

Willoughby Seago

Theoretical Physics

Statistical Physics

January 19, 2022

COURSE NOTES

Statistical Physics

Willoughby Seago

January 19, 2022

These are my notes from the course statistical physics. I took this course as a part of the theoretical physics degree at the University of Edinburgh.

These notes were last updated at 19:29 on January 14, 2024. For notes on other topics see <https://github.com/WilloughbySeago/Uni-Notes>.

0	1	1	1	0
0	1	1	2	1
1	3	5	3	2
1	1	3	2	2
1	2	3	2	1

Chapters

	Page
Chapters	ii
Contents	iii
List of Figures	v
I Monte Carlo and the Ising Model	1
1 Monte Carlo	2
2 Dynamics	7
3 Why Does This Work?	14
II Cellular Automata	16
4 The Game of Life	17
5 SIRS Model	21
III Partial Differential Equations	23
6 Initial Value Problems	24
7 Cahn–Hilliard Equation	27
8 Boundary Value Problems	30
9 Maxwell’s Equations	33
Index	35

Contents

	Page
Chapters	ii
Contents	iii
List of Figures	v
I Monte Carlo and the Ising Model	1
1 Monte Carlo	2
1.1 Averages	2
1.2 Monte Carlo Integration	2
1.2.1 Trapezium Rule	3
1.3 Statistical Physics	4
1.4 The Ising Model	4
2 Dynamics	7
2.1 Markov Chains and the Metropolis Algorithm	7
2.2 Glauber Dynamics	8
2.2.1 Computing Energy	8
2.2.2 Results of Glauber Dynamics	8
2.3 Kawasaki Dynamics	10
2.4 Measurements	10
2.5 Errors	12
2.5.1 Autocorrelation and Equilibration Times	12
2.5.2 Estimating Errors	12
3 Why Does This Work?	14
3.1 General Argument	14
II Cellular Automata	16
4 The Game of Life	17
4.1 Cellular Automata	17

4.2	The Game of Life	17
4.2.1	Final States	17
5	SIRS Model	21
5.1	The SIRS Model	21
5.2	Phases	21
5.3	Immunity	22
III	Partial Differential Equations	23
6	Initial Value Problems	24
6.1	Types of PDEs	24
6.2	IVP: Diffusion in One Dimension	24
6.2.1	More Dimensions	26
6.2.2	Stability	26
7	Cahn–Hilliard Equation	27
7.1	The Equation	27
7.2	Derivation	27
7.3	Chemical Potential	28
7.4	Discretisation	29
8	Boundary Value Problems	30
8.1	Jacobi Method	30
8.2	Gauss–Seidel Algorithm	31
8.3	Successive Over Relaxation	31
9	Maxwell’s Equations	33
9.1	The Equations	33
9.2	Poisson’s Equation	33
9.2.1	Electric Field	33
9.2.2	Magnetic Filed	33
	Index	35

List of Figures

	Page
1.1 Random Ising Model Start.	5
1.2 The Ising model.	5
2.1 Various results of running Glauber dynamics for $N = 50^2 = 2500$ spins. . .	9
2.2 Average absolute magnetisation and energy for an infinite number of spins.	11
4.1 Game of life block state.	18
4.2 Game of life beehive state.	18
4.3 Game of life blinker state.	19
4.4 Game of life glider state	20

Part I

Monte Carlo and the Ising Model

One

Monte Carlo

Monte Carlo algorithms, named after the Monte Carlo casino, are stochastic methods for computing integrals. In a Monte Carlo computation we approximate the value of an integral by a series of averages.

1.1 Averages

Given a random variable, x_i , the average is defined as

$$\langle x \rangle \doteq \frac{1}{n} \sum_{i=1}^n x_i \quad (1.1.1)$$

where the sum is over all measurements of x . Similarly if f is a function of the random variable x_i then the average value of this function is

$$\langle f(x) \rangle \doteq \frac{1}{n} \sum_{i=1}^n x_i f(x_i) = \sum_{i=1}^n x_i f_i \quad (1.1.2)$$

where we define $f_i = f(x_i)$.

Often we instead want to work with weighted averages, say if certain outcomes are more likely than others. If outcome i has weight w_i then the weighted average of the random variable x_i is

$$\langle x \rangle \doteq \frac{\sum_{i=1}^n x_i w_i}{\sum_{i=1}^n w_i}. \quad (1.1.3)$$

The weighted average of f is similarly

$$\langle f(x) \rangle \doteq \frac{\sum_{i=1}^n f(x_i) w_i}{\sum_{i=1}^n w_i}. \quad (1.1.4)$$

1.2 Monte Carlo Integration

Suppose we wish to compute the d -dimensional integral

$$I = \int_V f(\mathbf{r}) d^d r. \quad (1.2.1)$$

First note that we can write this as an integral over all space by writing it as

$$I = \int_{\mathbb{R}^d} f(\mathbf{r}) 1_V(\mathbf{r}) d^d r \quad (1.2.2)$$

where 1_V is the indicator function for V , defined by $1_V(\mathbf{r}) = 1$ if $\mathbf{r} \in V$ and $1_V(\mathbf{r}) = 0$ if $\mathbf{r} \notin V$.

Using the fact that

$$V = \int_V d^d r \quad (1.2.3)$$

is the volume of the integration region we can write the integral as

$$I = \frac{\int_V f(\mathbf{r}) d^d r}{\int_V d^d r} V. \quad (1.2.4)$$

Notice now the similarity to the weighted average with all weights being 1, which is just the normal average, simply change integrals to sums. We therefore have

$$I = \frac{\int_V f(\mathbf{r}) d^d r}{\int_V d^d r} V \approx \frac{\sum_i f(\mathbf{r}_i)}{\sum_i 1} = \langle f(\mathbf{r}) \rangle. \quad (1.2.5)$$

The process of calculating an integral becomes the process of computing an average.

If we take n samples then the sum in the denominator is simply n and we have

$$I \approx \frac{\sum_{i=1}^n f_i}{n} V. \quad (1.2.6)$$

This is our Monte Carlo estimate of the integral.

The error on this measurement is associated with the standard error on the mean, since we are calculating a sample mean but really we want the population mean, that is $n \rightarrow \infty$. The standard error on the mean is

$$\frac{\sigma}{\sqrt{n}} \sim n^{-1/2} \quad (1.2.7)$$

where σ is the standard deviation. We clearly want n to be large in order to have a small error.

1.2.1 Trapezium Rule

An alternative method for approximating integrals is to split the integration region into sections, we then approximate f as linear over these sections. Define \bar{f}_i to be the average value of this linear approximation of f in the i th section. The integral is then approximated as the total volume of these sections that we have split the integral into. That is

$$I = \int_V f(\mathbf{r}) d^d r \approx \sum_i \text{volume of } i\text{th trapezium} = \sum_i \delta^d \bar{f}_i \quad (1.2.8)$$

where δ^d is the area of the base of the section, assuming we split the space evenly into a grid spaced δ apart, and \bar{f}_i is the average height of the linear approximation of the function in this region. For a one-dimensional integral these small sections are trapeziums, hence the name trapezium rule.

The error in our approximation of the function is $O(\delta^2)$ since we have approximated f as linear. The error then scales as $n\delta^d\delta^2$ where n is the number of regions.

The number of regions is the integration volume divided by the volume of each region, $n = V/\delta^d$, and so $\delta^d \sim 1/n$, which means $\delta \sim n^{-1/d}$. The error therefore scales as $nn^{-1}n^{-2/d} = n^{-2/d}$.

Comparing this to the error for the Monte Carlo method we see that if $d > 4$ the error on the Monte Carlo method actually decreases faster than the error for the more complex trapezium rule.

1.3 Statistical Physics

In statistical physics the Monte Carlo method can be used to compute thermodynamic averages. A system with a fixed number of particles, N , is a canonical ensemble and the physics is dictated in part by the partition function

$$Z := \sum_{i \in \{\text{states}\}} e^{-\beta E_i} \quad (1.3.1)$$

where $\beta := 1/(k_B T)$ with E_i being the energy of state i , k_B being Boltzmann's constant and T the temperature.

The average of some observable, A , is then given by summing over all states, i :

$$\langle A \rangle := \frac{\sum_i A_i e^{-\beta E_i}}{\sum_i e^{-\beta E_i}} = \frac{1}{Z} \sum_i A_i e^{-\beta E_i}. \quad (1.3.2)$$

That is $\langle A \rangle$ is the weighted average with weight $w_i = e^{-\beta E_i}$.

Writing this same value as

$$\langle A \rangle \approx \frac{\sum_{i=1}^n A_i e^{-\beta E_i}}{n} \frac{n}{\sum_{i=1}^n e^{-\beta E_i}}, \quad (1.3.3)$$

here we've introduced a factor of $n/n = 1$, as well as approximating the sums over all states with sums over n states. Notice that the first term is the Monte Carlo estimate of the average of A and the second is the reciprocal of the Monte Carlo estimation of the partition function.

The question is how we sample the states. The obvious answer of uniformly sampling the states turns out not to work, as we will see in the next section.

1.4 The Ising Model

The Ising model is a simple model that can be used to demonstrate several concepts in statistical mechanics. The **Ising model** consists of a lattice of equally spaced points and we imagine that at each point there is a particle which can have either spin up or spin down. We will work with a two-dimensional grid. We index the points from 1 to N , although we could also index each point by its row and column, which is often more useful in code. To each point i we assign a value of $+1$ or -1 depending on if the spin is up or down. We start with the spins chosen randomly and uniformly, so approximately half of the spins will be up and half down.

The energy of the system depends only on how the spins align. We assume the grid spacing is large enough that only interactions with the nearest neighbours have effect. The total energy of the system is then

$$E = -J \sum_{\langle i, j \rangle} S_i S_j \quad (1.4.1)$$

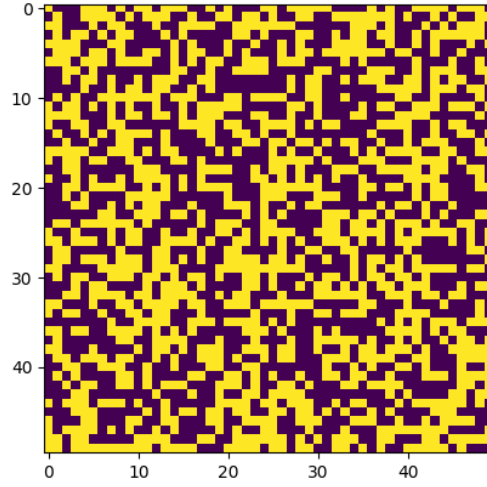


Figure 1.1: Initially the Ising model starts in a random state. Here opposite spins are represented by different colours. The axes are simply the row and column numbers, here $50^2 = 2500$ sites has been used.

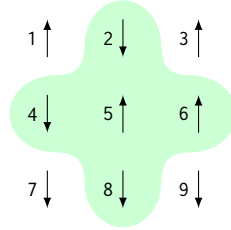


Figure 1.2: A 3×3 Ising model with spins represented by arrows. The shaded region is the nearest neighbours of spin 5, the central spin.

where J is a positive constant with dimensions of energy and $\langle i, j \rangle$ means that we sum over i and then j corresponds to the nearest neighbours.

Figure 1.2 shows a 3×3 Ising model. The nearest neighbours of the 5th point are points 2, 4, 6, and 8. The contribution to the sum when $i = 5$ is

$$-J \sum_{\langle 5, j \rangle} S_5 S_j = -J S_5 (S_2 + S_4 + S_6 + S_8) = -J 1(-1 - 1 + 1 - 1) = 2J \quad (1.4.2)$$

Notice that if we flip spin 5, that is $S_5 \rightarrow -S_5$, the change in energy is

$$\Delta E = -J(-S_5)(S_2 + S_4 + S_6 + S_8) - (-J)S_5(S_2 + S_4 + S_6 + S_8) \quad (1.4.3)$$

$$= 2J S_5 (S_2 + S_4 + S_6 + S_8) \quad (1.4.4)$$

$$= 2J 1(-1 - 1 + 1 - 1) \quad (1.4.5)$$

$$= -4J \quad (1.4.6)$$

This can be used to avoid having to calculate the energy of the entire model when we flip a single spin.

In a general Ising model of N sites, which in our example is 9, there are 2 possible values of the spin at each site and hence there are 2^N possible states. In our example

there are $2^9 = 512$ possible states. A more realistic size would be a 50×50 grid which has $N = 50^2 = 2500$, in which case $2^{2500} \approx 10^{752}$, which is huge.

The physics of the model is determined entirely by $e^{-\beta E}$. Since $E \propto J$ the only values that appear in the model, apart from the values of the spins, is the product $\beta J = J/(k_B T)$. We could use real values for these, for example J would be some factor of \hbar and $k_B \approx 10^{-23} \text{ J K}^{-1}$. However, these incredibly small numbers would cause huge floating point errors. Instead it is better to use the fact that these parameters only ever appear in ratio, J/k_B , and so we could just use $J/k_B \approx 10^{-12} \text{ s K}$. This is still tiny however and will lead to floating point issues. A better thing to do is to set $J = k_B = 1$, and then use T as the parameter measured in some units such that $J = k_B = 1$.

It is energetically favourable for all of the spins to align, either all up or all down. This means that there are two states which are much more likely than all other states, in particular they are much more likely than disordered states. On the other hand since 2^N is likely very large the probability of getting either of these two states randomly is 2^{-N} , which is approximately zero. Instead we should generate states with weight $e^{-\beta E_i}$. This is called **importance sampling**, where we generate states according to how likely they are, rather than uniform sampling. If we do this then we find that

$$\langle A \rangle = \frac{\sum_i A_i}{n}, \quad (1.4.7)$$

having already accounted for the factor of $e^{-\beta E_i}$ in our sampling.

There are two ways to do this, which will be the topic of the next chapter.

Two

Dynamics

2.1 Markov Chains and the Metropolis Algorithm

As discussed in the previous chapter random uniform sampling is too simplistic for most physics simulations. Things don't happen in isolation, the surroundings have an important effect. Our goal in this chapter will be to develop a way of simulating this that allows us to select states in a way that reflects the underlying physics. We will do so using Markov chains and a version of the Metropolis algorithm.

For our purposes a **Markov chain** is a way of generating a new state based only on the current state in a way that is physically reasonable. If we start with the state μ_1 , we can generate the state μ_2 , and then from this we generate μ_3 . Importantly the probability that μ_3 is a given state should depend only on μ_2 , and not on μ_1 . We say that the Markov chain is “memoryless” since it “can't remember μ_1 ” once it leaves this state.

After generating the states we will have a chain of states, $\mu_1 \rightarrow \mu_2 \rightarrow \mu_3 \rightarrow \dots \rightarrow \mu_n$. The process for generating the next state is what we call the dynamics of the system. We will consider two different dynamical algorithms later in this chapter.

The Metropolis algorithm is used in both dynamical algorithms which we will discuss. The general process for the metropolis algorithm is as follows. We start with the state μ_i and we want to generate a state μ_{i+1} in a physically reasonable way.

Code 2.1.1 — Metropolis Algorithm

```
1 generate a new state ,  $\mu_{i+1}$ 
2 compute the energy change ,  $\Delta E$ 
3 if  $\Delta E < 0$ :
4     accept the new state ,  $\mu_{i+1}$ 
5 else:
6     accept the new state with probability  $e^{-\beta\Delta E}$ 
7 repeat until a new state has been accepted
```

An alternative way to state the if clause is to accept the new state with probability $\min\{1, e^{-\beta\Delta E}\}$. This algorithm is very simple, we just have to specify how to generate μ_{i+1} .

2.2 Glauber Dynamics

Glauber dynamics is the simplest algorithm for generating the next state. We simply choose a random site and flip the spin there.

2.2.1 Computing Energy

When deciding if we accept the result of this flip we need to compute ΔE . The naive way to do this is to calculate the total energy after the flip and before the flip and compute the difference. However, when doing this there is a lot of extra computation occurring that is unnecessary. Suppose that we choose to flip the i th spin for some fixed i . Then

$$E_{\text{before}} = -J \sum_{\langle j,k \rangle} S_j S_k = -J S_i \sum_{\langle i,k \rangle} S_k - J \sum_{\substack{\langle j,k \rangle \\ j \neq i}} S_j S_k. \quad (2.2.1)$$

Here S_i is the value *before* the spin is flipped. Note that the sum over $\langle i, k \rangle$ keeps i fixed and sums k over the nearest neighbours of i . The energy after the spin is flipped is

$$E_{\text{after}} = J \sum_{\langle j,k \rangle} S_j S_k = J S_i \sum_{\langle i,k \rangle} S_k + J \sum_{\substack{\langle j,k \rangle \\ j \neq i}} S_j S_k. \quad (2.2.2)$$

Recall that the spins are represented by ± 1 so a spin flip is equivalent to a sign change. We can then compute the change in energy:

$$\Delta E = E_{\text{after}} - E_{\text{before}} = 2J S_i \sum_{\langle i,k \rangle} S_k. \quad (2.2.3)$$

This is much quicker to compute and better yet can be done in constant time, $O(1)$.

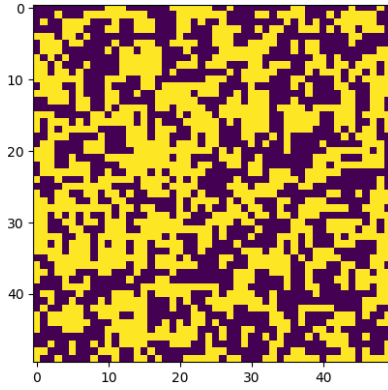
Sometimes we will need to compute the total energy of the system. The naive way to do this is to compute

$$E = -J \sum_{\langle i,j \rangle} S_i S_j \quad (2.2.4)$$

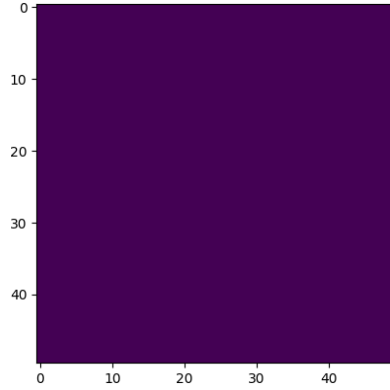
where i now ranges over all sites and j takes on the value of the nearest neighbours indices. A better way to do this relies on recognising that each term in the sum is symmetric in i and j . Therefore when computing, for example, the $S_5 S_6$ term we should also compute the $S_6 S_5$ term. For example, we may sum over the lattice but only account for the neighbours above and to the right, the neighbour to the left will be accounted for when we compute the term in the sum for the spin to the left and similarly with the site below. Both of these sums are $O(N)$ but by exploiting the symmetry like this the sum should be quicker to compute.

2.2.2 Results of Glauber Dynamics

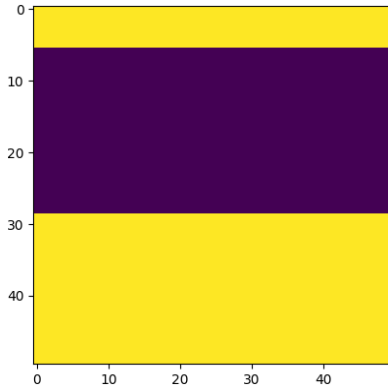
The results with Glauber dynamics depend on the temperature. If the temperature is very high then $e^{-\beta \Delta E} \approx 1$ and so we always swap the spin, resulting in essentially random swaps and no order even after a long time. If the temperature is very low then $e^{-\beta \Delta E} \approx 0$ and we only swap the spin if the energy decreases. This means that we end up with alignment of spins increasing so we will eventually achieve a state where all of the spins are aligned. At low temperatures we can get stuck in a



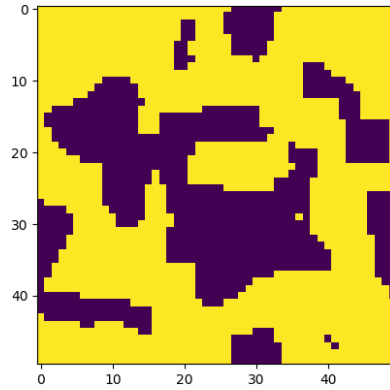
(a) The result of running Glauber dynamics at a high temperature. Here a “high temperature” is $T = 5$ in units where $k_B = J = 1$. Notice that the pattern of spins appears fairly random but there are slightly more yellow sites than purple.



(b) The result of running Glauber dynamics at a low temperature. Here a “low temperature” is $T = 0.01$ in units where $k_B = J = 1$.



(c) A metastable state in the Glauber dynamics. The boundary being straight lines minimises the number of neighbouring unaligned spins. This state will remain for a long time but eventually all spins will align.



(d) The result of 10 sweeps with low temperature Glauber dynamics. One sweep meaning attempting to change N spins, where N is the total number of spins (here $N = 2500$). Even after a relatively short time the spins separate into domains.

Figure 2.1: Various results of running Glauber dynamics for $N = 50^2 = 2500$ spins.

metastable state for a long time where we have two regions with opposite spins and the boundary contributes very little to the total energy and the chance of picking a state on the boundary is also low, resulting in a very slow change towards all the spins being aligned.

2.3 Kawasaki Dynamics

In Glauber dynamics we often end up with all spins aligned. This isn't always reasonable. Sometimes the net **magnetisation** of a system,

$$M = \sum_{i=1}^N S_i, \quad (2.3.1)$$

is fixed. One way to model this sort of system is with Kawasaki dynamics.

In Kawasaki dynamics we generate the next state by choosing two random spins and then swapping them. We now consider only what happens if the chosen spins are different since if their the same nothing changes. If these spins *aren't* nearest neighbours then the change in energy is simply

$$\Delta E = 2JS_i \sum_{\langle i,k \rangle} S_k + 2JS_j \sum_{\langle j,l \rangle} S_l \quad (2.3.2)$$

where i and j are the indices of the spins we swap. The logic for this is that this is essentially the same as repeating the Glauber dynamics process twice with these two spins chosen.

If the spins are nearest neighbours then we have to be more careful calculating the energy change but it can still be done in an $O(1)$ way by carefully considering which spins are actually effected by the swap.

Since we are only swapping spins the net magnetisation remains the same. This means that the results tend to have spins group into domains where they are all aligned since this minimises the energy. Of course if the temperature is high enough then entropy dominates and we get essentially random states indistinguishable from Glauber dynamics.

2.4 Measurements

Once we have a simulation running one of these dynamical algorithms we want to make measurements of the system. One of our goals will be to find the **critical temperature**, T_c , which is the temperature at which the system goes from ordered to random. This is the temperature which distinguishes the “low temperature” cases from the “high temperature” cases.

Two observables which we have already mentioned are the energy,

$$E = -J \sum_{\langle i,j \rangle} S_i S_j, \quad (2.4.1)$$

and magnetisation,

$$M = \sum_{i=1}^N S_i. \quad (2.4.2)$$

We expect that the energy will be much lower for an ordered phase and much higher for a disordered phase since spins align more in the ordered phase. For Glauber dynamics the magnetisation will tend to $\pm N$ for low temperatures as all spins align. For this reason we consider the absolute value of the magnetisation, since there is

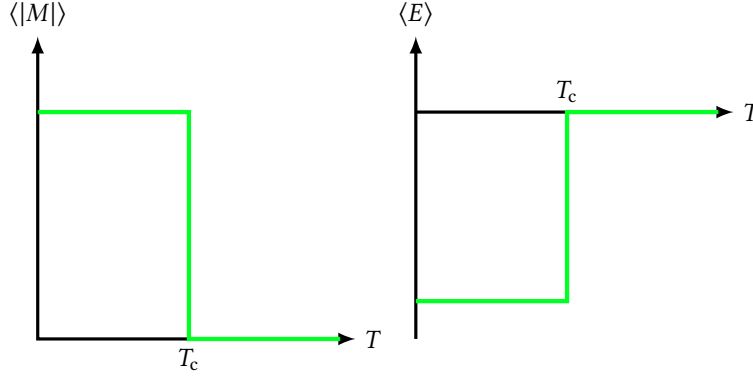


Figure 2.2: Plots of the average absolute magnetisation and energy for an infinite number of spins. Notice the sudden change at the critical temperature, T_c .

nor real logic to assigning one spin as +1 and the other as -1 so the sign cannot have physical meaning. For Kawasaki dynamics M is constant so not that useful.

The average energy is given by the Monte Carlo approximation as

$$\langle E \rangle = \frac{1}{n} \sum_{i=1}^n E(\mu_i) \quad (2.4.3)$$

where $\{\mu_i\}$ are all states visited and $E(\mu_i)$ is the energy in state μ_i , and n is the total number of states visited, we take n to be large for this to be a good approximation. Similarly the average absolute magnetisation is

$$\langle |M| \rangle = \frac{1}{n} \sum_{i=1}^n |M(\mu_i)| \quad (2.4.4)$$

where $M(\mu_i)$ is the magnetisation in state μ_i .

Consider what happens in Glauber dynamics. At a low temperature we expect that the spins align and so $\langle E \rangle = -2JN$, since either S_i and S_j are both +1 or both -1, either way their product is +1 and so we eventually achieve constant (up to random fluctuations of a few spins) energy $-2JN$ and over enough time the states that occurred before achieving this have negligible impact on the average. At high temperatures the spins are essentially random so S_i and S_j are just as likely to have the same sign as the opposite sign and hence in the energy sum most terms cancel out and we get $\langle E \rangle = 0$, again, up to random fluctuations.

The case of magnetisation is similar with $\langle |M| \rangle = N$ at low temperatures and $\langle |M| \rangle = 0$ at high temperatures.

How can we use this to find the critical temperature? First we consider what happens when the number of spins, N , tends to infinity. In this case we get two very clear regions of low and high temperature. This is shown in Figure 2.2. Notice the discontinuity at the critical temperature, T_c . For small N however this discontinuity can't be seen.

Instead of the discontinuity we use the fact that fluctuations tend to be very large near the critical temperature as the system moves back and forth between the two phases. One measure of the fluctuations is the standard deviation,

$$\sigma_E \coloneqq \langle E^2 \rangle - \langle E \rangle^2. \quad (2.4.5)$$

However, by dividing by $Nk_B T^2$ we can associate this with the specific heat capacity, C , per spin:

$$\frac{\langle E^2 \rangle - \langle E \rangle^2}{Nk_B T^2} = \frac{C}{N}. \quad (2.4.6)$$

Similarly for the magnetisation we can relate the standard deviation to the susceptibility, χ , per spin:

$$\frac{\langle M^2 \rangle - \langle M \rangle^2}{Nk_B T} = \frac{\chi}{N}. \quad (2.4.7)$$

Plotting these quantities will give a peak at T_c . The height of the peak will vary with $\ln N$ whereas away from the peak the values will be approximately constant with respect to N .

2.5 Errors

2.5.1 Autocorrelation and Equilibration Times

In order to avoid dependence on the initial conditions we should wait for the system to equilibrate before taking measurements. The easiest way to do this is to set some predefined wait time before starting measurements, rather than trying to assess whether the system is in equilibrium. The required time can be decreased when we are doing multiple simulation runs with slightly varied parameters by starting from the final state of the previous run. If we do this then an equilibration time of 100 sweeps is good enough for our purposes.

If we were to make a measurement every sweep then the results would be highly correlated between neighbouring measurements. Instead we should wait some time, τ_a , known as the **autocorrelation time**. This can be defined by

$$\frac{\chi(t - t')}{\chi(0)} = \exp \left[-\frac{t - t'}{\tau_a} \right], \quad (2.5.1)$$

where χ is the generalised susceptibility which we are measuring. The autocorrelation time is greatest near the point at which we get a phase transition, so we can compute a value at this point and use it for all simulation runs. It can be shown that an autocorrelation time of 10 sweeps between measurements is good enough for our purposes.

2.5.2 Estimating Errors

The error on zeroth order quantities like $\langle E \rangle$ or $\langle |M| \rangle$ is easily computed as the standard error on the mean, which is the standard deviation divided by the mean:

$$\delta E = \frac{\sigma_E}{\langle E \rangle} = \sqrt{\frac{\langle E^2 \rangle - \langle E \rangle^2}{n}}, \quad (2.5.2)$$

where n is the number of measurements of E that we made.

This doesn't work so well for first order quantities, like

$$c = \frac{C}{N} = \frac{\langle E^2 \rangle - \langle E \rangle^2}{Nk_B T^2} = \frac{1}{N} \frac{\partial \langle E \rangle}{\partial T}. \quad (2.5.3)$$

In order to get a good approximation of the error we would have to run the simulation lots of times and get lots of values of $\langle E \rangle$ and $\langle E^2 \rangle$ and then take averages of these. There are other methods better suited to this case using resampling.

The **Bootstrap algorithm** for estimating the error in $c = C/N$ is as follows:

- Start with n uncorrelated¹ measurements, e.g. for $n = 5$ we have E_1, \dots, E_5 .
- Resample, choose n measurements randomly from the set of all measurements, with replacement, e.g. we might choose $\{E_3, E_5, E_2, E_5, E_3\}$.
- Use the resampled set to compute $c_i = C/N$.
- Repeat k times to generate $\{c_1, \dots, c_k\}$.
- Estimate the error on c as $\sigma_c = \sqrt{\langle c^2 \rangle - \langle c \rangle^2}$ with $\langle c \rangle = \sum_{i=1}^k c_i/k$, and similar for $\langle c^2 \rangle$.

¹for correlated measurements we need a few extra things, but we assume uncorrelated measurements since we wait for longer than the autocorrelation time between measurements.

In practice we will take $k \approx 100$ as a reasonable number of repeats for the bootstrap algorithm.

The **jackknife algorithm** for estimating the error in $c = C/N$ is as follows:


- Start with n uncorrelated measurements, e.g. for $n = 5$ we have E_1, \dots, E_5 .
- Remove the first measurement for resampling, e.g. E_2, E_3, E_4, E_5 .
- Compute $c_1 = C/N$ using the data from the last step.
- Put the first measurement back and remove the second measurement, e.g. E_1, E_3, E_4, E_5 .
- Compute $c_2 = C/N$ using the data from the last step.
- \vdots
- Put the $(i-1)$ th measurement back and remove the i th, e.g. $E_1, \dots, E_{i-1}, E_{i+1}, \dots, E_5$.
- Compute $c_i = C/N$ using the data from the last step.
- \vdots
- Put the $(n-1)$ th measurement back and remove the n th, e.g. E_1, E_2, E_3, E_4 .
- Compute $c_n = C/N$ using the data from the last step.
- Estimate the error as $\sigma_c = \sqrt{\sum_{i=1}^k (c_i - \langle c \rangle)^2} = \sqrt{k(\langle c^2 \rangle - \langle c \rangle^2)}$.

Both the bootstrap and jackknife algorithms are reasonable for computing errors on first order measurements. However, the bootstrap algorithm has complexity $O(k)$ and the jackknife $O(n)$, and we will typically have $k = 100$ and $n \sim 10^3$, so bootstrap tends to be faster. Note that if there is still some correlation between measurements then both of these methods will under estimate the errors.

Three

Why Does This Work?

3.1 General Argument

 The notion of detailed balance, key to the following discussion, is covered in the statistical physics notes.

Suppose we have completed a Markov chain simulation with the metropolis algorithm. This gives a chain of states, μ_i . Consider some particular state, μ . The probability that the system is in state μ at time t is $p_\mu(t)$. We want to know how fast this changes. To do so we need to consider the flux of states into μ from some other state ν , and the flux of states out from μ into some other state ν , and then sum over all states ν . To do this we need the transition probability for transition from ν to μ , $P(\nu \rightarrow \mu)$. Putting this together we get

$$\frac{\partial}{\partial t} p_\mu(t) = \sum_{\nu \neq \mu} p_\nu(t) P(\nu \rightarrow \mu) - \sum_{\nu \neq \mu} p_\mu(t) P(\mu \rightarrow \nu). \quad (3.1.1)$$

The first term represents the flux of states into μ and the second the flux of states out of μ .

We want to find p_μ in steady state, that is when $\partial p_\mu(t)/\partial t = 0$. We expect that

$$p_\mu \propto \exp[-\beta E_\mu] \quad (3.1.2)$$

in steady state where $\beta = 1/(k_B T)$. This is just the Boltzmann distribution for a canonical ensemble.

We define $P(\mu \rightarrow \mu) = 0$ to allow us to write the equation above as a sum over all states. This means that in steady state we have

$$\frac{\partial p_\mu}{\partial t} = \sum_{\nu} [p_\nu P(\nu \rightarrow \mu) - p_\mu P(\mu \rightarrow \nu)] = 0. \quad (3.1.3)$$

It is not easy to solve this in general, but one solution is to have each term individually vanish. This is called **detailed balance**. That is we have

$$p_\nu P(\nu \rightarrow \mu) = p_\mu P(\mu \rightarrow \nu) \quad (3.1.4)$$

for all states μ and ν . Taking $\nu \neq \mu$ this gives

$$\frac{p_\mu}{p_\nu} = \frac{P(\nu \rightarrow \mu)}{P(\mu \rightarrow \nu)}. \quad (3.1.5)$$

Given our ansatz of a Boltzmann distribution in steady state we look for

$$\frac{p_\mu}{p_\nu} = \exp[-\beta(E_\mu - E_\nu)]. \quad (3.1.6)$$

In Glauber dynamics we have

$$P(\mu \rightarrow \nu) = g(\mu \rightarrow \nu)A(\mu \rightarrow \nu) \quad (3.1.7)$$

where g is the attempt rate, that is how often we select states μ and ν , and $A(\mu \rightarrow \nu)$ is the acceptance rate, that is, when we have selected states μ and ν what is the probability that we swap them. Recall that in Glauber dynamics one step involves flipping a single randomly chosen spin, and so

$$g(\mu \rightarrow \nu) = \begin{cases} 1/N & \text{if } \mu \text{ and } \nu \text{ differ by a single spin,} \\ 0 & \text{otherwise.} \end{cases} \quad (3.1.8)$$

Notice importantly that this means $g(\mu \rightarrow \nu) = g(\nu \rightarrow \mu)$, and so this factor cancels in the ratio. We therefore want to choose $A(\mu \rightarrow \nu)$ to give the desired ratio. The easiest way to do this is to take

$$A(\mu \rightarrow \nu) = \min\{1, \exp[-\beta(E_\nu - E_\mu)]\}. \quad (3.1.9)$$

We then have

$$\frac{p_\mu}{p_\nu} = \frac{A(\nu \rightarrow \mu)}{A(\mu \rightarrow \nu)} \quad (3.1.10)$$

$$= \begin{cases} \frac{\exp[-\beta(E_\mu - E_\nu)]}{1} & E_\mu > E_\nu \\ \frac{1}{\exp[-\beta(E_\nu - E_\mu)]} & E_\mu < E_\nu \end{cases} \quad (3.1.11)$$

$$= \exp[-\beta(E_\mu - E_\nu)], \quad (3.1.12)$$

which is the desired result.

We can make a similar argument to justify Kawasaki dynamics. The important thing is that, after a sufficient amount of time, both algorithms will take the system into an equilibrium state by imposing detailed balance. We are then free to take measurements and it will be as if we were drawing from a Boltzmann distribution.

Part II

Cellular Automata

Four

The Game of Life

4.1 Cellular Automata

Cellular automata consist of a grid and a list of update rules for the grid depending on its current condition. There are many types of cellular automata. For example, the Ising model as discussed in the previous section is a cellular automata which is stochastic, meaning there is an element of randomness, and sequential, meaning we update one site at a time. We will now discuss a different cellular automata called the game of life which is deterministic, meaning the result depends only on the initial condition, and parallel, meaning we update all sites at the same time.

4.2 The Game of Life

The **game of life** is a very simple cellular automata. To every cell we assign one of two states, alive or dead. We then update all cells at once based on the following algorithm.

- An alive cell with fewer than 2 neighbours becomes a dead cell.
- An alive cell with more than 3 neighbours becomes a dead cell.
- A dead cell with exactly three alive neighbours becomes alive.
- An alive cell with 2 or 3 alive neighbours remains alive.

Note that in the game of life neighbours includes diagonals, so each cell has 8 neighbours.

Despite the simple rules the game of life can develop very complex systems.

4.2.1 Final States

An **absorbing state** is one in which the system stops changing. An ergodic system, one in which all possible states are visited, cannot have absorbing states, but the game of life is not ergodic so this is not of concern. One example of an absorbing state is the block. This consists of a 2×2 block of alive cells surrounded by dead cells. This is shown in [Figure 4.1](#). Another slightly more complicated absorbing state is the beehive, shown in [Figure 4.2](#).

Another option is an oscillating state. These are states which change, but follow a fixed pattern which eventually repeats. The simplest oscillator is the blinker, which consists of a 3×1 block of alive states, which turns into a 1×3 block of

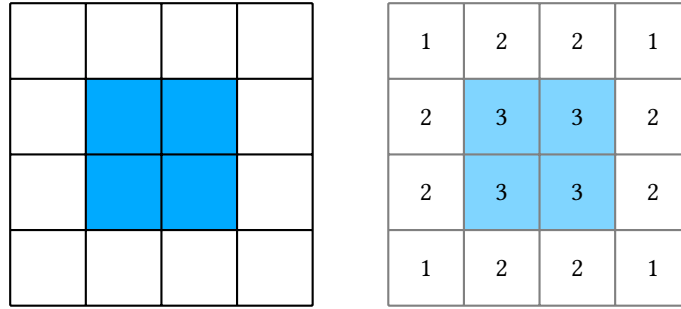


Figure 4.1: The block state is an absorbing state of the game of life. All alive cells have 3 neighbours and no dead cells have 3 neighbours, so nothing changes.

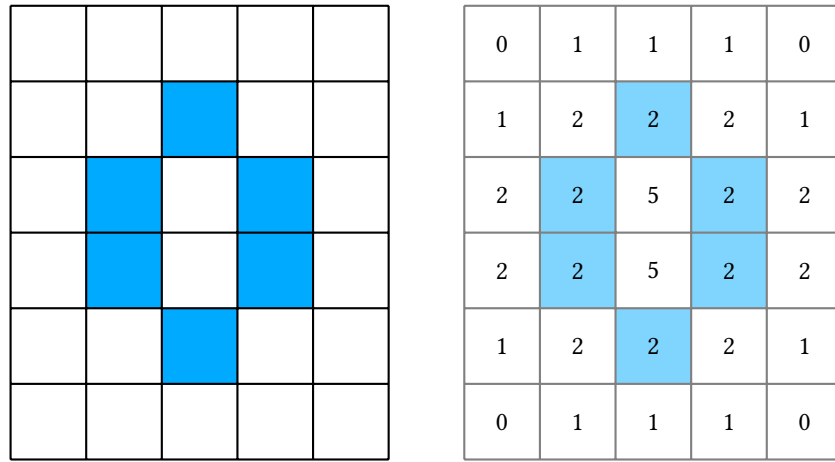


Figure 4.2: The beehive state is an absorbing state of the game of life. All alive cells have 2 neighbours and no dead cells have 3 neighbours, so nothing changes.

alive states, which turns back into the original 3×1 block of alive states. This is shown in [Figure 4.3](#).

The period of an oscillating state is how many states it visits before the pattern starts repeating. The blinker is a period 2 oscillator, since it has two states. Diagrammatically we can write a period two oscillator as

$$A \begin{array}{c} \curvearrowright \\ \curvearrowleft \end{array} B. \quad (4.2.1)$$

A period three oscillator can be represented as

$$\begin{array}{c} A \\ \curvearrowright \\ C \end{array} \begin{array}{c} \curvearrowleft \\ B \end{array}. \quad (4.2.2)$$

Notice that the existence of period three oscillators breaks time reversal symmetry since

$$\begin{array}{c} A \\ \curvearrowright \\ C \end{array} \begin{array}{c} \curvearrowleft \\ B \end{array} \neq \begin{array}{c} A \\ \curvearrowright \\ C \end{array} \begin{array}{c} \curvearrowleft \\ B \end{array}. \quad (4.2.3)$$

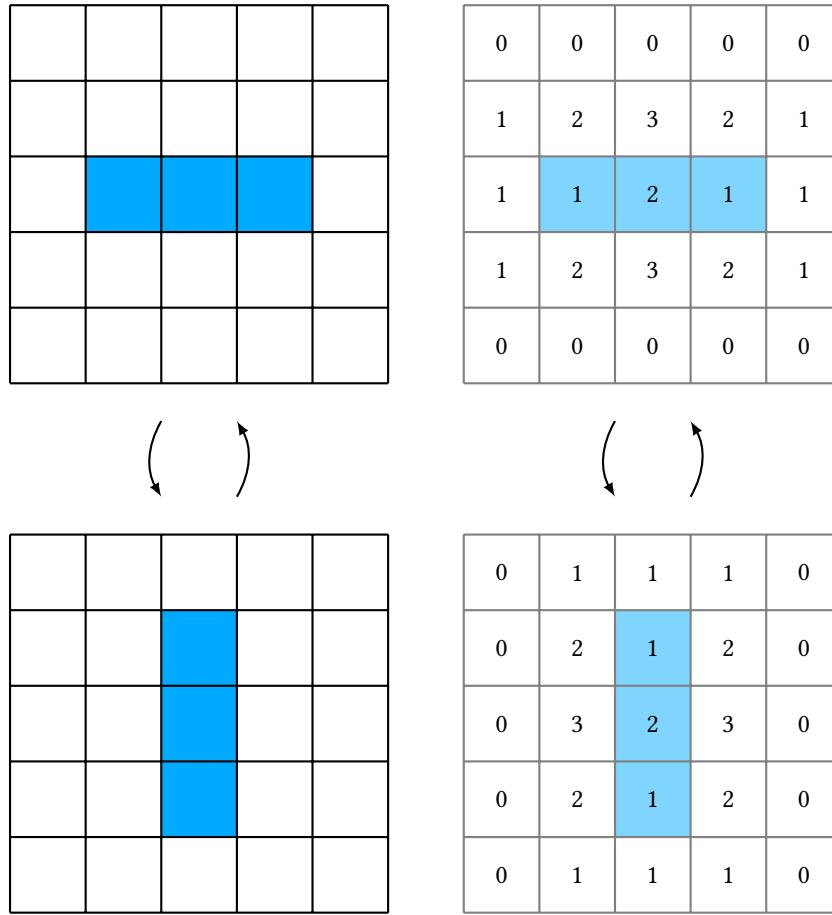


Figure 4.3: The blinker state is an oscillating state of the game of life. Two alive cells only have one alive neighbour, so become dead cells, and two dead cells have three alive neighbours and so become alive. The same then happens but rotated by 90° . This state then returns to the original state.

Another class of final states is travelling states. These are somewhat like oscillators, in that the same arrangement of alive cells occurs over and over, but the position of these cells changes. The simplest travelling state is the glider. This is shown in Figure 4.4. The glider has period four, meaning that there are four distinct arrangements of cells, but as a travelling state the game doesn't return to exactly the same state, rather a translated version.

One question we may want to ask is how fast a travelling state is. To do this we calculate the centre of mass of the state. This can be done using

$$\mathbf{r}_{\text{cm}} = \frac{1}{n} \sum_{i=2}^n \mathbf{r}_i a_i, \quad (4.2.4)$$

where n is the number of cells, \mathbf{r}_i is the position of the i th cell, and a_i is 0 if cell i is dead and 1 if cell i is alive. We can then work out the velocity of the centre of mass by working out the displacement after t steps and then dividing by t .

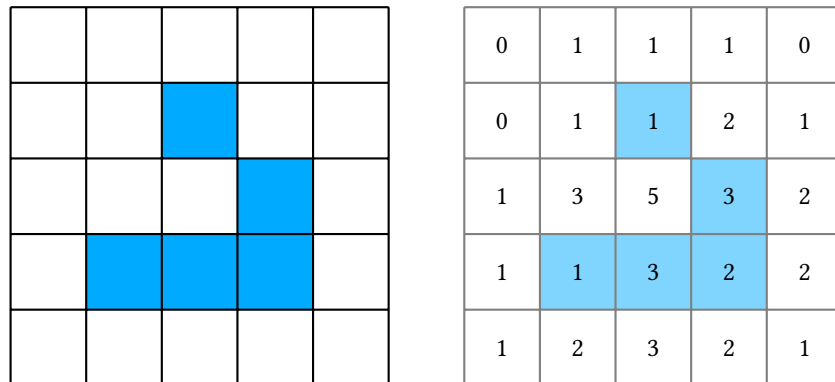


Figure 4.4: The glider state is a travelling state of the game of life. The current configuration will move diagonally down and right.

Another sensible question is how long it takes a given scenario to reach a final state, by which we mean a state in which nothing is really going to change, including states with non-interacting oscillators and travelling states. To do this we can simply count the number of alive cells over some period and if it doesn't change for a sufficiently large amount of time we say we've reached a final state. It is possible that a given initial state may not reach a final state by this definition, or may take a very long time, but most initial states that will result in a final state will do so in less than 5000 steps.

Five

SIRS Model

5.1 The SIRS Model

The **SIRS model** is a basic stochastic sequential cellular automata for modelling epidemics. There are three states, S, I, and R. These stand for susceptible, infected, and recovered. Each step a site is picked and the following rules applied:

- If the site is in state S and it has at least one infected neighbour it will change to state I with probability p_1 . This represents the infected neighbour infecting the susceptible site.
- If the site is in state I it will change to state R with probability p_2 . This represents the infected site recovering from the illness.
- If the site is in state R it will change to state S with probability p_3 . This represents the recovered site losing immunity.

Note that in the SIRS model we take only the cells to the left, right, above, and below as neighbours, not the diagonals.

It should be noted that the SIRS model is a slight modification of the SIR model, which doesn't allow for recovered states becoming susceptible again, essentially states in the SIR model have permanent immunity after recovering.

Note that once a rule has been applied we finish the step, otherwise it would be possible to have a single site go from S to I to R and back to S in one step.

5.2 Phases

There are three phases, which we characterise with the order parameter I/N , where I is the number of sites in state I and N is the number of steps completed. The first state is the absorbing state, where I/N eventually becomes 0. This represents no infected people, so only susceptible and recovered people. Eventually this will become only susceptible people, but this last bit isn't of interest, and we stop once the illness vanishes.

The second state is dynamic equilibrium. In this I/N is approximately constant with only small fluctuations. This represents something like a mild flu.

The final state is somewhere between these two states and is a wave state. In this I/N fluctuates between almost zero and most people being infected. We can identify this by considering the variance, $(\langle I^2 \rangle - \langle I \rangle^2)/N$, which will be large for a wave phase.

We can study these different phases by fixing p_3 , for example, and then considering a range of values for p_1 and p_2 and producing a heat map of $\langle I \rangle / N$. We can also fix, for example, p_2 , and consider just a range of values of p_1 if we want higher resolution measurements.

5.3 Immunity

One modification we can make is to introduce a fourth state, I_m , representing permanent immunity, say a genetic immunity that only some people have. Sites in this state cannot become another state, and likewise sites that don't start in this state cannot move into this state. We can then study the behaviour of $\langle I \rangle / N$ as a function of f_{I_m} , which is the fraction of people who start in state I_m . We expect that $\langle I \rangle / N$ decreases as f_{I_m} increases. At some point $\langle I \rangle / N$ will be zero for sufficiently large f_{I_m} . This represents herd immunity.

Part III

Partial Differential Equations

Six

Initial Value Problems

6.1 Types of PDEs

Broadly there are two types of partial differential equations (PDEs). These are initial value problems (IVPs) in which we have some system evolving in time and we know its state at time $t = 0$, and boundary value problems (BVPs) in which we have some system evolving in space and we know its value on a boundary. An example of an initial value problem is the diffusion equation,

$$\frac{\partial c}{\partial t} = D \nabla^2 c, \quad (6.1.1)$$

where $c(\mathbf{r}, t)$ is the concentration of some quantity at position \mathbf{r} and time t and D is a constant. In order to solve this we need to know $c(\mathbf{r}, 0)$ for all \mathbf{r} . An example of a boundary value problem is Poisson's equation,

$$\nabla^2 \phi = -\frac{\rho}{\epsilon}, \quad (6.1.2)$$

where $\phi(\mathbf{r})$ is the electric potential due to some fixed charge density, $\rho(\mathbf{r})$. In order to solve a boundary value problem we need to know boundary conditions. These come in two forms. **Dirichlet boundary conditions** involve knowing the value of ϕ along some boundary, and **von Neumann boundary conditions** involve knowing the value of $\hat{\mathbf{n}} \cdot \nabla \phi$ along some boundary where $\hat{\mathbf{n}}$ is a unit vector normal to the boundary.

In this chapter we are interested in initial value problems, which are typically easier to solve.

6.2 IVP: Diffusion in One Dimension

Consider the one-dimensional diffusion equation

$$\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2}, \quad (6.2.1)$$

where $c(x, t)$ is the concentration. We will solve this with a **finite difference** method, which entails approximating derivatives, defined by the limit

$$\frac{\partial c}{\partial t} := \lim_{\delta t \rightarrow 0} \frac{c(x, t + \delta t) - c(x, t)}{\delta t}. \quad (6.2.2)$$

by replacing δt with some small but non-zero value. To do so we discretise space into points separated by a distance δx . We then label these points by i . We also discretise time into steps of size δt , so that $t = n\delta t$, and we can use n to label times. We can then approximate the derivative at $x = i\delta x$ and time $t = n\delta t$ by

$$\frac{\partial c}{\partial t} \approx \frac{c[i; n+1] - c[i; n]}{\delta t}. \quad (6.2.3)$$

This is called the **forward difference** approximation, since it involves using the current time, n , and the next time, $n+1$. We can also compute the **backward difference** approximation:

$$\frac{\partial c}{\partial t} \approx \frac{c[i; n] - c[i; n-1]}{\delta t}. \quad (6.2.4)$$

These should give similar results for sufficiently small values of δt , and it is only when computing higher order derivatives, where errors compound, that we have to be careful about which we use.

Now consider $c(x + \delta x, t)$. We can Taylor expand this quantity, giving

$$c(x + \delta x, t) \approx c(x, t) + \delta x \left(\frac{\partial c}{\partial x} \right)_{(x,t)} + \frac{\delta x^2}{2} \left(\frac{\partial^2 c}{\partial x^2} \right)_{(x,t)} + O(\delta x^3). \quad (6.2.5)$$

Similarly we have

$$c(x - \delta x, t) \approx c(x, t) - \delta x \left(\frac{\partial c}{\partial x} \right)_{(x,t)} + \frac{\delta x^2}{2} \left(\frac{\partial^2 c}{\partial x^2} \right)_{(x,t)} + O(\delta x^3). \quad (6.2.6)$$

Adding these together and noting that the $O(\delta x^3)$ terms cancel like the $O(\delta x)$ terms we have

$$c(x + \delta x, t) + c(x - \delta x, t) \approx 2c(x, t) + \delta x^2 \left(\frac{\partial^2 c}{\partial x^2} \right)_{(x,t)} + O(\delta x^4). \quad (6.2.7)$$

Rearranging this we have

$$\left(\frac{\partial^2 c}{\partial x^2} \right)_{(x,t)} \approx \frac{1}{\delta x^2} [c(x - \delta x, t) + c(x + \delta x, t) - 2c(x, t)]. \quad (6.2.8)$$

This is called the **centred difference** approximation of the second derivative, since it uses both the previous, $x - \delta x$, and next, $x + \delta x$, positions. We then get the finite difference approximation

$$\frac{\partial^2 c}{\partial x^2} \approx \frac{1}{\delta x^2} (c[i-1; n] + c[i+1; n] - 2c[i; n]). \quad (6.2.9)$$

Combining these two approximations for the derivative the diffusion equation becomes

$$\frac{1}{\delta t} (c[i; n+1] - c[i; n]) = \frac{D}{\delta x^2} (c[i-1; n] + c[i+1; n] - 2c[i; n]). \quad (6.2.10)$$

Rearranging this for $c[i; n+1]$ we get

$$c[i; n+1] = c[i; n] + \frac{D \delta t}{\delta x^2} (c[i+1; n] + c[i-1; n] - 2c[i; n]). \quad (6.2.11)$$

This gives us a way to compute the value of c at the next time step, $n+1$, using its value at the current time, n , and just a few neighbouring values in space. We can easily write some code to iterate this to find the value at some later time from the given initial state.

6.2.1 More Dimensions

The only difference between the one-dimensional and N -dimensional diffusion equations is that the second order spatial derivative becomes the Laplacian. Fortunately our method generalises easily to N -dimensions. Exactly the same process of Taylor expanding works, for example in three dimensions we just have to consider $c(x, y + \delta x, z, t)$, $c(x, y - \delta x, z, t)$, $c(x, y, z + \delta x, t)$, and $c(x, y, z - \delta x, t)$ as well. Note that we assume even grid spacing in space, so all the differences are δx . Doing so we notice that the result is

$$\begin{aligned} c[i_1, \dots, i_N; n+1] = & c[i_1, \dots, i_N; n] + \frac{D \delta t}{\delta x^2} (c[i_1 - 1, i_2, \dots, i_N; n] \\ & + c[i_1 + 1, i_2, \dots, i_N; n] + \dots + c[i_1, \dots, i_{N-1}, i_N - 1; n] \\ & + c[i_1, \dots, i_{N-1}, i_N + 1; n] - 2Nc[i_1, \dots, i_N; n]). \end{aligned} \quad (6.2.12)$$

In three dimensions

$$\begin{aligned} c[i, j, k; n] = & c[i, j, k; n] + \frac{D \delta t}{\delta x^2} (c[i - 1, j, k; n] + c[i + 1, j, k; n] \\ & + c[i, j - 1, k; n] + c[i, j + 1, k; n] + c[i, j, k - 1; n] \\ & + c[i, j, k + 1; n] - 6c[i, j, k; n]). \end{aligned} \quad (6.2.13)$$

6.2.2 Stability

In order to have a numerically stable method it can be shown that we need to have

$$\frac{D \delta t}{\delta x^2} \leq \frac{1}{2N}. \quad (6.2.14)$$

Seven

Cahn–Hilliard Equation

7.1 The Equation

The **Cahn–Hilliard** equation governs the mixing of two substances, such as oil and water. Let $\rho_1(\mathbf{r})$ be the density of one substance, and $\rho_2(\mathbf{r})$ the density of the other. We define the order parameter

$$\varphi(\mathbf{r}) := \frac{\rho_1(\mathbf{r}) - \rho_2(\mathbf{r})}{\rho_1(\mathbf{r}) + \rho_2(\mathbf{r})}. \quad (7.1.1)$$

This will be 1 when there is only substance 1 at a point, -1 if there is only substance 2 at that point, and 0 for an even mix. The Cahn–Hilliard equation is then

$$\frac{\partial \varphi}{\partial t} = M \nabla^2 \mu \quad (7.1.2)$$

where M is a constant, called the mobility, and μ is the chemical potential. Notice that this is similar in form to the diffusion equation, except that we have the Laplacian of the chemical potential, which is a function of φ , whereas the diffusion equation would just have the Laplacian of φ .

7.2 Derivation

The derivation of the Cahn–Hilliard equation follows the derivation of the diffusion equation for an ideal gas, so we will cover this first.

Consider an ideal gas of density $\rho(\mathbf{r}, t)$. The number of particles in the gas is constant, and is given by

$$\int_V \rho(\mathbf{r}, t) dV. \quad (7.2.1)$$

This implies that there is a conservation law, which in turn gives us a continuity equation:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \mathbf{J}. \quad (7.2.2)$$

Here we identify \mathbf{J} as the mass current. We expect diffusion to occur away from areas of high density, which means we expect that $\mathbf{J} \propto -\nabla \rho$. Call the constant of proportionality D . We therefore have

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (-D \nabla \rho) = D \nabla^2 \rho. \quad (7.2.3)$$

which is the diffusion equation.

Now consider a mix of oil and water and define $\varphi(\mathbf{r}, t)$ as the order parameter from the previous section. Since the total amount of oil and water is fixed we have that

$$\int_V \varphi(\mathbf{r}, t) dV = \text{const.} \quad (7.2.4)$$

This means we have a conservation law, and hence there is a continuity equation

$$\frac{\partial \varphi}{\partial t} = -\nabla \cdot \mathbf{J} \quad (7.2.5)$$

for some \mathbf{J} .

Now consider the chemical potential, which is defined as

$$\mu = \frac{\partial F}{\partial N}, \quad (7.2.6)$$

where F is the free energy and N is the number of particles. Consider a region split into two parts, A and B , with N_A and N_B particles respectively. If we keep everything else constant but have a flux of particles between the two regions then we find that the change in free energy is

$$dF = \frac{\partial F}{\partial N_A} dN_A + \frac{\partial F}{\partial N_B} dN_B = \mu_A dN_A + \mu_B dN_B. \quad (7.2.7)$$

Keeping the total number of particles, $N = N_A + N_B$, constant means we have $dN = dN_A + dN_B = 0$, so $dN_B = -dN_A$. We therefore have

$$dF = (\mu_A - \mu_B) dN_A. \quad (7.2.8)$$

In equilibrium we have $dF = 0$, which gives $\mu_A = \mu_B$. This means that the chemical potential is constant in equilibrium, just like ρ is constant for an ideal gas in equilibrium. By analogy we then have that $\mathbf{J} \propto -\nabla \mu$. Call the constant of proportionality M . We then have

$$\frac{\partial \varphi}{\partial t} = -\nabla \cdot (-M \nabla \mu) = M \nabla^2 \mu, \quad (7.2.9)$$

which is the Cahn–Hilliard equation.

7.3 Chemical Potential

In order to solve the Cahn–Hilliard equation we need to rewrite the chemical potential in terms of φ . This can be done by identifying

$$\mu = \frac{\partial F}{\partial N} = \frac{\partial \left(\frac{F}{V} \right)}{\partial \left(\frac{N}{V} \right)} = \frac{\partial f}{\partial \varphi}, \quad (7.3.1)$$

¹In my lecture notes I had $f = F/N$ here and in equation Equation (7.3.1), so number density instead of volume density. It was pointed out that this is probably wrong, so I've changed it here, but it's been too long since I took this course for me to be sure (2024-01-14),

where $f = F/V$ is the free energy density¹, and we identify that N/V is the value of φ in equilibrium.

This means that if we know the free energy as a function of φ we can find the chemical potential as a function of φ . For example, consider the free energy density

$$f = -\frac{a}{2} \varphi^2 + \frac{a}{4} \varphi^4. \quad (7.3.2)$$

This has two minima, at $\pm\sqrt{2}$, which represent two separate phases. The chemical potential is then

$$\mu = \frac{\partial f}{\partial \varphi} = -a\varphi + a\varphi^3. \quad (7.3.3)$$

We then need to solve

$$\frac{\partial \varphi}{\partial t} = M\nabla^2(-a\varphi + a\varphi^3). \quad (7.3.4)$$

There is a problem with this, which is that it isn't numerically stable. To solve this we notice that we haven't accounted for surface tension, which corresponds to adding $-\kappa\nabla^2\varphi$ to the chemical potential. Hence we instead look to solve

$$\frac{\partial \varphi}{\partial t} = M\nabla^2(-a\varphi + a\varphi^3 - \kappa\nabla^2\varphi). \quad (7.3.5)$$

This corresponds to the free energy

$$f = -\frac{a}{2}\varphi^2 + \frac{a}{4}\varphi^4 + \frac{\kappa}{2}\nabla\varphi \cdot \nabla\varphi. \quad (7.3.6)$$

The first two terms favour phase separation, whereas the surface tension favours minimising the boundary.

7.4 Discretisation

In order to discretise the Cahn–Hilliard equation we need to discretise the $\partial\varphi/\partial t$ derivative, the $\nabla^2\mu$ term, and the $\nabla^2\varphi$ term appearing in the chemical potential. It's easier to discretise the two Laplacians separately, rather than trying to deal with a $\nabla^2\nabla^2\varphi$ term, since we have a very good approximation of the Laplacian which holds to $O(\delta x^4)$.

We will work in two dimensions, so we discretise into a square grid labelled by i and j , and discretise time and label it by n . The first step is to discretise μ :

$$\begin{aligned} \mu[i, j, n] = & -a\varphi[i, j, n] + a(\varphi[i, j, n])^3 - \frac{\kappa}{\delta x^2}(\varphi[i-1, j, n] + \varphi[i+1, j, n] \\ & + \varphi[i, j-1, n] + \varphi[i, j+1, n] - 4\varphi[i, j, n]). \end{aligned} \quad (7.4.1)$$

We can compute this given φ , which we will have as either the initial condition or the result of the previous step.

We then discretise the Cahn–Hilliard equation

$$\begin{aligned} \frac{\varphi[i, j, n+1] - \varphi[i, j, n]}{\delta t} = & \frac{M}{\delta x^2}(\mu[i-1, j, n] + \mu[i+1, j, n] + \mu[i, j-1, n] \\ & + \mu[i, j+1, n] - 4\mu[i, j, n]). \end{aligned} \quad (7.4.2)$$

Solving this for $\varphi[i, j, n+1]$ we get

$$\begin{aligned} \varphi[i, j, n+1] = & \varphi[i, j, n] + \frac{M\delta t}{\delta x^2}(\mu[i-1, j, n] + \mu[i+1, j, n] + \mu[i, j-1, n] \\ & + \mu[i, j+1, n] - 4\mu[i, j, n]). \end{aligned} \quad (7.4.3)$$

We can then iterate this to see the system evolve.

Eight

Boundary Value Problems

Boundary value problems are those without a time derivative, such as Poisson's equation,

$$\nabla^2 \varphi = -\frac{\rho}{\varepsilon}. \quad (8.0.1)$$

In order to solve boundary value problems we need a set of boundary conditions, specifying either φ or its derivatives on the boundary of some region.

Solving initial value problems was relatively straight forward because we just have to simulate the dynamics of the system. In order to solve boundary value problems we convert them into initial value problems, with additional boundary conditions, and then look for a steady state solution. For example, Poisson's equation becomes

$$\frac{\partial \varphi}{\partial t} = \nabla^2 \varphi + \frac{\rho}{\varepsilon}. \quad (8.0.2)$$

For a steady state solution $\partial \varphi / \partial t = 0$ and this reduces to Poisson's equation again. Note that we could also take

$$\frac{\partial \varphi}{\partial t} = -\nabla^2 \varphi - \frac{\rho}{\varepsilon}, \quad (8.0.3)$$

but this would lead to a numerically unstable algorithm.

8.1 Jacobi Method

The **Jacobi method** consists of solving the boundary value problem by converting it to an initial value problem and then iterating taking the time step to be as large as possible while maintaining the numerical stability of the solution. We then iterate until we reach a sufficiently steady state and take this as the answer.

For simplicity we consider Poisson's equation in one dimension, which we turn into the initial value problem

$$\frac{\partial \varphi}{\partial t} = \frac{\partial^2 \varphi}{\partial x^2} + \rho. \quad (8.1.1)$$

We set $\varepsilon = 1$ for simplicity also.

We can discretise this giving

$$\frac{\varphi[i; n+1] - \varphi[i; n]}{\delta t} = \frac{1}{\delta x^2} (\varphi[i+1; n] + \varphi[i-1; n] - 2\varphi[i; n]) + \rho[i]. \quad (8.1.2)$$

Solving this for $\varphi[i; n + 1]$ we get

$$\varphi[i; n + 1] = \varphi[i; n] + \frac{\delta t}{\delta x^2} (\varphi[i + 1; n] + \varphi[i - 1; n] - 2\varphi[i; n] + \delta x^2 \rho[i]). \quad (8.1.3)$$

Stability analysis reveals that we need to have $\delta t / \delta x^2 \geq 1/2$. Choosing $\delta x = 1$ this gives $\delta t = 1/2$ as the largest possible time step. The Jacobi algorithm is then to iterate

$$\varphi[i; n + 1] = \varphi[i; n] + \frac{1}{2} (\varphi[i + 1; n] + \varphi[i - 1; n] - 2\varphi[i; n] + \rho[i]) \quad (8.1.4)$$

$$= \frac{1}{2} [\varphi[i + 1; n] + \varphi[i - 1; n] + \rho[i]]. \quad (8.1.5)$$

In three dimensions the Jacobi algorithm is almost identical:

$$\begin{aligned} \varphi[i, j, k; n + 1] = & \frac{1}{6} (\varphi[i + 1, j, k; n] + \varphi[i - 1, j, k; n] + \varphi[i, j + 1, k; n] \\ & + \varphi[i, j - 1, k; n] + \varphi[i, j, k + 1; n] + \varphi[i, j, k - 1; n] + \rho[i, j, k]). \end{aligned} \quad (8.1.6)$$

We need to decide when to stop iterating, that is how do we identify a steady state? The easiest way is to define the error to be

$$\sum_{i,j,k} |\varphi[i, j, k; n + 1] - \varphi[i, j, k; n]|. \quad (8.1.7)$$

We then pick some tolerance value, say 10^{-2} or 10^{-3} , and we stop iterating when the error between iterations drops below the tolerance.

8.2 Gauss–Seidel Algorithm

The Jacobi algorithm has slow convergence, it is $O(N^2)$, where N is the extent of the system in one direction. So in three dimensions we might consider the volume to be an $N \times N \times N$ cube. The **Gauss–Seidel algorithm** is a slight modification on the Jacobi method which greatly improves the convergence time.

The idea for the Gauss–Seidel algorithm is that in steady state the value of φ shouldn't change much between iterations once we reach an almost-steady state. We exploit this in a for loop to avoid having to create a new array to store the values of φ calculated at $n + 1$, and instead use a single array and just update it as we go. The result is that in one dimension we instead compute

$$\varphi[i; n + 1] = \frac{1}{2} (\varphi[i + 1; n] + \varphi[i - 1; n + 1] + \rho[i]). \quad (8.2.1)$$

This converges slightly faster, since we use a more up to date value of the potential, and avoids time spent creating and copying arrays.

8.3 Successive Over Relaxation

Successive over relaxation (SOR) is a method that we apply to a differential equation of the form

$$\frac{dx}{dt} = g(x). \quad (8.3.1)$$

Discretising we have

$$\frac{x[n+1] - x[n]}{\delta t} = g[x[n]] \implies x[n+1] - x[n] = \delta t g[x[n]]. \quad (8.3.2)$$

Solving this for $x[n+1]$ we have

$$x[n+1] = \delta t g[x[n]] + x[n] = f[x[n]] + x[n], \quad \text{where} \quad f[x[n]] = \delta t g[x[n]]. \quad (8.3.3)$$

We generalise this by introducing a parameter, ω , called the **relaxation parameter**:

$$x[n+1] = \omega f[x[n]] + (1 - \omega)x[n]. \quad (8.3.4)$$

Notice that setting $\omega = 1$ we recover the original relation. We therefore have

$$x[n+1] = x[n] + \omega(f[x[n]] - x[n]) = x[n] + \omega \delta x[n]. \quad (8.3.5)$$

Here $\delta x[n] = x[n+1] - x[n]$.

The idea behind successive over relaxation is that we are not interested in how the system evolves, just the final state. If we set $\omega > 1$ then we will slightly overshoot the true value of $x[n+1]$, but we'll still be heading in the correct direction. We can then choose a value of ω and iterate this as we did for the other methods. For the value of ω we know that it needs to be greater than 0, in order to have anything change. It should also be greater than 1, otherwise we are taking smaller steps than necessary. There will also be some critical value of ω above which the algorithm becomes numerically unstable. Between these extremes there will be an optimal value of ω that gives the fastest possible convergence. It can be shown that for our purposes we have $0 < \omega < 2$.

We can apply successive over relaxation to the Gauss–Seidel algorithm by defining

$$\varphi_{\text{GS}}[i; n+1] = \frac{1}{2}(\varphi[i+1; n] + \varphi[i-1; n+1] + \rho[i]). \quad (8.3.6)$$

We then iterate

$$\varphi[i; n+1] = \omega \varphi_{\text{GS}}[i; n+1] + (1 - \omega)\varphi[i; n] \quad (8.3.7)$$

Nine

Maxwell's Equations

9.1 The Equations

Maxwell's equations are

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}, \quad \nabla \cdot \mathbf{B} = 0, \quad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \quad \text{and} \quad \nabla \times \mathbf{B} = \mu_0 \left(\mathbf{J} + \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right).$$

9.2 Poisson's Equation

We will solve Poisson's equation in three dimensions for two different cases of electromagnetism.

9.2.1 Electric Field

For a static field with $\partial \mathbf{B} / \partial t = 0$ we have $\nabla \times \mathbf{E} = 0$. This means we can write the electric field as the gradient of a scalar potential, $\mathbf{E} = -\nabla \varphi$. We then have

$$\nabla \cdot \mathbf{E} = \nabla \cdot (-\nabla \varphi) = \nabla^2 \varphi \quad (9.2.1)$$

and so

$$\nabla^2 \varphi = \frac{\rho}{\epsilon_0}. \quad (9.2.2)$$

This is Poisson's equation for the electric field.

9.2.2 Magnetic Field

Since $\nabla \cdot \mathbf{B} = 0$ we can write the magnetic field as the curl of a vector potential, $\mathbf{B} = \nabla \times \mathbf{A}$. We then have for a static field with $\partial \mathbf{E} / \partial t = 0$, that

$$\nabla \times \mathbf{B} = \nabla \times (\nabla \times \mathbf{A}) = \mu_0 \mathbf{J}. \quad (9.2.3)$$

We can use the following identity for the curl of the curl: $\nabla \times (\nabla \times \mathbf{A}) = \nabla(\nabla \cdot \mathbf{A}) - \nabla^2 \mathbf{A}$. We can work in the Coulomb gauge, where $\nabla \cdot \mathbf{A} = 0$, to give

$$-\nabla^2 \mathbf{A} = \mu_0 \mathbf{J}. \quad (9.2.4)$$

Consider a wire along the z -axis, so $\mathbf{J} = J \mathbf{e}_z$. We then have

$$\mathbf{B} = \nabla \times \mathbf{A} = \begin{pmatrix} \partial_y A_z - \partial_z A_y \\ \partial_z A_x - \partial_x A_z \\ \partial_x A_y - \partial_y A_x \end{pmatrix}. \quad (9.2.5)$$

Since $B_z = 0$ by symmetry we have $A_x = A_y = 0$. We also have that \mathbf{A} depends only on the distance from the wire, not distance along it, so $\mathbf{A} = A(x, y)$. Hence $\mathbf{A} = (0, 0, A_z(x, y))$. This gives Poisson's equation:

$$\nabla^2 A_z(x, y) = -\mu_0 J_z. \quad (9.2.6)$$

Index

A

absorbing state, [17](#)
autocorrelation time, [12](#)

B

backward difference, [25](#)
Bootstrap algorithm, [13](#)

C

Cahn–Hilliard, [27](#)
centred difference, [25](#)
critical temperature, [10](#)

D

detailed balance, [14](#)
Dirichlet boundary conditions, [24](#)

F

finite difference, [24](#)
forward difference, [25](#)

G

game of life, [17](#)
Gauss–Seidel algorithm, [31](#)

I

importance sampling, [6](#)
Ising model, [4](#)

J

jackknife algorithm, [13](#)
Jacobi method, [30](#)

M

magnetisation, [10](#)
Markov chain, [7](#)
Monte Carlo, [2](#)

R

relaxation parameter, [32](#)

S

SIRS model, [21](#)

V

von Neumann boundary conditions, [24](#)