d} and the autocorrelation function $\langle r_i r_{i+d} \rangle$ to be zero. Since all r_is are independent and equal:

$$\langle (h(k+d) - h(k))^2 \rangle = d \langle r^2 \rangle. \tag{3.33}$$

Comparing equations 3.31 and 3.33, we obtain:

$$w = d^{1/2} \sqrt{\langle r^2 \rangle}. \tag{3.34}$$

The width scales with the distance between the two points. The exponent of d, often denoted α, is the *roughness exponent* [8, 109]. The value $\alpha = 0.5$ is a well-known result for the two-dimensional Ising model [11] and the above derivation shows that it is a direct consequence of the assumption that the r_is are independent of each other. An alternative derivation using Fourier decomposition and equipartition of energy arrives at essentially the same conclusion [75]. If we take d to be the time separation between two points, random walks and Brownian motion have the same roughness exponent [8].

The interface width diverges as $d \to \infty$, so the interface is *rough*. The interface is *smooth* only when $kT = 0$. The transition from a smooth to a rough surface is a *roughening transition*, occurring at a transition temperature T_r [81]. In the two-dimensional Ising model, $T_r = 0$. That the interface is rough for $0 < T < T_c$ shows that surface tension does not automatically produce a smooth interface [2].

In three dimensions, $0 < T_r < T_c$. For the Ising model, the roughening temperature is $T_r \simeq 0.54 T_c$ [17, 52, 53, 84, 85, 118]. For the SOS model, the roughening transition occurs at $T_r \simeq 0.75 T_c$ [35]. At temperatures below T_r, the interface width remains finite, even when $d \to \infty$. Above T_r, the interface width diverges as $w^2 \sim \ln d$ [88].

The interface width w not only depends on d, it also depends on the temperature. The higher the temperature, the larger the width. We can readily calculate the root-mean-squared of r for the SOS model:

$$\begin{aligned}
\langle r^2 \rangle &= \sum_{r=-\infty}^{\infty} r^2 P_r \\
&= 2 \sum_{r=1}^{\infty} r^2 P_r \\
&= \frac{2}{\Xi} \sum_{r=1}^{\infty} r^2 e^{-[(2+2r)J_1 + 4rJ_2]/kT} \\
&= \frac{2\gamma_1^2}{\Xi} \sum_{r=1}^{\infty} r^2 (\gamma_1^2 \gamma_2^4)^r,
\end{aligned} \tag{3.35}$$

where I have used the definitions in equation 3.16. If we define:

$$y \equiv \sum_{r=1}^{\infty} (\gamma_1^2 \gamma_2^4)^r = \sum_{r=1}^{\infty} e^{-\varepsilon_0 r/kT}, \tag{3.36}$$

then:

$$\begin{aligned}
\sum_{r=1}^{\infty} r^2 (\gamma_1^2 \gamma_2^4)^r &= \frac{1}{\varepsilon_0^2} \frac{\partial^2 y}{\partial (1/kT)^2} \\
&= \frac{1}{\varepsilon_0^2} \frac{\partial^2}{\partial (1/kT)^2} \left(\frac{1}{e^{\varepsilon_0/kT} - 1} \right) \\
&= \frac{\cosh(\varepsilon_0/2kT)}{4 \sinh^3(\varepsilon_0/2kT)}.
\end{aligned} \tag{3.37}$$

Substituting equation 3.37 into equation 3.35 and using the Ξ for the SOS model (equation 3.17):

$$\langle r^2 \rangle = \frac{\gamma_1}{\gamma_2^2(1-\gamma_1^2\gamma_2^4+2\gamma_1^2)} \frac{\cosh(\varepsilon_0/2kT)}{\sinh^2(\varepsilon_0/2kT)}. \tag{3.38}$$

For the case $J_1 = J$, $J_2 = 0$, equation 3.38 reduces to:

$$\langle r^2 \rangle = \frac{1}{2\sinh^2(J/kT)}. \tag{3.39}$$

If $J_1 = J_2 = J$, equation 3.38 reduces to:

$$\langle r^2 \rangle = \frac{1}{\gamma(1-\gamma^6+2\gamma^2)} \frac{\cosh(3J/kT)}{\sinh^2(3J/kT)}. \tag{3.40}$$

In contrast, the interface width of a two-dimensional Ising model with nearest-neighbor interactions is [3, 4]:

$$\langle r^2 \rangle = \frac{1}{\sinh[2(K-K^*)]}, \tag{3.41}$$

where $K = J/kT$ and:

$$K^* \equiv -\frac{1}{2}\ln(\tanh K). \tag{3.42}$$

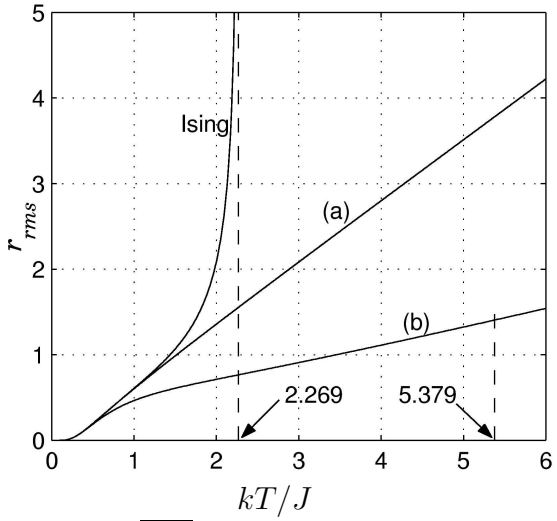


Figure 3.10. Interface width $\sqrt{\langle r^2 \rangle}$ of the Ising model and the SOS model in two special cases: (a) $J_1 = J$, $J_2 = 0$, (b) $J_1 = J_2 = J$. The arrows indicate kT_c/J for each case. Compare the result for the Ising model with (a).

Figure 3.10 plots all three cases. Although the SOS model predicts that the surface tension vanishes at T_c, the interface width remains finite. The comparison shows the remarkable breakdown of the SOS model as $T \to T_c$. In the Ising model, the interface width and the specific-heat capacity diverge at T_c while they remain finite in the SOS model. The calculation of the interface width from the SOS model agrees with that from the Ising model only up to $T \simeq 1.25J$.

3.6 Angular Dependence of Surface Tension

The one-dimensional Ising model is naturally paramagnetic. It cannot spontaneously magnetize for any temperature greater than zero [97]. To force it to have non-zero magnetization, requires an external magnetic field H_m to the Ising Hamiltonian:

$$H = \sum_{i,j} J_{ij}(1 - \delta_{\sigma_i,\sigma_j}) + H_m \sum_i \sigma_i, \tag{3.43}$$

where i is a sum over the entire lattice and j is a sum over the neighbors of i.

By the same analogy, the interface (for the Ising or SOS model) in a square lattice is naturally horizontal or vertical. In order to bias the interface to have a non-zero average slope, I introduce an external *torque* field τ [71]:

$$H = \sum_{i,j} J_{ij}(1 - \delta_{\sigma_i,\sigma_j}) + \tau h, \tag{3.44}$$

where h is the mean slope of the interface, defined to be:

$$h = \tan\theta = \sum_{r=-\infty}^{r=\infty} rP_r = \sum_{r=1}^{r=\infty} r(P_r - P_{-r}). \tag{3.45}$$

With the added term, the Helmholtz free energy is a function of T, L and h:

$$\begin{aligned} dU &= TdS + \sigma dL + \tau dh, \\ dF &= dU - TdS - SdT \\ &= -SdT + \sigma dL + \tau dh. \end{aligned} \tag{3.46}$$

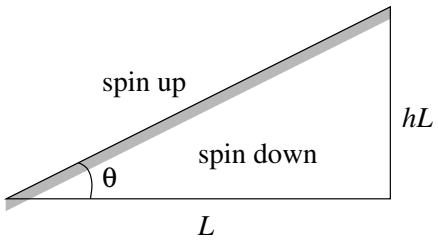


Figure 3.11. An interface at an angle θ relative to the horizontal.

Here, the external field τ is the independent variable and h is a function of τ. Defining a new free energy that is a function of τ instead of h is more convenient. I define a new free energy G, in line with the Gibbs free energy, where the pressure acts as the external field and the volume is the dependent variable:

$$\begin{aligned} G &\equiv F - \tau h = -kT \ln Z_\tau, \\ dG &= dF - \tau dh - h d\tau \\ &= -SdT + \sigma dL - h d\tau. \end{aligned} \tag{3.47}$$

G is the natural free energy of the new (T,L,τ) ensemble and Z_τ its partition function. The Helmholtz free energy becomes:

$$F = -kT \ln Z_\tau + \tau h = -kTL \ln \Xi_\tau + \tau h. \tag{3.48}$$

For zero field, F recovers the form of equation 3.18. The surface tension σ again equals the Helmholtz free energy per unit length:

$$\sigma = \frac{F}{\sqrt{h^2 L^2 + L^2}} = \frac{F}{L\sqrt{\tan^2\theta + 1}} = \frac{F \cos\theta}{L}. \tag{3.49}$$

The external field modifies the step energies (equation 3.4) to:

$$\varepsilon_r = (2 + 2|r|)J_1 + 4|r|J_2 - \tau r, \quad r \neq 0. \tag{3.50}$$

$\tau > 0$, biases the interface to have positive slope (energies for positive r decrease). We derive the partition function for a non-zero-slope interface in the same way as

for a horizontal interface. With the added field, the partition function per unit horizontal length, Ξ (equation 3.15) becomes:

$$\Xi_\tau = e^{-(2J_1+4J_2)/kT} + \sum_{r=1}^{\infty}(e^{-[(2+2r)J_1+4rJ_2+\tau r]/kT} + e^{-[(2+2r)J_1+4rJ_2-\tau r]/kT}). \quad (3.51)$$

Using the definitions of equation 3.16 and further defining,

$$\eta \equiv e^{\tau/kT}, \quad (3.52)$$

equation 3.51 simplifies to:

$$\Xi_\tau = \gamma_1^2\gamma_2^4 + \gamma_1^2 \sum_{r=1}^{\infty} \left[(\gamma_1^2\gamma_2^4\eta)^r + (\gamma_1^2\gamma_2^4\eta^{-1})^r\right]. \quad (3.53)$$

η is analogous to the thermodynamic definition of fugacity: $e^{\mu/kT}$ (where μ is the chemical potential) [60], hence I call η the *fugacity*. A few lines of algebra show that:

$$\Xi_\tau = \gamma_1^2\gamma_2^4 \left[\frac{1+\gamma_1^2(1-\gamma_2^4)(\eta+\eta^{-1})+\gamma_1^4\gamma_2^4(\gamma_2^4-2)}{1-\gamma_1^2\gamma_2^4(\eta+\eta^{-1})+\gamma_1^4\gamma_2^8}\right]. \quad (3.54)$$

For $h = 0$, $\tau = 0$, $\eta = 1$, Ξ_τ reduces to Ξ in equation 3.17. For $J_2 = 0$:

$$\Xi_\tau = \gamma_1^2 \left[\frac{1-\gamma_1^4}{1-\gamma_1^2(\eta+\eta^{-1})+\gamma_1^4}\right]. \quad (3.55)$$

For $J_1 = J_2 = J$:

$$\Xi_\tau = \gamma^6 \left[\frac{1+\gamma^2(1-\gamma^4)(\eta+\eta^{-1})+\gamma^8(\gamma^4-1)}{1-\gamma^6(\eta+\eta^{-1})+\gamma^{12}}\right]. \quad (3.56)$$

The slope h is a function of τ and kT. From equation 3.45:

$$\begin{aligned} h &= \frac{\sum_{-\infty}^{\infty} r\zeta_r}{\Xi_\tau} \\ &= kT\frac{\partial}{\partial\tau}\ln\Xi_\tau \\ &= \eta\frac{\partial}{\partial\eta}\ln\Xi_\tau \\ &\vdots \\ &= \frac{ag_-}{b+cg_+-dg_+^2}, \end{aligned} \quad (3.57)$$

where,

$$\begin{aligned} g_{\pm} &\equiv \eta \pm \eta^{-1}, \\ a &\equiv \gamma_1^2(1-\gamma_1^4\gamma_2^8), \\ b &\equiv (1+\gamma_1^4\gamma_2^8)(1+\gamma_1^4\gamma_2^8-2\gamma_1^4\gamma_2^4), \\ c &\equiv \gamma_1^2[1-2\gamma_2^4(1+\gamma_1^4\gamma_2^8)+3\gamma_1^4\gamma_2^8], \\ d &\equiv \gamma_1^4\gamma_2^4(1-\gamma_2^4). \end{aligned} \tag{3.58}$$

We could obtain τ (or η) as a function of h and kT by inverting equation 3.57, which involves solving a quartic equation and is extremely cumbersome. Instead, I will solve for τ numerically. Appendix B gives the Mathematica program that generates the polar plot in figure 3.13(b).

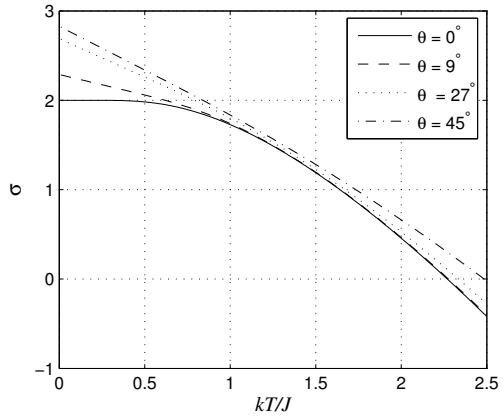
Again, comparing the SOS model with known results for the Ising model is instructive. I chose to compare the surface tensions of interfaces at various angles. I mentioned earlier (section 3.3) that the SOS model agrees with the Ising model only when $J_2 = 0$ and $\theta = 0°$. For angles greater than zero, the SOS model differs slightly from the Ising model, as in figure 3.12.

Equations 3.48, 3.49 and 3.55 give the surface tension of the SOS model. Because τ depends on both h and kT, I first numerically calculate it from equation 3.57 and then substitute its value into equations 3.48 and 3.55.

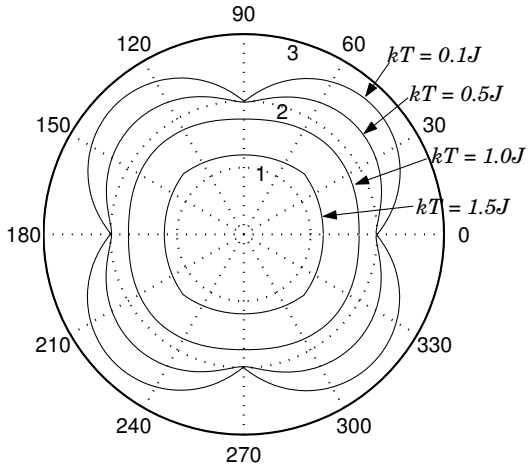
The surface tension of the nearest-neighbor Ising model for an arbitrary-angle interface is [106]:

$$\begin{aligned} \sigma_{Ising}(\theta, kT) &= kT\left[|\cos\theta|\sinh^{-1}(p|\cos\theta|) + |\sin\theta|\sinh^{-1}(p|\sin\theta|)\right], \\ p &= \frac{2}{q}\left[\frac{1-q^2}{1+(\sin^2 2\theta + q^2\cos^2 2\theta)^{1/2}}\right]^{1/2}, \\ q &= \frac{2\sinh(2J/kT)}{\cosh^2(2J/kT)}. \end{aligned} \tag{3.59}$$

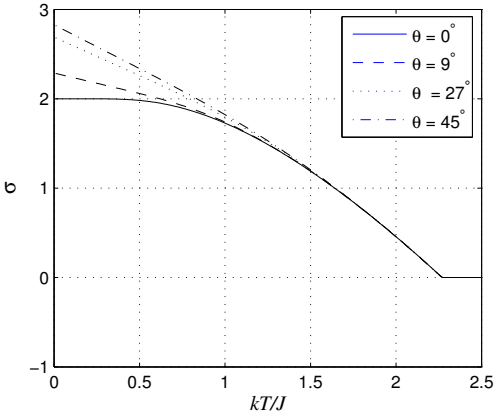
Equation 3.59 reduces to equation 3.20 when $\theta = 0$.



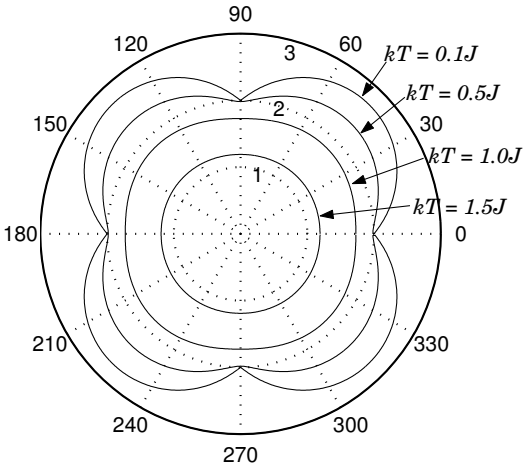
(a) SOS



(b) SOS



(c) Ising



(d) Ising

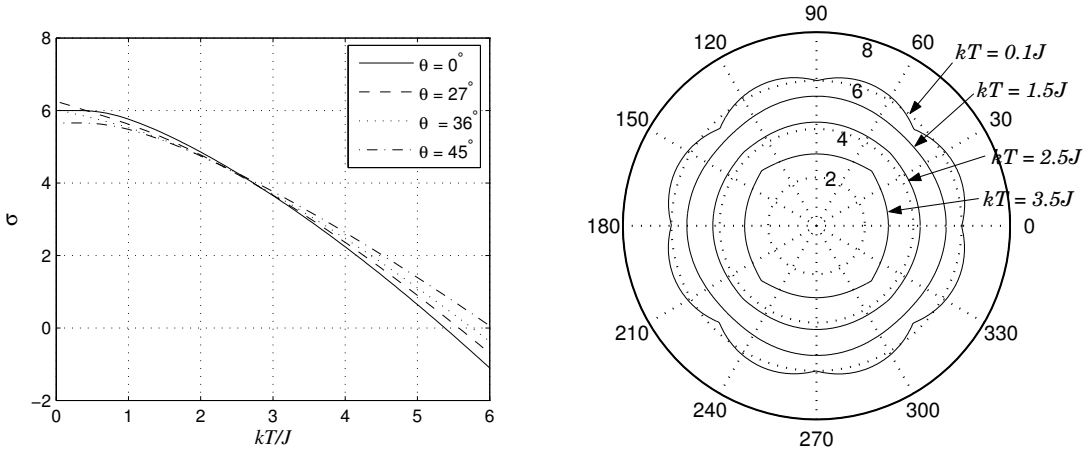
Figure 3.12. (a) and (c): Surface tension σ as a function of normalized temperature kT/J at various angles for the SOS and Ising models respectively. (b) and (d): Polar plots of surface tension at various temperatures for the SOS and Ising models respectively. Here, $J_2 = 0$ and all energies are in units of J.

Figure 3.12 plots the surface tension of the Ising and SOS models as a function of temperature (for fixed angle) and as a function of angle (for fixed temperature). The Ising model becomes more isotropic as the temperature increases (the polar plot of σ becomes rounder). The SOS model is most isotropic for $kT \simeq 1.25J$. Anisotropy increases for $kT \gtrsim 1.25$ (figures 3.12(a) and 3.14(a)), close to the temperature where

the interface widths of the SOS model and the Ising model start to diverge (figure 3.10). At the beginning of this chapter, I mentioned that the SOS model is a low-temperature approximation of the Ising model. The surface tension of the SOS model agrees well with that of the Ising model for $kT \lesssim 1.25J$ and $\forall\,\theta$.[2]

The SOS model agrees with the Ising model exactly for $\theta = 0, \forall\, kT \leqslant kT_c$ and for $kT = 0, \forall\,\theta$. The critical temperature (where $\sigma = 0$) for the Ising model is independent of the orientation of the interface, while the critical temperature of the SOS model increases with the interface angle (figure 3.12(a) and table 3.2).

Finally, I consider the equivalent-neighbor interactions, $J_1 = J_2 = J$. I calculate the τ field numerically by solving equation 3.57, then substituting the results into equations 3.48 and 3.56. Figure 3.13 shows the corresponding plots.



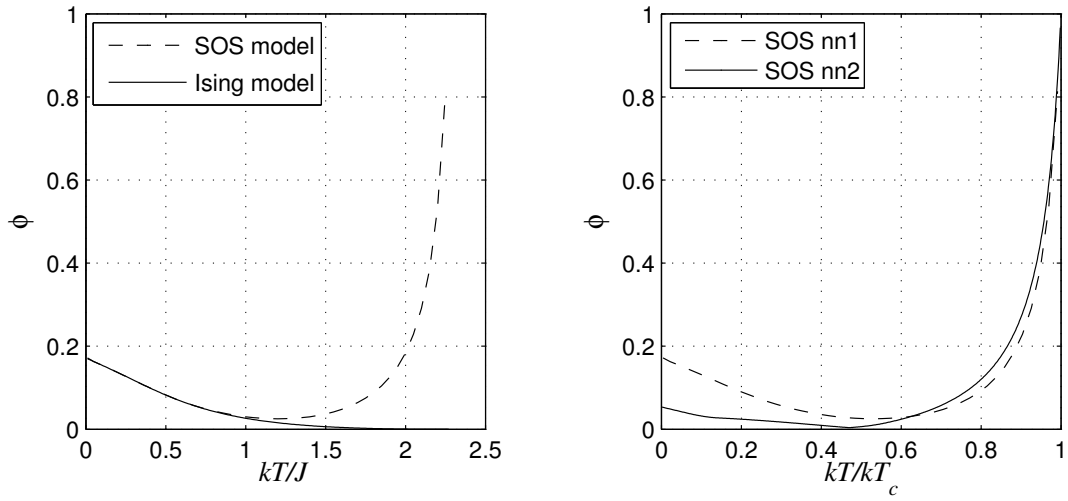
(a) (b)

Figure 3.13. Surface tension σ of the SOS model up to second-nearest-neighbor interactions (a) as a function of normalized temperature kT/J at various angles, (b) as a function of angle at various temperatures.

[2]I define the anisotropy to be:

$$\phi(kT) \equiv \frac{\max\{\sigma(kT)_{\text{polar}}\} - \min\{\sigma(kT)_{\text{polar}}\}}{\max\{\sigma(kT)_{\text{polar}}\} + \min\{\sigma(kT)_{\text{polar}}\}}.$$

$\sigma(kT)_{\text{polar}}$ is the polar plot of σ at temperature kT. $\phi = 0$ corresponds to the isotropic case. $\phi = 1$ corresponds to the most anisotropic case.



(a) (b)

Figure 3.14. Comparison of the anisotropy ϕ between (a) nearest-neighbor SOS and Ising models, (b) nearest- and second-nearest-neighbor ($J_1 = J_2$) SOS models as a function of temperature.

The second-nearest-neighbor SOS model is most isotropic at about $kT \simeq 2.7J$. Its critical temperature also increases with interface angle (figure 3.13 and table 3.2). Comparing the results for nearest-neighbor and second-nearest-neighbor interactions, shows that anisotropy due to lattice discretization decreases as the interaction range increases (figure 3.14(b)). In both cases (nearest- and second-nearest SOS model), the lattice is most isotropic (ϕ is smallest) at $kT \simeq 0.5kT_c$.

TABLE 3.2

CRITICAL TEMPERATURES IN THE SOS MODEL. NN1: NEAREST; NN2: UP TO SECOND-NEAREST-NEIGHBOR INTERACTIONS.

	$kT_c/J(\pm 0.0005)$	
Angle	nn1	nn2
0°	2.269	5.379
9°	2.276	5.398
18°	2.297	5.460
27°	2.334	5.573
36°	2.393	5.755
45°	2.485	6.041

CHAPTER 4

SIMULATION RESULTS FOR PHASE SEPARATION

In this chapter, I present Monte-Carlo-dynamics simulations of phase separation in the Ising model (equation 2.1) with $J_1 = J_2 = J_3 = J_4 = J$ using the three algorithms I defined in section 2.8. I calculate the surface tension, internal energy, entropy and specific heat of interfaces as a function of algorithm, temperature, interface angle and interaction range. I also analyze the interface width for zero-angle interfaces as a function of temperature and algorithm.

Since I wish to compare my Monte-Carlo simulations to theoretically derived functional forms, in this chapter, I also develop a consistent set of empirical formulae for thermodynamic quantities which are based on my understanding of the SOS model. For example, since my simulations allow for overhangs, I consider this effect by assuming the step energies are degenerate. Also, in an Ising model with third-neighbor or longer interactions adjacent steps interact; I include this effect by adding "virtual" steps between the physical steps. These types of assumptions allow me to derive a set of functional forms for the thermodynamic quantities my simulations calculate. The functional forms depend on five fitting parameters, three of which are energies and two of which are dimensionless. I call the resultant model the *modified SOS model*.

This model has the advantage of allowing me to extract consistent information about quantities like the critical temperature kT_e where the energy and the heat

capacity of the interface diverge. It is a purely empirical model.

4.1 Rate of Equilibration

In order to ensure that I gather Monte-Carlo simulation results after the interface has equilibrated, I need to find the rate of equilibration.

I begin with a horizontal interface and equilibrate it using either Algorithm One, Two or Three at different temperatures. I calculate the energy per unit lateral length (step energy in the SOS model) after each Monte-Carlo step, where one Monte-Carlo step equals $L \times L$ spin-flip attempts (whether successful or not). I sometimes gather data after multiple or fractional Monte-Carlo steps. For example, a half Monte-Carlo steps equals $\frac{1}{2}L \times L$ spin-flip attempts.

Figure 4.1 shows the average step energies as a function of time at various temperatures. The interaction range is up to fourth-nearest neighbors and all interactions are of equal strength. I show the simulation results for Algorithms One and Two together to highlight their similarities and differences. The higher the temperature, the higher the average energy per step. The equilibrium average energies of both algorithms are the same, but Algorithm One equilibrates faster than Algorithm Two. Algorithm Three resembles Algorithms One and Two but, with lower equilibrium energies (figure 4.2).

I analyze of the dependence of equilibrium step energies on temperatures in the next section. Here, I focus on the time dependence of the step energy. I fit the average step energy, $\langle \varepsilon \rangle$, to the empirical equation that I discovered through trial and error:

$$R \equiv \frac{\langle \varepsilon \rangle - 22}{\langle \varepsilon \rangle_{eq} - 22} = 1 - \frac{1}{(1 + t/t_r)^{\beta}}, \tag{4.1}$$

where $\langle \varepsilon \rangle_{eq}$ is the equilibrium step energy (when $t \to \infty$) and t is the time in units of Monte-Carlo steps (MCS). The fitting parameters are the characteristic

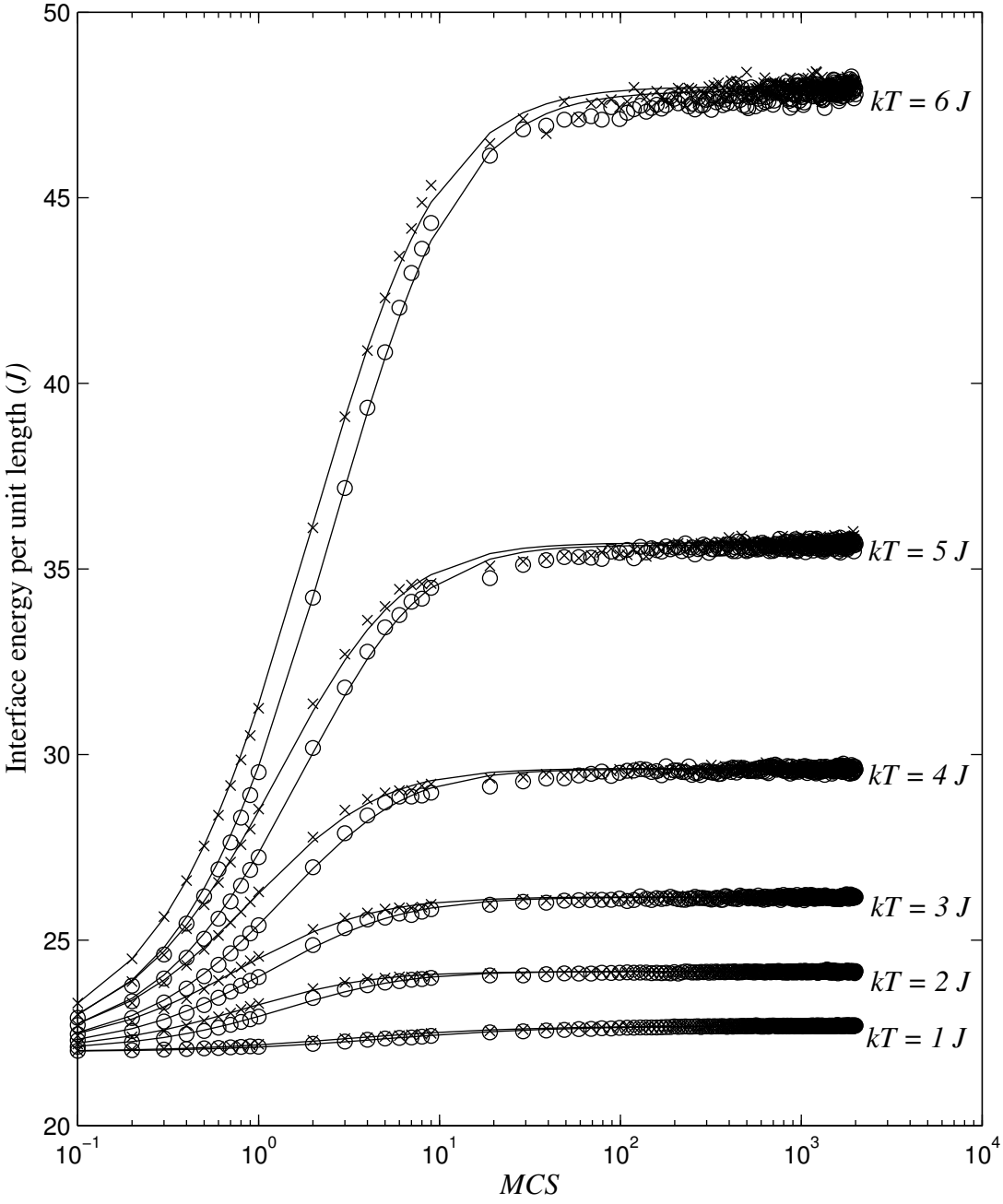


Figure 4.1. The average energy per unit lateral length (average step energy for the SOS model) of a horizontal interface as a function of time (MCS). $\times$: Algorithm One, $\bigcirc$: Algorithm Two. Each data point is a result of averaging 100 simulations. Solid lines are best fits to equation 4.1. The interaction range is up to fourth-nearest neighbors. All interactions are of equal strength, $J_1 = J_2 = J_3 = J_4 = J$.

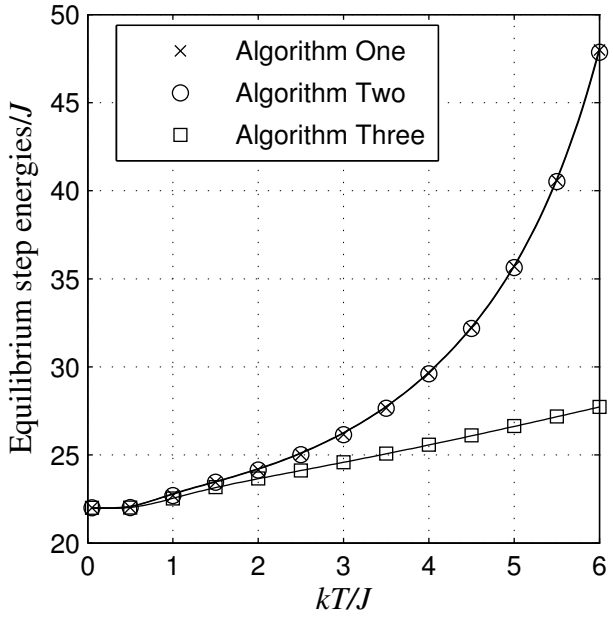


Figure 4.2. The equilibrium step energies in the SOS model (energies per unit lateral length) as a function of algorithm and temperature. Error bars are smaller than the data points. Solid lines are best fits to equation 4.11.

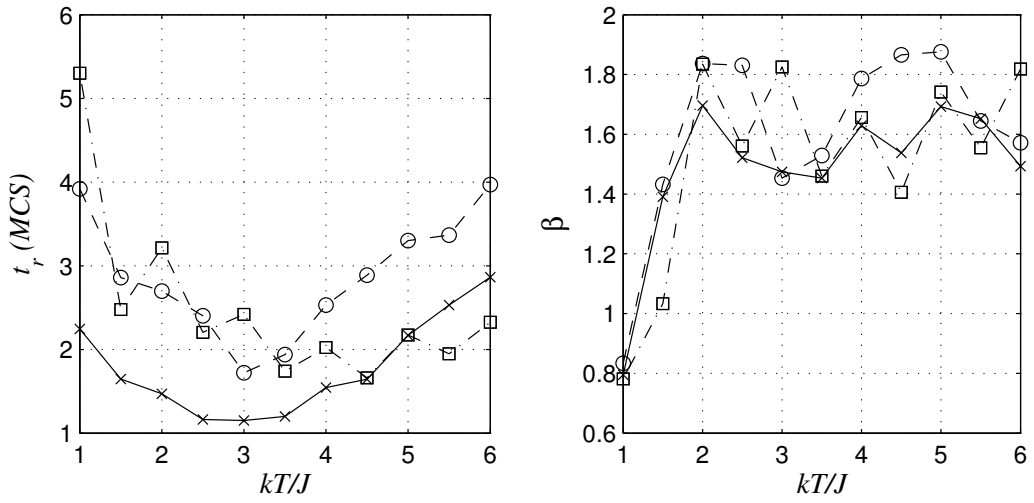
time constant t_r and the growth exponent β. I subtract $22J$ from $\langle\varepsilon\rangle$ because at $t = 0$, the interface is perfectly horizontal. For interactions up to fourth-nearest neighbors, the energy per unit length of a flat horizontal interface is $22J$. Therefore, the *normalized average step energy*, R, ranges from 0 to 1 as t goes from 0 to ∞. Table 4.1 and figure 4.3 show the results for the fitting parameters.

From figure 4.3(a), t_r's dependence on temperature is similar for all three algorithms. The time t_r first decreases with temperature, and then increases again after reaching a minimum. For Algorithms One and Two, t_r is minimal for $kT \simeq 3.0J$ while for Algorithm Three, t_r is minimal for $kT \simeq 4.5J$. The t_r of Algorithm Two is always greater than that of Algorithm One. If we look at figure 4.2, the equilibrium step energy for Algorithms One and Two for $kT \simeq 3J$ is roughly the same as that for Algorithm Three for $kT \simeq 4.5J$. The location of the minimal t_r should be a function of the surface roughness (equilibrium step energy), and should not depend on the algorithm. Although the interface is not rough at low temperatures, due to

TABLE 4.1

THE CHARACTERISTIC TIME CONSTANTS, t_r, AND GROWTH EXPONENTS, β, OF THE EQUILIBRIUM STEP ENERGY FOR ALGORITHMS ONE, TWO AND THREE AS A FUNCTION OF TEMPERATURE FOR FOURTH-NEAREST-NEIGHBOR INTERACTIONS ($J_1 = J_2 = J_3 = J_4 = J$).

kT/J	$t_r \pm 0.05MCS$			$\beta \pm 0.01$		
	Algorithm			Algorithm		
	One	Two	Three	One	Two	Three
1.0	2.24	3.92	5.30	0.80	0.83	0.78
1.5	1.65	2.86	2.48	1.39	1.43	1.03
2.0	1.47	2.70	3.21	1.70	1.84	1.83
2.5	1.16	2.40	2.20	1.52	1.83	1.56
3.0	1.15	1.72	2.42	1.47	1.45	1.83
3.5	1.20	1.94	1.74	1.45	1.53	1.46
4.0	1.54	2.53	2.02	1.63	1.79	1.66
4.5	1.64	2.89	1.66	1.54	1.86	1.41
5.0	2.17	3.30	2.17	1.69	1.88	1.74
5.5	2.53	3.37	1.95	1.65	1.64	1.56
6.0	2.86	3.97	2.32	1.49	1.57	1.82



(a) (b)

Figure 4.3. (a) Variation of the time constant t_r with temperature kT. (b) Variation of growth exponent β with temperature kT. $\times$: Algorithm One. $\bigcirc$: Algorithm Two. $\square$: Algorithm Three.

weak thermal agitations, the surface still takes a long time to equilibrate. At high temperatures, the interface is very rough, and it takes a correspondingly longer time to reach its final roughness. Between these two regimes the surface is smooth enough and the temperature high enough for the surface to reach the final roughness in a relatively short time.

Figure 4.3(b) shows that the temperature dependence of the growth exponents, β is similar for all three algorithms. For temperatures $kT \leqslant 2J$, β is roughly proportional to temperature. For temperatures $kT \geqslant 2J$, β remain roughly constant, fluctuating around $\beta = 1.6$. Table 4.2 shows the average values of the growth exponent, β, for each algorithm and for temperatures $kT \geqslant 2J$.

TABLE 4.2

AVERAGE GROWTH EXPONENT β FOR $2J \leqslant kT \leqslant 6J$.

Algorithm	$\langle\beta\rangle$
One	1.57 ± 0.03
Two	1.71 ± 0.06
Three	1.65 ± 0.05

At $t = 0$, $\langle\varepsilon\rangle = 22J$ and $R = 0$. As t increases, $\langle\varepsilon\rangle \rightarrow \langle\varepsilon\rangle_{eq}$ and $R \rightarrow 1$. Therefore, $1 - R$ measures the closeness of the step energy $\langle\varepsilon\rangle$ to the equilibrium value $\langle\varepsilon\rangle_{eq}$. The smaller the value of $1 - R$, the closer $\langle\varepsilon\rangle$ to $\langle\varepsilon\rangle_{eq}$. Figure 4.4 shows $1 - R$ as a function of time. The solid line (best fit) shows that $1 - R$ decreases towards zero as time increases, but the simulated values of $\langle\varepsilon\rangle$ fluctuate in a range of about $\pm 1\%$ ($|1 - R| = 10^{-2}$) around their equilibrium value. When the step energy lies within 1% of its equilibrium value, I consider the interface fully equilibrated.

The smaller the value of β, the slower the equilibration. Equilibration is slower at low temperatures. At higher temperatures ($kT \geqslant 2J$), β is roughly constant ($\simeq 1.6$). The rate of equilibration depends only on t_r to which it is inversely proportional. Table 4.3 shows that for $\beta = 1.6$ the average step energy lies within 1% of its

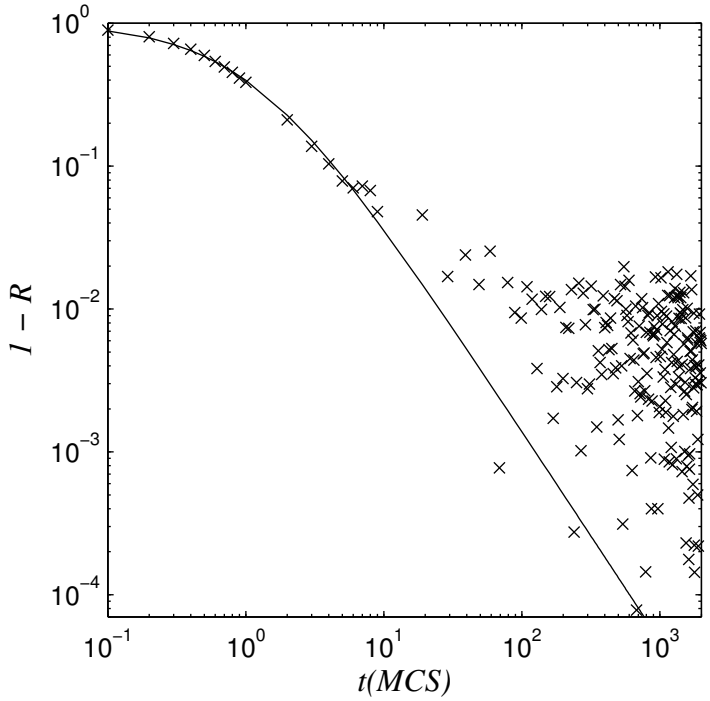


Figure 4.4. $1 - R$ *vs.* t for Algorithm One, at $kT = 3J$, where R is the normalized average step energy. The solid line is the best fit to equation 4.1. At $t = 100\,MCS$, $1 - R \leqslant 10^{-2}$, therefore, R is within 1% of its equilibrium value. Regardless of temperature, fluctuation amplitudes remain mostly in the range between 10^{-2} and 10^{-3}.

TABLE 4.3

NORMALIZED AVERAGE STEP ENERGY, R AS A FUNCTION OF TIME FOR FOUR DIFFERENT VALUES OF β.

	R			
t/t_r	$\beta = 1.6$	$\beta = 1.0$	$\beta = 0.8$	$\beta = 0.5$
1	0.65	0.50	0.43	0.29
2	0.81	0.67	0.58	0.42
3	0.88	0.75	0.67	0.50
4	0.91	0.80	0.72	0.55
5	0.93	0.83	0.76	0.59
10	0.97	0.90	0.85	0.70
20	0.99	0.95	0.91	0.78
50	1.00	0.98	0.95	0.86
100	1.00	0.99	0.98	0.90

equilibrium value after about $20t_r$. For $t_r \simeq 3\,MCS$, this time is about $60\,MCS$.

For $J \leqslant kT < 2J$ ($0.8 < \beta < 1.6$), equilibration takes about $100t_r$, anywhere between $200\,MCS$ and $500\,MCS$. For $kT < J$ ($\beta < 0.8$), equilibration may take more than $1000\,MCS$. My simulations are not long enough ($t_{max} = 2000MCS$) to determine the exact values of t_r and β for $kT < J$.

To analyze $\langle\varepsilon\rangle_{eq}$ (figure 4.2) as a function of temperature, I need to explain the fitting equations I use, which I base on the SOS model in chapter 3.

4.2 Modified SOS model

I wish to compare my Monte-Carlo simulation results to those for the SOS model. I modified the second-nearest-neighbor SOS model from chapter 3. Through trial and error, I found that an optimal fit requires five parameters. I describe here my logic for using these five parameters.

The step energy of the SOS model with up to second-nearest-neighbor interactions and equal coupling constants ($J_1 = J_2$) is:

$$\varepsilon_r = (2 + 6|r| + 4C_r)J, \tag{4.2}$$

where C_r comes from equation 3.3. For arbitrary-range interactions, I propose a general equation:

$$\varepsilon_r = (\varepsilon_a + \varepsilon_b|r| + \varepsilon_c C_r)J, \tag{4.3}$$

where ε_a, ε_b and ε_c are fitting parameters. In order to allow for overhangs, I add a degeneracy term (δ) to the partition function per step. In the SOS model, the interface is single-valued with no overhangs. Overhangs cause the interface to be multi-valued, increasing the degeneracy of the step energy. Therefore, I fit to a modified partition function per step:

$$\Xi = \sum_{r=-\infty}^{\infty} \delta^{|r|}\zeta_r, \tag{4.4}$$

where:

$$\zeta_r = e^{-\varepsilon_r/kT}. \tag{4.5}$$

The previous chapter showed that for third-nearest-neighbor or longer interactions, adjacent steps interact, with the energy of a step depending not only on its height r, but also on the height of its neighbors. The situation is analogous to a one-dimensional Ising model where each spin interacts with its nearest neighbors. The method of transfer matrices [111, 120] easily solves the partition function of the one-dimensional Ising model. Because the spin values are either +1 or -1, the dimension of the transfer matrix is only 2×2. Unfortunately, in the SOS model, the height r ranges from $-\infty$ to ∞. Therefore, the corresponding transfer matrix is $\infty \times \infty$. I derived this matrix by finding a consistent way of defining the interaction energy between neighboring steps for third-nearest-neighbor interactions, but will leave the daunting task of solving it to future investigations.[1] Here I account for the interactions between neighboring steps by adding a fifth fitting parameter. I propose that a step's free energy:

$$F = -kT \ln \Xi^{\mathcal{A}}, \tag{4.6}$$

instead of:

$$F = -kT \ln \Xi. \tag{4.7}$$

The interactions between steps are akin to adding "virtual" steps between the physical steps. The parameter $\mathcal{A}$ therefore accounts for the interactions between steps by including more steps than the physical width of the lattice. With these assumptions, the five fitting parameters are ε_a, ε_b, ε_c, δ and $\mathcal{A}$.

[1] Kramers and Wannier [66] showed that the partition function of the two-dimensional Ising model is equal to the largest eigenvalue of an infinite matrix of simple structure. Domb [26] showed that a number of problems in statistical mechanics of cooperative phenomena reduce to determining the largest eigenvalue of an infinite matrix of characteristic bi-diagonal structure.

If we substitute equation 4.2 into equation 4.4 and follow the same procedures as those after equation 3.15, we obtain:

$$\Xi = \gamma_a \left(\gamma_c + \frac{2\delta\gamma_b}{1-\delta\gamma_b} \right), \tag{4.8}$$

where:

$$\begin{aligned} \gamma_a &\equiv e^{-\varepsilon_a/kT}, \\ \gamma_b &\equiv e^{-\varepsilon_b/kT}, \\ \gamma_c &\equiv e^{-\varepsilon_c/kT}. \end{aligned} \tag{4.9}$$

The Helmholtz free energy is:

$$F = \mathcal{A} \left[\varepsilon_a - kT \ln \left(\gamma_c + \frac{2\delta\gamma_b}{1-\delta\gamma_b} \right) \right]. \tag{4.10}$$

The internal energy is:

$$\begin{aligned} \langle \varepsilon \rangle_{eq} = U &= -(kT)^2 \frac{\partial}{\partial(kT)} \left(\frac{F}{kT} \right) \\ &= \mathcal{A} \left\{ \varepsilon_a + \frac{2\varepsilon_b \delta\gamma_b + \varepsilon_c \gamma_c (1-\delta\gamma_b)^2}{(1-\delta\gamma_b)[2\delta\gamma_b + (1-\delta\gamma_b)\gamma_c]} \right\} \\ &= \mathcal{A} \left[\varepsilon_a + \frac{2\varepsilon_b \gamma_a \delta\gamma_b}{\Xi(1-\delta\gamma_b)^2} + \frac{\varepsilon_c \gamma_a \gamma_c}{\Xi} \right] \\ &= \mathcal{A}(\varepsilon_a + \varepsilon_b \langle |r| \rangle + \varepsilon_c \langle C_r \rangle). \end{aligned} \tag{4.11}$$

The third and fourth lines of equation 4.11 are equal term-by-term. The entropy and specific-heat capacity come from the usual thermodynamic relations:

$$\begin{aligned} S &= \frac{U-F}{T} \\ &= -\frac{\partial F}{\partial T}, \\ C_v &= \frac{\partial U}{\partial T}. \end{aligned} \tag{4.12}$$

I do not write out the explicit and cumbersome form of S and C_v.

4.3 Temperature Dependence of $\langle\varepsilon\rangle_{eq}$ for a Zero-Angle Interface

I simulated phase separation of a horizontal interface using the Ising-model Hamiltonian (equation 2.1) and the modified Metropolis Algorithms (One, Two and Three) from section 2.8. Figure 2.6 shows snapshots of the typical interface for each of the algorithms. The range of interactions is up to fourth-nearest neighbors with equal coupling constants ($J_1 = J_2 = J_3 = J_4 = J$). I calculate the interface energy E and divided the energy by the length of the lattice (L) to obtain the average energy per unit lateral length ($\langle\varepsilon\rangle = E/L$), which I compare to the step energy of my modified SOS model. I use a model similar to the second-nearest neighbor SOS model in chapter 3.

Note that equations 4.10 to 4.12 relate the free energy, internal energy, entropy and heat capacity to one set of five parameters: ε_a, ε_b, ε_c, δ and $\mathcal{A}$. The model these equations represent is empirical, but has the advantage that one set of five parameters provides all thermodynamic quantities.

I use equation 4.11 to fit the simulation results for the average energy per unit lateral length, using the *nlinfit* function in Matlab 7 (appendix C). The nlinfit function uses the Levenberg-Marquardt method [103] to find the optimal values of the fitting parameters and determine their 95% confidence intervals. Table 4.4 shows the curve-fitting results, figure 4.2 the corresponding plots.

The optimal parameter values need not be unique and I found many sets of fitting parameters that fit the simulation results well. To weed out unphysical results, I use the following acceptance criteria:

1. At $kT = 0$, the interface is perfectly flat; therefore the energy per unit length must equal $22J$, *i.e.* $\mathcal{A}(\varepsilon_a + \varepsilon_c) = 22J$.
2. For interactions up to fourth-nearest neighbors, the energy of a step of size one ($r = 1$) is $30J$, therefore $\mathcal{A}(\varepsilon_a + \varepsilon_b) \simeq 30J$.
3. Since step energy increases with step size, ε_b must be positive.

4. The best fit should reproduce the characteristic peak of C_v at low temperatures (Figure 3.5(d)).

5. The order of magnitude of ε_a, ε_b and ε_c should be smaller than simulation results for $\langle\varepsilon\rangle$, otherwise accounting for the values of $\langle\varepsilon\rangle$ if $\varepsilon_{a,b,c} \gg \langle\varepsilon\rangle$ (equation 4.3) would be difficult.

TABLE 4.4

RESULTS OF FITTING AVERAGE STEP ENERGIES TO EQUATION 4.11 FOR A ZERO-ANGLE INTERFACE WITH $J_1 = J_2 = J_3 = J_4 = J$. THE PARAMETERS δ AND $\mathcal{A}$ ARE DIMENSIONLESS.

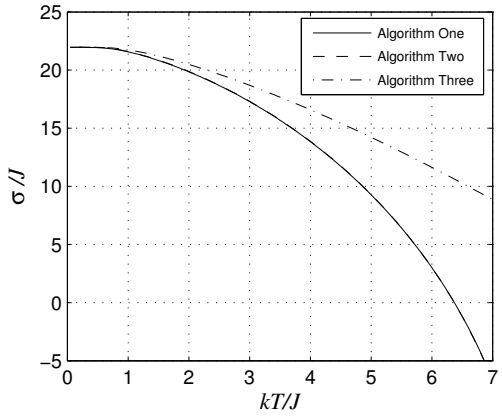
Algorithm	ε_a/J	ε_b/J	ε_c/J	δ	$\mathcal{A}$
One	14.78±1.47	5.78±0.91	3.36±0.72	2.05±0.22	1.21±0.05
Two	14.78±1.40	5.81±0.86	3.38±0.68	2.06±0.21	1.21±0.05
Three	7.01±0.98	7.52±0.86	4.93±0.72	0.89±0.04	1.84±0.08

Figure 4.2 shows the internal energies simulated using the three algorithms as a function of temperature. Figure 4.5 shows the surface tension, entropy and specific-heat for the three algorithms. In all cases, Algorithm One coincides with Algorithm Two over the entire range of temperatures.

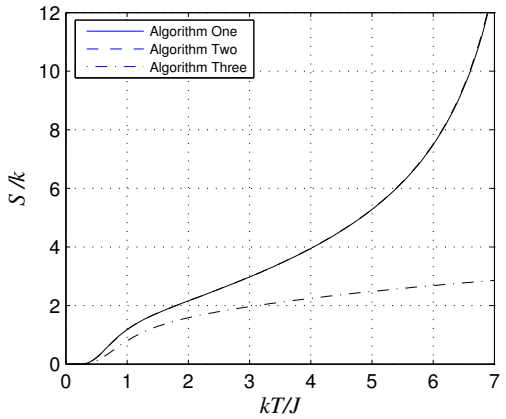
Solving $\sigma = 0$ numerically gives the critical temperatures, kT_c, for each algorithm (table 4.5). kT_c is largest for Algorithm Three. From equation 4.11, U diverges when $\delta\gamma_b = 1$. From this observation, we can derive another critical temperature kT_e where both U and C_v diverge (table 4.5). For all algorithms, $kT_e > kT_c$. The kT_e of Algorithm Three is infinite, as in the SOS model, while the behavior of kT_e for Algorithms One and Two is closer to the behavior in the Ising model.

4.4 Interface Width

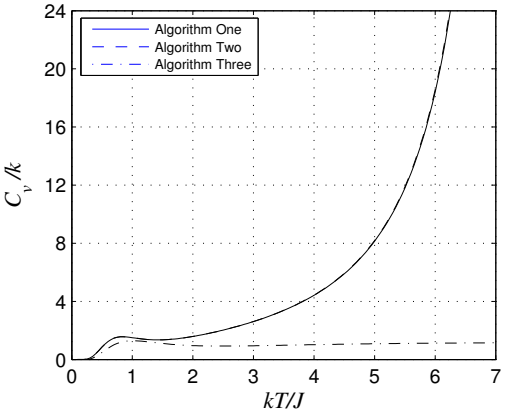
Section 4.3 compared the thermodynamic properties of an interface for the three algorithms. This section analyzes the interfacial geometry, specifically the interface width. Following the steps in section 3.5, I derive the root-mean-squared width of



(a) σ



(b) S



(c) C_v

Figure 4.5. (a) Surface tension σ, (b) entropy S per step and (c) specific-heat capacity C_v per step. Results for $J_1 = J_2 = J_3 = J_4 = J$, a zero-angle interface and simulations for kT up to $6.0J$. In all cases, Algorithm One is indistinguishable from Algorithm Two.

TABLE 4.5

CRITICAL TEMPERATURES IN THE SOS MODEL FOR A HORIZONTAL INTERFACE FOR DIFFERENT ALGORITHMS USING AN EQUIVALENT NEIGHBOR MODEL ($J_1 = J_2 = J_3 = J_4 = J$).

Algorithm	kT_c/J	kT_e/J
One	6.37±0.38	8.04±0.10
Two	6.37±0.36	8.05±0.09
Three	9.90±1.02	∞

the interface in the modified SOS model:

$$\begin{aligned}\langle r^2\rangle &= 2\sum_{r=1}^{\infty} r^2 P_r \\ &= \frac{2}{\Xi}\sum_{r=1}^{\infty} r^2\delta^r e^{-\varepsilon_r/kT} \\ &= \frac{2}{\Xi}\sum_{r=1}^{\infty} r^2\delta^r e^{-(\varepsilon_a+\varepsilon_b r)/kT} \\ &= \frac{2}{\Xi}\gamma_a\sum_{r=1}^{\infty} r^2(\delta\gamma_b)^r. \end{aligned} \tag{4.13}$$

If we define:

$$y \equiv \sum_{r=1}^{\infty}(\delta\gamma_b)^r = \frac{1}{(\delta\gamma_b)^{-1}-1}, \tag{4.14}$$

then:

$$\begin{aligned}\sum_{r=1}^{\infty} r^2(\delta\gamma_b)^r &= \frac{1}{\varepsilon_b^2}\frac{\partial^2 y}{\partial(1/kT)^2} \\ &= \frac{\cosh(\frac{\varepsilon_b/kT-\ln\delta}{2})}{4\sinh^3(\frac{\varepsilon_b/kT-\ln\delta}{2})}. \end{aligned} \tag{4.15}$$

Substituting equation 4.15 into equation 4.13 and using Ξ from equation 4.8:

$$\langle r^2\rangle = \frac{(\delta\gamma_b)^{1/2}}{2\delta\gamma_b+(1-\delta\gamma_b)\gamma_c}\frac{\cosh(\frac{\varepsilon_b/kT-\ln\delta}{2})}{\sinh^2(\frac{\varepsilon_b/kT-\ln\delta}{2})}. \tag{4.16}$$

Substituting the parameter values from table 4.4 into equation 4.16 gives $r_{rms} \equiv \sqrt{r^2}$ as a function of temperature which I plot in figure 4.6. The interface width diverges when $\varepsilon_b/kT = \ln\delta$, giving the same kT_e value as in the previous section (section 4.3). The interface width r_{rms} diverges at the same temperature as the interface's internal energy U and specific heat C_v.

In the Ising model, the specific heat capacity and interface width diverge at kT_c while the interface width in the SOS model does not diverge at a finite temperature. Equation 4.16 shows that when we include an overhang term (δ), the interface width of the modified SOS model diverges at a finite temperature kT_e. Although the temperature kT_e at which the internal energy, specific heat and interface width

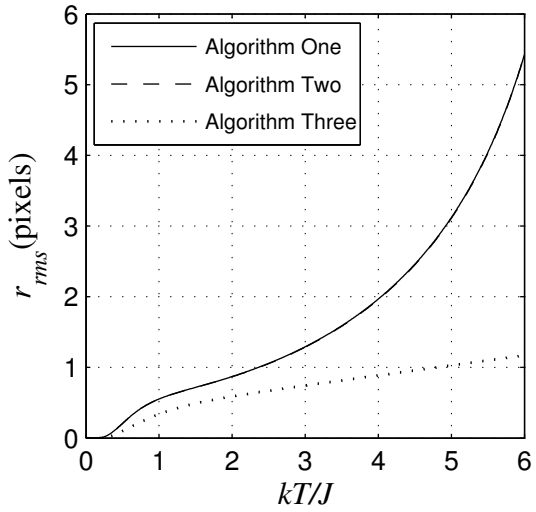


Figure 4.6. r_{rms} (equation 4.16) *vs.* kT. Algorithm One is indistinguishable from Algorithm Two.

diverge differs from the critical temperature kT_c, kT_e is still finite, and of the same order of magnitude as kT_c. I speculate that the exclusion of overhangs in the SOS model destroys the second-order phase transition which occurs at kT_c in the Ising model. Including overhangs in an empirical fashion restores the phase transition, albeit at a slightly higher temperature.

The exclusion of overhangs not only affects the existence of phase transitions, it also changes the interface morphology. Requiring the interface to be single valued, *i.e.* prohibiting overhangs, changes the interface from self similar to self affine [22]. A *self-similar* function obeys the scaling rule:

$$f(x) \sim \frac{1}{b} f(bx), \tag{4.17}$$

while a *self-affine* function obeys the scaling rule:

$$f(x) \sim \frac{1}{b^{\alpha}} f(bx). \tag{4.18}$$

Self-similar functions look the same when we rescale both the x and y-axes by the same factor b. A self-affine function looks the same if we rescale the x- and y-axes

by different amounts, *i.e.* b and b^α where α is the *Holder exponent* or *self-affine exponent* [8].

Equation 3.34 shows that w is a self-affine function of d with $\alpha = 0.5$. Here, the self-affine exponent is the roughness exponent. In Brownian motion, the self-affine exponent is also called the *Hurst exponent* [36, 98].

Previous investigations [91, 92] of interfaces driven by external forces (random-field Ising model) have shown that at large length scales the interface is self-affine with $\alpha = 0.5$.

For distances d of the order of interface width, the self-affine exponent relates to the fractal dimension of the interface [8, 76, 114]:

$$D_f = 2 - \alpha. \tag{4.19}$$

For $\alpha = 0.5$, the corresponding fractal dimension is $D_f = 1.5$. For distances much larger than a few interface widths, the interface looks smooth and one-dimensional, and the corresponding fractal dimension at these length scales is $D_f = 1$. The fractal dimensions of some rough surfaces is close to 1.5. For example, the fractal dimension of the external perimeter of gravity-invasion percolation is $D_f \simeq 1.36 \pm 0.3$ [13]. A previous Monte-Carlo simulation of the SOS model found $D_f = 1.58 \pm 0.06$ [81]. The fractal dimension of diffusion fronts is $D_f \simeq 1.75$ [107].

Experimentally, we can define and measure the interface width in many ways, *e.g.* by calculating the standard deviation of the interface height over horizontal intervals of length d:

$$w_1(d) = \sqrt{\langle (h - \langle h \rangle_d)^2 \rangle_d} = \sqrt{\langle h^2 \rangle_d - \langle h \rangle_d^2}\,, \tag{4.20}$$

where the subscript d indicates that the average is over an interval of length d. The widths $w_1(d)$ of several intervals with the same horizontal length d can provide an average value of $w_1(d)$. Alternatively, we can define the width as the average height

difference between two points separated by a horizontal distance d:

$$w_2(d) = \langle |h(x+d) - h(x)| \rangle, \tag{4.21}$$

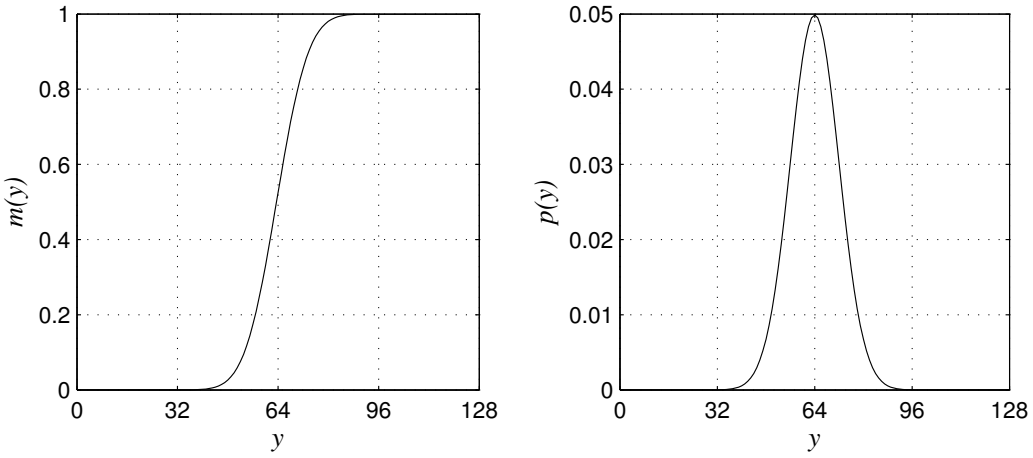
or equivalently:

$$w_3(d) = \sqrt{\langle (h(x+d) - h(x))^2 \rangle}\ . \tag{4.22}$$

All three definitions (4.20, 4.21, 4.22) should give the same roughness exponent when we fit w_1, w_2, w_3 to:

$$w_{1,2,3}(d) = w_o d^{\alpha}. \tag{4.23}$$

In the presence of overhangs, the height of the interface $h(x)$ is multi-valued and I include all heights at a given x in the average. For large values of d ($\gg$ interface width), a single-valued function $h_{sv}(x)$, defined as the highest point for each x can approximate the interface. My simulations show that widths calculated with $h(x)$ and $h_{sv}(x)$ are within error of each other.



(a) (b)

Figure 4.7. (a) Schematic of a typical magnetization profile across an interface. (b) The first derivative (slope) of the magnetization profile.

An alternative definition of width equates the interface width to the width of the first derivative of the magnetization profile across the interface (figure 4.7) [84].

The magnetization is 0 for spin 0 and 1 for spin 1. I take the interface location to be $m(y) \simeq 0.5$.

The slope of the magnetization profile is:

$$p(y) = \frac{m(y + \Delta y) - m(y)}{\Delta y}, \tag{4.24}$$

and the interface width is:

$$\begin{aligned} W_m &= \sqrt{\langle y^2 \rangle - \langle y \rangle^2}, \\ \langle y \rangle &= \int y p(y) dy, \\ \langle y^2 \rangle &= \int y^2 p(y) dy. \end{aligned} \tag{4.25}$$

I simulate the phase separation of a horizontal interface over $10,000\, MCS$. At $t = 0\, MCS$, the interface is perfectly horizontal and its width is zero. As t increases, the interface roughness and width increase. I calculate $w(d)$ at regular time intervals. Figure 4.8(a) shows a typical log-log plot of w *vs.* d at the beginning and the end of a simulation. The slope gives the roughness exponent α of the interface. The intercept of the plot gives w_o. However, I use the first data point to approximate w_o since the result is considerably less noisy. The difference between the first data points and the actual intercept is the multiplicative factor 2^α.

The final slope α (between $d = 2$ and $d = 64$) is the same regardless of temperature and algorithm (figures 4.8(b) and 4.8(c)). The intercept w_o increases with temperature and depends on the algorithm (figures 4.8(b) and 4.8(d)). I calculated the equilibrium value of α by averaging the slope between $t = 2000\, MCS$ and $10000\, MCS$. Similarly, w_o averages the first data point for t between $2000\, MCS$ and $10000\, MCS$.

Figure 4.9 shows my analysis of the interface width using w_3. The results for w_1 and w_2 (not shown) are similar to w_3 both qualitatively and quantitatively. The

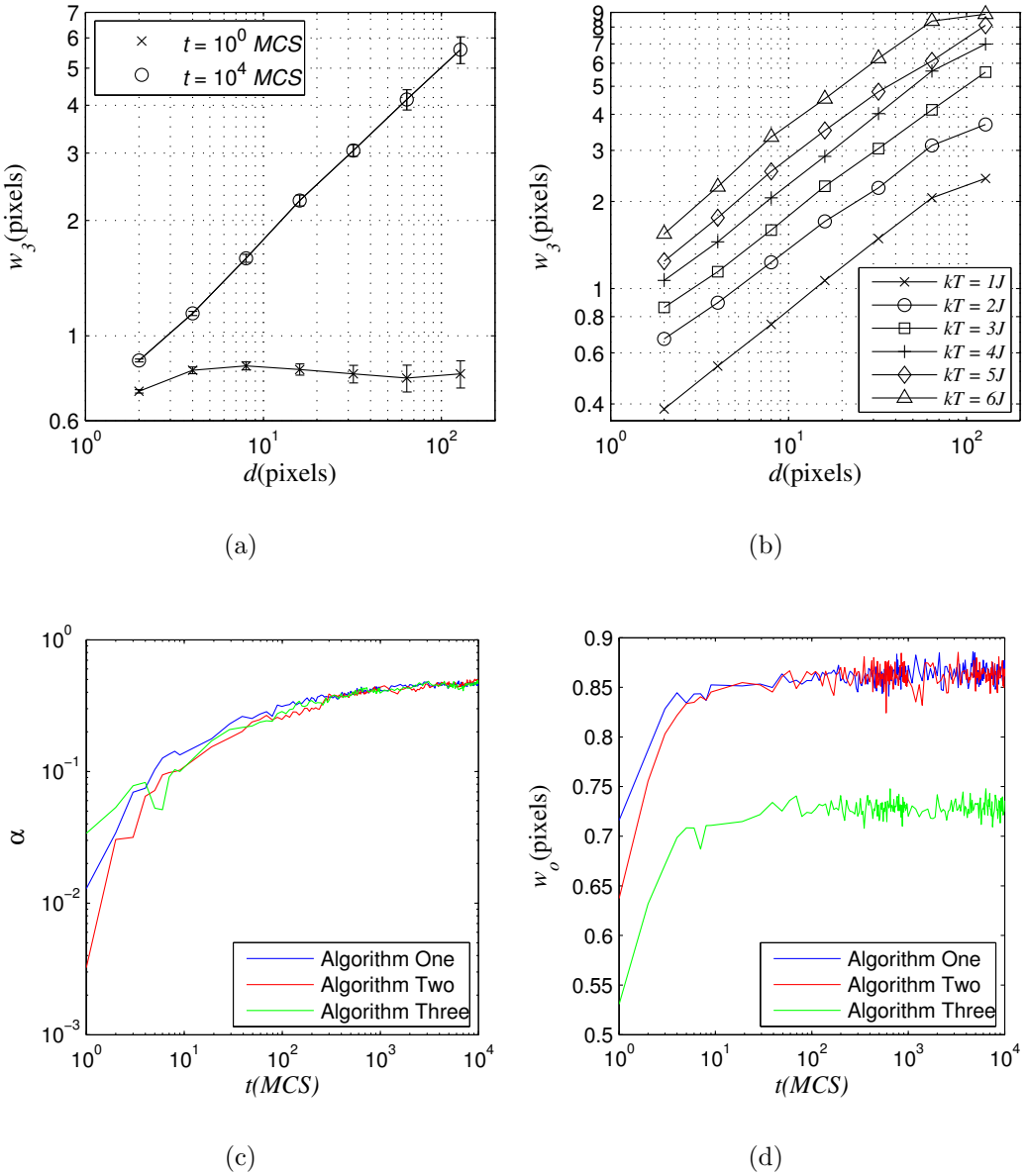
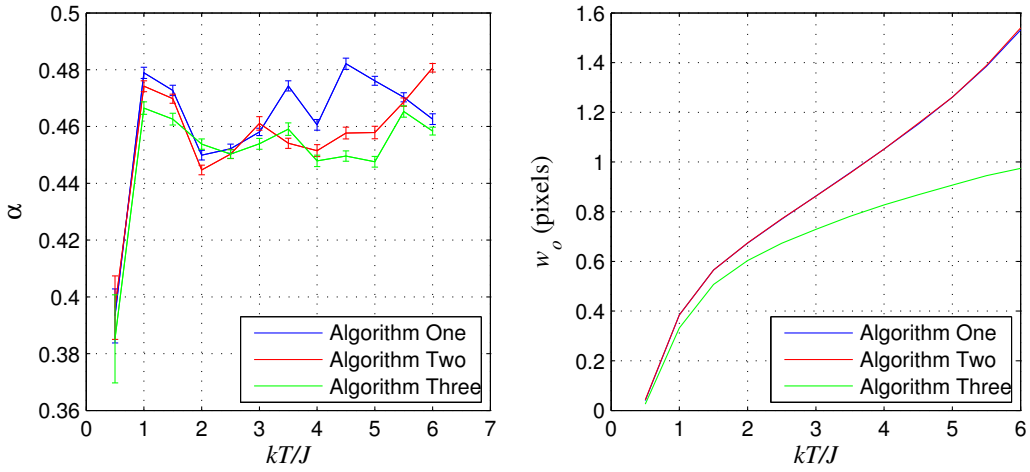


Figure 4.8. (a) Log-log plot of w_3 *vs.* d at the beginning and the end of simulations of phase separation using Algorithm One at fixed $kT = 3.0J$. The lateral length of the interface is 512 pixels. Each data point averages 20 independent simulations. (b) Log-log plot of w_3 *vs.* d at $t = 10,000\,MCS$ at various temperatures. (c) Variation of the slope α in time for fixed $kT = 3.0J$ for different algorithms. (d) Variation of w_o with time at fixed $kT = 3.0J$ for different algorithms.



(a) (b)

Figure 4.9. (a) Average α and (b) average w_o of w_3 (equation 4.23) as a function of temperature for phase-separation simulations using Algorithms One, Two and Three. Error bars in (b) are as thin as the lines.

TABLE 4.6

THE AVERAGE VALUE OF THE ROUGHNESS EXPONENT α OF THE INTERFACE FOR ALGORITHMS ONE, TWO AND THREE. THE RANGES OF TEMPERATURES OVER WHICH I AVERAGE α ARE: FOR w_1, FROM $1.5J$ TO $6.0J$; FOR w_2, FROM $2.0J$ TO $6.0J$; FOR w_3, FROM $1.0J$ TO $6.0J$.

Algorithm	$\langle\alpha\rangle$		
	w_1	w_2	w_3
One	0.435 ± 0.004	0.496 ± 0.004	0.467 ± 0.003
Two	0.426 ± 0.004	0.487 ± 0.003	0.461 ± 0.003
Three	0.429 ± 0.005	0.498 ± 0.004	0.456 ± 0.002

results for w_m are noisy, but generally agree with figure 4.9(b). I did not calculate w_m as a function of the distance d. In figure 4.9(a), excluding the data point at $kT = 0.5J$, the average value of α for all three algorithms is about 0.46. Table 4.6 gives the average values of α for w_1, w_2 and w_3. The overall average $\alpha = 0.46 \pm 0.01$. Using equation 4.19, the fractal dimension D_f of the interface is 1.54 ± 0.01, in good agreement with Mon's result, $D_f = 1.58 \pm 0.06$ [81].

4.5 Non-zero-angle Interfaces

All my simulations begin with a flat interface, either horizontal, or at an angle. In section 3.2, the interface may fluctuate, but on average, the interface is either horizontal or vertical ($\theta = 0°$, $90°$). Because of the symmetry of the square lattice, a horizontal interface is energetically the same as a vertical interface. For any initial angle, the equilibrated interface using any algorithm is either horizontal or vertical. In order to simulate an interface at an angle, I have to skew the lattice so that the ground state of the interface is at an angle relative to the horizontal axis.

First, I draw a straight line at the desired angle (line AB in figure 4.10), dividing the lattice into two phases: spin-0 and spin-1. Pixels that lie in the path of the straight line are partly in the both spin-0 and spin-1 regions. I adopt the simple rule that if more than half of the pixel lies in the spin-0 region, then it has spin 0, otherwise, it has spin 1. The result is a staircase-like interface (figure 4.10).

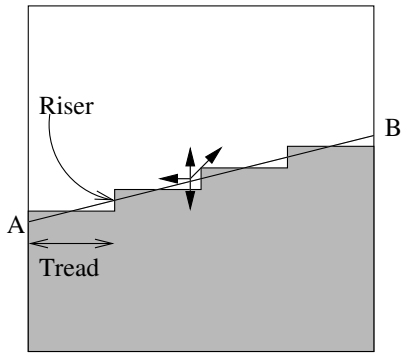


Figure 4.10. The figure illustrates how I skew the lattice to simulate interfaces at an angle.

I want this staircase-like interface instead of the horizontal interface to be the ground-state configuration, *i.e.* have the lowest energy. For temperatures greater than zero, the interface should fluctuate around this tilted interface instead of the horizontal interface. This method is similar to applying a torque field to the SOS model in section 3.6.

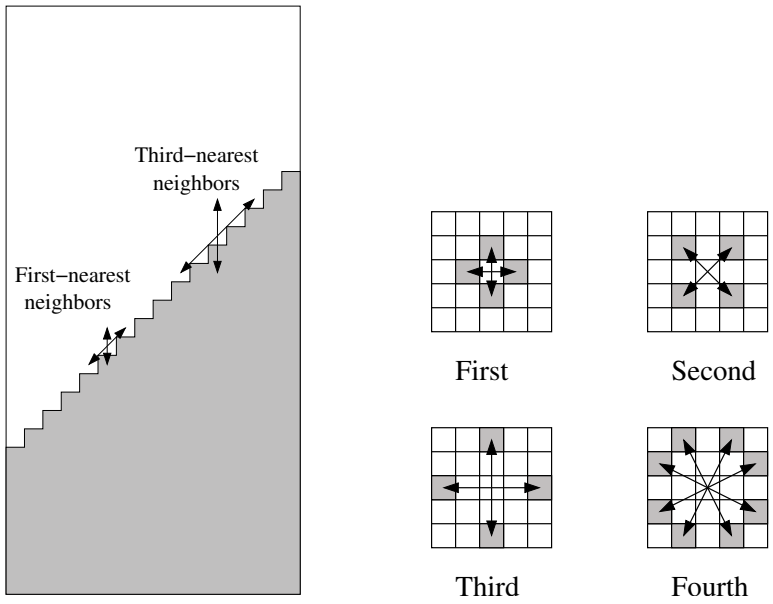
Borrowing terminologies used in carpentry, the horizontal parts of the steps are *treads* and the vertical parts *risers*. Figure 4.10 shows how I redefine the nearest neighbor of a pixel facing a riser (black arrows). The second, third and fourth neighbors change similarly. In other words, all neighbors that lie to the right of a riser shift up by one pixel and all neighbors that lie to the left of a riser shift down by the same amount. If a neighbor is neither to the right nor left of a riser, then I use the regular, unbiased neighbor list. All pixels in the same column (including the bulk pixels) have similar new neighbor lists.

The periodic boundary condition joins points A and B and all other edge pixels which are in the same relative position to each other as A and B, making the lattice look like the thread of a screw. I fix the spin of the top row pixels to be 0 and the bottom row to be 1. Any neighbors above the top row have spin 0 and below the bottom row spin 1. For example, the bottom-right pixel has neighbors that lie below the bottom row; by default, those neighbors have spin 1. Similarly, the upper-right pixel's neighbor is above the top row, its neighbor's spin is therefore 0. For interfaces with angles close to 45°, I simulate the interface on a lattice with height > length (figure 4.11(a)). For the 45° interface, first- and second-nearest neighbors shift (up or down) by 1 pixel, while third- and fourth-nearest neighbors shift (up or down) by 2 pixels.

For these new neighbor lists, the tilted interface is the lowest-energy configuration. The energy of the tilted interface calculated using the skewed neighbor list is the same as the energy of the horizontal interface calculated using regular, unbiased neighbor list:

$$E(\theta \neq 0, \text{skewed neighbor list for angle } \theta) = E(\theta = 0, \text{regular neighbor list}). \tag{4.26}$$

I simulate the interface using the new neighbor lists. At zero temperature,



(a) 45° neighbor list (b) 0° neighbor list

Figure 4.11. (a) A lattice with a 45° interface. I show only first- and third-neighbor lists. (b) Regular (0°) neighbor lists (arrows and shaded pixels).

the interface is a staircase similar to figure 4.11(a). For non-zero temperatures, the interface fluctuates around its tilted minimum-energy configuration. Since the energy of the tilted interface (using the skewed neighbor list) is the same as that of the zero-angle interface (using the regular neighbor list), if we calculate the energy of the tilted interface at any angle using the skewed neighbor list, the result is the same as that for the zero-angle interface I discussed in section 4.3. In order to distinguish among interfaces of different angles, during the gathering of Monte-Carlo data, I calculate the energy of the tilted interface using the regular, unbiased neighbor lists. For this set of simulations, I averaged only twenty simulations instead of one hundred as for the zero-angle case. The range of interaction is up to fourth-nearest neighbors with equal interactions for all neighbors.

I divide the interface energy U (calculated using the regular, unbiased neighbor list) by the length of the interface $L/cos\theta$, where L is the horizontal length of the

lattice and θ is the angle of the interface with respect to the horizontal (figure 3.6). The result is the energy per unit interface length $U\cos\theta/L$. At zero temperature, the internal energy U equals the Helmholtz free energy F, therefore, the surface tension σ (equation 1.36) also equals $U\cos\theta/L$. Figure 4.12 plots the surface tension *vs.* angle at $kT = 0$. The longer-range interaction reduces anisotropy compared to figures 3.12(b) and 3.13(b). At $kT = 0$, the anisotropy ϕ for the equivalent fourth-nearest-neighbor interaction is 0.018 compared to 0.050 for second-nearest neighbor (figure 3.14(b)) and 0.154 for nearest-neighbor interaction (figure 3.14(a)).

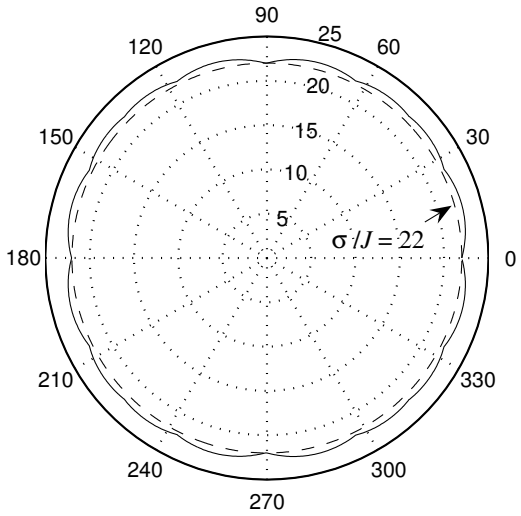


Figure 4.12. Surface tension σ for up to fourth-nearest-neighbor interactions at $kT = 0$ for the Ising model with $J_1 = J_2 = J_3 = J_4 = J$. The dashed circle is at $\sigma/J = 22$.

To find the fitting parameters for an interface at an angle, we could add a torque field τ and use the same Mathematica program I used previously (section 3.6). However, τ itself depends on the fitting parameters, so we face a chicken and egg situation. we could guess a value of τ, find the best values of the fitting parameters, then recalculate τ and iterate until the values converge.[2] This method is very laborious and the final values may depend on the initial guess. We might even fall

[2]Not only does τ depend on the fitting parameters, it also depends on the interface angle and temperature. Therefore, we need to calculate a new value of τ for each angle and temperature.

into a limit cycle. Instead, I absorb τ into the fitting parameters and use equation 4.11 directly. I fit the simulation results using the same method I used in section 4.3. Table 4.7 presents the results. Figure 4.13 shows a few of the corresponding plots.

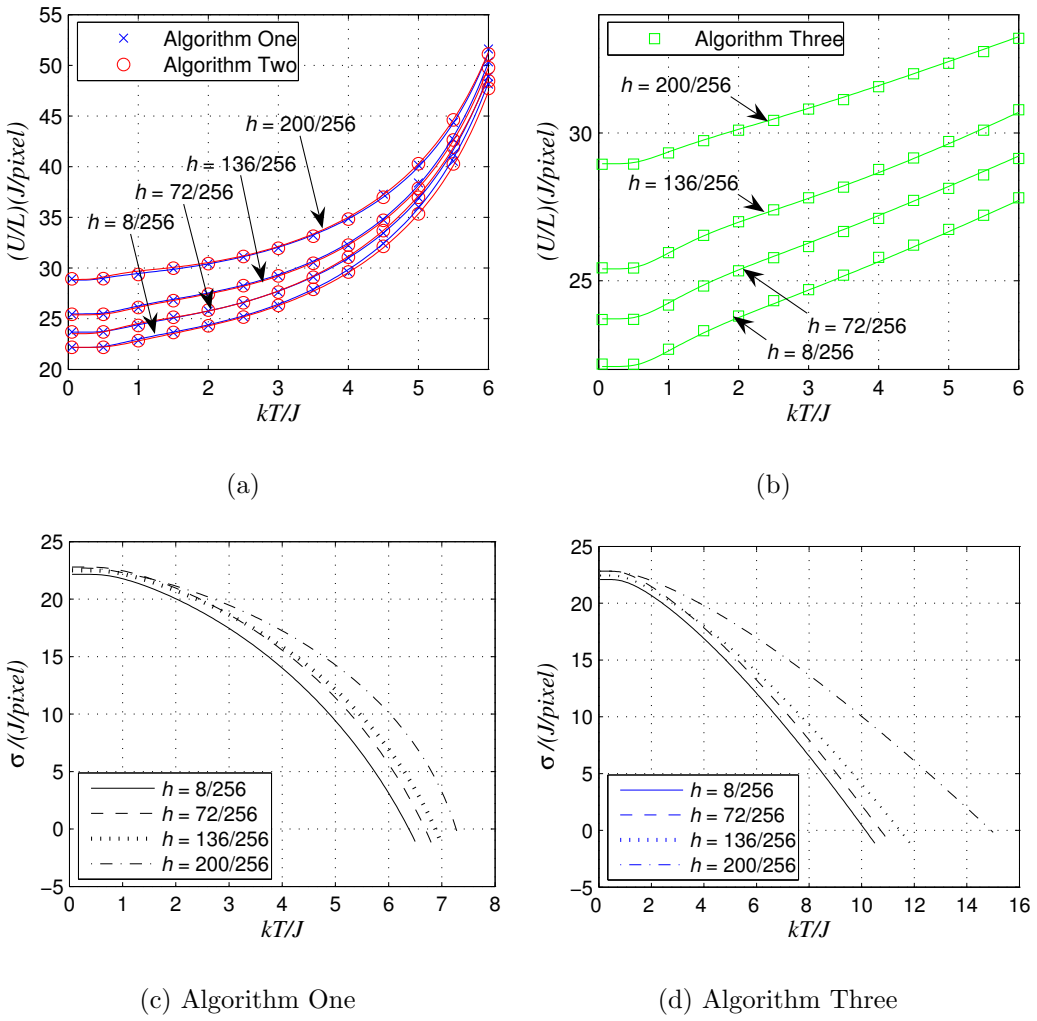


Figure 4.13. (a) and (b) Plots of U/L *vs.* kT. Solid lines are best fits. The statistical error for each point is no bigger than the symbols. (c) and (d) Surface tensions $\sigma = F\cos\theta/L$ *vs.* kT for Algorithms One and Three respectively.

The results of Algorithm One agree with Algorithm Two. Owing to better statistics, Algorithm One agrees best with Algorithm Two for the zero-angle case.

TABLE 4.7

RESULTS OF FITTING SIMULATED VALUES OF ENERGY PER UNIT LENGTH U/L TO EQUATION 4.11 OF THE MODIFIED SOS MODEL FOR INTERFACES AT VARIOUS ANGLES $h = \tan\theta$.

$h = \tan\theta$	Algorithm	ε_a/J	ε_b/J	ε_c/J	δ	$\mathcal{A}$	kT_c/J	kT_e/J
$\frac{8}{256}$	One	14.60±5.20	6.01±3.12	3.57±2.52	2.11±0.79	1.22±0.18	6.38±1.39	8.05±0.54
	Two	16.95±3.74	5.89±2.57	3.03±1.90	2.10±0.67	1.11±0.10	6.60±0.76	7.92±0.39
	Three	6.94±2.77	7.61±2.48	4.94±2.10	0.86±0.09	1.86±0.20	10.15±3.19	∞
$\frac{40}{256}$	One	18.40±6.78	4.73±4.74	2.42±3.18	1.82±1.06	1.10±0.19	6.59±0.82	7.92±1.05
	Two	16.22±5.17	5.88±3.12	3.41±2.46	2.09±0.79	1.16±0.16	6.54±1.17	7.98±0.53
	Three	6.94±3.41	7.92±3.21	5.32±2.71	0.88±0.15	1.87±0.29	10.25±3.86	∞
$\frac{72}{256}$	One	17.90±6.27	5.73±3.79	3.24±2.91	2.06±0.95	1.12±0.18	6.70±1.14	7.95±0.70
	Two	13.25±4.73	6.51±2.31	4.44±2.10	2.19±0.57	1.33±0.20	6.39±1.34	8.33±0.39
	Three	7.31±2.90	7.83±2.68	5.17±2.27	0.84±0.11	1.90±0.22	10.77±3.38	∞
$\frac{104}{256}$	One	17.96±2.22	7.17±1.28	4.15±1.05	2.47±0.39	1.11±0.06	6.85±0.35	7.92±0.12
	Two	16.86±1.84	6.55±0.99	4.00±0.83	2.26±0.27	1.17±0.06	6.74±0.36	8.05±0.10
	Three	7.86±3.26	7.41±2.72	4.91±2.31	0.81±0.09	1.91±0.25	10.98±3.78	∞
$\frac{136}{256}$	One	18.43±5.81	6.88±2.97	4.32±2.55	2.38±0.86	1.12±0.16	6.87±0.91	7.91±0.45
	Two	18.79±2.13	6.98±1.10	4.29±0.94	2.41±0.32	1.10±0.06	6.92±0.33	7.93±0.11
	Three	8.94±2.91	8.82±2.83	6.18±2.40	0.96±0.20	1.68±0.27	11.41±2.41	∞
$\frac{168}{256}$	One	25.69±6.04	5.57±3.35	3.12±2.48	2.07±0.88	0.94±0.12	7.14±0.12	7.67±0.58
	Two	21.66±5.93	6.45±2.78	4.12±2.36	2.26±0.76	1.05±0.15	7.10±0.70	7.91±0.44
	Three	14.20±3.53	8.40±2.71	5.72±2.26	1.04±0.19	1.37±0.20	12.85±2.17	197±145
$\frac{200}{256}$	One	28.75±7.26	4.51±3.29	2.91±2.46	1.79±0.74	0.91±0.14	7.25±0.21	7.71±0.67
	Two	22.52±6.90	6.38±2.69	4.72±2.43	2.22±0.70	1.06±0.17	7.21±0.90	7.99±0.47
	Three	21.94±5.01	5.89±2.04	3.46±1.63	0.95±0.05	1.14±0.16	14.98±3.56	∞
$\frac{232}{256}$	One	30.15±7.14	5.22±2.75	3.91±2.27	1.97±0.68	0.90±0.13	7.32±0.09	7.67±0.49
	Two	22.63±8.90	7.09±3.17	5.84±2.93	2.42±0.88	1.07±0.23	7.29±1.15	8.01±0.57
	Three	32.11±6.26	6.34±2.33	3.67±1.84	1.09±0.09	0.86±0.11	16.85±3.26	74±22

The predicted errors for the fitting parameters are very large, in one case (Algorithm One, $\tan\theta = 40/256$), the errors are even bigger than the predicted values. However, the 95% confidence fits differ little from the best fits. In other words, big changes in the values of the fitting parameters translate to small changes in the values of the fitting equation, indicating that the empirical equation 4.11 does not have the optimal functional form. That is, the simulated results contain physics beyond what I have considered in my modified SOS model, which is not surprising, given the heuristic nature of my model.

We cannot draw any quantitative conclusion beyond that Algorithms One and Two agree qualitatively. Even taking the errors into account, Algorithm Three definitely differs from Algorithms One and Two.

Figure 4.13 plots the variation of the surface tension $\sigma = F\cos\theta/L$ with temperature kT. We can deduce the anisotropy of the lattice by drawing a vertical line at a temperature kT on figures 4.13(c) and 4.13(d). If the σs for different angles at that temperature are the same, then the lattice is isotropic. If the values differ, then the lattice is anisotropic. The greater the spread of values, the more anisotropic the lattice. Since the spread of values increases with increasing temperature, the anisotropy increases with temperature, showing that the lattice is most isotropic for $kT = 0$, unlike the SOS models with first- and second-nearest-neighbor interactions which are most isotropic for $kT \neq 0$ (figures 3.12(a) and 3.13(a)). Algorithms One and Two are similar while Algorithm Three is more anisotropic.

I do not show the entropy and specific heat capacity *vs.* kT as they are similar to figures 4.5(b) and 4.5(c).

4.6 Dependence of kT_c on the Range of Interaction

In the last chapter (chapter 3), I analyzed the critical temperatures in the second-nearest-neighbor SOS model as a function of the coordination number z by changing the ratio of the first- and second-neighbor coupling constants J_2/J_1. I found that the SOS model agrees with Domb and Potts's Ising model results (figure 3.9). Here, I keep the ratios $J_4/J_1,\ J_3/J_1,\ J_2/J_1$ equal to one and increase the coordination number z by increasing the interaction range. Because all coupling constants are equal, the coordination number equals the number of neighbors within the interaction range.

I simulated a horizontal interface using Algorithms One and Three, but did not use Algorithm Two. As in section 4.5, I calculate the energy of interface after it has equilibrated and divide the energy by the length of the lattice L. Figures 4.14 show the normalized energy (divided by $\varepsilon_{r=0}$, table 4.8) per unit length as a function of temperature for four interaction ranges.

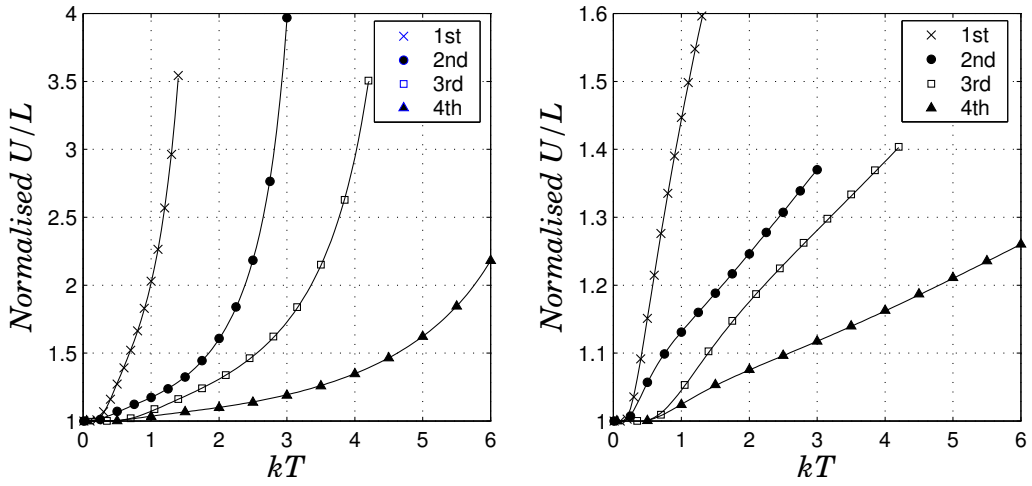
TABLE 4.8

COORDINATION NUMBERS AND ENERGIES OF STEPS OF HEIGHT ZERO AND ONE AS A FUNCTION OF THE INTERACTION RANGE FOR EQUAL JS ON A SQUARE LATTICE.

Up to nth neighbors	z	$\varepsilon_{r=0}$	$\varepsilon_{r=1}$
1st	4	$2J$	$4J$
2nd	8	$6J$	$8J$
3rd	12	$10J$	$18J$
4th	20	$22J$	$30J$

I then fit the energy per unit length to equation 4.11 using the same Matlab program as before (appendix C). I found different sets of parameter values that fit the simulation results well, but only one that satisfied all the criteria I discussed in section 4.3. Table 4.9 shows the results of this curve fitting. I also solve for kT_c by

solving $\sigma = 0$ numerically. kT_e equals to $\varepsilon_b / \ln \delta$. Table 4.10 presents the results. Finally, figure 4.15 plots $zK_c (= z/kT_c)$ *vs.* z for Algorithms One and Three and the results of Domb and Potts (equation 3.28).



(a) Algorithm One (b) Algorithm Three

Figure 4.14. Normalized energy per unit length for four different interaction ranges. Error bars are smaller than the size of the symbols. Solid lines are best fits.

TABLE 4.9

RESULTS OF FITTING EQUATION 4.11 TO SIMULATION RESULTS IN FIGURE 4.14. THE TOP FOUR ROWS ARE RESULTS FOR ALGORITHM ONE, THE BOTTOM FOUR ROWS ARE FOR ALGORITHM THREE.

Range	ε_a	ε_b	ε_c	δ	$\mathcal{A}$
1st	0.44±0.23	2.67±0.38	1.25±0.25	3.81±0.79	1.19±0.05
2nd	5.05±1.88	3.26±1.78	1.74±1.30	2.53±1.27	0.90±0.08
3rd	5.94±1.05	6.15±1.25	2.69±0.84	3.30±0.81	1.16±0.04
4th	14.78±1.47	5.78±0.91	3.36±0.72	2.05±0.22	1.21±0.05
1st	0.18±0.88	1.74±0.72	0.66±0.66	0.55±0.11	2.40±0.68
2nd	-0.45±0.16	4.04±0.19	3.02±0.16	0.64±0.01	2.33±0.06
3rd	-0.91±1.01	8.52±1.72	5.48±1.51	0.79±0.19	2.18±0.34
4th	7.01±0.98	7.52±0.86	4.93±0.72	0.89±0.04	1.84±0.08

In figure 4.15, the simulation results display similar trends to theoretical predictions for the Ising model, but the critical temperatures from simulations are

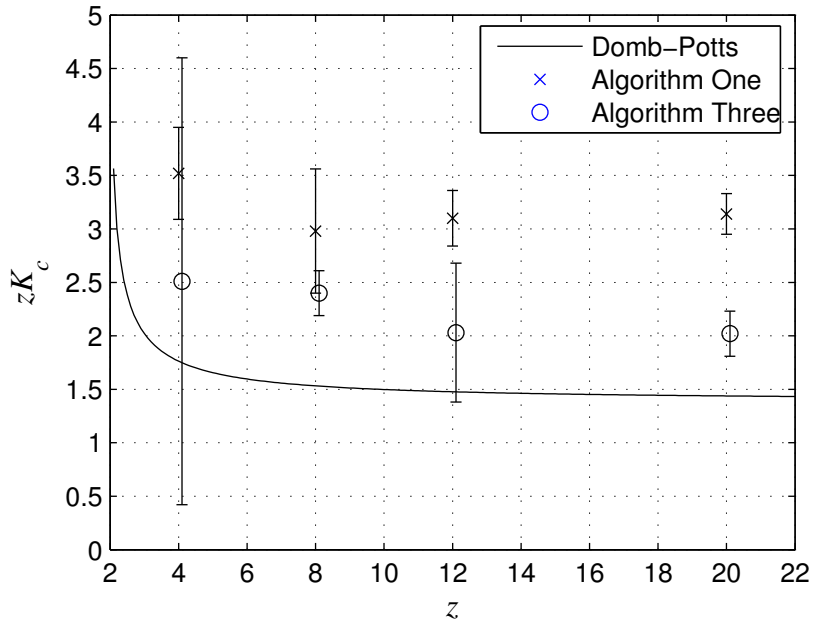


Figure 4.15. Comparison of zK_c *vs.* z for Algorithms One and Three (table 4.10) and the Ising model as predicted by Domb and Potts (equation 3.28).

TABLE 4.10

CRITICAL TEMPERATURES OF VARIOUS INTERACTION RANGES FOR THE SOS MODEL WITH EQUAL JS. TOP FOUR ROWS ARE FOR ALGORITHM ONE, BOTTOM FOUR FOR ALGORITHM THREE.

Range	z	kT_c	kT_e	z/kT_c	z/kT_e
1st	4	1.13±0.14	1.99±0.00	3.52±0.43	2.01±0.00
2nd	8	2.68±0.52	3.51±0.27	2.98±0.58	2.28±0.17
3rd	12	3.87±0.32	5.15±0.09	3.10±0.26	2.33±0.04
4th	20	6.37±0.38	8.04±0.10	3.14±0.19	2.49±0.03
1st	4	1.59±1.32	∞	2.51±2.09	0
2nd	8	3.33±0.29	∞	2.40±0.21	0
3rd	12	5.92±1.89	∞	2.03±0.65	0
4th	20	9.90±1.02	∞	2.02±0.21	0

consistently lower than those the theory predicts.[3] This discrepancy could happen if I consistently overestimated the energy per unit length by systematically underestimating the length of the interface. I set the interface length to the lattice length L regardless of the simulation temperature and interface roughness. I present two

[3]Higher values of K_c mean lower values of kT_c, because $K_c = J/kT_c$.

possible methods to improve my estimate of the interface length.

The first method simply calculates the contribution to the total energy from first-neighbor interactions. The energy from first-neighbor interactions divided by two is the interface length, l_{segment}, because I double count links.

The second method counts the number of surface pixels, $l_{\text{surface pixel}}$, defined as pixels with at least one mismatched nearest neighbor. Again, I divide the total number of surface pixels by two because of double counting.

When the interface is flat (either 0° or 90°), both methods give the same length, equal to L. Obviously, at higher temperatures, the first yields a longer interface length than the second because a surface pixel may have more than one mismatched link. Therefore,

$$\ell_{\text{segment}} \geqslant \ell_{\text{surface pixel}} \geqslant L, \tag{4.27}$$

which implies:

$$\langle \varepsilon \rangle_{\text{segment}} \leqslant \langle \varepsilon \rangle_{\text{surface pixel}} \leqslant \langle \varepsilon \rangle_{L}. \tag{4.28}$$

As the length of the interface is no longer constant as the SOS model assumes, but increases with temperature, we must express the surface tension as the derivative of the surface free energy with respect to the interface length (equation 1.10), instead of the ratio of the free energy to the interface length:

$$\sigma = \left(\frac{\partial F}{\partial \ell}\right)_T = -\left(\frac{\partial F}{\partial T}\right)_\ell \Big/ \left(\frac{\partial \ell}{\partial T}\right)_F. \tag{4.29}$$

However, one problem remains. As $\ell = \ell(T)$ and $F(\ell, T) = F(\ell(T), T)$, the total derivative of F with respect to T is:

$$\begin{aligned} \frac{dF}{dT} &= \left(\frac{\partial F}{\partial T}\right)_\ell + \left(\frac{\partial F}{\partial \ell}\right)_T \frac{d\ell}{dT} \\ &= \left(\frac{\partial F}{\partial T}\right)_\ell + \sigma \frac{d\ell}{dT}. \end{aligned} \tag{4.30}$$

I have $\frac{dF}{dT}$ and $\frac{d\ell}{dT}$ from simulation results, but not $\left(\frac{\partial F}{\partial T}\right)_\ell$. In order to proceed, one must first separate the ℓ dependence of F from the explicit T dependence of F. I leave this problem to future research.

4.7 Discussion

I have successfully used three different modifications of the Metropolis algorithm in my phase-separation simulations. Two of the algorithms, One and Two, which violate detailed balance minimally, produce interfaces that agree with each other thermodynamically and morphologically. Dynamically, Algorithms One and Two are not exactly the same. Algorithm One equilibrates faster than Algorithm Two. Algorithm Three, which violates detailed balance to a greater degree, gives interfaces decidedly different from the first two, in both morphology, dynamics and thermodynamics.

Most current simulations [46, 47, 48, 63, 117] use Algorithm Three. I strongly recommend that simulators modify their acceptance algorithm using equation 2.34 to convert Algorithm Three to Algorithm Two. The minimal increase in complexity will greatly increase the accuracy of their results. Algorithm Two also converges significantly faster than Algorithm Three.

This chapter has also introduced an empirical, modified SOS model. The interface in the modified SOS model has two critical temperatures. The first, kT_c is the temperature at which the surface tension vanishes. The second, kT_e is the temperature at which the internal energy, entropy, specific heat and interface width diverge. kT_e is always greater than kT_c. kT_e relates to the existence of overhangs. In calculations of phase separation using the full Ising model, we expect these temperatures to be the same, $kT_c = kT_e$. Since the modified SOS model predicts a finite value for the temperature kT_e, the modified SOS model represents an improvement

over the standard SOS model.

Results from all three algorithms lie somewhere between those of the Ising model and the SOS model. Algorithms One and Two behave more like the Ising model where the interface width diverges at a finite temperature, while Algorithm Three behaves more like the SOS model where the interface width does not diverge at a finite temperature.

Although at zero temperature the square lattice is more isotropic as the range of interaction increases, it becomes more anisotropic as the temperature increases. This result is unexpected.

I suggest a number of interesting extensions:

1. Extend the simulation to three dimensions. In three dimensions, the roughening transition temperature, kT_r, is finite, as opposed to zero in two dimensions.
2. Perform simulations in triangular and hexagonal lattices to see if the specific-heat capacity peaks at low temperatures and if the lattices become less isotropic as the interaction range increases at non-zero temperatures. Investigate the dependence of critical temperature on the type of lattice and dimensionality.
3. Simulate with nonequal coupling constants in the x and y directions.
4. Simulate with nonequal coupling constants for different interaction ranges and to improve isotropy.
5. Simulate more accurately in the $kT < J$ and $\theta > 0$ regions.
6. Solve the third-nearest-neighbor interaction problem analytically using the transition matrix method.
7. Calculate w_m (equation 4.25) as a function of distance d.
8. I assumed that the interface height has no autocorrelation. Checking this assumption would be interesting. If it fails, what is the correlation length? Does it depend on the size of the lattice? At zero temperature, the interface is perfectly ordered, so the correlation length is infinite. Does the correlation length go to zero for all non-zero temperatures or at kT_c?

CHAPTER 5

SIMULATIONS OF BROWNIAN MOTION

In this chapter I simulate the Brownian motion of droplets using the algorithms from chapter 2 to investigate the effects of temperature and droplet size on the droplet's diffusion constant and the medium's viscosity and compare them to theory. I also investigate the equilibrium droplet shape and energy.

Jan Ingenhousz first observed Brownian motion in 1785 [32, 96]. Botanist Robert Brown subsequently rediscovered it in 1827 when he observed under a microscope that pollen suspended in water moved erratically. Einstein (1905), in one of the early confirmations of the atomic theory, first explained Brownian motion as the result of the random bombardment of molecules obeying the Maxwell velocity distribution [33]. Although the most celebrated, Einstein's paper was not the final word on Brownian motion. Norbert Wiener [29] devised a purely mathematical model of stochastic processes to explain Brownian motion. The Wiener process is a form of Markov process where the position of a particle depends only on its immediately preceding position. Asymptotically, Brownian motion approaches a Weiner process at large times [41]. Langevin, Smoluchowski, Uhlenbeck, Orstein and many others refined and developed the theory. For a brief historical account and the theory of Brownian motion, please refer to [68, 77, 89, 120] or any statistical thermodynamics textbook. For a selection of early papers, please see [116].

Consider a macroscopic (compared to the molecules of a fluid) particle or droplet

of mass m and radius a that is suspended in a fluid medium. Because of the constant bombardment of molecules, the particle or droplet moves erratically. Analysis of Brownian motion aims to derive the displacement (or rather, the mean-squared displacement) of the droplet as a function of time and other fundamental parameters like temperature, viscosity and droplet size. I will only state the governing equation of Brownian motion and its assumptions. I encourage the reader to refer to one of the references for further details.

The usual starting point for modeling Brownian Motion is the Langevin equation:[1]

$$m\frac{\partial v}{\partial t} = -\Omega v + F_r(t), \tag{5.1}$$

where Ω is the friction coefficient and $F_r(t)$ is the random force due to collisions with molecules. Deriving $v(t)$ and ultimately $r(t)$, where r is the displacement from an initial position, requires a few assumptions:

1. The friction force obeys Stoke's law [69],
$$\Omega v = 6\pi\eta a v, \tag{5.2}$$
where η is the dynamic viscosity of the fluid.
2. $F(t)$ is independent of $v(t)$.
3. Because $F_r(t)$ is completely random, the ensemble average $\langle F_r(t)\rangle$ is zero.
4. $F_r(t)$ is an extremely fast-varying function of t. A time interval Δt exists such that $F_r(t)$ varies many times and $v(t)$ is essentially constant within the interval. The correlation function $\langle F_r(t')F_r(t'')\rangle$ is a rapidly decreasing function of $|t' - t''|$. For $|t' - t''| \simeq \Delta t$, $\langle F_r(t')F_r(t'')\rangle = 0$.

Using the equipartition theorem, the solution of the Langevin equation is:

$$\langle x^2\rangle = \frac{2mkT}{\Omega^2}(\frac{\Omega}{m}t - 1 + e^{-\frac{\Omega}{m}t}). \tag{5.3}$$

Figure 5.1 plots $\log\langle x^2\rangle$ vs $\log(t)$. For small t, the slope is 2 (dotted line). For large t, the slope is 1. $\Omega/m = 1$ in this example.

[1] An alternative approach uses the Fokker-Planck Equation.

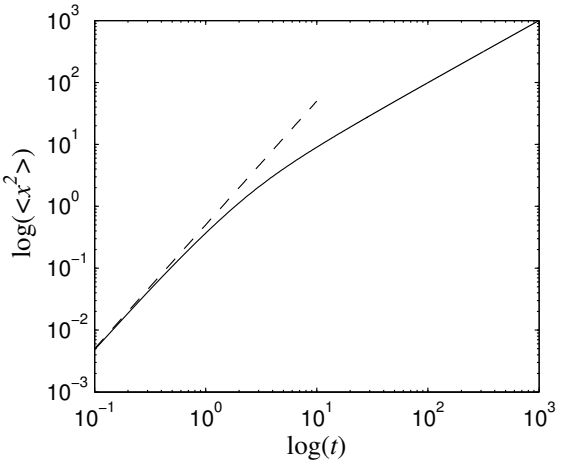


Figure 5.1. Log-log plot of the solution of the Langevin equation 5.3. In this plot, I have set $\Omega/m = 1$ and $2kT/\Omega = 1$.

Over very short times, $\frac{\Omega}{m}t \ll 1$ (dotted line in figure 5.1), equation 5.3 reduces to:

$$\lim_{t \ll m/\Omega} \langle x^2 \rangle = \frac{kT}{m} t^2. \tag{5.4}$$

For very long times, $\frac{\Omega}{m}t \gg 1$, equation 5.3 reduces to Einstein's result:

$$\lim_{t \gg m/\Omega} \langle x^2 \rangle = \frac{2kT}{\Omega} t = \frac{kT}{3\pi\eta a} t. \tag{5.5}$$

If we define the diffusion constant to be:

$$D = \frac{2kT}{\Omega} = \frac{kT}{3\pi\eta a}, \tag{5.6}$$

then for long times,

$$\lim_{t \gg m/\Omega} \langle x^2 \rangle = Dt, \tag{5.7}$$

and, since we expect all dimensions to be equivalent,

$$\begin{aligned} \lim_{t \gg m/\Omega} \langle r^2 \rangle_{2D} &= 2\langle x^2 \rangle &= 2Dt, \\ \lim_{t \gg m/\Omega} \langle r^2 \rangle_{3D} &= 3\langle x^2 \rangle &= 3Dt. \end{aligned} \tag{5.8}$$

A plot of $\langle r^2 \rangle$ vs t at large times should yield a straight line with a slope proportional to D.

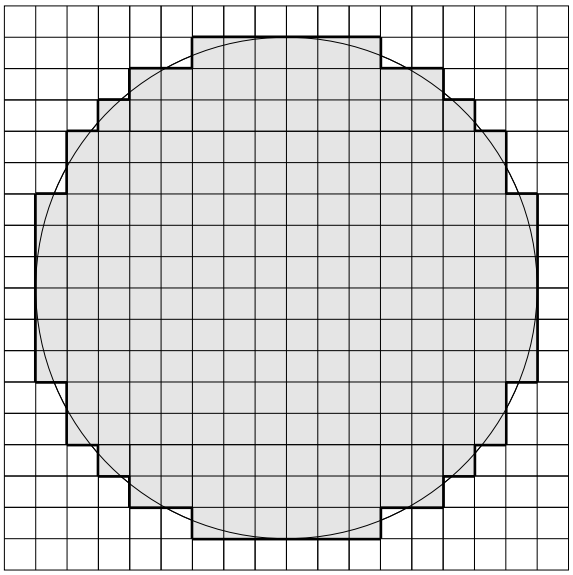


Figure 5.2. A digitized droplet of radius 8 pixels. Pixels within the droplet are gray (spin 1). A medium (white pixels) having spin 0 surrounds the droplet.

To simulate Brownian motion on a discrete lattice, I first represent a droplet of radius a on the lattice, as I did interfaces with non-zero slopes. I draw a circle of radius a on the lattice. If a pixel lies more that half within the circle, then it receives spin 1, otherwise, it receives spin 0. The Mathematica program in appendix D automates the digitization of droplets. Figure 5.2 shows a digitized droplet of radius 8 pixels.

Table 5.1 and figure 5.3(a) show how the area A_T of the digitized droplet varies with its radius a. Table 5.1 and figure 5.3(b) show how the energy of a droplet varies with its radius for various interaction ranges. The energy of the medium equals to the energy of the droplet, so the energy of the entire lattice is twice that of the droplet.

The area A_T of the droplet is near πa^2 for large radii. The difference between A_T and πa^2 is less than 5% for $a \geqslant 4$ pixels. I fit the droplet energy to:

$$E = pa + q. \tag{5.9}$$

TABLE 5.1

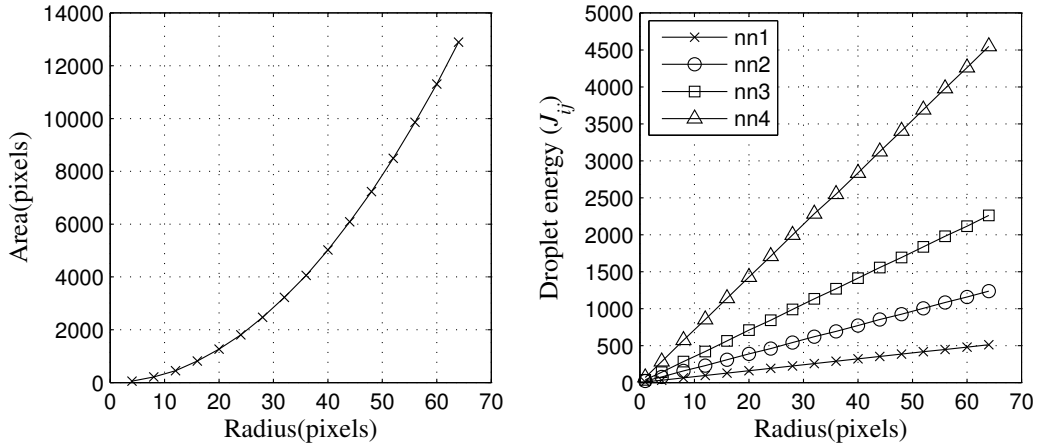
AREAS (NUMBER OF PIXELS) OF DIGITIZED DROPLETS AS A FUNCTION OF RADIUS. NN4 MEANS UP TO FOURTH-NEAREST-NEIGHBOR INTERACTIONS. ALL INTERACTIONS ARE OF EQUAL STRENGTH.

radius	area	Droplet energy(J_{ij})			
a	A_T	nn1	nn2	nn3	nn4
3	32	24	60	108	220
4	52	32	76	140	284
5	80	40	100	180	356
6	112	48	116	212	420
7	156	56	132	244	500
8	208	64	156	284	572
12	448	96	228	420	852
16	812	128	308	564	1140
20	1264	160	388	708	1428
24	1804	192	460	844	1708
28	2472	224	540	988	1996
32	3228	256	620	1132	2284
36	4060	288	692	1268	2548
40	5024	320	772	1412	2836
44	6092	352	852	1556	3124
48	7232	384	924	1692	3404
52	8492	416	1004	1836	3692
56	9856	448	1084	1980	3980

Table 5.2 gives the results of linear regression fits to equation 5.9. Within error, we may assume $q = 0$, so the droplet energy is proportional to its radius.

5.2 Modifying the Ising Model and the Metropolis Algorithm

If we use the Ising-model Hamiltonian in equation 2.1 in conjunction with any of the modified Metropolis Algorithms in section 2.8, the Brownian droplet would disappear within a few MCS. The reason is that the lowest energy of a lattice is zero, *i.e.* when all the pixels are of the same spin. As the dynamics tries to minimize the energy of the lattice, the droplet pixels will flip into spins of the surrounding medium.



(a) (b)

Figure 5.3. Properties of digitized droplets. (a) Area *vs.* radius. Solid line is πa^2. (b) Energy *vs.* radius. Solid lines are best fits.

TABLE 5.2

FITS OF THE ENERGY OF A DIGITIZED CIRCULAR DROPLET TO EQUATION 5.9. $p/2\pi$ IS THE ENERGY PER UNIT PERIMETER LENGTH. THE TOTAL ENERGY OF THE LATTICE IS TWICE THE ENERGY OF THE DROPLET. THE ENERGY PER UNIT LENGTH OF A CIRCULAR DROPLET IS CONSISTENTLY HIGHER THAN THAT OF A FLAT INTERFACE (SEE TABLE 4.8).

nn	$p(J/\text{pixel})$	$p/2\pi(J/\text{pixel})$	$q(J)$
1	8	1.273±0.000	0
2	19.31±0.02	3.073±0.003	-0.3±0.8
3	35.31±0.02	5.620±0.003	-0.3±0.8
4	71.02±0.06	11.303±0.010	-0.6±1.9

The problem does not occur in my simulations of phase separation because neither set of spins completely encircles the other. Whenever a different spin encircles a region of uniform spin, the enclosed region will eventually vanish. Nothing in the original Hamiltonian (equation 2.1) prevents the droplet from vanishing or energy from continuously dissipating in the lattice. In other words, the model has no latent heat. In real life, we need energy to vaporize liquid. The Ising model incurs no such

penalty.

In order to prevent the Brownian droplet from vanishing, we need to add a term to the Hamiltonian that penalizes the droplet if it strays from its *target area*. This *area constraint* balances the surface tension and keeps the size of the Brownian droplet close to its target area. I propose the following modification to the Ising model to simulate Brownian motion, it is the same constraint as in reference [48]:

$$H_{\text{droplet}} = \sum_{i \in \text{droplet}} \sum_{j=1}^{z} J_{ij}(1 - \delta_{\sigma_i,\sigma_j}) + J_A(A(t) - A_T)^2, \tag{5.10}$$

where i ranges over all the pixels within the droplet and j ranges over the neighbors of i. J_{ij} is the coupling constant between pixel i and pixel j. If all couplings are equal, $J_{ij} = J$. J_A is the strength of the area constraint. Its units are energy per unit area squared. J_A is equivalent to the latent heat of vaporization. $A(t)$ and A_T are the area at time t and the target area of the droplet. Henceforth, I will drop the subscript "droplet" as I am only concerned with the droplet energy.

How does the area constraint affect detailed balance? Algorithms One and Three will remain the same and use the Metropolis acceptance probability in equation 2.24. We need to modify the acceptance probability of Algorithm Two (equation 2.34) so that it normalizes properly and the ratio of forward and backward probabilities satisfies the Boltzmann ratio.

I divide the droplet energy into two parts: E_J and E_A, the spin-spin-interaction energy and the area-constraint energy respectively:

$$\begin{aligned} E &= E_J + E_A, \\ E_J &= \sum_{i,j} J_{ij}(1 - \delta_{\sigma_i,\sigma_j}), \\ E_A &= J_A(A(t) - A_T)^2. \end{aligned} \tag{5.11}$$

Since all the algorithms only flip single spins at each attempt, the change in droplet area is always ± 1 pixel. Denoting the initial area A_μ and the final area A_ν, the

change in area is $\Delta A = A_\nu - A_\mu$. If the forward direction is $\Delta A = +1$, then the backward direction is $\Delta A = -1$, and *vice versa*. The change in E_A is thus:

$$\begin{aligned} \Delta E_A &= J_A[(A_\nu - A_T)^2 - (A_\mu - A_T)^2] \\ &= J_A \Delta A[\Delta A + 2(A_\mu - A_T)]. \end{aligned} \tag{5.12}$$

From equation 5.12, one can easily show:

$$\Delta E_A(\mu \to \nu) = -\Delta E_A(\nu \to \mu), \tag{5.13}$$

and since from section 2.6,

$$\Delta E_J(\mu \to \nu) = -\Delta E_J(\nu \to \mu), \tag{5.14}$$

therefore,

$$\Delta E(\mu \to \nu) = -\Delta E(\nu \to \mu). \tag{5.15}$$

Because of equation 5.15, the Metropolis Algorithm satisfies the condition of detailed balance for all transitions. For Algorithm One, all transitions except nucleation satisfy detailed balance. As before, Algorithm Three still violates detailed balance for all the transitions. However, the modified acceptance probability in Algorithm Two:

$$A(\mu \to \nu) = \begin{cases} \frac{z+\Delta E_J}{z-\Delta E_J} e^{-\Delta E/kT}, & \text{if } \Delta E > 0 \\ 1, & \text{if } \Delta E \leq 0 \end{cases}, \tag{5.16}$$

no longer satisfies detailed balance, since the prefactor involves the spin-spin interaction while the exponent involves the total energy. The problem with equation 5.16 is that ΔE_A is independent of ΔE_J. We cannot normalize the probability when we combine ΔE_A and ΔE_J together and consider them as a whole (as in ΔE). To circumvent this problem, we must consider four cases separately: $\Delta E_J > 0, \Delta E_A > 0$;

$\Delta E_J > 0, \Delta E_A \leqslant 0$; $\Delta E_J \leqslant 0, \Delta E_A > 0$; $\Delta E_J \leqslant 0, \Delta E_A \leqslant 0$; and modify the acceptance probability to:

$$A(\mu \to \nu) = \begin{cases} \frac{z+\Delta E_J}{z-\Delta E_J} e^{-(\Delta E_J + \Delta E_A)/kT}, & \text{if } \Delta E_J > 0, \Delta E_A > 0 \\ \frac{z+\Delta E_J}{z-\Delta E_J} e^{-\Delta E_J/kT}, & \text{if } \Delta E_J > 0, \Delta E_A \leqslant 0 \\ e^{-\Delta E_A/kT}, & \text{if } \Delta E_J \leqslant 0, \Delta E_A > 0 \\ 1, & \text{if } \Delta E_J \leqslant 0, \Delta E_A \leqslant 0 \end{cases} . \tag{5.17}$$

The reader can substitute equation 5.17 into equation 2.33 to prove to herself that the correct Boltzmann ratio results. I call this Algorithm Two-A. Algorithm Two-A follows the same procedures as Algorithm Two, using equation 5.17 instead of equation 5.16 as the acceptance probability.

To bridge the gap between Algorithm Two-A and Three, I define Algorithm Two-B which follows Algorithm Three but modifies its acceptance probability to:

$$A(\mu \to \nu) = \begin{cases} e^{-(\Delta E_J + \Delta E_A)/kT}, & \text{if } \Delta E_J > 0, \Delta E_A > 0 \\ e^{-\Delta E_J/kT}, & \text{if } \Delta E_J > 0, \Delta E_A \leqslant 0 \\ e^{-\Delta E_A/kT}, & \text{if } \Delta E_J \leqslant 0, \Delta E_A > 0 \\ 1, & \text{if } \Delta E_J \leqslant 0, \Delta E_A \leqslant 0 \end{cases} . \tag{5.18}$$

Droplet simulations begin by creating a circular digitized droplet of area A_T at the center of the lattice, then employ Algorithms Two-A, Two-B and Three. Although no molecules bombard the droplet, each spin-flip attempt is analogous to a collision between the droplet and a molecule. I track the center of mass of the droplet at regular intervals up to 10,000 Monte-Carlo steps. To obtain the mean-squared displacement, I average the squared displacements of one hundred droplets. I simulated droplets of radii 3, 4, 5, 6, 7, 8, 12 and 16 pixels, corresponding to target areas of 32, 52, 80, 112, 156, 208, 448 and 812 pixels. For each droplet size, I also varied the temperature from $kT = 0.5J$ to $6.0J$. All simulations used fourth-nearest-neighbor interactions with $J_1 = J_2 = J_3 = J_4 = J \equiv J_A$.

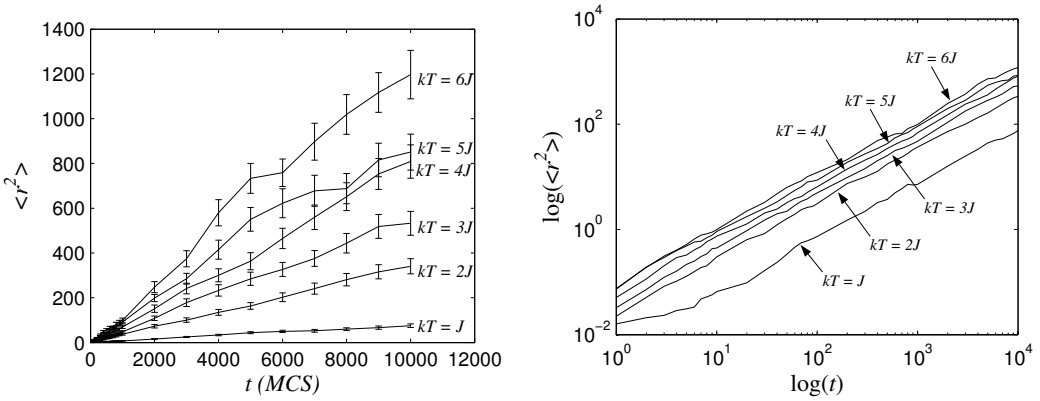


Figure 5.4. (a) The mean-squared displacement *vs.* time for droplets of radius 3 pixels ($A_T = 32$ pixels) using Algorithm Two-A. The unit of mean-squared displacement is pixels squared. The slope is proportional to the diffusion constant D. (b) log-log plot of (a). The intercept is proportional to D. The slope is close to 1, which we expect because the mean-squared displacement is linear in t.

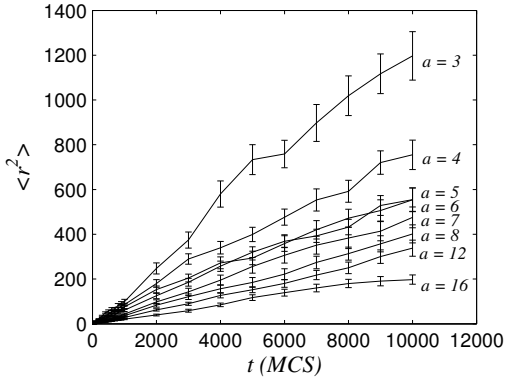


Figure 5.5. The mean-squared displacement *vs.* time for droplets of different radii for $kT = 6.0J$ using Algorithm Two-A. The unit of radius is pixels. The smaller the droplet, the higher the rate of diffusion, as equation 5.6 predicts.

Figure 5.4(a) shows that the mean-squared displacement of droplets is indeed proportional to time. Also, the higher the temperature, the greater the rate of diffusion, as equation 5.6 predicts. Figure 5.4(b) shows a log-log plot of figure 5.4(a) to emphasize the linearity at small times.

Because the Hamiltonian has no inertia (equation 5.10), the motion is infinitely overdamped and the droplet moves for a distance much smaller than its radius.

If we set $m = 0$, then equation 5.3 reduces to equation 5.5 for all t. The time-constant m/Ω that marks the transition from short-time behavior ($\sim t^2$) to long-time behavior ($\sim t$) is zero. For temperatures higher than J, the slope of the log-log plot reaches one as early as $0.01MCS$ (not shown in figure 5.4(b)).

For temperatures $kT = J$ and lower, the slope of the log-log plot at small times is less than one because the droplet takes a finite time to equilibrate. As I found in section 4.1, the lower the temperature, the longer the interface takes to equilibrate.

Figure 5.5 plots the mean-squared displacement *vs.* time for droplets of different sizes at the same temperature. The bigger the droplet, the slower the diffusion. The corresponding figures for Algorithms Two-B and Three are similar to figures 5.4 and 5.5 but with different slopes.

Since the theory of Brownian motion predicts that the mean-squared displacement $\langle r^2 \rangle$ is proportional to $(kT/a)t$ (equation 5.5), plotting $\langle r^2 \rangle$ *vs.* the rescaled time $(kT/a)t$ should collapse all the lines into a single line with slope $2/(3\pi\eta)$. Thus the slope of the graph directly determines the viscosity η of the simulations. Figures 5.6 superimpose the different plots of mean-squared displacement *vs.* $(kT/a)t$ for all droplet sizes and temperatures $kT \geqslant 2J$.

Table 5.3 gives the average slope for each droplet size and temperature for each algorithm.

TABLE 5.3

AVERAGE SLOPE OF $\langle r^2 \rangle$ VS. $(kT \cdot t)/a$.

Algorithm	Slope$(=\frac{2}{3\pi\eta})$	$\eta(\frac{J \cdot MCS}{pixel^3})$
Two-A	0.0554±0.0006	3.83±0.04
Two-B	0.0482±0.0004	4.40±0.04
Three	0.0414±0.0006	5.13±0.07

The simulation temperatures ranged from $0.5J$ to $6.0J$, but I only used temperatures at or above $2.0J$ in calculating the viscosity because the lines for temperatures

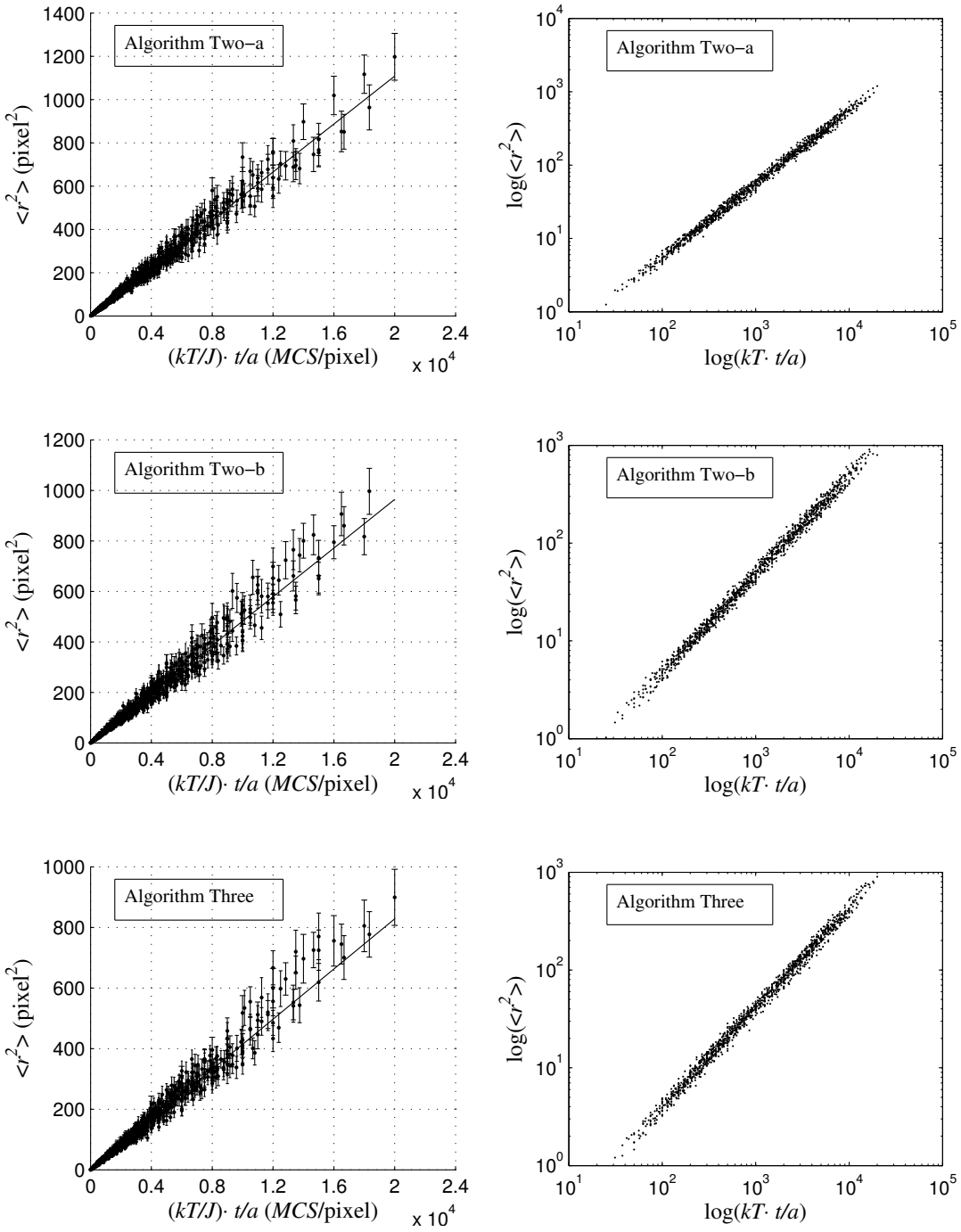


Figure 5.6. Mean-squared displacement *vs.* rescaled time $(kT/a)t$ for Algorithms Two-A, Two-B and Three. Droplet radii included in the plots are 3, 4, 5, 6, 7, 8, 12 and 16 pixels. For each droplet size, I include temperatures $2.0J$, $2.5J$, $3.0J$, $3.5J$, $4.0J$, $4.5J$, $5.0J$, $5.5J$ and $6.0J$. The straight lines are results from linear regressions. On the right are the corresponding log-log plots for each algorithm.

below $kT = 2J$ do not collapse onto the same straight lines in figure 5.6.

Figure 5.7 shows the low-temperature and high-temperature plots together. The slopes in the two regimes clearly differ.

In figure 5.7, the viscosity depends both on the temperature and droplet size. For a fixed droplet size (left column), the lower the temperature, the smaller the slope and the greater the viscosity. However, the slope does not increase monotonically with temperature. Above a critical temperature, the slope and viscosity stay constant. The bigger the droplet, the higher the critical temperature. For Algorithm Two-A, the critical temperatures range from $1.0J$ to $1.5J$. For Algorithms Two-B and Three, the critical temperatures range from $1.5J$ to $2.0J$.

At fixed temperatures (right column) below the critical temperature, the bigger the droplet, the smaller the slope and the greater the viscosity. As the temperature approaches the critical temperature, the viscosities of smaller droplets approach their final values faster than that of those bigger droplets. Above the critical temperature all droplets diffuse at the same rate. In short, at low temperatures, the viscosity depends on droplet size; at high temperatures, the viscosity is independent of droplet size.

The viscosity of a liquid, which is a property of the liquid itself, should be independent of the size of the droplet. Although I will continue to use the word viscosity, I need to reexamine the detailed mechanism that gives rise to diffusion in Monte-Carlo simulations. The droplet does not behave like a solid sphere but its surface fluctuates like an amoeba. The surface fluctuation is a function of temperature. As the previous chapter 4 showed, the higher the temperature, the greater the surface fluctuations.

When the droplet experiences an external force, it does not respond in the same manner as a solid sphere. The droplet is a viscoelastic material. It has both elastic

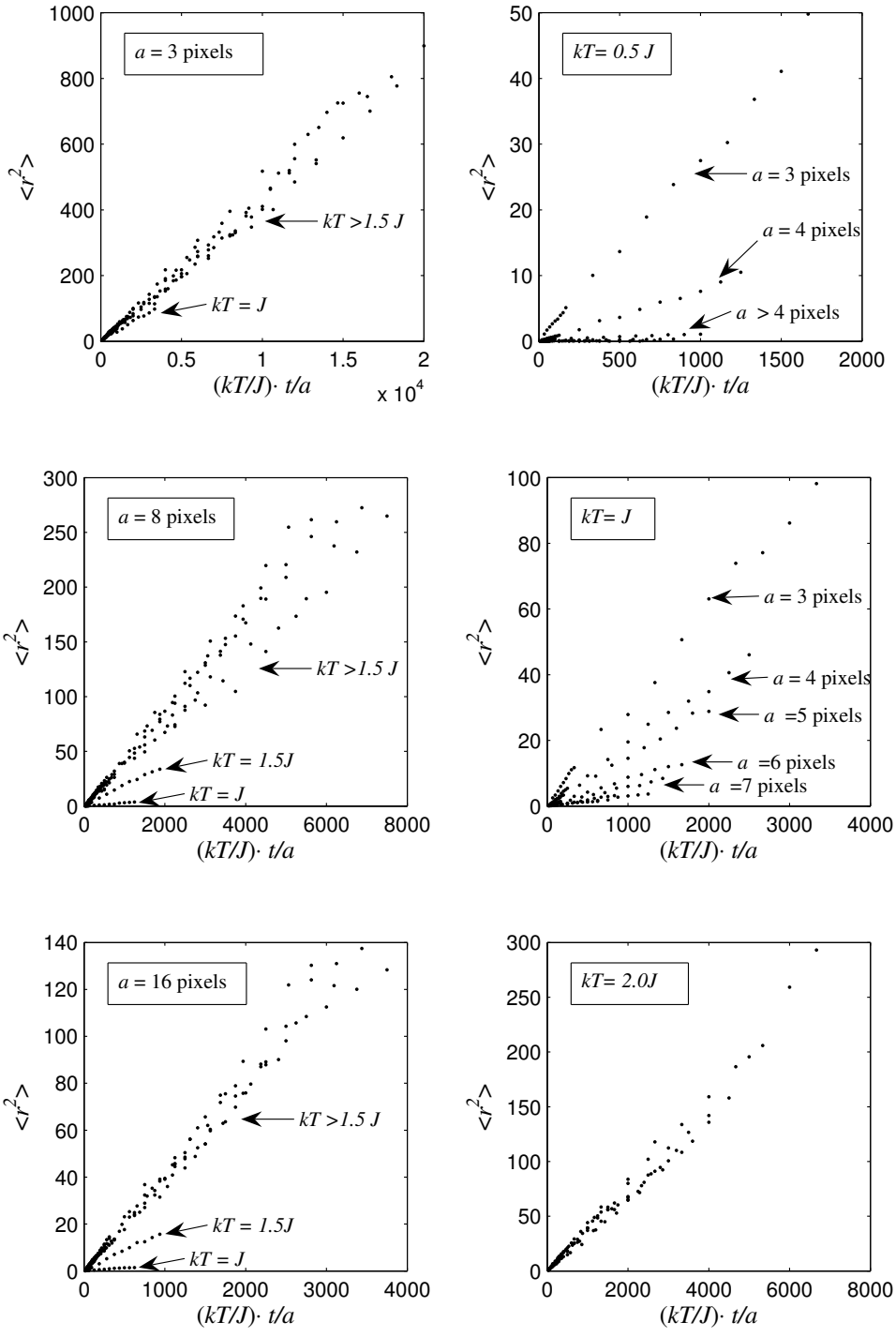
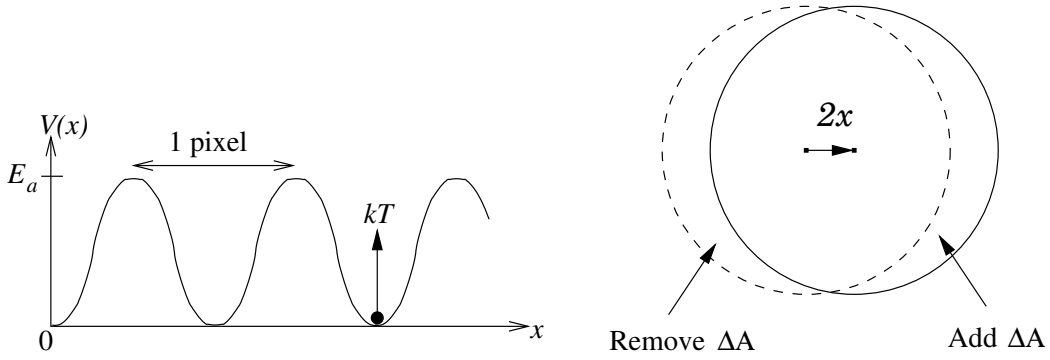


Figure 5.7. Results of $\langle r^2 \rangle$ (pixels^2) *vs.* rescaled time $(kT/a)t$ for Algorithm Three. Left column: superposition of plots for all temperatures for three different droplet sizes ($a = 3, 8, 16$ pixels). Right column: superposition of plots for all radii at three different temperatures ($kT = 0.5J, 1.0J, 2.0J$). I omit error bars for clarity.

properties like a spring and viscous properties like a liquid. A spring responds to external force by deforming while liquids respond to external force by flowing. At low temperatures, when the frequency of collisions is low and each collision is weak, the droplet changes shape rather than moving in the direction of the force. The droplet has time to recover to its initial shape before the next collision occurs. [2]

If two collisions occur before the droplet has time to recover or if the force of a collision is beyond what the surface deformation can absorb, the droplet will not be able to respond by changing shape, but will move in the direction of the force. Since collisions are more frequent at high temperatures, the droplet behaves like a solid sphere. Since bigger droplets have more surface area to deform than smaller droplets, the critical temperature is higher for bigger droplets.



(a) (b)

Figure 5.8. (a) An imaginary potential that exists on a continuum extension of a regular lattice. The black dot representing the center of mass is initially in one of the potential's minima. The black arrow indicates the magnitude of the temperature. (b) Moving a droplet entails moving its surface layer.

In order to cast the viscoelastic properties of the discretized droplet into concrete mathematical form, we may think of the droplet as a particle that wades through

[2]The center of mass of a drop can move in units less than one pixel. The smallest move is one pixel/droplet area. The point is that the lowest possible energy of any droplet configuration with center of mass not aligned with a pixel is larger than the lowest possible energy of a droplet whose center of mass is so aligned and that this energy increases monotonically as the center moves to being half way between two pixels.

a continuum periodic potential with regular maxima and minima. Because the underlying space (the lattice) is discrete, the particle can only occupy certain discrete positions. We imagine that the potential energy is zero at allowed positions of the particle, and positive at forbidden positions (figure 5.8(a)). The "wave length" of the potential is one pixel, reflecting the underlying square lattice. The droplet has lowest energy when its center of mass is at one of the minima.

Any droplet configuration with center of mass not centered on a pixel is less symmetrical than one that is, thus has longer perimeter and larger surface energy. If the droplet could lose energy by shortening its boundary and losing or gaining a pixel of volume, the energy might not increase monotonically as the droplet moves towards a half-pixel offset, however, the symmetrical configuration with aligned center of mass should still have lowest energy.

At zero temperature, the droplet remains at the bottom of one of the potential wells. As the temperature increases, the droplet starts to vibrate inside the well. In order to move to the next potential well, the droplet needs to overcome the potential barrier E_a.

E_a is a function of the droplet radius. The bigger the droplet, the higher the barrier. We may approximate how E_a depends on a. In order to move the center of mass of the droplet by one pixel, only a layer of the surface pixels need to move from one side of the droplet to the other (Figures 5.8(b)). Naturally, the bigger the droplet, the more surface pixels need to move. If the radius of the droplet is a, then the center of mass of the ΔA sliver is roughly a away from the center of the droplet. Removing the ΔA portion, the center of mass of the droplet moves by x pixels:

$$x = \frac{(\Delta A)a}{A - \Delta A}. \tag{5.19}$$

If the ΔA portion moves from one side of the droplet to the other side, then the center of mass of the droplet moves a total distance of $2x$. If we further make the

approximation $A \gg \Delta A$ and $A \sim a^2$, then equation 5.19 reduces to:

$$x \simeq \frac{\Delta A}{a}. \tag{5.20}$$

If $x \simeq 1$ pixel, then $\Delta A \simeq a$. Therefore, in order to move the center of mass by 1 pixel, we have to move a layer of pixels which is proportional to the radius of the droplet. Since I expect E_a to be proportional to ΔA, E_a is also proportional to a.

The probability of the droplet jumping to the next well is proportional to the Boltzmann distribution:

$$P \propto e^{-E_a/kT}. \tag{5.21}$$

The higher the temperature, the higher the probability of crossing the barrier. When the temperature is higher than E_a, the potential no longer inhibits motion of the droplet, which can freely diffuse. This analogy explains why below a certain critical temperature, the droplet diffuses slower than the classical Langevin equation predicts and the diffusivity (viscosity) depends on the size of the droplet. Above the critical temperature, the normal Brownian motion the Langevin equation (equation 5.1) predicts applies. The additional potential modifies the Langevin equation to:

$$m\ddot{x} = -\Omega\dot{x} - \frac{\partial V(x)}{\partial x} + F_r(t), \tag{5.22}$$

where a dot above a symbol denotes a time derivative. The negative sign in front of the potential indicates that the force due to the potential always points towards the bottom of the potential well.

As my simulations found, the motion of the droplet is overdamped ($m/\Omega \ll 1$). So, we may ignore the fast-changing term in the Langevin equation, *i.e.* $\ddot{x}$. Such *adiabatic elimination* [50] simplifies the Langevin equation to:

$$\Omega\dot{x} = F_v(x) + F_r(t), \tag{5.23}$$

where:

$$\begin{aligned} F_v(x) &\equiv -\frac{\partial V(x)}{\partial x}, \\ \langle F_r(t)F_r(t')\rangle &= 2\Omega kT\delta(t-t'). \end{aligned} \tag{5.24}$$

The corresponding Fokker-Planck equation for overdamped motion (the high-friction limit) is the *Smoluchowski* equation [105]:

$$\frac{\partial P(x,t)}{\partial t} = \frac{1}{\Omega}\left[-\frac{\partial}{\partial x}F_v(x) + kT\frac{\partial^2}{\partial x^2}\right]P(x,t), \tag{5.25}$$

where $P(x,t)dx$ is the probability of finding the droplet between x and $x+dx$ at time t. When $V(x) =$ constant $\forall x$, the Fokker-Planck equation reduces to the familiar diffusion equation. The first term on the right-hand side of equation 5.25 is the *drift* term, the second term on the right-hand side is the *diffusion* term.

Brownian motion in a periodic potential and the Smoluchowski equation arise in various fields, *e.g.* solid-state physics, chemical physics, communications theory, laser mode-locking, Josephson tunneling and superionic conductors. Both the stationary and time-dependent probability densities $P(x,t)$ have known solutions. Reference [105] gives a list of related references.

I follow closely the derivation in [105]. Consider a particle moving in a periodic potential with period 2π. Apply a constant force F_c representing a gravitational force, chemical attractants/repellents or other forces independent of position. For $F_c \neq 0$, the net drift velocity should be non-zero. I will later set F_c to zero. The Smoluchowski equation thus becomes:

$$\frac{\partial P}{\partial t} = \frac{1}{\Omega}\frac{\partial}{\partial x}\left[-(F_c + F_v(x)) + kT\frac{\partial}{\partial x}\right]P = -\frac{\partial \mathcal{J}}{\partial x}. \tag{5.26}$$

In the last equality of equation 5.26, $\mathcal{J}$ is the probability current, which satisfies:

$$\frac{\partial P}{\partial x} - \frac{F_c + F_v(x)}{kT}P = -\frac{\Omega\mathcal{J}(x)}{kT}. \tag{5.27}$$

For this choice of the probability current (equation 5.27), the Smoluchowski equation transforms into a continuity equation:

$$\frac{\partial P}{\partial t} + \nabla \cdot (\vec{v}P) = 0. \tag{5.28}$$

The probability current $\mathcal{J}$ relates to the mean drift velocity $\langle \vec{v} \rangle$.

I want the stationary solution for the probability distribution density $P(x,t)$ in the long-time limit, *i.e.* at equilibrium. Then, P is independent of time and $\mathcal{J}$ is a constant and equation 5.27 is just a first-order ordinary differential equation. Multiplying all the terms by an integrating factor $\exp((V(x) - F_c x)/kT)$ and integrating the resultant equation gives the solution:

$$P(x) = e^{-V'(x)/kT} \left(\mathcal{N} - \frac{\Omega \mathcal{J}}{kT} \int_0^x e^{V'(x')/kT} dx' \right), \tag{5.29}$$

where I have defined $V'(x) \equiv V(x) - F_c x$ and $\mathcal{N}$ is the integration constant. Determining the constants $\mathcal{J}$ and $\mathcal{N}$ requires two equations. One equation is the normalization condition of $P(x)$, the other is the requirement that $P(x)$ be bounded for large x. Consider the integral in equation 5.29:

$$\begin{aligned} \int_0^{2\pi n + x} e^{V'(x')/kT} dx' = & \int_0^{2\pi} e^{V'(x')/kT} dx' + \int_{2\pi}^{4\pi} e^{V'(x')/kT} dx' + \ldots \\ & + \int_{2\pi(n-1)}^{2\pi n} e^{V'(x')/kT} dx' + \int_{2\pi n}^{2\pi n + x} e^{V'(x')/kT} dx'. \end{aligned} \tag{5.30}$$

In equation 5.30, n is an integer and x takes values between 0 and 2π. Because of the periodicity in $V(x)$:

$$\begin{aligned} V'(2\pi m + x) &= V(2\pi m + x) - F_c \times (2\pi m + x) \\ &= V(x) - F_c x - 2\pi m F_c \\ &= V'(x) - 2\pi m F_c. \end{aligned} \tag{5.31}$$

Ssubstitution of equation 5.31 into equation 5.30 gives the same factor $I_+(2\pi)$ for

all but the last of the terms in equation 5.30:

$$\begin{aligned}\int_0^{2\pi n+x} e^{V'(x')/kT}dx' &= I_+(2\pi)\sum_{m=0}^{n-1}\phi^m + I_+(x)\phi^n \\ &= I_+(2\pi)\left(\frac{1-\phi^n}{1-\phi}\right) + I_+(x)\phi^n, \end{aligned} \tag{5.32}$$

where:

$$\begin{aligned} I_\pm(x) &\equiv \int_0^x e^{\pm V'(x')/kT}dx', \\ \phi &\equiv e^{-2\pi F_c/kT}. \end{aligned} \tag{5.33}$$

Substituting equations 5.31 and 5.32 back into equation 5.29 and collecting all the terms that depend on n, gives:

$$P(2\pi n+x) = e^{-V'(x)/kT}\left\{\left[\mathcal{N} - \frac{\Omega\mathcal{J}}{kT}\frac{I_+(2\pi)}{(1-\phi)}\right]\phi^{-n} + \frac{\Omega\mathcal{J}}{kT}\left[\frac{I_+(2\pi)}{1-\phi} - I_+(x)\right]\right\}. \tag{5.34}$$

The term that depends on n grows exponentially with n. Therefore, for $P(x)$ to be bounded requires that:

$$\mathcal{N} = \frac{\Omega\mathcal{J}}{kT}\frac{I_+(2\pi)}{(1-\phi)}. \tag{5.35}$$

Putting equation 5.35 back into equation 5.34, gives:

$$P(2\pi n+x) = e^{-V'(x)/kT}\left(\mathcal{N} - \frac{\Omega\mathcal{J}}{kT}I_+(x)\right) = P(x). \tag{5.36}$$

Therefore, $P(x)$ is also periodic with period 2π. Since $P(x)$ is periodic, I normalize $P(x)$ over the interval 0 to 2π:

$$\mathcal{N}I_-(2\pi) - \frac{\Omega\mathcal{J}}{kT}\int_0^{2\pi} e^{-V'(x)/kT}I_+(x)dx = 1. \tag{5.37}$$

Eliminating $\mathcal{N}$ from equations 5.35 and 5.37 yields an expression for $\mathcal{J}$:

$$\mathcal{J} = \frac{kT(1-\phi)}{\Omega\left[I_+(2\pi)I_-(2\pi) - (1-\phi)\int_0^{2\pi} e^{-V'(x)/kT}I_+(x)dx\right]}. \tag{5.38}$$

As I mentioned earlier, the mean drift velocity relates to the probability current:

$$\begin{aligned}\langle v\rangle = \langle \dot{x}\rangle &= \frac{1}{\Omega}\langle F_c + F_v(x) + F_r(x)\rangle \\ &= \frac{1}{\Omega}\langle F_c + F_v(x)\rangle \\ &= \frac{1}{\Omega}\int_0^{2\pi}(F_c + F_v(x))P(x)dx \\ &= \frac{1}{\Omega}\int_0^{2\pi}\left(kT\frac{\partial P}{\partial x} + \Omega\mathcal{J}\right)dx \\ &= 2\pi\mathcal{J}. \end{aligned} \tag{5.39}$$

If $F_c = 0$, $\phi = 1$, then $\mathcal{J} = \langle v\rangle = 0$, as we would expect. Although the mean drift velocity is zero in the absence of a constant applied force, the *mobility* μ_m which is the ratio between the drift velocity and the applied field is finite in the limit $F_c \to 0$:

$$\begin{aligned}\lim_{F_c\to 0}\mu_m &\equiv \lim_{F_c\to 0}\frac{\langle v\rangle}{F_c/m} \\ &= \lim_{F_c\to 0}\frac{m2\pi\mathcal{J}}{F_c} \\ &= \frac{4\pi^2 m}{\Omega I_+(2\pi)I_-(2\pi)}, \end{aligned} \tag{5.40}$$

with $V'(x) = V(x)$. From Einstein's relation, the diffusion constant relates to the mobility by [104]:[3]

$$D = 2\frac{kT}{m}\mu_m, \tag{5.41}$$

in accordance with equation 5.6's definition of the diffusion constant. Therefore, without a constant external force, the diffusion constant in a periodic potential is:

$$D_{F_c=0} = \frac{8\pi^2 kT}{\Omega I_+(2\pi)I_-(2\pi)}. \tag{5.42}$$

If the potential is zero, $I_\pm(2\pi) = 2\pi$, and equation 5.42 reduces to equation 5.6.

In order to proceed further, I need to choose a form for the periodic potential. I expect that the qualitative behavior of $D_{F_c=0}$ is the same regardless of the exact

[3]The particle mass m in the present discussion replaces the electronic charge e in Einstein's relation.

form of the potential. Since I do not know exactly how $V(x)$ depends on the droplet size, I apply equation 5.42 to three typical potentials:

$$\begin{aligned}
\text{square-wave:} \qquad V(x) &= \begin{cases} E_a, & 2n\pi \leqslant x \leqslant (2n+1)\pi \\ 0, & (2n+1)\pi \leqslant x \leqslant (2n+2)\pi \end{cases}, \\
\text{saw-tooth:} \qquad V(x) &= \begin{cases} E_a[-x/\pi + (2n+1)], & 2n\pi \leqslant x \leqslant (2n+1)\pi \\ E_a[x/\pi - (2n+1)], & (2n+1)\pi \leqslant x \leqslant (2n+2)\pi \end{cases}, \\
\text{cosine:} \qquad V(x) &= \frac{E_a}{2}(\cos(x)+1),
\end{aligned} \tag{5.43}$$

with $n = 0, 1, 2, \ldots$. In each case, $V(x)$ ranges from 0 to E_a.

Substituting the various forms of $V(x)$ into equation 5.42 and integrating is simple. For the piece-wise-linear potentials, we can solve the diffusion constant analytically. For the cosine potential, using the identities:

$$\int_0^{2\pi} e^{z\cos x} dx = 2\int_0^{\pi} e^{z\cos x} dx, \tag{5.44}$$

and:

$$\int_0^{2\pi} e^{-z\cos x} dx = \int_0^{2\pi} e^{z\cos x} dx, \tag{5.45}$$

the diffusion constant is a function of the zeroth-order Modified Bessel Function of the first kind:

$$J_o(z) = \frac{1}{\pi}\int_0^{\pi} e^{z\cos x} dx. \tag{5.46}$$

The solutions for each case are:

$$\begin{aligned}
\text{square-wave:} \qquad D_{F_c=0} &= \frac{2kT}{\Omega \cosh^2(R)}, \\
\text{saw-tooth:} \qquad D_{F_c=0} &= \frac{2kTR^2}{\Omega \sinh^2(R)}, \\
\text{cosine:} \qquad D_{F_c=0} &= \frac{2kT}{\Omega [J_o(R)]^2},
\end{aligned} \tag{5.47}$$

where the ratio $R \equiv E_a/(2kT)$. In comparison, the diffusion constant for the Langevin equation in the long-time regime is $D = 2kT/\Omega$ (equation 5.6). In each case, $D_{F_c=0}$ reduces to $2kT/\Omega$ for $kT \gg E_a$ *i.e.* $R \to 0$.

Figure 5.9 plots $D_{F_c=0}$ (in units of $2kT/\Omega$) as a function of the temperature (in units of E_a). The diffusivities of all three potentials follow the same trend. The diffusion constant is zero at zero temperature and reaches a maximal value of $2kT/\Omega$ as $kT \gg E_a$, with no transition at $kT = E_a$. The diffusivity (inverse viscosity) increases smoothly with the temperature and saturates.

Since I expect E_a to be proportional to the droplet radius, bigger droplets reach maximal mobility at higher temperatures and diffuse slower than the Langevin equation predicts at low temperatures.

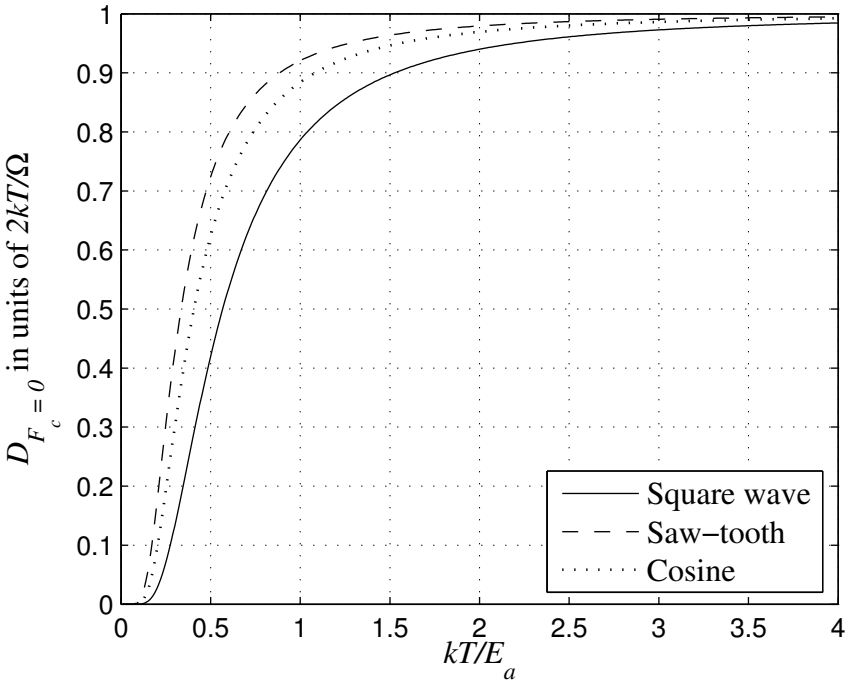


Figure 5.9. The diffusion constant $D_{F_c=0}$ as a function of kT/E_a for three different potentials.

5.4 Frequency of Collisions

A concept which relates to Brownian motion is the *mean free time*, the average time between two consecutive collisions. For a random process like radioactive decay or molecular collision, where every event is uncorrelated to every other event, the

number of events n that occur within a time interval t follows a Poisson probability distribution:

$$P_n(t) = \frac{(\lambda t)^n e^{-\lambda t}}{n!}, \tag{5.48}$$

where λ is the average rate or frequency of events (the number of events per unit time). The probability that a molecule passes a time t without collisions is:

$$P_0(t) = e^{-\lambda t}. \tag{5.49}$$

Therefore the average time between two consecutive collisions (the mean free time) is:

$$t_{mf} = \frac{\int_0^\infty t P_0(t) dt}{\int_0^\infty P_0(t) dt} = \frac{1}{\lambda}. \tag{5.50}$$

Equation 5.50 agrees with my identification of λ with the rate of collisions. The *mean free path*, or the average distance a molecule travels between two consecutive collisions is:

$$x_{mf} = \upsilon t_{mf} = \upsilon/\lambda, \tag{5.51}$$

where υ is the average speed of a molecule.[4] For a concentration ρ (number of molecules per unit volume) of hard sphere molecules of radius a, the mean free path is:

$$x_{mf} = \frac{1}{\rho \sigma_c}, \tag{5.52}$$

where σ_c is the collision cross-section, which is πa^2 for hard spheres or $2a$ for hard circles.[5] Combining equations 5.51 and 5.52 and using the equipartition theorem:

$$\lambda = \rho \sigma_c \upsilon \sim \rho a \sqrt{kT}. \tag{5.53}$$

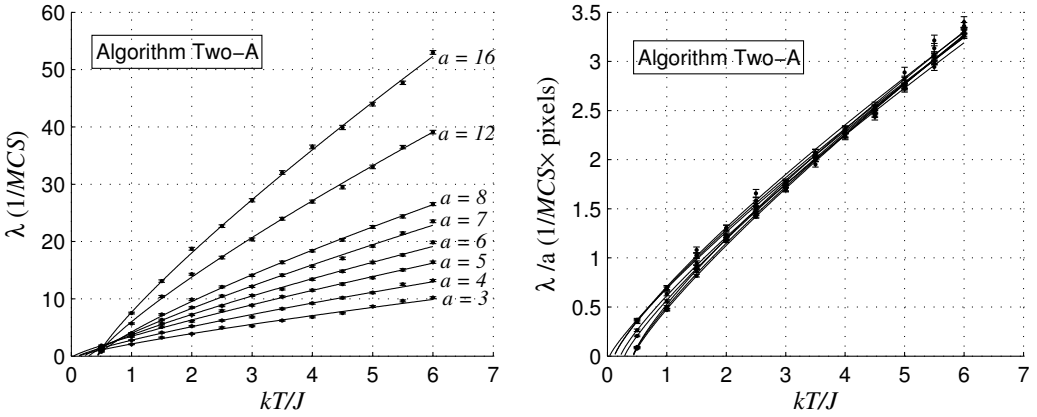
[4]More accurately, υ is the average relative speed between molecules and the Brownian particle. Since the molecules are usually much lighter than the Brownian particle, the speed of the Brownian particle is negligible compared to the molecular speeds. Therefore, to first approximation, the relative speed equals the average molecular speed.

[5]Because the Brownian particle is usually much bigger than the molecules, the size of the Brownian particle dominates the collision cross-section. Therefore, a is the radius of the Brownian particle.

I counted the number of spin flips (analogous to collisions) per Monte-Carlo step as a function of droplet radius and temperature and fit the counts to the empirical equation:

$$\lambda = ca(kT - kT_a)^d, \tag{5.54}$$

where c is a proportionality constant. kT_a measures the minimum energy needed to add or subtract one pixel from a droplet. Figure 5.10 and table 5.4 give the results of the fits.



(a) (b)

Figure 5.10. (a) Number of spin flips per Monte-Carlo step, λ, *vs.* temperature for various droplet radii for the Ising model using Algorithm Two-A. (b) Normalized number of spin flips per Monte-Carlo step, λ/a, *vs.* kT. Solid lines are best fits to equation 5.54. Plots for Algorithms Two-B and Three are similar.

TABLE 5.4

RESULTS OF FITTING NUMBER OF SPIN FLIPS PER MONTE-CARLO STEP TO EQUATION 5.54.

Algorithm	$c(1/(MCS \cdot \text{pixel} \cdot J))$	d
Two-A	0.77±0.01	0.82±0.01
Two-B	0.61±0.02	0.87±0.02
Three	0.51±0.01	0.88±0.02

For all three algorithms, kT_a ($\lesssim 0.5J$) is a slowly increasing function of a, sug-

gesting that adding or subtracting a pixel takes less energy on a curved interface than a flat interface. The bigger the droplet, the flatter the surface and the harder adding or subtracting a pixel from the droplet surface becomes. The exponent d is bigger than 0.5 because the droplet surface become rougher as the temperature increases, thus creating more surface area on which spin flips can take place.

5.5 Energy-Area Distribution of Droplets

Figures 5.11, 5.12 and 5.13 show the energy-area distributions of droplets at different temperatures. The peaks and averages of the distributions always center around an $A < A_T$, with two or even three peaks at low temperatures. As the temperature increases, the distribution broadens. In general, the bigger the area, the higher the droplet energy. Multiple peaks in the distribution always occur at different droplet areas. Each droplet size has a unique, minimum-energy droplet shape.

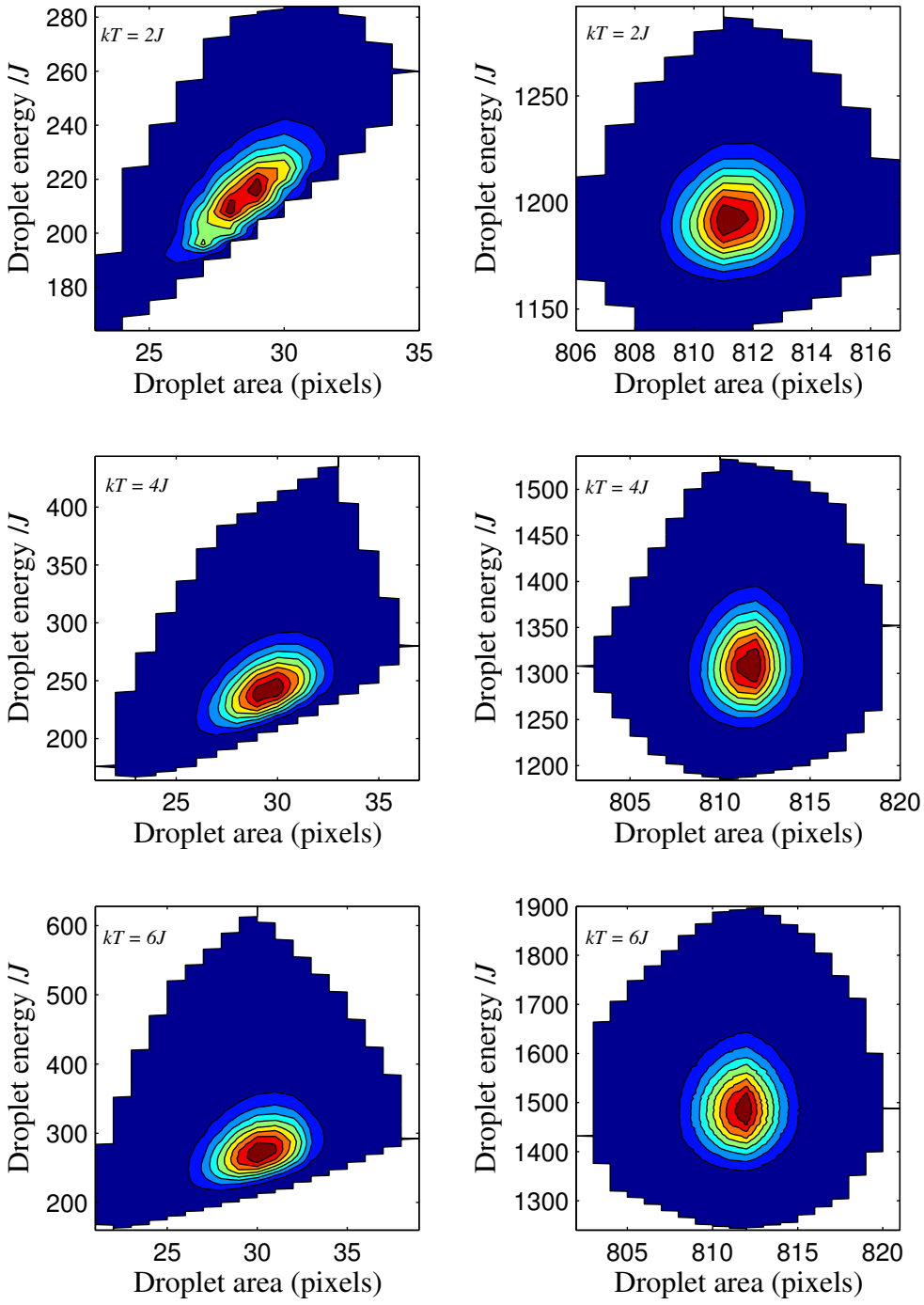
At zero temperature, a droplet's energy is proportional to its radius or the square root of its area (figure 5.3(b)). As temperature increases, we expect the energies of droplets at fixed areas should increase. I fit the droplet's average energy as a function of its average area with the equation:

$$E_J = p' A^{q'}. \tag{5.55}$$

At $kT = 0$, $A = \pi a^2$, I expect $q' \simeq 0.5$ and $p'\sqrt{\pi} \simeq p$ (from table 5.2 with nn = 4). Either p' or q' or both may be a function of temperature.

Figure 5.14 shows the average droplet energy *vs.* the average droplet area for different fixed temperatures using Algorithm Two-A. Table 5.5 shows the results of fitting the average droplet energy to the average droplet area using equation 5.55 for all three algorithms.

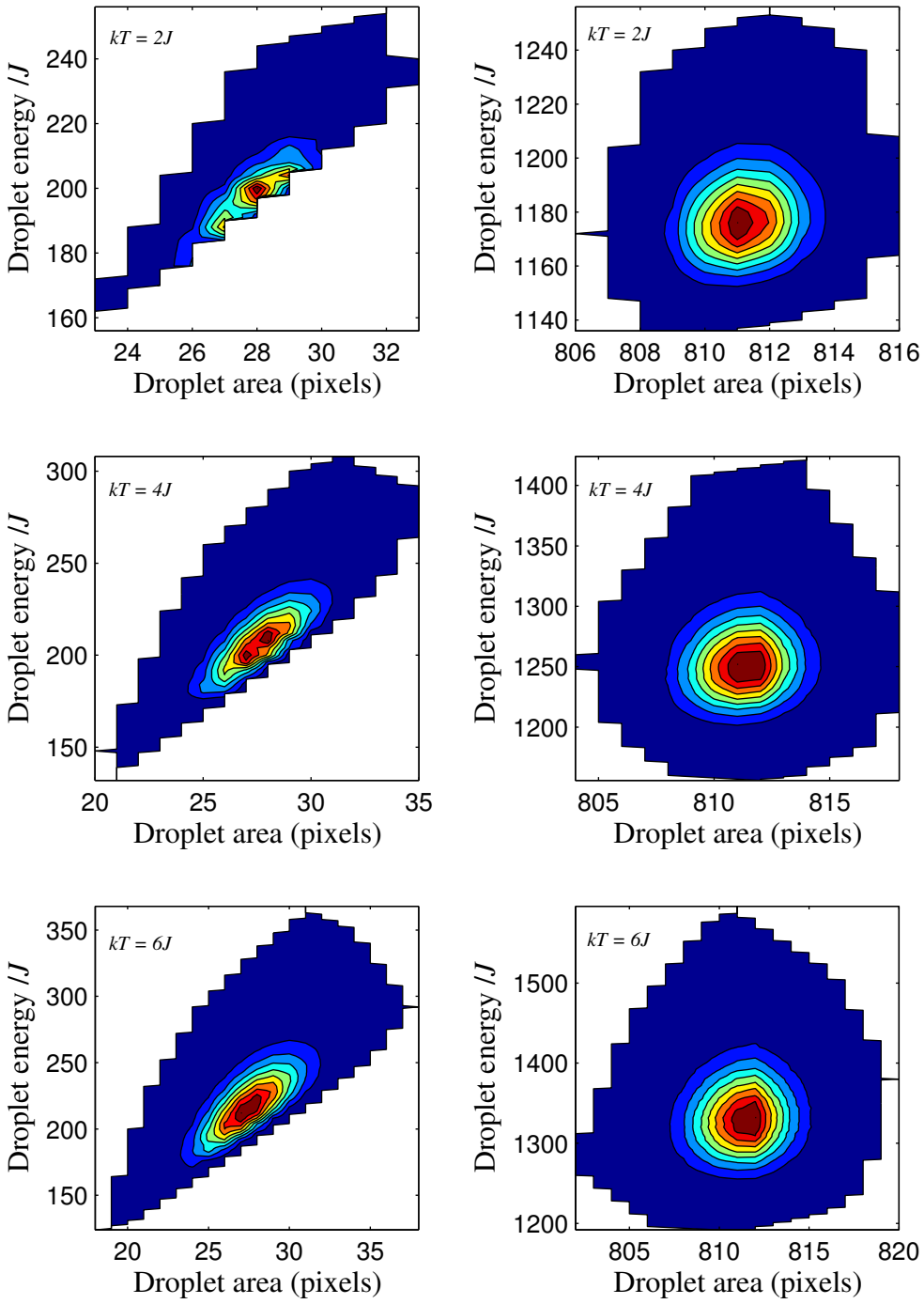
With E_J from equation 5.55 and E_A from equation 5.11, the total energy $E = E_J + E_A$ is an asymmetrical function with minimum close to A_T (figure 5.15(a)).



(a) a = 3, $A_T = 32$.

(b) a = 16, $A_T = 812$.

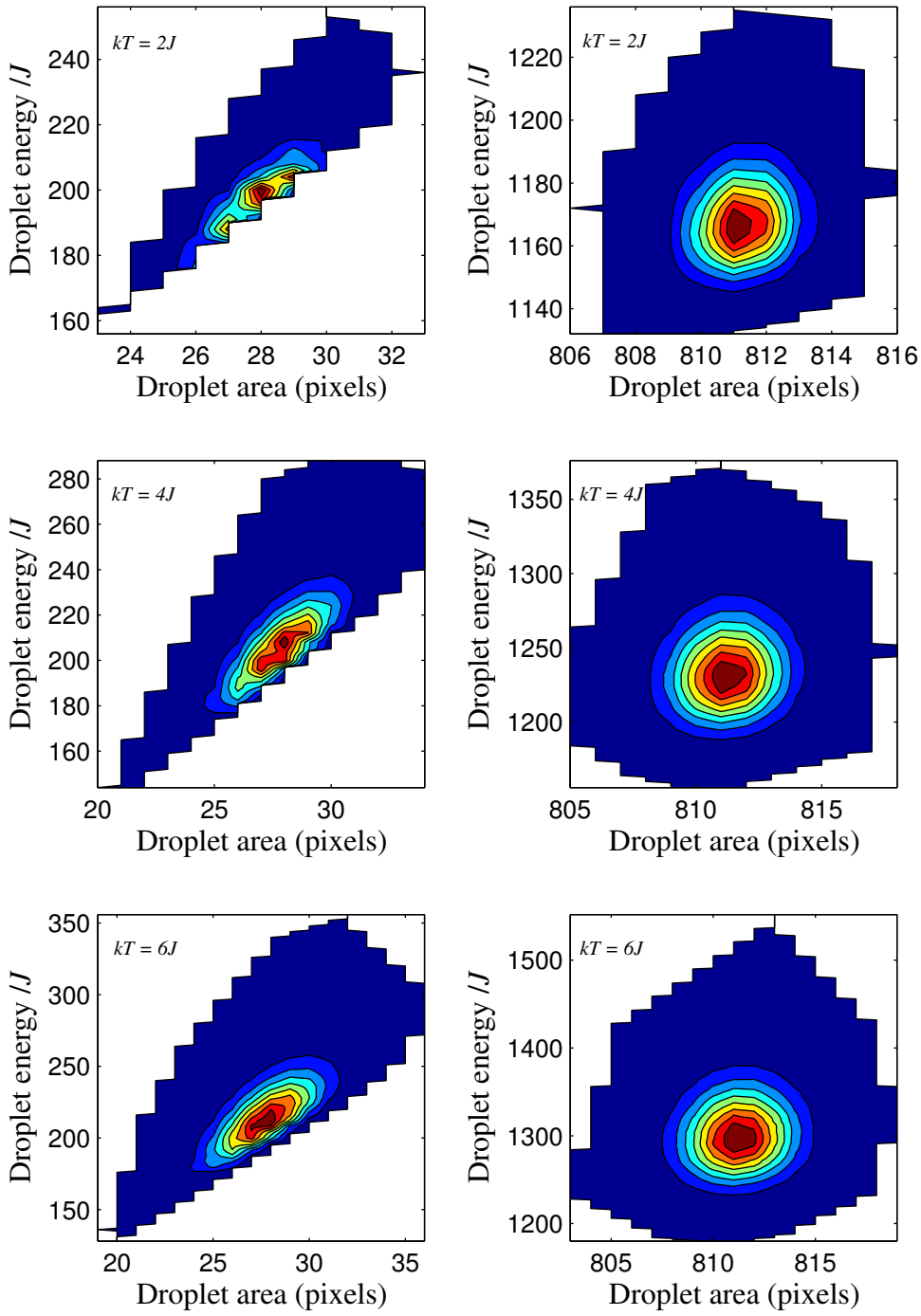
Figure 5.11. Energy-area distribution of droplet simulations for the Ising model and with Algorithm Two-A. (a) Left column: $a = 3$ pixels. (b) Right column: $a = 16$ pixels. Probability is in descending order from red to blue.



(a) a = 3, $A_T = 32$.

(b) a = 16, $A_T = 812$.

Figure 5.12. Energy-area distribution of droplet simulations for the Ising model and with Algorithm Two-B. (a) Left column: $a = 3$ pixels. (b) Right column: $a = 16$ pixels. Probability is in descending order from red to blue.



(a) a = 3, $A_T = 32$.

(b) a = 16, $A_T = 812$.

Figure 5.13. Energy-area distribution of droplet simulations for the Ising model and with Algorithm Three. (a) Left column: $a = 3$ pixels. (b) Right column: $a = 16$ pixels. Probability is in descending order from red to blue.

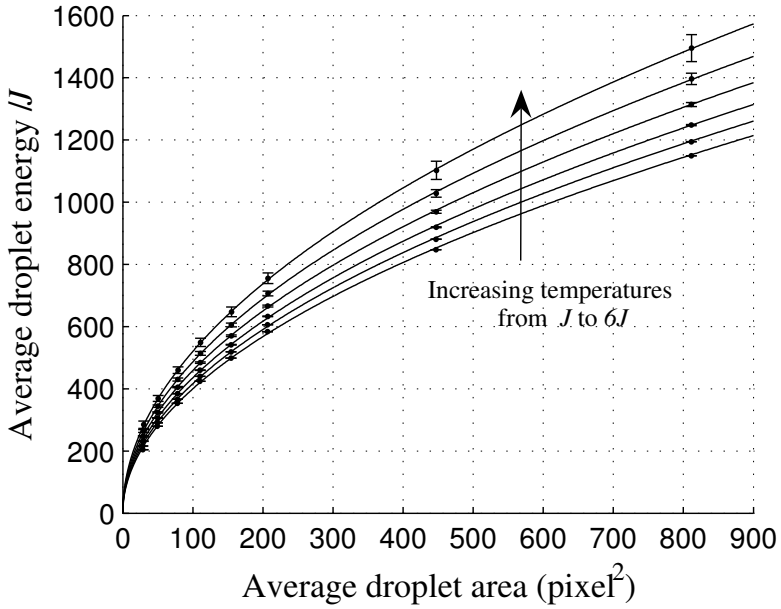


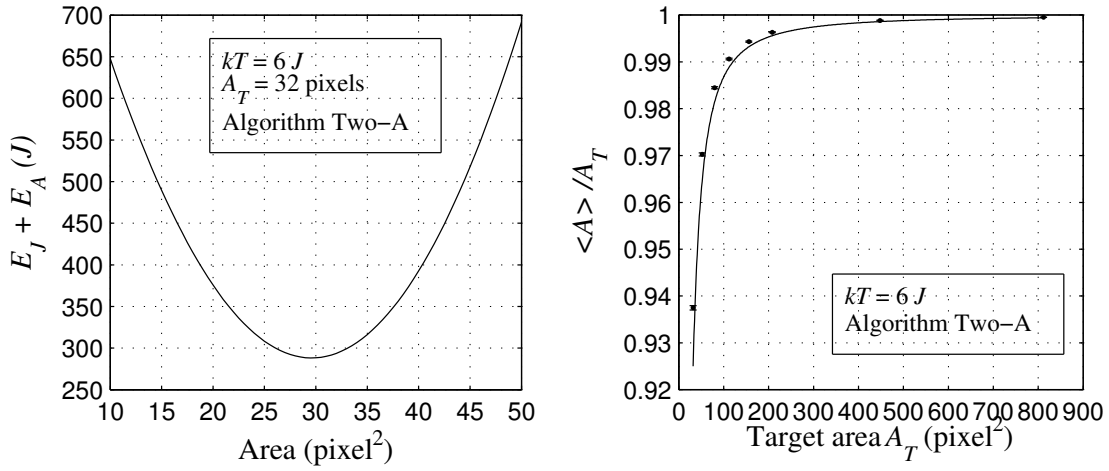
Figure 5.14. Average droplet energy *vs.* average droplet area at different temperatures. Solid lines are best fits to equation 5.55. Simulations used Algorithm Two-A. Results from Algorithms Two-B and Three are similar.

TABLE 5.5

RESULTS OF FITTING AVERAGE DROPLET ENERGY AS A FUNCTION OF AVERAGE DROPLET AREA USING EQUATION 5.55.

kT	$p'(\pm 0.5J)$			$q' \pm 0.002$		
(J)	Two-A	Two-B	Three	Two-A	Two-B	Three
0.5	38.7	37.7	37.7	0.504	0.508	0.508
1.0	39.4	37.7	37.8	0.504	0.510	0.509
1.5	40.1	37.9	37.9	0.504	0.511	0.510
2.0	41.1	38.3	38.2	0.503	0.512	0.511
2.5	42.1	38.8	38.6	0.503	0.512	0.511
3.0	43.2	39.3	39.0	0.502	0.513	0.512
3.5	44.3	39.8	39.4	0.502	0.513	0.512
4.0	45.5	40.4	39.8	0.502	0.513	0.513
4.5	46.7	40.8	40.3	0.502	0.513	0.513
5.0	48.1	41.5	40.7	0.503	0.514	0.514
5.5	49.6	42.0	41.2	0.503	0.514	0.514
6.0	51.2	42.5	41.6	0.504	0.515	0.515

Because E_J is a monotonically increasing function of A, the minimum of E always occurs at $A < A_T$. Therefore, the average area of droplets is always smaller than A_T. Physically, because the surface tension tries to reduce the surface energy and hence area, the resultant area is always slightly less than the target area. The ratio of A/A_T will increase with area constraint, J_A. Since the slope of E_J decreases with increasing A, the asymmetry in E decreases with increasing A; hence the average area A is closer to A_T for bigger droplets (Figure 5.15(b)). The fitting equation 5.55 is a good approximation for bigger droplets when the underlying discreteness of the lattice is small compared to the size of the droplets.



(a) (b)

Figure 5.15. (a) Total droplet energy $E = E_J + E_A$ *vs.* area A. The target area is 32 pixel^2. The minimum of E occurs at $A < A_T$. (b) The ratio of average area to target area as a function of the target area. The solid line is the locus of minima of E for different target areas A_T.

5.6 Discussion and Conclusion

I have implemented three algorithms to simulate Brownian motion in the Ising model. I found similarities and dissimilarities to classical Brownian motion. The droplet behaves like a viscoelastic material with a freely fluctuating surface and

slightly fluctuating area. The droplet has an associated bulk modulus. At low temperatures, the droplet diffuses much slower than a Brownian particle. At high temperatures, elasticity gives way to viscosity and the droplet diffuses as the theory predicts. The "transition temperature" from elasticity to viscosity depends on the size of the droplet. The bigger the droplet, the higher the "transition temperature." Future simulations need to ascertain the exact "transition temperature" and its dependence on droplet radius.

Although the area constraint ensures that droplet stay close to their target areas, the surface energy alone is insufficient to hold droplets together when $kT \gg J$. As a result, droplets become more diffuse and their effective surface increases with temperature.

The diffusion of Brownian droplets depends only quantitatively and not qualitatively on the choice of algorithm dynamics. Enforcing detailed balance makes droplets diffuse faster. Considering four spin-flip cases (Algorithm Two-B) instead of two (Algorithm Three) increases the diffusion rate. Therefore, $D_{2A} > D_{2B} > D_3$. The surface energies of a droplet are almost equal in Algorithms Two-B and Three. The surface energy in Algorithm Two-A (the only one that satisfies detailed balance) is much higher. Algorithms Two-B and Three only differ in the number of spin-flip cases in their corresponding acceptance probabilities (equations 5.18 and 2.24); both violate detailed balance. Their diffusion rates differ but their droplet surface energies are similar. Thus, detailed balance affects both the droplet energy and diffusion rate, while considering the spin-flip cases in finer detail (four cases in Algorithm Two-B *vs.* two cases in Algorithm Three) only affects the diffusion rate.

Time scale in Monte-Carlo simulations is an important and interesting question. A problem that researchers often face when doing Monte-Carlo simulations of the type I have discussed is the question of simultaneity. In Monte-Carlo simulations,

spin flips occur sequentially. Therefore, at fine time-scales (in the order of a few spin flip attempts), events do not happen concurrently. At coarse time-scales, however, events do appear to happen concurrently. In real life, causally unconnected can happen simultaneously. Therefore, simulations are most realistic at coarse time-scales. My Brownian motion simulations indicate that the motion is overdamped with a time constant is smaller than one Monte-Carlo step. Simulations that last thousands of Monte-Carlo steps, they are safely in the coarse-time-scale regime.

Possible extensions of this work include:

1. Simulating Brownian motion in three dimensions to compare the ratio of diffusion constants in three and two dimensions to the theoretical prediction of 3/2.
2. Simulating Brownian motion with different interaction ranges to determine if the range affects the diffusion rate, *i.e.* the viscosity.
3. More detailed simulations at low temperatures to determine the exact temperature where the viscosity ceases to depend on the temperature.
4. Determine the relation between the magnitude of the imaginary potential, E_a, and the droplet radius a.
5. Changing the value of the area constraint J_A to determine its effect on the viscosity and the point where the area dependence of viscosity ceases.
6. Adding a perimeter constraint to prevent the droplet surface from breaking apart to determine the effects of breakup on viscosity and the frequency of spin flips.
7. Adding a constant force (*e.g.* a gravitational force) and measuring the drift velocity. The implementation of gravity is similar to the area constraint. We add another term (the gravitational potential energy) to the Ising model Hamiltonian:

$$H_{\text{droplet}} = \sum_{i \in \text{droplet}} \sum_{j=1}^{z} J_{ij}(1 - \delta_{\sigma_i,\sigma_j}) + J_A(A(t) - A_T)^2 + mg(h_{cm}(t)), \quad (5.56)$$

where g is the "gravitational acceleration" and $h_{cm}(t)$ is the height of the droplet's center-of-mass at time t. The resultant simulation is equivalent to Stokes flow. Because the droplet falls at a constant velocity, the net force acting on the droplet is zero, therefore:

$$mg = 6\pi\eta a\langle v\rangle, \quad (5.57)$$

where $\langle v\rangle$ is the mean drift velocity of the droplet. Since the simulator sets the value of mg and $\langle v\rangle$ is a measured value, such simulations provide another method to determine the viscosity η.

5.7 Lessons for Future Research

The most important consequence of this work is likely to be in biological simulations. I have presented a systematic way of analyzing three different Monte-Carlo algorithms. The three algorithms give similar results for dynamical simulations. They produce different rate of diffusion and hence different effective time-scales.

In biological simulations, the driving force is often not gravitational, but chemical attraction or repulsion, *chemotaxis*. Usually, the driving force is proportional to the local gradient of chemical concentration. However, the speed of the simulated droplet/cell may not be strictly proportional to the strength of the external force, especially when the speed approaches the maximum speed achievable in Monte-Carlo simulations (the "speed of sound"), which equals one pixels per Monte-Carlo step.[6] When the speed of the droplet is close to the maximum speed, increasing the external force will not proportionally increase the speed. Such simulations may resemble reality qualitatively, but not quantitatively. This thesis has explored and presented a systematic way to determine the regimes in which particular Monte-Carlo models apply.

[6]On average, each pixel attempts to flip once every Monte-Carlo step, therefore, the maximum speed of a boundary is one pixel per Monte-Carlo step.

APPENDIX A

Euler's Theorem

Let $f(x_1, \ldots, x_N)$ be a homogeneous function of degree n:

$$f(\lambda x_1, \ldots, \lambda x_N) = \lambda^n f(x_1, \ldots, x_N). \tag{A.1}$$

Differentiate both sides of equation A.1 with respect to λ:

$$\begin{aligned} \frac{d}{d\lambda} f(\lambda \vec{x}) &= \frac{d}{d\lambda} \lambda^n f(\vec{x}) \\ \sum_i^N \frac{\partial \lambda x_i}{\partial \lambda} \frac{\partial f(\lambda \vec{x})}{\partial \lambda x_i} &= n\lambda^{n-1} f(\vec{x}) \\ \sum_i^N x_i \frac{\partial f(\lambda \vec{x})}{\partial \lambda x_i} &= n\lambda^{n-1} f(\vec{x}). \end{aligned} \tag{A.2}$$

Setting $\lambda = 1$, we obtain Euler's theorem:

$$\vec{x} \cdot \nabla f(\vec{x}) = n f(\vec{x}). \tag{A.3}$$

APPENDIX B

Generating the Surface Tension Polar Plot of the SOS model

The following Mathematica progam generates the surface tension polar plot (figure 3.13(b)) of the SOS model for temperatures $kT = 0.1J$, $0.5J$, $1.5J$, $2.5J$, $3.5J$, $4.5J$. The default coupling constants are $J_1 = J_2 = J = 1$. Equation 3.54 gives the partition function, Ξ. Depending on the computing power, it takes about one minute to finish calculations. For nearest-neighbor interactions, set $J_2 = 0$. For nearest-neighbor interactions, the maximum temperature must be less than $2.269J$.

```
Clear[J1, J2, x, τ, θ];
J1 = 1;
J2 = 1;
g1[x_] := Exp[-J1 / x];
g2[x_] := Exp[-J2 / x];
g12[x_] := g1[x]^2;
g14[x_] := g1[x]^4;
g24[x_] := g2[x]^4;
g28[x_] := g2[x]^8;
η[x_, τ_] := Exp[τ / x];
gp[x_, τ_] := 2 Cosh[τ / x];
Ξ[x_, τ_] := (g12[x] g24[x] (1 + g12[x] (1 - g24[x]) gp[x, τ] + g14[x] g24[x] (g24[x] - 2))) /
             (1 - g12[x] g24[x] gp[x, τ] + g14[x] g28[x]);
F[x_, τ_, θ_] := τ * Tan[θ] - x * Log[Ξ[x, τ]];
α[x_, τ_, θ_] := F[x, τ, θ] Cos[θ];

(* note that there are no colons in the following definitions *)
h[x_, τ_] = x * D[Log[Ξ[x, τ]], τ];
U[x_, τ_] = x^2 * D[Log[Ξ, τ], x];
```

```
<< Graphics`Graphics`
nmax = 6;
myplot = Table[{}, {n, 1, nmax}];
kT = {0.1, 0.5, 1.5, 2.5, 3.5, 4.5};
dash = {{0.0, 0.0}, {0.05, 0.05}, {0.025, 0.025},
   {0.01, 0.01}, {0.01, 0.01, 0.03, 0.03}, {0.005, 0.03}, {0.05, 0.01}};

For[n = 1, n ≤ nmax, n++,
 Clear[θ, τ, tau, tau2, αlist45, αlist90, αlist180, αlist360];
 θ = Table[i, {i, π / 200, 49 π / 200, π / 200}];
 tau = Map[NSolve[Simplify[h[kT[[n]], τ]] == Tan[#], τ] &, θ];
 tau2 = τ /. tau;
 tau = Select[Flatten[tau2], TrueQ[Im[#] == 0 && ! (# < 0)] &];
 tau = Join[{0}, tau];
 θ = Join[{0}, θ];
 αlist45 = MapThread[α[kT[[n]], #1, #2] &, {tau, θ}];
 αlist90 = Join[αlist45, Reverse[αlist45]];
 αlist180 = Join[αlist90, Reverse[αlist90]];
 αlist360 = Join[αlist180, Reverse[αlist180]];
 myplot[[n]] =
  PolarListPlot[αlist360, PlotJoined → True, PlotStyle → Dashing[dash[[n]]], DisplayFunction → Identity];
]
allplots = Take[myplot, nmax];
Show[allplots, DisplayFunction → $DisplayFunction];
```

APPENDIX C

Using the Matlab nlinfit Function for Curve-fitting

The fitting equation is 4.11. The inputs of the *sosfit* function are:

1. An initial set of values for the fitting parameters $(\varepsilon_a, \varepsilon_b, \varepsilon_c, \delta, \mathcal{A})$.
2. A column of values of the independent variable, x.
3. A column of values of the dependent variable, y.
4. An initial guess of the value of kT_c.

The outputs of the sosfit function are:

1. A plot of x *vs.* y, best-fit (solid blue line) and 95% confidence interval of the best fit (dashed red line).
2. Best-fit values of the fitting parameters, kT_c and kT_e, including their 95% confidence intervals.

sosfit.m

```
% Nonlinear curve-fitting using matlab nlinfit() function.
% y = efit(p, x) is the fitting function (equaation 4.11).
% To use this program, save this file, sosfit.m in a directory.
% Run matlab 7 in the same directory as sosfit.m.
% In the command window, type sosfit(p, x, y, guesskTc) where
% p a column containing initial values of the fitting parameters:
% p(1) = ea, p(2) = eb, p(3) = ec, p(4) = delta, p(5) = A.
% x is an n x 1 column of temperatures.
% y is an n x 1 column of energies at the corresponding values of x.
% guesskTc is an initial guess of kTc.

function output = sosfit(p, x, y, guesskTc)

  % define anonymous functions of variable x and parameters p:
   ga   = @(p, x) exp(-p(1)./x);
   gb   = @(p, x) p(4)*exp(-p(2)./x);
   gc   = @(p, x) exp(-p(3)./x);
   zeta = @(p, x) ga(p, x).*(gc(p, x) + 2*gb(p, x)./(1 - gb(p, x)));
   fun1 = @(p, x) 2*p(2)*ga(p, x).*gb(p, x)./(zeta(p, x).*(1 - gb(p, x)).^2);
   fun2 = @(p, x) p(3)*ga(p, x).*gc(p, x)./zeta(p, x);
   efit = @(p, x) p(5)*(p(1) +  fun1(p, x) + fun2(p, x));

  % using nlinfit() function to curve-fit efit() to (x,y) data.
   [pfinal, resid, J] = nlinfit(x, y, efit, p);

  % compute 95% confidence interval for the fitting parameters.
   pci = nlparci(pfinal, resid, J);

  % compute predicted y and predicted 95% confidence interval of y.
   [yfinal, ysigma] = nlpredci(efit, x, pfinal, resid, J);

  % plot experimental results with best fit + error.
   plot(x,y,'x', x,yfinal,'b', x,yfinal-ysigma,'--r', x,yfinal+ysigma,'--r');

  % find the temperature at which the Helmhotz free energy is zero (kTc).
   pf = pfinal;
   Helm = @(x) -pf(5)*x.*log(zeta(pf, x));
   kTc = fzero(Helm, guesskTc);
   pf = pci(:,1);
   Helm = @(x) -pf(5)*x.*log(zeta(pf, x));
   kTcci = fzero(Helm, guesskTc);
   sigmakTc = abs(kTc - kTcci);

  % find the temperature at which efit diverges (kTe).
   kTe = NaN;
   sigmakTe = NaN;
   if pfinal(4) > 1.0
     kTe = pfinal(2)/log(pfinal(4));
   end
   if pci(4,2) > 1.0
     kTeci = pci(2,2)/log(pci(4,2));
     sigmakTe = abs(kTe - kTeci);
   end

   % output results:
   % ea    +/- sigma(ea)
   % eb    +/- sigma(eb)
   % ec    +/- sigma(ec)
   % delta +/- sigma(delta)
   % A     +/- sigma(A)
   % kTc   +/- sigma(kTc)
   % kTe   +/- sigma(kTe)
   pdelta = (pci(:,2) - pci(:,1))/2;
   output = [pfinal' pdelta; kTc sigmakTc; kTe sigmakTe];
```

APPENDIX D

Digitising a Droplet

The following Mathematica program produces digitized two-dimensional circular droplets and calculates their energies. To use the program, first press the [shift] and [return] keys together to activate the Mathematica kernel. To digitize a circle of radius $r = 8$, for example, type:

MyCircle[8] followed by [shift]+[return].

to create a vector $\boldsymbol{M}$ whose elements are the number of pixels in each column of the digitised droplet. Because of the symmetry along the x and y-axes, **M** only stores the number of pixels for the first quadrant of the droplet. To see the vector **M**, type:

M followed by [shift]+[return].

For example, for an $r = 8$ droplet, **M** = {8, 8, 8, 7, 7, 6, 5, 3} (refer to figure 5.2). To obtain the area of the droplet, type:

Area followed by [shift]+[return].

To obtain the number of boundary pixels up to nth-nearest neighbors, type:

Perimeter[n] followed by [shift]+[return].

To obtain the energy of the droplet up to nth-nearest neighbors, type:

PerimeterEnergy[n,n] followed by [shift]+[return].

The values of n are 1, 2, 3 or 4.

```
FF[r_, x_] := If[r^2 - x^2 ≤ 0, 0, √(r^2 - x^2)];
GG[r_, x_, y_] := If[FF[r, x] ≥ y, y, FF[r, x]];
HH[r_, x1_, x2_, y1_, y2_] := NIntegrate[GG[r, x, y2], {x, x1, x2}] - NIntegrate[GG[r, x, y1], {x, x1, x2}];
II[r_, x_, y_] := HH[r, x - 1, x, y - 1, y];
JJ[r_, x_, y_] := If[II[r, x, y] ≥ 0.5, y, y - 1];
KK[r_, x_, top_] := FixedPoint[JJ[r, x, #1] &, top];

LL[r_] := Module[{top = r},
                 Do[top = M[[x]] = KK[r, x, top];
                    If[x ≥ top, Break[]],
                   {x, 1, r}]
                 ];

MyCircle[r_] := Module[{hh = 0, top, A},
                 M = Table[0, {r}];
                 LL[r];
                 A = Reverse[Union[M]];
                 Do[top = A[[y]];
                    If[M[[top]] =!= 0, Break[]];
                    hh += Count[M, A[[y]]];
                    M[[top]] = hh,
                    {y, 1, Length[A]}];
                 Export["circle" <> ToString[r] <> ".dat", M, "List"];
                 ];

NN1 = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
NN2 = {{1, 1}, {-1, 1}, {1, -1}, {-1, -1}};
NN3 = {{2, 0}, {-2, 0}, {0, 2}, {0, -2}};
NN4 = {{2, 1}, {-2, 1}, {1, 2}, {-1, 2}, {2, -1}, {-2, -1}, {1, -2}, {-1, -2}};
NN1to4 = Join[{NN1}, {NN2}, {NN3}, {NN4}];

IsBoundaryPixel[r_, x_, xn_, yn_] :=
     If[xn > r, 1, xn2 = If[xn ≤ 0, Abs[xn] + 1, xn];
                   hh = M[[x]] + yn;
                   hh = If[hh ≤ 0, Abs[hh] + 1, hh];
                   If[xn2 > r, 1, If[hh > M[[xn2]], 1, 0]]]

BoundaryPixelList[n_] :=
  Module[{xn, yn, r = Length[M], BPL},
           BPL = Table[0, {r}];
           Do[Do[Catch[Do[xn = x + NN1to4[[n, i, 1]];
                          yn = -y + NN1to4[[n, i, 2]];
                          If[IsBoundaryPixel[r, x, xn, yn] == 1, Throw[BPL[[x]]++]],
                          {i, 1, Length[NN1to4[[n]]]}];
                         Break[]],
                 {y, 0, M[[x]] - 1}],
              {x, 1, r}];
           Return[BPL]
       ];

ConstantJij = {1, 1, 1, 1};
AttenuatedJij = {1, 1 / Sqrt[2], 1 / 2, 1 / Sqrt[5]};

PixelEnergy[r_, x_, y_, e_] := Module[{xn, yn, energy = 0},
                                  For[j = 1, j ≤ e, j++,
                                     For[i = 1, i ≤ Length[NN1to4[[j]]], i++,
                                        xn = x + NN1to4[[j, i, 1]];
                                        yn = -y + NN1to4[[j, i, 2]];
                                        energy += ConstantJij[[j]] *
                                                  IsBoundaryPixel[r, x, xn, yn]
                                        ]
                                     ];
                                   Return[energy]
                                 ];

BoundaryEnergyList[n_, e_] := Module[{r = Length[M], BPL, BEL},
                                 BPL = BoundaryPixelList[n];
                                 BEL = Table[0, {r}];
                                 Do[Do[BEL[[x]] += PixelEnergy[r, x, y, e],
                                       {y, 0, BPL[[x]] - 1}],
                                    {x, 1, r}];
                                 Return[BEL]
                                 ];

Perimeter[n_] := 4 * Apply[Plus, BoundaryPixelList[n]];

PerimeterEnergy[n_, e_] := 4 * Apply[Plus, BoundaryEnergyList[n, e]];

Area := 4 * Apply[Plus, M];
```

APPENDIX E

A C++ Object-Oriented Program for Brownian Motion Simulation

The basic object in this object-oriented program is the *simple-lattice-point*, which is equivalent to pixels in the text. The *simple-lattice-point* object holds a single piece of information, a pointer to the droplet/bubble/cell it belongs to. The object has a number of functions which allow it to compare different *simple-lattice-point*s, update the spin of a *simple-lattice-point*, print-out the spin of a *simple-lattice-point*, *etc.*

The next level up is the *bubble* object. The *bubble* object is a generic object which represents a bubble, a biological cell, or a droplet. It holds information about the bubble's spin, type, area, target area and center-of-mass of bubble. The object has a number of functions that assign and update the values of the various parameters it holds.

The header file `brownian2d.h` contains the highest-level object is the lattice itself. The *brownian2d* object contains the entire lattice, which consists of a two-dimensional array of *simple-lattice-point* objects. The dimensions of the lattice are X and Y. The default size is 256×256 which the user changed in the `brownian2d.cc` source code, subject to the constraint that $X \times Y$ must be a power of 2. The *brownian2d* object reads variables from the text file `variables.txt` and the vectors $\mathbf{M}$ the Mathematica program generates in appendix D. Files named `circle<r>.dat`, where `<r>` is the radius of the digitised circle store the vectors $\mathbf{M}$. All the `circle<r>.dat` files live in a subdirectory called **Circle_file**.

The *brownian2d* object then initializes the lattice, creates the droplets, simulates Brownian motion using one of the three algorithms (2a, 2b, 3) and finally writes its output to files in the directory **data**. The user specifies the algorithm to simulate at run-time.

The brownian2d object uses a Matlab engine to generate images of the lattice during simulations. The *brownian2d* object first stores the spins of the entire lattice in a file and then calls the Matlab program named visualize1 to convert the file to an image and display it on the user's screen. This process slows down the simulation significantly, but is very useful for debugging. The user must include the proper Matlab library during compilation. Different operating systems place the library in different directories. For example, in Sun Solaris, compile the code by typing:

g++ -L/usr/local/src/matlab6.5/extern/lib/sol2 -leng -lmx -lm
-I/usr/local/src/matlab6.5/extern/include
lattice_point.h bubble.h ij.h ran3_long.h ran3_float.h brownian2d.h
brownian2d.cc -o brownian2d

Consult the local documentation for proper library paths. To execute the object code, type:

brownian2d 2a or **brownian2d 2b** or **brownian2d 3**

Three objects: *simple-lattice-point*, *bubble* and *brownian2d* live in their respective header files. The source code which connects the user to the objects is in the file `brownian2d.cc`.

The pseudo-random number generators are `ran3_float.h` and `ran3_long.h`. The first returns a value between 0 and 1 and determines the transition probabilities. The second returns an integer between 0 and $2^{30} - 1$ and for random selecting of candidate and target pixels.

Finally, the *ij* object holds two values: i and j. The *map* container then associates

an *ij* object to a value a_{ij}. For example, i and j may represent the area and energy of droplets respectively, and a_{ij} may represent the number of occurences of a particular set of ij values. The indices of a matrix or a two-dimensional array always begin with 0. Therefore, an array is cumbersome and wasteful if we do not know its size beforehand.

```
#ifndef LATTICE_POINT_CLASS_H
#define LATTICE_POINT_CLASS_H
#include <iostream>
#include "bubble.h"

//enum where {interior, surface, edge, vertex};

class simple_lattice_point {
 protected:
  bubble* bub;                      // The bubble it belongs to

 public:
  simple_lattice_point(bubble* bb = 0) : bub(bb) {}
  ~simple_lattice_point() {}

  // setting spin of lattice point, used during initialization
  void set_spin(bubble* bb) { bub = bb; }
  void set_spin(simple_lattice_point p) { bub = p.bub; }
  void update_sum_xy(int x, int y) { bub->update_sum_xy(x, y); }

  // overloaded assignment operator, used during coarsening
  simple_lattice_point operator=(const simple_lattice_point &p) {
    bub = p.bub;
    return *this; }

  // overloaded << operator
  friend std::ostream& operator<<(std::ostream &strm,
                                  const simple_lattice_point &lp) {
    strm.width(2);
    strm << lp.get_spin();
    return strm;
  }

  // getting spins. simple_lattice_point is declared as a friend of bubble,
  // so it can directly access bubble's spin
  const int get_spin() const { return bub->spin; }
  bubble* get_bubble_pointer() const { return bub; }
  const int get_type() const { return bub->type; }
  const int get_area() const { return bub->area; }
  const int get_target_area() const { return bub->target_area; }
  const double get_bubble_energy() const { return bub->energy; }

  // comparing spins between two lattice points
  bool operator==(simple_lattice_point& p) {return bub->spin == (p.bub)->spin;}
  bool operator!=(simple_lattice_point& p) {return bub->spin != (p.bub)->spin;}

  // increase and decrease bubbble's volume
  void increase_bubble_area() { bub->increase_area(); }
  void decrease_bubble_area() { bub->decrease_area(); }
  void increment_bubble_energy(int de) { bub->increment_energy(de); }
};

// compound_lattice_point inherits from simple_lattice_point.
// compound_lattice_point also stores the neighbouring lattice points.
// The reason for not combining the two into a single class is to
// avoid infinite recursion when initializing lattice points :
// initializing a lattice point leads to initializing its neighbours,
// which then leads to initializing neighbours' neighbours, ad infinitum.
class compound_lattice_point : public simple_lattice_point {
 private:
  simple_lattice_point** neighbour;  // The neighbouring lattice points

 public:
  compound_lattice_point() { neighbour = new simple_lattice_point* [33]; }
  ~compound_lattice_point() { delete [] neighbour; }

  void set_neighbour(int n, simple_lattice_point* p) { *(neighbour + n) = p; }
  simple_lattice_point** list_of_neighbours() { return neighbour; }
};

#endif
```

```
#ifndef BUBBLE_H
#define BUBBLE_H
#include <iostream>

class bubble {
 private:
  int spin;
  int type;                     // 0 = medium(background) 1 = bubble
  int area;
  int target_area;
  int sum_x, sum_y;             // ctr_x = sum_x/area, ctr_y = sum_y/area
  int energy;
  float initial_ctr_x, initial_ctr_y;
  friend class simple_lattice_point;

 public:
  bubble(int sp = 0, int ty = 0, int ta = 0) :
    spin(sp), type(ty), target_area(ta), area(0), energy(0) {}
  ~bubble() {}

  void set_spin(int sp) { spin = sp; }
  void set_type(int ty) { type = ty; }
  void set_area(int a) { area = a; }
  void increase_area(const int& a=1) { area += a; }
  void decrease_area(const int& a=1) { area -= a; }
  void set_target_area(int ta) { target_area = ta; }
  void set_sum_xy(int x, int y) { sum_x = x; sum_y = y; }
  void update_sum_xy(const int& x, const int& y) { sum_x += x; sum_y += y; }
  void set_initial_ctr() { initial_ctr_x = float(sum_x)/area;
                           initial_ctr_y = float(sum_y)/area; }
  void set_energy(int e) { energy = e; }
  void increment_energy(int de) { energy += de; }

  int get_spin() const { return spin; }
  int get_type() const { return type; }
  int get_area() const { return area; }
  int get_target_area() const { return target_area; }
  int get_energy() const { return energy; }
  float get_ctr_x() const { return float(sum_x)/area; }
  float get_ctr_y() const { return float(sum_y)/area; }
  float get_initial_ctr_x() const { return initial_ctr_x; }
  float get_initial_ctr_y() const { return initial_ctr_y; }
  float get_rsquare() const {
    float x = get_ctr_x();
    float y = get_ctr_y();
    return ((x - initial_ctr_x)*(x - initial_ctr_x) +
            (y - initial_ctr_y)*(y - initial_ctr_y));
  }
};

#endif
```

```
#ifndef BROWNIAN2D_H
#define BROWNIAN2D_H
#include <iostream>
#include <vector>                  // for vector<>
#include <cmath>                   // for exp(), sqrt()
#include <iomanip>                 // for setw(), setfill()
#include <cstdlib>                 // for exit(), system(), srand(), rand()
#include <fstream>
#include <strstream>
#include <ctime>                   // for time(NULL)
#include <cstdio>                  // for sprintf()
#include <string>
#include <map>                     // associate ij object with entries
#include "bubble.h"
#include "lattice_point.h"
#include "ij.h"                    // an object that holds indices of a table.
#include "ran3_long.h"
#include "ran3_float.h"
#include "mat.h"
#include "engine.h"                // Matlab engine for visualization
#define  MOD(X, Y)     (X & (Y-1))
#define  DS            256         // default square lattice dimension
#define  NB            4           // # of droplets to be created
#define  NRUN          1           // # of simulation runs
#define  NMCS          100         // # of mcs each run
#define  NEminus1      0.01        // data collected every 0.01 mcs
#define  NE0           0.1         // data collected every 0.1 mcs
#define  NE1           1           // data collected every 1 mcs
#define  NE2           10          // data collected every 10 mcs
#define  NE3           100         // data collected every 100 mcs
#define  NE4           1000        // data collected every 1000 mcs
#define  NTsmall       19          // number of times data is collected t<1
#define  NT            56          // total number of times data is collected
#define  NN1           4           // up to 1st nearest neighbours
#define  NN2           8           // up to 2nd nearest neighbours
#define  NN3           12          // up to 3rd nearest neighbours
#define  NN4           20          // up to 4th nearest neighbours
#define  NNn           NN4
using namespace std;

/*************************************************************************/
/*         Started on 27/2/2002                                          */
/*         Helical boundary condition                                    */
/*         Dry foam                                                      */
/*         Default size is 128*128;                                      */
/*         Use only powers of 2 for lattice dimensions                   */
/*         Coupling between bubble and medium is 0                       */
/*         Note: in order for MOD() to work, XY must be a power of 2.    */
/*                                                                       */
/*         This is version 2 (21/3/2003)                                 */
/*         It includes codes to calculate displacement of droplets       */
/*         as well as the energy and area of droplets.                   */
/*                                                                       */
/*         This is version 3 (14/1/2004)                                 */
/*         Incorporates algorithm 2 and 3 by way of virtual function.    */
/*         Default size is DS x DS.                                      */
/*         Uses the result from previous Mathematica program             */
/*         to produce an initial circular droplet.                       */
/*                                                                       */
/*         (8/9/2004)                                                    */
/*         Gather data at smaller t (0.1 mcs, 0.01 mcs)                  */
/*                                                                       */
/*         (12/10/2004)                                                  */
/*         Count the number of spin-flips in each Monte-Carlo steps.     */
/*************************************************************************/

class brownian2d {
 protected:
  int X, Y, XY;
  int disp[NN4+1];
  float SpinFlipsCount[12][NMCS+1];
  float countspinflips;
  double kT;
  float Ja;                    // area constraint
  float Jp;                    // perimeter constraint
  float Jij[2][2];             // coupling constants
  float RR[NT][NRUN*NB];
  float meanRR[NT][3];
  long ran_seed1, ran_seed2, ran_seed3;
  std::vector<simple_lattice_point> element;
  bubble* bub_array[NB+1];     // An array of pointers to droplets
  map<ij, int> ae_count;       // tabulates the area-energy distribution

  void initialize();
  void set_disp();
  void read_variables();
  void reset_spin_count();
  void generate_random_seeds();
  void create_droplets(int);
  void create_droplet_at(int, int, int, int);
  void show_all_lattice_points() const;
  void save_img(const char*, int) const;
  void save_data(const char*, int) const;
  void save_spin_counts(const char*, int) const;
  void commence_droplet_tracking();
  void reset();
  void gather_data_smallt(int&, float, int);
  void gather_data(int&, int, int, Engine* ep1);
  void calc_meanRR();
  virtual bool flip_spin(const float);
  bool flip_routine(simple_lattice_point&, simple_lattice_point&, int&, int);

 public:
  brownian2d(int x = DS) : X(x), Y(x), XY(x*x) { initialize(); }
  brownian2d(int x, int y) : X(x), Y(y), XY(x*y) { initialize(); }
  ~brownian2d() { for(int i=0; i<NB; i++)  delete bub_array[i]; };

  int get_X() const { return X; }
  int get_Y() const { return Y; }
  int get_XY() const { return XY; }
  void show_spins_of_all_lattice_points() const;
  void show_droplets_info() const;
  void minimize(const char*);
};

// derived class (algorithm 2a is in brownian2d class).
// everything in algorithm3 is the same as brownian2d except for the
// flip_spin() function.
class algorithm3 : public brownian2d {
 protected:
  virtual bool flip_spin(const float);
 public:
  algorithm3(int x = DS) : brownian2d(x) { initialize(); }
  algorithm3(int x, int y) : brownian2d(x, y) { initialize(); }
  ~algorithm3() { for(int i=0; i<NB; i++)  delete bub_array[i]; };
};

class algorithm2b : public brownian2d {
 protected:
  virtual bool flip_spin(const float);
 public:
  algorithm2b(int x = DS) : brownian2d(x) { initialize(); }
  algorithm2b(int x, int y) : brownian2d(x, y) { initialize(); }
  ~algorithm2b() { for(int i=0; i<NB; i++)  delete bub_array[i]; };
};
```

```
void brownian2d::initialize() {
  set_disp();
  read_variables();
  element.reserve(XY);
  generate_random_seeds();

  bub_array[0] = new bubble(0, 0);           // spin 0, type 0 (medium)
  bub_array[0]->set_area(0);
  for(int i=1; i<=NB; i++)
    bub_array[i] = new bubble(i, 1);         // spin i, type 1 (droplet)
}

void brownian2d::set_disp() {

  disp[0] = 0;

  // 1st nearest               // 2nd nearest
  disp[1] = 1;                 disp[5] = 1 + X;
  disp[2] = -1;                disp[6] = 1 - X;
  disp[3] = X;                 disp[7] = -1 + X;
  disp[4] = -X;                disp[8] = -1 - X;

  // 3rd nearest               // 4th nearest
  disp[9] = 2;                 disp[13] = 2 + X;
  disp[10] = -2;               disp[14] = 2 - X;
  disp[11] = 2*X;              disp[15] = 1 + 2*X;
  disp[12] = -2*X;             disp[16] = 1 - 2*X;
                               disp[17] = -2 + X;
                               disp[18] = -2 - X;
                               disp[19] = -1 + 2*X;
                               disp[20] = -1 - 2*X;
}

void brownian2d::read_variables() {

  char comment[100];

  fstream datafile("variables.txt", ios::in);
  if(!datafile) {
    cerr << "Fail to open variables.txt file.\n";
    exit(EXIT_FAILURE);
  }

  datafile >> Ja;          datafile.getline(comment, 100);  // use '\n'
  datafile >> Jp;          datafile.getline(comment, 100);  // as delimiter
  datafile >> Jij[0][0];   datafile.getline(comment, 100);
  datafile >> Jij[0][1];   datafile.getline(comment, 100);
  datafile >> Jij[1][0];   datafile.getline(comment, 100);
  datafile >> Jij[1][1];   datafile.getline(comment, 100);

  datafile.close();
}

void brownian2d::reset_spin_count() {

  for(int i=0; i<12; i++)
    for(int t=0; t<=NMCS; t++)
      SpinFlipsCount[i][t] = 0;
}

void brownian2d::generate_random_seeds() {

  srand(time(NULL));

  ran_seed1 = -rand();
  ran_seed2 = -rand();
  ran_seed3 = -rand();
}
```

```
void brownian2d::create_droplets(int radius) {

  // location of droplets
  int xc[NB + 1] = {X/2, X/4, X/4, 3*X/4, 3*X/4};
  int yc[NB + 1] = {Y/2, Y/4, 3*Y/4, Y/4, 3*Y/4};

  // declare the droplets including the medium
  for(int i=0; i<=NB; i++) {
    bub_array[i]->set_area(0);
    bub_array[i]->set_sum_xy(0, 0);
    bub_array[i]->set_energy(0);
  }
  // initialize all lattice points to point to medium
  for(int coord=0; coord<XY; coord++) {
    element[coord].set_spin(bub_array[0]);
    element[coord].increase_bubble_area();
  }
  // create droplet i
  for(int i=1; i<=NB; i++)
    create_droplet_at(radius, i, xc[i], yc[i]);

  // calculate initial droplet energy
  for(int coord=0; coord<XY; coord++)
    for(int n=1; n<=NNn; n++)
      if(element[coord] != element[MOD(coord + disp[n], XY)])
        element[coord].increment_bubble_energy(1);

}

void brownian2d::create_droplet_at(int radius, int i, int xc, int yc) {

  char* filename = new char[80];
  sprintf(filename, "./Circle_file/circle%d.dat", radius);
  fstream circlefile(filename, ios::in);
  if(!circlefile) {
    cerr << "Fail to open file " << filename << endl;
    exit(EXIT_FAILURE);
  }
  // Create the droplet at (xc, yc)
  int pix;
  int x0, y0;
  int x1, y1;
  int target_area = 0;
  for(int x=0; x<radius; x++) {
    circlefile >> pix;
    for(int y=0; y<pix; y++) {
      x0 = x + xc; x1 = -x -1 + xc;
      y0 = y + yc; y1 = -y -1 + yc;
      element[y0*X + x0].set_spin(bub_array[i]);
      element[y0*X + x1].set_spin(bub_array[i]);
      element[y1*X + x0].set_spin(bub_array[i]);
      element[y1*X + x1].set_spin(bub_array[i]);
      target_area += 4;
      bub_array[i]->increase_area(4);
      bub_array[0]->decrease_area(4);
      bub_array[i]->update_sum_xy(x0,y0);
      bub_array[i]->update_sum_xy(x0,y1);
      bub_array[i]->update_sum_xy(x1,y0);
      bub_array[i]->update_sum_xy(x1,y1);
    }
  }
  bub_array[0]->set_target_area(XY - target_area);
  bub_array[i]->set_target_area(target_area);
  circlefile.close();

}
```

```
void brownian2d::show_all_lattice_points() const {

  for(int j=0; j<Y; j++) {
    cout << endl;
    for(int i=0; i<X; i++)
      cout << element[j*X + i] << " ";
    cout << endl;
  }

}

void brownian2d::show_droplets_info() const {
  for(int i=1; i<=NB; i++) {
    cout << "spin = " << bub_array[i]->get_spin() << "\t";
    cout << "type = " << bub_array[i]->get_type() << "\t";
    cout << "target area = " << setw(4)
         << bub_array[i]->get_target_area() << "\t";
    cout << "actual area = " << bub_array[i]->get_area() << "\t";
    cout << "energy = " << bub_array[i]->get_energy() << endl;
  }
}

void brownian2d::save_img(const char* argv, int mcs) const {
  char* name = new char[32];

  ostrstream filename(name, 32);
  if(!filename) {
    cerr << "Fail to create filename " << argv << endl;
    exit(EXIT_FAILURE);
  }
  filename.seekp(0);
  filename << "./movies/" << argv;
  filename << ".dat" << ends;
  fstream frame_file(name, ios::out);
  if(!frame_file) {
    cerr << "Fail to open file " << name << endl;
    exit(EXIT_FAILURE);
  }

  for(int i=0; i<XY; i++)
    frame_file << element[i];

  frame_file.close();
  delete name;
}

void brownian2d::save_data(const char* algo, int radius) const{

  char* name = new char[32];

  ostrstream filename(name, 32);
  if(!filename) {
    cerr << "Fail to create filename " << endl;
    exit(EXIT_FAILURE);
  }
  filename.seekp(0);   // diffusion data
  filename << "./data/difusn" << algo << "_" << radius << "_";
  filename << setfill('0') << setw(2) << 10*kT << ".dat" << ends;
  fstream ctr_m_file(name, ios::out);
  if(!ctr_m_file) {
    cerr << "Fail to open file " << name << endl;
    exit(EXIT_FAILURE);
  }
  filename.seekp(0);   // area-energy distribution
  filename << "./data/ae_dist" << algo << "_" << radius << "_";
  filename << setfill('0') << setw(2) << 10*kT << ".dat" << ends;
  fstream ae_file(name, ios::out);
  if(!ae_file) {
    cerr << "Fail to open file " << name << endl;
    exit(EXIT_FAILURE);
  }

  filename.seekp(0);   // spin-flips count
  filename << "./data/sp_count" << algo << "_" << radius << ".dat" << ends;
  fstream sp_file(name, ios::out);
  if(!sp_file) {
    cerr << "Fail to open file " << name << endl;
    exit(EXIT_FAILURE);
  }
  for(int t=0; t<NTsmall; t++)
    ctr_m_file << meanRR[t][0] << "\t"
               << setw(10) << meanRR[t][1] << " "
               << setw(10) << meanRR[t][2] << endl;

  // Normal iterator can be used to modify the contents of maps.
  // Since save_data() is a constant function (doesn't allow
  // modification of data), the compiler will complain if
  // we use normal iterator. Solution: use const_iterator.

  map<ij, int>::const_iterator it;
  for(it=ae_count.begin(); it!=ae_count.end(); it++)
    ae_file << (*it).first.getI() << "\t" << (*it).first.getJ() << "\t"
            << (*it).second << endl;

  ctr_m_file.close();
  ae_file.close();

  delete name;
}

void brownian2d::save_spin_counts(const char* algo, int radius) const {
  char* name = new char[32];

  ostrstream filename(name, 32);
  if(!filename) {
    cerr << "Fail to create filename " << endl;
    exit(EXIT_FAILURE);
  }
  filename.seekp(0);   // spin-flips count
  filename << "./data/sp_count" << algo << "_" << radius << ".dat" << ends;
  fstream sp_file(name, ios::out);
  if(!sp_file) {
    cerr << "Fail to open file " << name << endl;
    exit(EXIT_FAILURE);
  }
  for(int i=0; i<12; i++) {
    for(int t=0; t<=NMCS; t++)
      sp_file << SpinFlipsCount[i][t] << " ";
    sp_file << endl;
  }
  delete name;
}

void brownian2d::commence_droplet_tracking() {

  for(int i=0; i<=NB; i++)
    bub_array[i]->set_initial_ctr();

}

void brownian2d::reset() {

  for(int t=0; t<NT; t++) {
    meanRR[t][0] = meanRR[t][1] = meanRR[t][2] = 0;
    for(int r=0; r<NRUN*NB; r++)
      RR[t][r] = 0;
```

```
  }
}

void brownian2d::minimize(const char* algo) {

  string data_directory = "mkdir data";
  string movie_directory = "mkdir movies";
  system(data_directory.c_str());
  system(movie_directory.c_str());

  /*Engine *ep1;
  if(!(ep1 = engOpen("\0"))) {
    cerr << "Can't start MATLAB engine1\n";
    exit(EXIT_FAILURE);
  }
  */
  int rad[8] = {3,4,5,6,7,8,12,16};
  int radius;
  for(int r=0; r<8; r++) {
    radius = rad[r];
    cout << "radius = " << radius << endl;

    reset_spin_count();
    for(kT=0.5; kT<=6.0; kT+=0.5) {
      cout << "\t temp = " << kT << endl;
      reset();
      SpinFlipsCount[int(kT/0.5-1)][0] = kT;

      for(int run=1; run<=NRUN; run++) {
        generate_random_seeds();
        create_droplets(radius);

        for(int wm=1; wm<=200; wm++) // warm-up
          flip_spin(1.0);

        commence_droplet_tracking();
        int t = 0;
        bool flag = false;

        /*for(float mcs=NEminus1; mcs<=NE0; mcs+=NEminus1){
          flip_spin(NEminus1);
          gather_data_smallt(t, mcs, run);
        }
        for(float mcs=NE0+NE0; mcs<NE1; mcs+=NE0){
          flip_spin(NE0);
          gather_data_smallt(t, mcs, run);
        }
        */
        for(int mcs=NE1; mcs<=NMCS; mcs+=NE1) {
          countspinflips = 0;
          if(flip_spin(NE1)) {
            flag = true;    // droplet vanished or touching the edge of lattice
            cout << "run " << run << "\tmcs " <<  mcs << endl;
            break;
            }
          //gather_data(t, mcs, run, ep1);
          SpinFlipsCount[int(kT/0.5-1)][mcs] = countspinflips/NB;
        }

        if(flag) {          // Repeat this run if flag is true
          generate_random_seeds();
          run--;
          continue;
        }
      }
      //calc_meanRR();
      //save_data(algo, radius);
      //ae_count.clear();     // clear the area-energy distribution table
      }
      save_spin_counts(algo, radius);
    }
    //engClose(ep1);
  }

  void brownian2d::gather_data_smallt(int& t, float mcs, int run) {

    meanRR[t][0] = mcs;
    for(int i=1; i<=NB; i++) {
      meanRR[t][1] += bub_array[i]->get_rsquare();
      RR[t][NB*(run-1) + i - 1] = bub_array[i]->get_rsquare();
    }
    t++;
  }

  void brownian2d::gather_data(int& t, int mcs, int run, Engine* ep1) {

    char* action = new char[80];
    // gather area-energy distribution data (allow 100 mcs warmup)
    if(mcs >= 100) {
      ij area_energy;
      for(int i=1; i<=NB; i++) {
        area_energy.setIJ(bub_array[i]->get_area(), bub_array[i]->get_energy());
        ae_count[area_energy]++;
      }
    }

    // gather diffusion data
    if((mcs <= 10    && mcs%NE1 == 0) ||
       (mcs <= 100   && mcs%NE2 == 0) ||
       (mcs <= 1000  && mcs%NE3 == 0) ||
       (mcs <= 10000 && mcs%NE4 == 0)) {
      meanRR[t][0] = mcs;
      for(int i=1; i<=NB; i++) {
        meanRR[t][1] += bub_array[i]->get_rsquare();
        RR[t][NB*(run-1) + i - 1] = bub_array[i]->get_rsquare();
        }
      cout << t << "\t" << mcs << endl;
      t++;
      //show_droplets_info();
      /*save_img("frame", mcs);
        sprintf(action, "visualize1 ./movies/frame.dat %d %d %d", mcs, X, Y);
        engEvalString(ep1, action);
      */
    }
    delete action;
  }

  void brownian2d::calc_meanRR() {

    double errRR;
    for(int t=0; t<NT; t++) {
      meanRR[t][1] /= (NRUN*NB);
      errRR = 0;

      for(int r=0; r<NRUN*NB; r++)
        errRR += pow(RR[t][r] - meanRR[t][1], 2);

      meanRR[t][2] = sqrt(errRR/(NRUN*NB*(NRUN*NB - 1)));
    }
  }

  bool brownian2d::flip_spin(const float ne) {

    int coord1, coord2, coordj;
    int type1, type2, typej;
    int index2;
```

```
  int dN;
  double J_ij = Jij[1][1];
  double fac[NNn];
  double dEJ, dEA;

  for(int dn=0; dn<NNn; dn++)
    fac[dn] = (double(NNn + dn)/double(NNn - dn));

  for(int i=0; i<int(ne*XY); i++) {
    dEJ = dEA = 0;
    dN = NNn;
    coord1 = MOD(ran3l(&ran_seed1), XY);
    index2 = ran3l(&ran_seed2)%NN2 + 1;
    coord2 = MOD(coord1 + disp[index2], XY);

    if(element[coord1] != element[coord2]) {
      type1 = element[coord1].get_type();
      type2 = element[coord2].get_type();

      if(type1 == type2) {
        cout << "droplets are touching each other!\t";
        return true;
      }
      // delta_E due to interaction (spin 1 -> spin 2)
      for(int i=1; i<=NNn; i++) {
        coordj = MOD(coord1 + disp[i], XY);
        if(element[coordj] != element[coord1])
          dN -= 2;
      }
      dEJ = dN*J_ij;

      // delta_E due to area constraint.
      // Bubble1 - 1 pixel, bubble2 + 1 pixel.
      // There is no area constraint on medium.
      // This is ensured by the value of "type".
      dEA = (Ja*type1*(2*(element[coord1].get_target_area() -
                         element[coord1].get_area()) + 1) +
             Ja*type2*(-2*(element[coord2].get_target_area() -
                           element[coord2].get_area()) + 1));

      // flip spins
      if(dEJ<=0 && dEA<=0){
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
      else if((dEJ<=0 && dEA>0) && ran3f(&ran_seed3) < exp(-dEA/kT)) {
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
      else if((dEJ>0 && dEA<=0) && ran3f(&ran_seed3) < fac[dN]*exp(-dEJ/kT)) {
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
      else if(ran3f(&ran_seed3) < fac[dN]*exp(-(dEJ + dEA)/kT)) {
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
    }
  }
  return false;
}

bool brownian2d::flip_routine(simple_lattice_point& spin1,
                      simple_lattice_point& spin2, int& coord1, int dEJ) {
  spin1.decrease_bubble_area();
  spin2.increase_bubble_area();
  spin1.increment_bubble_energy(dEJ);
  spin2.increment_bubble_energy(dEJ);
  if(spin1.get_area() == 0) {
    cout << "droplet vanished!\t";
    return true;
  }
  int x = coord1%X;
  int y = coord1/X;
  if(x == 0 || x == X-1 || y == 0 || y == Y-1) {
    cout << "droplet at lattice edge!\t";
    return true;
  }
  spin1.update_sum_xy(-x, -y);
  spin2.update_sum_xy(x, y);
  spin1 = spin2;
  countspinflips++;
  return false;

}

bool algorithm3::flip_spin(const float ne) {

  int coord1, coord2, coordj;
  int type1, type2, typej;
  int index2;
  int dN;
  double J_ij = Jij[1][1];
  double dE, dEJ;

  for(int i=0; i<int(ne*XY); i++) {
    dE = 0;
    dN = NNn;
    coord1 = MOD(ran3l(&ran_seed1), XY);
    index2 = ran3l(&ran_seed2)%NN2 + 1;
    coord2 = MOD(coord1 + disp[index2], XY);

    if(element[coord1] != element[coord2]) {
      type1 = element[coord1].get_type();
      type2 = element[coord2].get_type();

      if(type1 == type2) {
        cout << "droplets are touching each other!\t";
        return true;
      }
      // delta_E due to interaction
      for(int i=1; i<=NNn; i++) {
        coordj = MOD(coord1 + disp[i], XY);
        if(element[coordj] != element[coord1])
          dN -= 2;
      }
      dE = dN*J_ij;
      dEJ = dE;

      // delta_E due to area constraint.
      // droplet1 - 1 pixel, droplet2 + 1 pixel.
      // There is no area constraint on medium.
      // This is ensured by the value of "type".
      dE += (Ja*type1*(2*(element[coord1].get_target_area() -
                         element[coord1].get_area()) + 1) +
             Ja*type2*(-2*(element[coord2].get_target_area() -
                           element[coord2].get_area()) + 1));

      // flip spins
      if(dE <= 0) {
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
      else if(kT > 0 && ran3f(&ran_seed3) < exp(-dE/kT))
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
```

```
    }
  }
  return false;
}

bool algorithm2b::flip_spin(const float ne) {

  int coord1, coord2, coordj;
  int type1, type2, typej;
  int index2;
  int dN;
  double J_ij = Jij[1][1];
  double dEJ, dEA;

  for(int i=0; i<int(ne*XY); i++) {
    dEJ = dEA = 0;
    dN = NNn;
    coord1 = MOD(ran3l(&ran_seed1), XY);
    index2 = ran3l(&ran_seed2)%NN2 + 1;
    coord2 = MOD(coord1 + disp[index2], XY);

    if(element[coord1] != element[coord2]) {
      type1 = element[coord1].get_type();
      type2 = element[coord2].get_type();

      if(type1 == type2) {
        cout << "droplets are touching each other!\t";
        return true;
      }
      // delta_E due to interaction
      for(int i=1; i<=NNn; i++) {
        coordj = MOD(coord1 + disp[i], XY);
        if(element[coordj] != element[coord1])
          dN -= 2;
      }
      dEJ = dN*J_ij;

      // delta_E due to area constraint.
      // droplet1 - 1 pixel, droplet2 + 1 pixel.
      // There is no area constraint on medium.
      // This is ensured by the value of "type".
      dEA = (Ja*type1*(2*(element[coord1].get_target_area() -
                         element[coord1].get_area()) + 1) +
             Ja*type2*(-2*(element[coord2].get_target_area() -
                           element[coord2].get_area()) + 1));

      // flip spins
      if(dEJ<=0 && dEA<=0){
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
      else if((dEJ<=0 && dEA>0) && ran3f(&ran_seed3) < exp(-dEA/kT)) {
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
      else if((dEJ>0 && dEA<=0) && ran3f(&ran_seed3) < exp(-dEJ/kT)) {
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
      else if(ran3f(&ran_seed3) < exp(-(dEJ + dEA)/kT)) {
        if(flip_routine(element[coord1], element[coord2], coord1, (int)dEJ))
          return true;
      }
    }
  }
  return false;
}
#endif
```

```
#include <iostream>
#include "brownian2d.h"
using namespace std;

int main(int argc, char* argv[]) {

  system("date");

  if(argc == 1) {
    cout << "Please enter the version of algorithm to use: 2a or 2b or 3\n";
    exit(EXIT_FAILURE);
  }

  string algo = argv[1];

  if(algo == "2a") {
    cout << "Algorithm 2a: randomly select a neighbour's spin.\n"
         << "4-cases modified Metropolis acceptance ratios.\n";
    brownian2d simulation;
    simulation.minimize(argv[1]);
  }
  else if(algo == "2b") {
    cout << "Algorithm 2b: randomly select a neighbour's spin.\n"
         << "4-cases Metropolis acceptance ratios.\n";
    algorithm2b simulation;
    simulation.minimize(argv[1]);
  }
  else if(algo == "3") {
    cout << "Algorithm 3: randomly select a neighbour's spin.\n"
         << "2-cases Metropolis acceptance ratios.\n";
    algorithm3 simulation;
    simulation.minimize(argv[1]);
  }
  else {
    cout << "Please enter a valid version of algorithm: 2a or 2b or 3\n";
    exit(EXIT_FAILURE);
  }
  system("date");

}
```

```
#ifndef RAN3_FLOAT_H
#define RAN3_FLOAT_H

/*************************************************************/
/*                                                           */
/*           Lagged Fibonacci random number generator        */
/*           ----------------------------------------        */
/*                                                           */
/*              X_n = (X_(n-55) - X_(n-24)) mod m            */
/*                                                           */
/*                Documentation                              */
/*                -------------                              */
/*                                                           */
/* 30/8/2001    MBIG = 1000000000                            */
/* 17/9/2001    MBIG = (1L<<30)                              */
/* 17/9/2001    #define mod_diff(x,y) (((x)-(y))&(MBIG-1))    */
/*              ran3f() returns a number in the range [0, 1] */
/*                                                           */
/*                    Usage                                  */
/*                    -----                                  */
/*                                                           */
/*        Call with negative seed idum                       */
/*                                                           */
/*************************************************************/

#include <cstdlib>
#define MBIG (1L<<30)           // modulus
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)
#define SLAG 24                 // short lag
#define LLAG 55                 // long lag
#define mod_diff(x,y)   (((x)-(y))&(MBIG-1))  // (x-y)mod MBIG

float ran3f(long *idum)
{
        int i,ii,k;
        static int iff=0;
        static int inext,inextp;
        static long ma[LLAG+1];
        long mj,mk;

        if (*idum < 0 || iff == 0) {
                iff=1;
                mj=MSEED-(*idum < 0 ? -*idum : *idum);
                mj %= MBIG;
                ma[LLAG]=mj;
                mk=1;
                for (i=1;i<=(LLAG-1);i++) {
                        ii=(21*i) % LLAG;
                        ma[ii]=mk;
                        mk=mj-mk;
                        if (mk < MZ) mk += MBIG;
                        mj=ma[ii];
                }
                for (k=1;k<=4;k++)
                        for (i=1;i<=LLAG;i++) {
                                ma[i] -= ma[1+(i+30) % LLAG];
                                if (ma[i] < MZ) ma[i] += MBIG;
                        }
                inext=0;
                inextp= (LLAG - SLAG);
                *idum=1;
        }
        if (++inext == (LLAG+1)) inext=1;
        if (++inextp == (LLAG+1)) inextp=1;
        mj = mod_diff(ma[inext], ma[inextp]);
        ma[inext]=mj;
        return mj*FAC;
}

#undef MBIG
#undef MSEED
#undef MZ
#undef FAC
#undef SLAG
#undef LLAG
#undef mod_diff

#endif
/* (C) Copr. 1986-92 Numerical Recipes Software W$5&+1*?$. */
```

```
#ifndef RAN3_LONG_H
#define RAN3_LONG_H

/*************************************************************/
/*                                                           */
/*           Lagged Fibonacci random number generator        */
/*           ----------------------------------------        */
/*                                                           */
/*              X_n = (X_(n-55) - X_(n-24)) mod m            */
/*                                                           */
/*                Documentation                              */
/*                -------------                              */
/*                                                           */
/* 30/08/2001    MBIG = 1000000000                           */
/* 17/09/2001    MBIG = (1L<<30)                             */
/* 17/09/2001    #define mod_diff(x,y) (((x)-(y))&(MBIG-1))   */
/* 21/09/2001    change ran3l() so that it returns an integer */
/*               between 0 and MBIG-1 (1073741823)           */
/*                                                           */
/*                   Usage                                   */
/*                   -----                                   */
/*                                                           */
/*    Call with either positive or negative seed idum        */
/*                                                           */
/*************************************************************/

#define MBIG (1L<<30)        // modulus
#define MSEED 161803398
#define MZ 0
#define SLAG 24              // short lag
#define LLAG 55              // long lag
#define mod_diff(x,y)  (((x)-(y))&(MBIG-1))  // (x-y)mod MBIG

long ran3l(long *idum)
{
        int i, ii, k;
        static int initialize = 1;
        static int inext,inextp;
        static long ma[LLAG+1];
        long mj,mk;

        if (*idum < 0 || initialize) {
                initialize = 0;
                mj=MSEED-(*idum < 0 ? -*idum : *idum);
                mj %= MBIG;
                ma[LLAG]=mj;
                mk=1;
                for (i=1;i<=(LLAG-1);i++) {
                        ii=(21*i) % LLAG;
                        ma[ii]=mk;
                        mk=mj-mk;
                        if (mk < MZ) mk += MBIG;
                        mj=ma[ii];
                }
                for (k=1;k<=4;k++)
                        for (i=1;i<=LLAG;i++) {
                                ma[i] -= ma[1+(i+30) % LLAG];
                                if (ma[i] < MZ) ma[i] += MBIG;
                        }
                inext = 0;
                inextp = (LLAG - SLAG);
                *idum = 1;
        }
        if (++inext == (LLAG+1)) inext=1;
        if (++inextp == (LLAG+1)) inextp=1;
        mj = mod_diff(ma[inext], ma[inextp]);
        ma[inext]=mj;
        return mj;
}

#undef MBIG
#undef MSEED
#undef MZ
#undef SLAG
#undef LLAG
#undef mod_diff

#endif
/* (C) Copr. 1986-92 Numerical Recipes Software W$5&+1*?$. */
```

```
#ifndef IJ_H
#define IJ_H

class ij {
 private:
  int vi;
  int vj;

 public:
  ij(const int& i=0, const int& j=0) : vi(i), vj(j) {}
  ij(const ij& p) { vi = p.vi; vj = p.vj; }
  ~ij() {}
  void setI(const int& i) { vi = i;}
  void setJ(const int& j) { vj = j;}
  void setIJ(const int& i, const int& j) { vi = i; vj = j;}
  int getI() const { return vi;}
  int getJ() const { return vj;}
  bool operator==(const ij& p) const { return (vi==p.vi)&&(vj==p.vj); }
  bool operator!=(const ij& p) const { return (vi!=p.vi)||(vj!=p.vj); }
  bool operator>(const ij& p) const {
    return (vi>p.vi)||((vi==p.vi)&&(vj>p.vj));
  }
  bool operator<(const ij& p) const {
    return (vi<p.vi)||((vi==p.vi)&&(vj<p.vj));
  }
};

#endif
```

```
1.0           // Ja  = area constraint
0.1           // Jp  = perimeter constraint
0.0           // J00 = medium - medium
1.0           // J01 = medium - bubble
1.0           // J10 = bubble - medium
2.0           // J11 = bubble - bubble (between different bubbles)
```

```
% pic.dat constains a 1-d array of integers
% The first element of pic.dat i.e. pic(1), is the monte carlo step
% reshape(pic, dim, dim) reshapes the 1-d array to a dim x dim array
% matlab starts index at 1, the colormap converts 1 to RGB[1 1 1], white;
% 2 to RGB[1 0 0], red; 3 to RGB[0 1 0], green; and 4 to RGB[0 0 1] blue
% Since pic contains 0 and it won't be rendered by the colormap,
% add 1 to every element

%clear all

function output = visualize1(filename, mcs, x_dim, y_dim)
fid   = fopen(filename, 'r');
frame = fscanf(fid, '%d');
frame = frame + 1;
mcs   = strread(mcs);     % Converting string to integer
x_dim = strread(x_dim);
y_dim = strread(y_dim);
A = reshape(frame, x_dim, y_dim);
cmap = [1 1 1; 1 0 0; 0 1 0; 0 0 1; 1 .8 0; 0 0 0];
output = image(A);
colormap(cmap);
axis image;
title(mcs);
%filename = sprintf('./movies/frame%d.tiff', mcs);
%imwrite(A, cmap, filename, 'tiff');
%I = imread(filename, 'tiff');
%imshow(I);
```

BIBLIOGRAPHY

[1] D. Abraham, G. Gallavotti and A. Martin-Lof, Surface Tension in the Ising Model. *Lettere Al Nuovo Cimento*, 2: 143–146 (1971).

[2] D. B. Abraham and P. Reed, Phase Separation in the Two-Dimensional Ising Ferromagnet. *Phys. Rev. Lett.*, 33: 377–379 (1974).

[3] D. B. Abraham and P. Reed, Interface Profile of the Ising Ferromagnet in Two Dimensions. *Commun. Math. Phys.*, 49: 35–46 (1976).

[4] Y. Akutsu and N. Akutsu, Intrinsic Structure of the Phase-Separation Line in the Two-Dimensional Ising Model. *J. Phys. A: Math. Gen.*, 20: 5981–5990 (1987).

[5] R. Aldrovandi, *Special Matrices of Mathematical Physics: Stochastic, Circulant and Bell Matrices*, p. 25–53. World Scientific, Singapore (2001).

[6] P. Ball, *The Self-Made Tapestry: Pattern Formation in Nature*. Oxford University Press, Oxford (1999).

[7] R. B. Bapat and T. E. S. Raghavan, *Nonnegative Matrices and Applications*, p. 1–24. Cambridge University Press, Cambridge (1997).

[8] A.-L. Barabasi and E. H. Stanley, *Fractal Concepts in Surface Growth*, p. 1–37. Cambridge University Press, New York (1995).

[9] B. A. Berg, U. Hansmann and T. Neuhaus, Properties of Interfaces in the Two and Three Dimensional Ising Model. *Z. Phys. B*, 90: 229–239 (1993).

[10] H. A. Bethe, Statistical Theory of Superlattices. *Proc. Roy. Soc. A*, 150: 552–575 (1935).

[11] K. Binder, Monte Carlo Calculation of the Surface Tension for Two- and Three-dimensional Lattice-gas Models. *Phys. Rev. A*, 25: 1699–1709 (1982).

[12] K. Binder and D. P. Landau, Phase Diagrams and Critical Behavior in Ising Square Lattices with Nearest- and Next-Nearest-Neighbor Interactions. *Phys. Rev. B*, 21: 1941–1961 (1980).

[13] A. Birovljev, L. Furuberg, J. Feder, T. Jossang, K. J. Maloy and A. Aharony, Gravity Invasion Percolation in Two Dimensions: Experiment and Simulation. *Phys. Rev. Lett.*, 67: 584–587 (1991).

[14] W. L. Bragg and E. J. Williams, The Effect of Thermal Agitation on Atomic Arrangement in Alloys. *Proc. Roy. Soc. A*, 145: 699–730 (1934).

[15] S. G. Brush, History of the Lenz-Ising Model. *Rev. Mod. Phys.*, 39: 883–893 (1967).

[16] T. W. Burkhardt, Interface Free Energy and Critical Line for the Ising Model with Nearest and Next-Nearest-Neighbor Interactions. *Z. Physik B*, 29: 129–132 (1978).

[17] E. Burkner and D. Stauffer, Monte Carlo Study of Surface Roughening in the Three-Dimensional Ising Model. *Z. Phys. B*, 53: 241–243 (1983).

[18] W. K. Burton, N. Cabrera and F. C. Frank, The Growth of Crystals and the Equilibrium Structure of their Surfaces. *Philos. Trans. Roy. Soc.*, 243: 299–358 (1951).

[19] F. Calheiros, S. Johannesen and D. Merlini, Surface Tension and SOS Limit in the 2D Ising Model. *J. Phys. A: Math. Gen.*, 20: 5991–6000 (1987).

[20] D. Chandler, *Introduction to Modern Statistical Mechanics*, p. 44–49. Oxford University Press, New York (1986).

[21] T. S. Chang, An Extension of Bethe's Theory of Order-Disorder Transitions in Metallic Alloys. *Proc. Roy. Soc. A*, 161: 546–563 (1937).

[22] M. Cieplak, A. Maritan and J. R. Banavar, Interfacial Geometry and Overhanging Configurations. *J. Phys. A: Math. Gen.*, 27: L765–L769 (1994).

[23] C. A. Croxton, *Statistical Mechanics of the Liquid Surface*, p. 1–20. John Wiley & Sons, Chichester (1980).

[24] N. W. Dalton, Distant-neighbour Interactions in Ferromagnetic Systems (BPW approximation). *Proc. Phys. Soc.*, 89: 845–857 (1966).

[25] N. W. Dalton and D. W. Wood, Critical Point Behavior of the Ising Model with Higher-Neighbor Interactions Present. *J. Math. Phys.*, 10: 1271–1302 (1969).

[26] C. Domb, Order-Disorder Statistics. I. *Proc. Roy. Soc. (London)*, 196: 36–50 (1949).

[27] C. Domb and N. W. Dalton, Crystal Statistics with Long-Range Forces I. The Equivalent Neighbour Model. *Proc. Roy. Soc. A*, 89: 859–871 (1966).

[28] C. Domb and R. B. Potts, Order-Disorder Statistics. IV. A Two-Dimensional Model with First and Second Interactions. *Proc. Roy. Soc. A*, 210: 125–141 (1951).

[29] J. L. Doob, Wiener's Work in Probability Theory. *Bulletin American Math. Soc.*, 72(number 1, part 2): 69–72 (1966).

[30] K. J. Dormer, *Fundamental Tissue Geometry for Biologists*. Cambridge University Press, Cambridge (1980).

[31] A. W. Drake, *Fundamentals of Applied Probability Theory*, p. 163–193. McGraw-Hill, New York (1967).

[32] W. Ebeling and F. Schweitzer, Self-Organization, Active Brownian Dynamics and Biological Applications. *Nova Acta Leopoldina NF*, 88: 169–188 (2003).

[33] A. Einstein, On the Motion of Small Particles Suspended in Liquids at Rest Required by the Molecular Kinetic Theory of Heat. *Ann. der Physik*, 19: 371–381 (1906).

[34] D. R. Ekserova, D. Exerowa and P. M. Kruglyakov, *Foam and Foam Films: Theory, Experiment, Application*. Studies in Interface Science, Elsevier, Amsterdam (1998).

[35] H. G. Evertz, M. Hasenbusch, M. Marcu and K. Pinn, The Solid-on-Solid Surface Width around the Roughening Transition. *Physica A*, 199: 31–39 (1993).

[36] F. Family and T. Vicsek, Dynamics Scaling of Growing Surfaces. In F. Family and T. Vicsek, editors, *Dynamics of Fractals Surfaces*, p. 73–77, World Scientific, Singapore (1991).

[37] C. Fan and F. Y. Wu, Ising Model with Second-Neighbor Interaction. I. Some Exact Results and an Approximate Solution. *Phys. Rev.*, 179: 560–570 (1969).

[38] P. J. Flory, *Statistical Mechanics of Chain Molecules*. Interscience, New York (1969).

[39] H. Flyvbjerg and C. Jeppesen, A Solvable Model for Corsening Soap Froths and Other Domain Boundary Networks in Two Dimensions. *Physica Scripta*, T38: 49–54 (1991).

[40] G. F. Fröbenius, *Über Matrizen aus nichtnegativen Elementen*, p. 456–477. S.-B. Preuss. Akad. Wiss., Berlin (1912).

[41] G. Gallavotti, *Statistical Mechanics: A Short Treatise*, p. 233–251. Springer, Berlin (1999).

[42] G. Gallavotti and A. Martin-Lof, Surface Tension in the Ising Model. *Commun. Math. Phys.*, 25: 87–126 (1972).

[43] R. W. Gibberd, Next-Nearest-Neighbor Ising Model. *J. Math. Phys.*, 10: 1026–1029 (1969).

[44] J. W. Gibbs, *Elementary Principles in Statistical Mechanics*. Ox Bow Press, Woodridge (1902, reprinted 1981).

[45] J. A. Glazier, *Dynamics of Cellular Patterns*. Ph.D. thesis, University of Chicago (1989).

[46] J. A. Glazier, M. P. Anderson and G. S. Grest, Coarsening in the Two Dimensional Soap Froth and the Large-Q Potts Model: A Detailed Comparison. *Phil. Mag. B*, 62: 615–645 (1990).

[47] J. A. Glazier and F. Graner, Simulation of the Differential Adhesion Driven Rearrangement of Biological Cells. *Phys. Rev. E*, 47: 2128–2154 (1993).

[48] F. Graner and J. A. Glazier, Simulation of Biological Cell Sorting Using a Two-Dimensional Extended Potts Model. *Phys. Rev. Lett.*, 69: 2013–2016 (1992).

[49] D. t. Haar, *Elements of Statistical Mechanics*, p. 251–295. Rinehart, New York (1954).

[50] H. Haken, *Synergetics. An Introduction. Nonequilibrium Phase Transitions and Self-Organization in Physics, Chemistry and Biology.*, p. 158–179. Springer-Verlag, Berlin (1983).

[51] M. Hasenbusch, Direct Monte Carlo Measurement of the Surface Tension in Ising Models. *J. Phys. I. (France)*, 3: 753–765 (1993).

[52] M. Hasenbusch and S. Meyer, Cluster-Update Acceleration of Interface Roughening in the 3D Ising Model. *Phys. Rev. Lett.*, 66: 530–533 (1991).

[53] M. Hasenbusch, S. Meyer and M. Putz, The Roughening Transition of the 3D Ising Interface: A Monte Carlo Study. *J. Stat. Phys.*, 85: 383–401 (1996).

[54] M. Hasenbusch and K. Pinn, Surface Tension, Surface Stiffness, and Surface Width of the 3-dimensional Ising Model on a Cubic Lattice. *Physica A*, 192: 342–374 (1993).

[55] M. Hasenbusch and K. Pinn, The Interface Tension of the 3-Dimensional Ising Model in the Scaling Region. *Physica A*, 245: 366–378 (1997).

[56] C. Herring, The Use of Classical Macroscopic Concepts in Surface-Energy Problems. In R. Gomer and C. S. Smith, editors, *Structure and Properties of Solid Surfaces*, p. 5–30, University of Chicago, Chicago (1952).

[57] S. Hildebrandt and A. Tromba, *The Parsimonious Universe: Shape and Form in The Natural World*. Copernicus, New York (1996).

[58] B. J. Hiley and G. S. Joyce, The Ising Model with Long Range Interactions. *Proc. Phys. Soc.*, 85: 493–507 (1965).

[59] T. Hill, *An Introduction to Statistical Thermodynamics*, p. 214–231. Addision-Wesley, Reading, Massachusetts (1960).

[60] K. Huang, *Introduction to Statistical Physics*, p. 260–270. Taylor and Francis, London (2001).

[61] C. Isenberg, *The Science of Soap Films and Soap Bubbles*, p. 1–24. Dover, New York (1992).

[62] E. Ising, The Theory of Ferromagnetism. *Z. Physik*, 31: 253–258 (1925).

[63] Y. Jiang, P. J. Swart, A. Saxena, M. Asipauskas and J. A. Glazier, Hysteresis and Avalanches in Two-Dimensional Foam Rheology Simulations. *Phys. Rev. E*, 59: 5819–5832 (1999).

[64] D. E. Knuth, *The Art of Computer Programming, Volume 2*, p. 1–179. Addison-Wesley, Reading, Massachusetts. (1997).

[65] S. Kobe, Ernst Ising - Physicist and Teacher. *J. Stat. Phys.*, 88: 991–995 (1997).

[66] H. A. Kramers and G. H. Wannier, Statistics of the Two-Dimensional Ferromagnet. Part I. *Phys. Rev.*, 60: 252–262 (1941).

[67] R. Kubo, H. Ichimura, T. Usui and N. Hashitsume, *Statistical Mechanics*, p. 77–78. North-Holland, Amsterdam (1965).

[68] R. Kubo, M. Toda and N. Hashitsume, *Statistical Physics II. Nonequilibrium Statistical Mechanics*, volume 31 of *Springer Series on Solid-State Science*. Springer-Verlag, Berlin (1991).

[69] P. K. Kundu, *Fluid Mechanics*. Academic Press, San Diego (1990).

[70] D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, Cambridge (2000).

[71] H. J. Leamy, G. H. Gilmer and K. A. Jackson, Statistical Thermodynamics of Clean Surfaces. In J. M. Blakely, editor, *Surface Physics of Materials: Volume 1*, p. 121–188, Academic Press, New York (1975).

[72] J. C. Lee, *Thermal Physics: Entropy and Free Energies*, p. 56–67. World Scientific, River Edge, New Jersey (2002).

[73] W. Lenz, Note on the Explanation of Magnetic Phenomena in Solid Bodies. *Z. Physik*, 21: 613–615 (1920).

[74] S. Ling, M. P. Anderson, G. S. Grest and J. A. Glazier, Comparison of Soap Froth and Simulation of Large-Q Potts Model. *Materials Science Forum*, 94-96: 39–51 (1992).

[75] S.-K. Ma, *Statistical Mechanics*, p. 485–487. World Scientific, Philadelphia (1985).

[76] B. B. Mandelbrot, Self-Affine Fractals and Fractal Dimension. In F. Family and T. Vicsek, editors, *Dynamics of Fractals Surfaces*, p. 11–20, World Scientific, Singapore (1991).

[77] D. A. McQuarrie, *Statistical Mechanics*. Harper & Row, New York (1975).

[78] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.*, 21: 1087–1092 (1953).

[79] K. H. Meyer and C. Ferri, Sur l'élasticité du caoutchouc. *Helv. Chim. Acta.*, 18: 570–589 (1935).

[80] F. Mohling, *Statistical Mechanics: Methods and Applications*, p. 545–550. John Wiley & Sons, New York (1982).

[81] K. K. Mon, Self-Similarity and Fractal Dimension of a Roughening Interface by Monte Carlo Simulations. *Phys. Rev. Lett.*, 57: 866–868 (1986).

[82] K. K. Mon and D. Jasnow, Monte Carlo Evaluations of Interfacial Tension and Universal Amplitude Ratios of the Three-Dimensional Ising Model. *Phys. Rev. A*, 31: 4008–4011 (1985).

[83] K. K. Mon and D. Jasnow, Surface Tension and Universality in the Three-Dimensional Ising Model. *J. Stat. Phys.*, 41: 273–280 (1985).

[84] K. K. Mon, D. P. Landau and D. Stauffer, Interface Roughening in the Three-Dimensional Ising Model. *Phys. Rev. B*, 42: 545–547 (1990).

[85] K. K. Mon, S. Wansleben, D. P. Landau and K. Binder, Anisotropic Surface Tension, Step Free Energy, and Interfacial Roughening in the Three-Dimensional Ising Model. *Phys. Rev. Lett.*, 60: 708–711 (1988).

[86] E. Muller-Hartmann and J. Zittartz, Interface Free Energy and Transition Temperature of the Square-Lattice Ising Antiferromagnet at Finite Magnetic Field. *Z. Physik B*, 27: 261–266 (1977).

[87] M. Nauenberg and B. Nienhuis, Critical Surface for Square Ising Spin Lattice. *Phys. Rev. Lett.*, 33: 944–946 (1974).

[88] D. Nelson, T. Piran and S. Weinberg, *Statistical Mechanics of Membranes and Surfaces*, p. 1–4. World Scientific, Singapore (1989).

[89] E. Nelson, *Dynamical Theories of Brownian Motion*. Princeton University Press, New Jersey (1967).

[90] M. E. J. Newman and G. T. Barkema, *Monte Carlo Methods in Statistical Physics*. Clarendon Press, Oxford (1999).

[91] C. S. Nolle, B. Koiller, N. Martys and M. O. Robbins, Morphology and Dynamics of Interfaces in Random Two-Dimensional Media. *Phys. Rev. Lett.*, 71: 2074–2077 (1993).

[92] C. S. Nolle, B. Koiller, N. Martys and M. O. Robbins, Effect of Quenched Disorder on Moving Interfaces in Two Dimensions. *Physica A*, 205: 342–354 (1994).

[93] J. Oitmaa, The Square-Lattice Ising Model with First and Second Neighbour Interactions. *J. Phys. A: Math. Gen.*, 14: 1159–1168 (1981).

[94] S. Ono and S. Kondo, Molecular Theory of Surface Tension in Liquids. In S. Flugge, editor, *Encyclopedia of Physics: Structure of Liquids*, volume 10, p. 134–280, Springer-Verlag, Berlin (1960).

[95] L. Onsager, Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition. *Phys. Rev.*, 65: 117–149 (1944).

[96] J. M. R. Parrondo, Brownian Motion and Gambling: From Ratchets to Paradoxical Games. *Contemporary Physics*, 45: 147–157 (2004).

[97] P. A. Pearce, Principles of Statistical Mechanics. In V. V. Bazhanov and C. J. Burden, editors, *Statistical Mechanics and Field Theory*, p. 1–25, World Scientific, Singapore (1995).

[98] H.-O. Peitgen, H. Jurgens and D. Saupe, *Chaos and Fractals: New Frontiers of Science*, p. 487–496. Springer-Verlag, New York (1992).

[99] H. Pelzer, Kinetic Theory of the Elasticity of Rubber. *Reports on Progress in Physics*, 6: 330–334 (1939).

[100] O. Perron, Theorie der Über Matrizen. *Math. Ann.*, 64: 248–263 (1907).

[101] R. B. Potts. Ph.D. thesis, Oxford University (1951).

[102] R. B. Potts, Some Generalized Order-Disorder Transformations. *Proc. Camb. Phil. Soc.*, 48: 106–109 (1952).

[103] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C, 2nd ed.*, p. 274–283. Cambridge University Press, Cambridge (1999).

[104] F. Reif, *Fundamentals of Statistical and Thermal Physics*, p. 560–582. McGraw-Hill, New York (1965).

[105] H. Risken, *The Fokker-Planck Equation. Methods of Solution and Applications*, p. 276–300. Springer-Verlag, Berlin (1989).

[106] C. Rottman and M. Wortis, Exact Equilibrium Crystal Shapes at Nonzero Temperature in Two Dimensions. *Phys. Rev. B*, 24: 6274–6277 (1981).

[107] B. Sapoval, M. Rosso and J. F. Gouyet, The Fractal Nature of a Diffusion Front and the Relation to Percolation. *J. Physique Lett.*, 46: L149–L156 (1985).

[108] E. H. Stanley, *Introduction to Phase Transitions and Critical Phenomena*, p. 1–38. Oxford University Press, New York (1971).

[109] H. E. Stanley, E. F. Taylor and P. A. Trunfio, *Fractals in Science. An Introductory Course*, p. 191–202. Springer-Verlag, Berlin (1994).

[110] R. H. Swendsen and S. Krinsky, Monte Carlo Renormalization Group and Ising Models with $n \geqslant 2$. *Phys. Rev. Lett.*, 43: 177–180 (1979).

[111] C. J. Thompson, One-Dimensional Models–Short Range Forces. In C. Domb and M. S. Green, editors, *Phase Transitions and Critical Phenomena*, volume 1, p. 192–193, Academic Press, London (1972).

[112] D. W. Thompson, *On Growth and Form*. MacMillan, New York (1945).

[113] L. R. G. Treloar, *The Physics of Rubber Elasticity*, p. 19–55. Clarendon Press, Oxford (1949).

[114] R. F. Voss, Random Fractals: Characterization and Measurement. In F. Family and T. Vicsek, editors, *Dynamics of Fractals Surfaces*, p. 40–50, World Scientific, Singapore (1991).

[115] G. H. Wannier, The Statistical Problem in Cooperative Phenomena. *Rev. Mod. Phys.*, 17: 50–60 (1945).

[116] N. Wax, *Selected Papers on Noise and Stochastic Processes*. Dover, New York (1954).

[117] D. Weaire and J. A. Glazier, Modelling Grain Growth and Soap Froth Coarsening: Past, Present and Future. *Materials Science Forum*, 27: 94–96 (1992).

[118] J. D. Weeks, G. H. Gilmer and H. J. Leamy, Structural Transition in the Ising-Model Interface. *Phys. Rev. Lett.*, 31: 549–551 (1973).

[119] B. Widom, Surface Tension of Fluids. In C. Domb and M. S. Green, editors, *Phase Transitions and Critical Phenomena*, volume 2, p. 79–88, Academic Press, London (1972).

[120] S. Wilde, Richard E. and Singh, *Statistical Mechanics: Fundamentals and Modern Applications*, p. 87–98. Wiley-Interscience, New York (1997).

[121] M. Zajac, *Modeling Convergent Extension by Way of Anisotropic Differential Adhesion*. Ph.D. thesis, University of Notre Dame (2002).