Science and Computers II: Project 2

Part I
Numerical Techniques

1 Using computers to solve physics problems

The vast majority of real life physics problems do not have simple exact analytic solutions that can be easily obtained. Even for your lecture courses, your professors often go to great lengths to think up special case examples that you can solve analytically. This leaves two options:

- make approximations so that an approximate solution can be found (perturbation theory in quantum mechanics is a good example of this);
- solve the mathematics numerically after completing as much of the calculation anaytically as is practical.

In this course, we will focus on the latter.

In most cases, numerical problems in physics boil down to one of four things:

Algebra: the manipulation of numbers, vectors, matrices etc., whose manipulation, though conceptually straightforward, is too long and arduous to do by hand.

Numerical Integration: Most often our analytic calculations are stopped by encountering a nasty integral. Maybe it doesn't have a closed form, or if it does, it is too complicated to reliably write down, or maybe we just want the flexibility to change the integral's boundary conditions without having to recalculate everything. This is perhaps the most important class of numerical problem for physics, so this is where we will spend a great deal of our time.

Differential Equations: Many of the laws of physics are described in terms of differential equations. For example, wave mechanics, quantum mechanics, fluid mechanics, thermodynamics, electromagnetism, general relativity and even classical dynamics, all involve differential equations.

Root Finding: Often in solving equations in physics one has to find the roots of some complicated function or polynomial. This can be a highly non-trivial task, and must often be done numerically.

This categorization is not exclusive - there are many other cases - and, of course, it is rather fluid. Many problems that appear at first sight to require a numerical treatment of differential equations are more easily done via numerical integration. An obvious example would be solving the differential equation,

$$\frac{dy}{dx} = f(x), \tag{1}$$

which can be expressed instead as,

$$y = \int f(x)\,dx. \tag{2}$$

(One cannot always do this though since the right-hand side may be a function of both x and y, i.e. $f(x, y)$.)

2 Numerical Integration

2.1 Integrals as averages

In essence, all numerical integrations are attempts to find the mean value of a function over a (possibly multi-dimensional) integral,

$$I = \int_{x_1}^{x_2} f(x)\,dx = (x_2 - x_1)\,\langle f(x)\rangle. \tag{3}$$

The most obvious way to calculate this numerically is simply to take some example choices of x, let's call them x_i with $i = 1 \ldots N$, calculate $f(x_i)$ and take the average value of $f(x_i)$ to approximate the mean,

$$I \approx (x_N - x_1)\frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{4}$$

We could be slightly more sophisticated and put *weights* w_i on each of these to make some choices of x_i more important than others,

$$I \approx (x_N - x_1)\frac{1}{W}\sum_{i=1}^{N} w_i f(x_i), \tag{5}$$

where the extra normalization factor is $W = \sum_{i=1}^{N} w_i$. Different numerical integration techniques differ in their choice of how to pick a representative sample of x_i and their relative importance (what weights to choose).

2.2 Trapezoidal Rule

The trapezoidal rule, in its simplest form, is simply approximating the area under a curve by that of a trapezoid. Clearly, this is Eq.(5) with $N = 2$ and $w_i = 1/2$. We could go further and split up the integration region into lots of little trapezoids of equal spacing in x. The total area

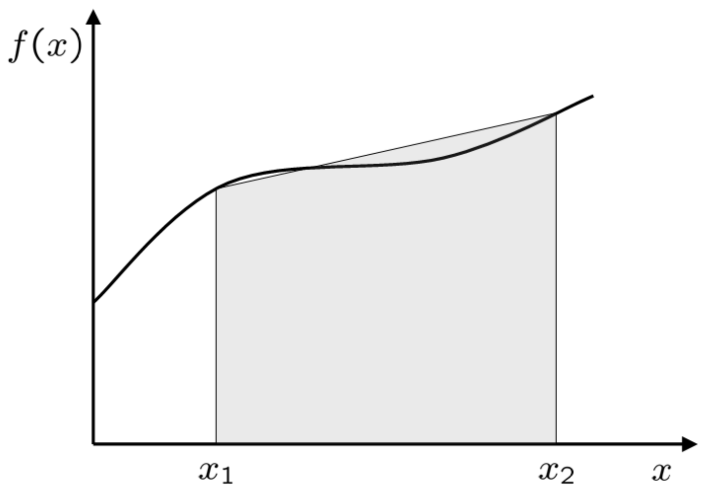

Figure 1: A graphical representation of the simplest ($N = 2$) trapezoidal rule.

will then be the sum of these smaller areas which we can approximate via a trapezoid. So,

$$\begin{aligned} I &\approx (x_2 - x_1)\frac{1}{2}(f(x_1) + f(x_2)) + (x_3 - x_2)\frac{1}{2}(f(x_2) + f(x_3)) \\ &\quad +(x_4 - x_3)\frac{1}{2}(f(x_3) + f(x_4)) + \ldots + (x_N - x_{N-1})\frac{1}{2}(f(x_{N-1}) + f(x_N)) \qquad (6) \\ &= (x_N - x_1)\frac{1}{N-1}\left[\frac{1}{2}f(x_1) + f(x_2) + f(x_3) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right], \qquad (7) \end{aligned}$$

where, in the last step, we have assumed that all the intervals are of equal size (so for example, $(x_2 - x_1) = (x_N - x_1)/(N - 1)$). Clearly this is Eq.(5) with $w_1 = w_N = 1/2$ and $w_i = 1$ for $1 < i < N$.

Exercise 1: **Exercise 1: Simple implementation of the trapezoidal rule**

Write C code to do the following

```
double trap( double (*func)(double x), double a, double b, double h )
```

where `a` and `b` are the the end-points of the integration, `h` is the interval size to be used $[h = (x_N - x_1)/(N - 1)]$ and `func` is the name of the function to be integrated, which should have the prototype

```
double my_function( double x )
```

So, for example, to evaluate

$$\int_0^1 \exp\left(-x^2\right)\, dx$$

you would write

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

double trap( double (*func)(double x), double a, double b, double h )
{
  ...
  /* to evaluate the function at a value x call */
  f = func( x );
  ...
}

double myfun( double x )
{
  return exp( - (x*x) );
}

int main( int argc, char *argv[] )
{
  double a = 0.0;
  double b = 1.0;
  double h = 0.1;
  double ans;

  ans = trap( myfun, a, b, h );

  fprintf( stdout, "Answer = %e\n", ans );

  return 0;
}
```

If you are not sure how to pass a function name as an argument to another function, see me. Use your `C` code to calculate the integral

$$I = \int_0^2 \left[2 + \cos(2\sqrt{x})\right] \, dx,$$

using the trapezoidal rule. In this case, the integral is doable analytically, so compare your answer to the correct expression for $N = 2, 4, 8, 16$. Can you see any pattern in the error's dependence on N? Consider the Taylor expansion for the Trapezoidal Rule to determine the dependence on N (or h). Is the error linear in h? Quadratic?

2.3 Simpson's Rule

In the trapezoidal rule we are essentially approximating the curve by a set of straight lines between fixed points on the curve. We could have instead approximated the curve using a set

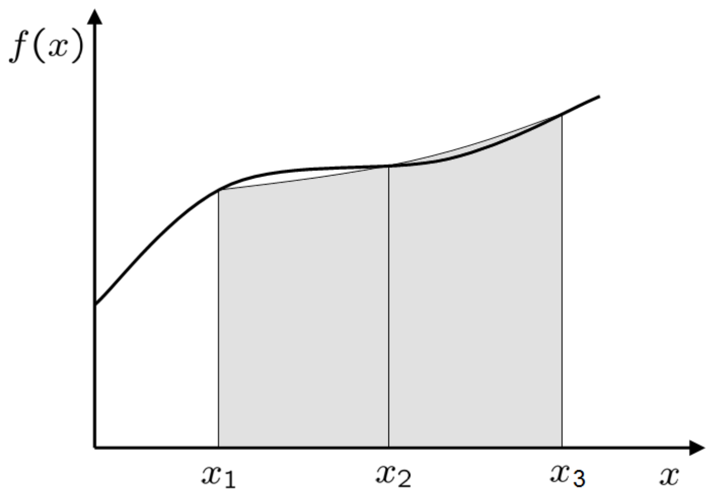

Figure 2: A graphical representation of the simplest ($N = 3$) Simpson's rule.

of parabolas rather than straight lines, and if the curve is quite "wiggly" (that is, not straight) we might expect to do a little better, see Figure 2.

You can calculate the area under a parabola quite easily, and anchoring it at the end and in the middle we find,

$$I \approx (x_3 - x_1)\frac{1}{2}\left[\frac{1}{3}f(x_1 = a) + \frac{4}{3}f(x_2 = m) + \frac{1}{3}f(x_3 = b)\right]. \tag{8}$$

This does much better than the simplest trapezoidal rule. The generalization to more points (in exactly the same way as we did for the trapezoidal rule) is,

$$\begin{aligned} I \;\approx\; & (x_N - x_1)\frac{1}{N-1}\left[\frac{1}{3}f(x_1) + \frac{4}{3}f(x_2) + \frac{2}{3}f(x_3) + \frac{4}{3}f(x_4) + \ldots\right. \\ & \left. + \frac{2}{3}f(x_{N-2}) + \frac{4}{3}f(x_{N-1}) + \frac{1}{3}f(x_N)\right], \end{aligned} \tag{9}$$

or in other words Eq. (5) with $w_1 = w_N = 1/3$, $w_i = 4/3$ for $1 < i < N$ and i even, and $w_i = 2/3$ for $1 < i < N$ and i odd.

Exercise 2: Repeat the last exercise using Simpson's Rule. What is the dependence of the error on N this time? Try to understand this dependence from the Taylor series expansion for the integral.

2.4 Gaussian Quadrature

In the previous example, Simpson's Rule, we used a parabola to mimic the behavior of the function over the range (x_1, x_3) and then extended this by dividing our integration up into lots of "mini-integrations". We could have taken an alternative approach, and instead of dividing up the region, we could have mimiced the function with more complicated polynomials. Clearly these more complicated polynomials would also require more points in order to specify them, but if we choose a set of polynomials which is *complete* (i.e. any well behaved function can be built out of them) then we should be able to get arbitrarily close to the exact answer just by including more of them. The terminology is that an n-point Gaussian Quadrature yields an exact result for the integration of a $2n-1$ degree polynomial.

There are many different types of Gaussian Quadrature, depending on the polynomials you choose to mimic your function with. Examples of these functions are Legendre polynomials, Jacobi polynomials, Chebyshev polynomials, Laguerre polynomials, Hermite polynomials and many more. Since you may not have met most of these functions yet, this integration method is a little too advanced for this course and we won't take it any further.

2.5 Monte Carlo Integration

In principle, it shouldn't matter how we choose our x_i and our weights w_i, we should eventually, for N large enough, reproduce the correct answer for any well behaved function. (Although how we choose our x_i and w_i does affect how fast we converge on the correct result.) Monte Carlo integration takes this principle to the extreme and chooses the x_i *randomly*!

Looking again at Eq.(5) (this time with weights $w_i = 1$ for simplicity),

$$I \approx (x_N - x_1)\frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{10}$$

we see no specification for the values x_i. We just assumed that they should be uniformly spaced, but this didn't need to be the case.

Exercise 3: Repeat the numerical integration of

$$I = \int_0^2 \left[2 + \cos(2\sqrt{x})\right]\, dx,$$

this time using random values for x_i and unit weights. You can use the `C` function `rand()` to generate random numbers. How fast does this converge?

If you have done the above tasks correctly, you will see that the Trapezoidal Rule has an error $\propto 1/N^2$, Simpson's rule has an error $\propto 1/N^4$ and the Monte Carlo integration has an error $\propto 1/\sqrt{N}$.

In fact, the error of a Monte Carlo integration is related to the standard deviation, σ, of the integrand over the region. For a one dimensional integration, this is defined by,

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left[f(x_i) - \bar{f}\right]^2 = \frac{1}{N}\left(\sum_{i=1}^{N}\left[f(x_i)\right]^2\right) - \bar{f}^{\,2} \tag{11}$$

where the mean is,

$$\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{12}$$

Then the Monte Carlo error is

$$\sigma_{\text{MC}} = \frac{\sigma}{\sqrt{N}}. \tag{13}$$

2.6 The advantages of Monte Carlo integration

So if the convergence of Monte Carlo integration is slower than the other methods we have already looked at, what is it good for?

- **Fast convergence for many dimensions:** Although the other methods give much smaller errors for one dimensional integrations, their errors grow with the number of dimensions. For a d dimensional integration, the Trapezoidal Rule's error is $\propto N^{-2/d}$ and Simpson's Rule's error is $\propto N^{-4/d}$, but a Monte Carlo integration error is always $\propto 1/\sqrt{N}$.

- **Easy to incorporate awkward boundary conditions:** In the one dimensional cases we have looked at so far, the boundary conditions were rather trivial (just numbers), but for higher dimensional integrations they can become quite complex (surfaces). This make the other rules quite difficult to program because you need to know explicit formulae for the boundaries in terms of the variable you are integrating over. For Monte Carlo integration, you can simply choose to generate the $\vec{x}_i$ over a larger space which has simple boundaries, check whether or not the point lies in the space you are interested in, and throw it away if it isn't.

- **Small feasibility limit:** You can see that the trapezoidal rule and Simpson's rule both need at least two or three x_i before you even start. While this is fine for small numbers of dimensions, for larger dimensional integrals you need to have at least this number of points raised to the power of the number of dimensions. For example, for a 10-dimensional integration, Simpson's rule needs a minimum of $3^{10} = 59049$ points! This is not the case for a Monte Carlo integration where in principle one point (or two if you want an estimate of the error) will do (but of course would give very bad errors).

- **Ability to improve the calculation:** For other integration methods you are "locked in" to the accuracy you choose at the start of your calculation. For example, in Simpson's rule, you choose N and then set your x_i and w_i accordingly. If you then decide that your answer is not accurate enough, you need to increase N and start the calculation again. For a Monte Carlo integration you can just keep adding points until you get the accuracy you want - N is not fixed at the start.

- **Integration as physical simulation:** Many physics systems are stochastic, relying on apparently random effects (e.g. thermodynamics) or possible even truely random effects (e.g. quantum mechanics). Monte Carlo's can then model physical systems where each generated point is analogous to a physical event. This gives a much more intuitive physical picture of what is going on in the Monte Carlo integration/simulation.

Exercise 4: To demonstrate the second of these advantages, write a `C` program to calculate the volume of a sphere embedded in n dimensions. Check that for 10 dimensions, its volume is,

$$V = \frac{\pi^5}{120} r^{10},$$

where r is the radius. (This problem is much easier than it sounds, so if you are having trouble figuring it out, please *ask*!)

Exercise 5: Consider a crescent, as seen in Figure 3, and bounded by the curves,

$$\begin{aligned} x^2 + y^2 &= a^2, &(14)\\ (x-c)^2 + y^2 &= b^2, &(15) \end{aligned}$$

where a, b and c are constants with $b < a$ and $a - b < c < a + b$. Assuming it is uniform in the z-direction (out of the page) write a `C` program to calculate its moment of inertia about the axis $x = y = 0$.

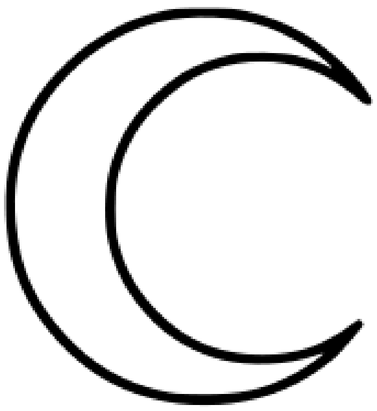

Figure 3: A crescent.

Note that both of the above tasks would be very difficult to do using non-Monte Carlo numerical integrations (though the first one is straightforward to do analytically).

2.7 The importance of flat distributions

It should be fairly obvious that if the distribution we are integrating is perfectly flat, then $\langle f(x) \rangle = f(x)$ for all x and all these integration methods give a perfect result with smallest N. Of course, in this case one would not be integrating it numerically! However, it is generally true that these methods all work better if the integrand is reasonably flat, and work poorly if the integrand has a strong dependence on the parameter you are integrating over. So, if we can manipulate our integration to make it reasonably flat before numerical integration, we will need far fewer points for the integration to reach the desired accuracy.

So how do we do this? Consider integrating a function $f(x)$, and suppose we know of a function $g(x)$ which *we know how to integrate analytically* which is approximately the same as $f(x)$ over the range of x we are integrating over. Then,

$$\int f(x)dx = \int \frac{f(x)}{g(x)}dy, \tag{16}$$

where

$$dy = g(x)dx \quad \Rightarrow \quad y = \int g(x)dx. \tag{17}$$

Since we know how to integrate $g(x)$ we can work out y analytically. A numerical integration over y will be more efficient than an integration over x since $f(x)/g(x)$ is reasonably flat.

Let's have a concrete example. Consider the integration,

$$I = \int_{-10}^{10} \frac{1+\frac{1}{4}\cos x}{1+x^2}\,dx, \tag{18}$$

As shown in figure 4. Unaltered, this would be difficult to integrate numerically since our Monte Carlo will very often choose values of x that contribute little to the mean.

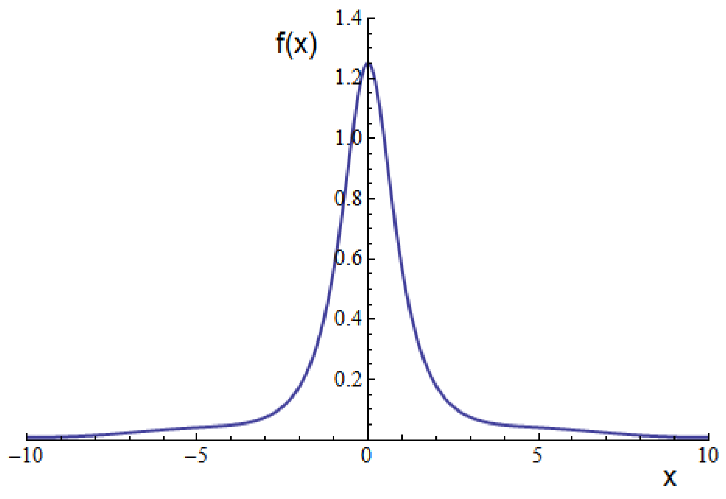

Figure 4: The function $f(x) = \left(1+\frac{1}{4}\cos x\right)/(1+x^2)$.

However, the function $g(x) = 1/(1+x^2)$ is easy to integrate,

$$\int \frac{1}{a^2+x^2}\,dx = \tan^{-1} x + \text{constant}, \tag{19}$$

so we can change variables to $y = \tan^{-1} x$ and write,

$$I = \int_{-10}^{10} \frac{1+\frac{1}{4}\cos x}{1+x^2}\,dx = \int_{-\tan^{-1}10}^{\tan^{-1}10} \left[1+\frac{1}{4}\cos(\tan y)\right]\,dy. \tag{20}$$

This is now quite flat, so the error is greatly reduced.

Exercise 6: Calculate the integral Eq.(18) using a Monte Carlo integration, first using x as your integration variable and then $y = \tan^{-1} x$. In both cases, quantify the Monte Carlo error (as in Eq.(13)) and compare the two methods.

2.8 Monte Carlo as a physical simulation

So far, Monte Carlo's have been just a tool for performing integrations. However, as described in section 2.6 they can be used to simulate experiments with some random element.

To understand this, let's again consider an example: Buffon's Needle. In 1777 the *Comte de Buffon* posed the following problem: imagine a board with lines drawn on it at equal spacing t, upon which a needle of length l is randomly thrown; what is the probability of the needle landing across one of the lines. See Figure 5 for a pictorial representation.

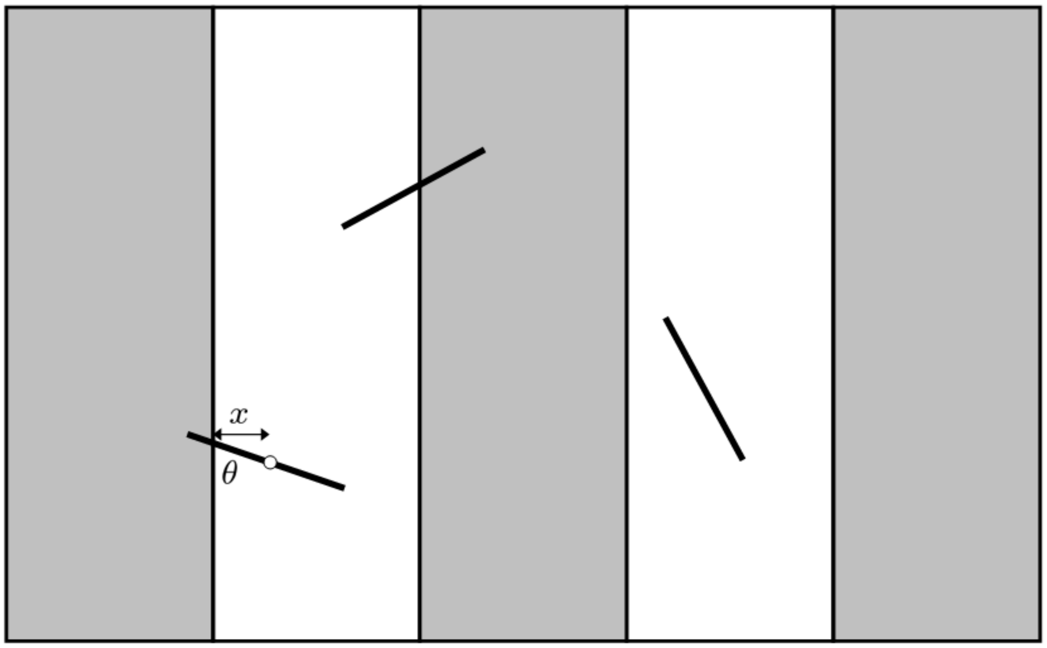

Figure 5: Buffon's Needle problem. A needle of length l is randomly thrown onto a board with lines drawn on it at equal spacing t. What is the probability of the needle landing on one of the lines?

Analytically, this is quite straightforward. If x is the distance of the centre of the needle to the closest line, and θ is the angle between the needle and the lines, then it will cross the line if,

$$x \leq \frac{l}{2} \sin\theta. \tag{21}$$

Since x is random between $0 < x < \frac{t}{2}$ and the angle θ is random between $0 < \theta < \pi/2$ then the probability of the needle landing on a line is,

$$\frac{\displaystyle\int_0^{\pi/2} d\theta \int_0^{(l/2)\sin\theta} dx}{\displaystyle\int_0^{\pi/2} d\theta \int_0^{t/2} dx} = \frac{\displaystyle\int_0^{\pi/2} d\theta\, \frac{l}{2}\sin\theta}{\displaystyle\int_0^{\pi/2} d\theta\, \frac{t}{2}} = \frac{l/2}{\pi t/4} = \frac{2l}{\pi t}. \tag{22}$$

So one could, in principle, determine the value of π by throwing lots of needles on such a board: if you throw N needles and n land on a line, then,

$$\pi \approx \frac{2lN}{nt}. \tag{23}$$

This experiment has been done many times, most famously in 1901 by an Italian mathematician called Lazzerini, who found $\pi = 3.1415929$ after 3408 throws.

We can *simulate* this experiment on a computer by generating random numbers and using them to give us values of x and θ for each needle thrown. For each x and θ choice we then decide if the needle crosses a line (i.e. if $x \leq (l/2)\sin\theta$) and add one to n if it does. In essence, we are using a Monte Carlo to do the integration in Eq.(22), but due to the probabilistic nature of the experiment, this is also a simulation.

NB: You should realize that this is not an actual calculation of π since we need to input π into our program in order to generate a random number between 0 and $\pi/2$ (for θ). This is only a *simulation* of an experiment. (Can you think how we might modify it to actually calculate π with no π inputed?)

Exercise 7: Write a Monte Carlo to simulate Buffon's experiment. Quantify your Monte Carlo error on the measurement.

3 The 2d Ising Model

The two-dimensional Ising Model is a model of ferromagnetism in solids. It is important because it was the first and for a long time the only exactly solvable model with a phase transition. Phase transitions are important throughout physics, from condensed matter to particle physics. The Ising model is a prototype for the study of phase transitions, and is of broad interest, i.e. there is more to it than just the study of magnetic materials.

The Hamiltonian, or total energy of the system in a state $\{s_{i,j}\}$ is

$$H(\{s_{i,j}\}) = -J\sum_{i,j} s_{i,j}(s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}), \tag{24}$$

where we assume only nearest neighbor interactions with coupling J and periodic boundary conditions. The probability of finding the system in a particular state $\{s_{i,j}\}$ is given by

$$P(\{s_{i,j}\}) = \frac{1}{Z(\beta)}\exp[-\beta H(\{s_{i,j}\})], \tag{25}$$

where $\beta = 1/(k_B T)$, with k_B Boltzmann's constant, T the temperature, and

$$Z(\beta) = \sum_{s_{i,j}} \exp[-\beta H(\{s_{i,j}\})], \tag{26}$$

where Z is called the partition function.

In order to calculate macroscopic quantities like the magnetization, you need to sum over all spin configurations weighted by their respective probabilities. This can be done using the Metropolis algorithm (see, e.g. arXiv:0803.0217 for the basic idea). The mean magnetization can be computed using

$$\langle M \rangle = \frac{1}{N} \sum_{\{s_{i,j}\}} M(\{s_{i,j}\}), \tag{27}$$

where one averages over the N configurations generated using the probability $P(\{s_{i,j}\})$.

Exercise 8:

- Make sure that you have a random number generator that creates pseudo random numbers uniformly distributed between $[0, 1)$.
- Create a one-dimensional array that contains the spin (1 or -1) for every site of a $L \times L$ lattice. Initialize your lattice with all spins set to 1.
- Use the Metropolis algorithm to update your system.
- Display the Monte Carlo time evolution using the animation software provided.
- Calculate the magnetization as a function of the effective $\beta_{\text{eff}} = J/(k_B T)$. Include an estimate of the statistical error. Where is the phase transition?
- A better way to locate the phase transition is using the magnetic susceptibility $\chi_M = \langle M^2 \rangle - \langle M \rangle^2$, which diverges at the phase transition (in the infinite volume limit). Compute this quantity as a function of β_{eff} and locate the critical β_c. How does this compare to the analytical result of $\beta_c = \frac{1}{2}\ln(1 + \sqrt{2})$?

4 Project Report

You should write a report on your Ising Model simulations (Exercise 8), and, in addition, you should choose *one* of the exercises from 4-7 to include in your report. Your report should be written using LaTeX(unless you have a very strong preference for another package with similar capabilities). You should include an abstract, and an introduction that explains why the exercises are interesting and provides a summary of the projects. You should explain how you solved the problem, including the algorithm you used (without detailed `C` code) and present your results. Remember to make an estimate of your errors! Finally you should draw some conclusions and summarize your work, making suggestions on how your calculation could be improved (or not!). Your `C` code and outputs should be reserved for an appendix. If you use anybody else's algorithm, results, formulas or information, be sure to reference them.