Advanced Monte Carlo Methods

Problems

September-November, 2012

Contents

1 Integration with the Monte Carlo method

1.1 Non-uniform random numbers

Monte Carlo simulations are based on random sampling of the quantities to compute. This sampling relies upon a random number generator (RNG). Most high level languages (java, C++, Matlab, ...) contain RNGs that generate (quasi-)random numbers uniformly between 0 and 1. For instance in Matlab (or Octave) this is done by the function **rand()**. In Matlab one also has **randn()**, which generates random numbers from a Gaussian distribution[1]. In many Monte Carlo problems it is important to generate non-uniform random numbers. This exercise presents two different ways to do so.

Let us suppose that we want to generate random numbers with a given distribution $f(x)$. Here $f(x) \geq 0$ and it is normalized $\int_{-\infty}^{+\infty} f(x)dx = 1$. Examples are $f(x) = e^{-x}$ in the interval $[0, +\infty)$ (and zero otherwise), or $f(x) = 1 - x^2$ in $[-1, 1]$. We consider the cumulative distribution function:

$$F(x) = \int_{-\infty}^{x} dy\ f(y) \tag{1}$$

for which $0 < F(x) \leq 1$. If we now draw random numbers x_i from the uniform distribution in $[0, 1]$ then the random numbers $y_i = F^{-1}(x_i)$ will be distributed with probability density $f(x)$. To show this is easy and left to you as exercise.

This first approach requires that the function $f(x)$ can be integrated analytically and that $F(x)$ can be inverted. This is not the case for instance for[2] $f(x) = e^{-x^2}/\sqrt{\pi}$.

Another way to generate random number with a given distribution $f(x)$ on a finite interval (meaning that $f(x) \neq 0$ only in some interval $[a, b]$) is the *hit-and-miss* method. Let $f(x)$ be defined in $[a, b]$ and let M such that $M > f(x)$ for every x in $[a, b]$. We now generate two random numbers t and s uniformly distributed in $[a, b]$ and $[0, M]$, respectively. The case $s > f(t)$ corresponds to a miss and the generated numbers are neglected. If instead $s \leq f(t)$, we keep and store the value of t, which is the output of our RNG algorithm. Note that s is only a checking variable and its value is not returned.

Note: the hit-and-miss method has a drawback. All the misses are generated and not actually used. This is a waste of computer resources.

Questions

- Using uniform random numbers in the interval $[0, 1]$ and the inverse cumulative distribution function as defined in Eq. (1), generate random numbers (plot the histograms) which are distributed
  - uniformly in $[-2, 1]$
  - exponentially in $\mathbb{R}^+$: $f(x) = e^{-x}$ for $x \geq 0$ and $f(x) = 0$ for $x < 0$
  - linearly in the interval $[-1, 1]$: $f(x) = (x + 1)/2$ and zero elsewhere.

[1] Try **hist(rand(1,10000))** and **hist(randn(1,10000))** to plot the histograms of the random numbers generated by the two functions

[2] A generator of Gaussian random numbers can be obtained using an appropriate transformation (the Box-Muller transformation).

- Using the hit-and-miss method generate random numbers (plot the histograms) which are distributed
  - linearly in the interval $[-1, 1]$: $f(x) = (x + 1)/2$ and zero elsewhere.
  - distributed as $f(x) = -x(1 - x)e^{-x^2} \log(1 - x)$ in $[0, 1]$
- Using the hit-and-miss method generate random points in a two-dimensional plane uniformly distributed inside a circle of radius $R = 1$.
- Generate the same uniform distribution (without the hit-and-miss method) using an appropriate distribution function $f(r, \theta)$, where r and θ are polar coordinates.

1.2 Gaussian RNG

A way to generate random numbers from a Gaussian distribution is by the so-called Box-Muller transformation. Suppose that we have two numbers x and y drawn from a standard Gaussian distribution. Their combined density function then becomes

$$f(x, y)dxdy = \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right) dxdy \tag{2}$$

Transforming this into polar coordinates gives

$$f(r, \theta)drd\theta = \frac{1}{2\pi} \exp\left(-\frac{r^2}{2}\right) rdrd\theta \tag{3}$$

So generating random numbers r and θ according to above distribution and transforming them back to cartesian coordinates would yield two random numbers x and y which are standard Gaussian distributed. The θ angle is drawn from a uniform distribution in $[0, 2\pi]$. The r-dependent part is solvable as before, since here we can calculate and invert the cumulative density function:

$$F(r) \quad = \quad \int_0^r \exp\left(-\frac{z^2}{2}\right) zdz \tag{4}$$

So generating a uniform θ, and a r by inverting $F(r)$ allows us to compute two independent Gaussian random numbers x and y by making the transformation from polar to cartesian coordinates.

Questions

- Program above algorithm and show that this indeed generates couples of normally distributed random variables.
- How do we need to adapt the algorithm when we want the random numbers to be distributed according to a Gaussian distribution with average μ and variance σ^2?

1.3 Monte Carlo integration (I)

In this exercise we will estimate the value of the following integral with Monte Carlo methods:

$$\mathcal{I} = \int_{-\infty}^{\infty} \exp\left(-\frac{r^2}{2}\right)(x+y+z)^2 dxdydz \tag{5}$$

where $r^2 = x^2 + y^2 + z^2$.

To perform the calculation we take the factor $e^{-r^2/2}$ as a density function on the domain of the integral. When doing so, one has to be careful with the normalization factor. This integral can also be calculated analytically, and also using another type of Monte Carlo sampling based on the Metropolis algorithm (see next exercise). We can thus compare these different methods.

Questions

- Compute $\mathcal{I}$ analytically.
- Use a Gaussian RNG to generate x, y and z and estimate the integral with this method. Compare this result with the analytical result.
- Calculate the variance σ^2 of the sampled values for different numbers of sampled points N, and show that $\sigma^2 \sim 1/N$.
- Why can't we turn the two factors around: Use $(x+y+z)^2$ as density, and $e^{-r^2/2}$ as integrand?

2 Markov chain Monte Carlo

2.1 Monte Carlo integration (II)

Here we repeat the calculation of the same integral using a different Monte Carlo method: the *Metropolis algorithm*. We need for this purpose to define a Markov chain characterized by some transition probabilities $P(\vec{r} \to \vec{r}\,')$. We would like that the "dynamics" so defined converges to the distribution $w(\vec{r}) = (2\pi)^{-3/2} \exp(-r^2/2)$.

We know that if we require detailed balance, i.e. that the rates for any pairs of points $\vec{r}$ and $\vec{r}\,'$ fulfill

$$q(\vec{r})P(\vec{r} \to \vec{r}\,') = q(\vec{r}\,')P(\vec{r}\,' \to \vec{r}) \tag{6}$$

then the dynamics converges to the distribution $q(\vec{r})$. Therefore we choose the transition probabilities such

$$\frac{P(\vec{r} \to \vec{r}\,')}{P(\vec{r}\,' \to \vec{r})} = \frac{w(\vec{r}\,')}{w(\vec{r})} = \exp\left(-\frac{r'^2 - r^2}{2}\right) \tag{7}$$

The transition probabilities are usually split as $P(\vec{r} \to \vec{r}\,') = g(\vec{r} \to \vec{r}\,')A(\vec{r} \to \vec{r}\,')$ where g is the selection probability and A the acceptance ratio.

The g is generated as follows: starting from an initial point $\vec{r}$ we select a new point $\vec{r}\,' = \vec{r} + \vec{\delta}$ where $\delta = (\delta_x, \delta_y, \delta_z)$ is a random shift and the three components are independently chosen from $W_h(\delta)$ a uniform distribution centered in 0 and of width h

$$W_h(\delta) = \begin{cases} 1/h & \text{if } |\delta| < h/2 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The acceptance ratio for the Metropolis algorithm is given by:

$$A(\delta) = \begin{cases} 1 & \text{if } r'^2 - r^2 < 0 \\ \exp(-\frac{r'^2 - r^2}{2}) & \text{otherwise} \end{cases} \tag{9}$$

Shortly: we jump to a new site and accept this jump if the new site is closer to the origin. If we are further away from the origin we accept the jump with a probability $\exp(-\frac{r'^2 - r^2}{2}) < 1$.

Questions

- Using the Metropolis algorithm plot the trajectory followed by the x, y-coordinates of the points using $N = 10^4$ steps starting from an initial position at the origin $\vec{r} = (0, 0, 0)$ and at the point $\vec{r} = (10, 10, 10)$. Repeat the calculation for $h = 1$ and $h = 0.1$. By plotting the histogram of the positions for a run with $N = 10^6$ points verify indeed that the dynamics generates an equilibrium distribution of points proportional to $\exp(-r^2/2)$.

- Using the Metropolis algorithm estimate numerically the integral and compare its value to that of the previous exercise.

- Depending on the starting point $\vec{r}$ a given number of Metropolis steps N_{eq} are necessary for "equilibration". Estimate N_{eq} for an initial point at $\vec{r} = (10, 10, 10)$ for $h = 1$ and $h = 0.1$.

- Which algorithm do you expect is better for this calculation: the Gaussian random number generator of the first exercice session of the Metropolis algorithm discussed here?

2.2 Stochastic matrices

A $N \times N$ matrix P is called *stochastic* if a) all its elements are non-negative ($\forall i, j \; P_{ij} \geq 0$) and b) the sum of all columns are equal to 1, i.e. $\forall i$:

$$\sum_{i=1}^{N} P_{ij} = 1 \tag{10}$$

Given a system with N states, the stochastic matrix generates the dynamics of the system. The element P_{ij} is the transition rate that the system does in a time interval Δt from the state j to the state i. One also uses the notation $P_{ij} = p(j \to i)$.

Therefore if $w(t)$ is a vector, whose elements $w_i(t)$ are probabilities of finding the system in a state i at time t then at time $t + \Delta t$, the probabilities are the elements of the vector obtained by multiplication with the matrix P:

$$w(t + \Delta t) = Pw(t) \tag{11}$$

Note that a state of the system is described by a vector of non-negative entries $w_i \geq 0$. We will call a non-negative vector a vector whose elements are all non-negative.

Questions

- Show that given a non-negative initial vector w the stochastic matrix generates non-negative vectors at all later times.
- Show that the product of two stochastic matrices is a stochastic matrix
- We introduce now the norm $||w||_1 = \sum_j |w_j|$. Consider an arbitrary vector y (with entries which can also be negative).

  Show that the following inequality holds

  $$\|Py\|_1 \leq \|y\|_1 \tag{12}$$

  and deduce that if λ is eigenvalue of P then $|\lambda| \leq 1$
- Show that a stochastic matrix has at least an eigenvalue $\lambda = 1$. (*Hint: think of a suitable left eigenvector of the stochastic matrix.*)

2.3 Detailed balance

Consider a system with three states with energies $E_1 < E_2 < E_3$. The dynamics is such that the system can make these transitions $1 \to 2$, $2 \to 3$ and $3 \to 1$, or stay.

1) Assigning arbitrary rates to these transitions write the 3×3 stochastic matrix which describes the evolution of the process.

2) Does this dynamics satisfy detailed balance? Is it ergodic?

3) Show that the rates can be chosen such that the equilibrium distribution is the Boltzmann distribution, i.e. that the probability of finding the system in the state i is

$$p_i = \frac{e^{-\beta E_i}}{Z} \tag{13}$$

where $Z = \sum_{i=1}^{3} e^{-\beta E_i}$ is the partition function.

Note: this problem is an example of the fact that "detailed balance" is not a necessary condition to get the Boltzmann distribution as equilibrium of the dynamics

3 Ising and Potts models

3.1 Ising Model: Uniform sampling

In the two dimensional Ising model spins $s_i = \pm 1$ are located at the vertices of a square lattice. The energy of a configuration is given by

$$E(\{s_k\}) = -J \sum_{\langle ij \rangle} s_i s_j \tag{14}$$

where the sum is extended to the four nearest neighboring spins. Each spin has four nearest neighbors, two in the vertical and two in the horizontal directions. The positive constant $J > 0$ represents the strenght of the interaction. We set it here equal to $J = 1$.

We consider now a Monte Carlo algorithm which samples uniformly all configurations of the model, *just to show that this is very inefficient sampling.*

Generating a lattice configuration and boundary conditions

The first thing we have to do is to generate a square lattice, which is not very difficult in most programming languages. We could use e.g. a matrix in Matlab, or a double array in C. One of the things we now must decide is what boundary conditions we will take.

The most used boundary condition is the periodic boundary condition. This is easy to program with e.g. the modulo operation (the symbol % in C, and the function `mod` in Matlab). We indicate in what follows with $s(i, j)$ the value of the spin at lattice position (i, j). With periodic boundary conditions, the neighboring spins are given by[3]:

- The spin above and below are $s(\mathrm{mod}(i \pm 1 + N, N), j)$
- The spin right and left are $s(i, \mathrm{mod}(j \pm 1 + N, N))$

There is however a second type of boundary conditions that is numerically much faster: *helical boundary conditions.* We label now the N^2 spins as a vector $s(i)$ and fill up the lattice from bottom left to top right as in the example in Fig. 1(b). The $N \times N$ lattice is repeated in all directions, but shifted of one unit vertically. The consequences are:

- The spin above and below are $s(\mathrm{mod}(i \pm N + N^2, N^2))$
- The spin left and right are $s(\mathrm{mod}(i \pm 1 + N^2, N^2))$

These are the boundary conditions that we will use.

Uniform sampling

We now go back to the problem of uniform sampling. The algorithm works as follows

Let ω be an empty vector

1) Generate a random configuration of the N^2 spins.
2) Calculate the energy E of the generated configuration from Eq. (14) and update the vector by adding an extra element equal to E to it (in Matlab $\omega = [\omega\ E]$).
3) Go back to 1.

(a)

				(1,4)	(2,4)	(3,1)	(4,4)				
				(1,3)	(2,3)	(3,1)	(4,3)				
				(1,2)	(2,2)	(3,1)	(4,2)				
				(1,1)	(2,1)	(3,1)	(4,1)				
(1,4)	(2,4)	(3,1)	(4,4)	(1,4)	(2,4)	(3,1)	(4,4)	(1,4)	(2,4)	(3,1)	(4,4)
(1,3)	(2,3)	(3,1)	(4,3)	(1,3)	(2,3)	(3,1)	(4,3)	(1,3)	(2,3)	(3,1)	(4,3)
(1,2)	(2,2)	(3,1)	(4,2)	(1,2)	(2,2)	(3,1)	(4,2)	(1,2)	(2,2)	(3,1)	(4,2)
(1,1)	(2,1)	(3,1)	(4,1)	(1,1)	(2,1)	(3,1)	(4,1)	(1,1)	(2,1)	(3,1)	(4,1)
				(1,4)	(2,4)	(3,1)	(4,4)				
				(1,3)	(2,3)	(3,1)	(4,3)				
				(1,2)	(2,2)	(3,1)	(4,2)				
				(1,1)	(2,1)	(3,1)	(4,1)				

(b)

				(13)	(14)	(15)	(16)				
				(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
				(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
(13)	(14)	(15)	(16)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(1)	(2)	(3)	(4)
(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
				(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
								(1)	(2)	(3)	(4)

Figure 1: (a) Periodic boundary conditions. (b) Helical boundary conditions

If we run this algorithm k times we generate a vector containing all the energies of the sampled configurations $\omega = (E_1, E_2, \ldots E_k)$. Instead of writing the total energy in the vector we could also write the energy per bond $e = E/2N^2$. This quantity is probably more informative[4] since it varies in the interval [-1,1], where -1 is the energy of one of the two ground states where spins are either all $s = +1$ or all $s = -1$. Can you draw the configuration with energy $e = 1$? Note: instead of storing the vector with all the values of the energy you can also generate an histogram by dividing [-1,1] in M equally spaced intervals and by binning the values of e in those intervals.

Boltzmann sampling with hit-and-miss method

We use now the hit-and-miss method to generate energies distributed according to $\exp(-\beta E)$, where $\beta = 1/k_B T$ is the inverse temperature. Select a value of β first and generate an empty vector ω. We indicate with E_0 the ground state energy ($E_0 = -2N^2$).

1) Generate a random configuration of the N^2 spins.

2) Calculate the energy E of the generated configuration from Eq. (14).

3) Generate a uniform random number r in $[0, 1]$.

4) If $r \leq \exp(-\beta(E - E_0))$ update the vector ω as above, by adding the energy E (this is a hit!).

5) Go back to 1.

Note that the hit-and-miss method at an infinite temperature ($\beta = 0$) reduces to the uniform sampling discussed above.

Questions

- For a given N (e.g. $N = 16$) generate the histogram of energies per bond ($e = E/2N^2$) using the uniform sampling method running the algorithm a large number of times (say $10^4 - 10^5$).

- Use the hit-and-miss method to generate configurations with temperatures $T = 10^5$, $T = 10^3$, $T = 10$ and $T = 1$. (Note: we will always take $k_B = 1$ thoughout this course). What is for each of the four temperatures the fraction of the number of hits over the total number of configurations generated? Conclude that the hit-and-miss method performs very poorly.

- Another possibility is to use the so-called reweighting technique. From the histogram of the total energy generated by the uniform sampling $P(E)$, we can generate that for a given temperature T by $P_T(E) = \exp(-E/T)P(E)$. Plot the histograms for $T = 10^5$, $T = 10^3$, $T = 10$ and $T = 2$. Does this looks like a reasonable method to you? We will later compare these with those obtained from the Metropolis algorithm.

[3] Be careful with the mod operation on negative numbers.

[4] e is an intensive variable, its range does not change if we change the lattice size.

3.2 Metropolis algorithm for the Ising model

We discuss here the Metropolis algorithm for the Ising model. In this algorithm spin configurations are sampled by a Markov chain, which is characterized by a transition probability $P(\mu \to \nu)$ where μ and ν are two spin configurations. In the algorithm $P(\mu \to \nu) \neq 0$ only if the two configurations differ by a single spin. We initialize the system by selecting a random configuration μ of a $N \times N$ lattice with helical boundary conditions. The practical implementation of the algorithm is as follows:

1) Select one of the N^2 spins at random, say spin $s(i)$.

2) Calculate $\Delta E = E_{\rm fin} - E_{\rm in}$, where $E_{\rm in}$ is the energy of the initial configuration and $E_{\rm fin}$ the energy of the configuration obtained by flipping the spin $s(i) \to -s(i)$.

3) Flip the spin $s(i) \to -s(i)$ if $\Delta E \leq 0$ or if $r \leq \exp(-\Delta E/k_B T)$, where r is a uniform random number in [0,1].

4) Goto 1).

This cycle is repeated a large number of times (e.g. $\sim 10^6 - 10^8$). Usually in Monte Carlo simulations we define the unit of time in sweeps. One sweep corresponds to one attempted step per spin, thus in a lattice with N^2 spins one sweep corresponds to N^2 Monte Carlo steps. Typical observables one computes in the Ising model are the energy, or the magnetization per spin defined as

$$m = \frac{1}{N^2} \sum_i s(i) \tag{15}$$

or spin-spin correlation functions. These quantities reach their equilibrium value after some equilibration time $\tau_{\rm eq}$. Another quantity we compute is the (temporal) autocorrelation function which is defined for a system in which time is a discrete variable

$$\chi(t) \;\; = \;\; \frac{1}{t_f} \sum_{s=\tau_{eq}}^{\tau_{eq}+t_f} [m(s) - \langle m \rangle][m(s+t) - \langle m \rangle] \tag{16}$$

where $m(t)$ is the magnetization at timestep t, and $\langle m \rangle$ is the average equilibrium magnetization. This equation is only valid when the Monte Carlo run has reached an equilibrium configuration ($t > \tau_{\rm eq}$). The sum in Eq. (16) runs from τ_{eq}, a timestep where the simulation definitely has reached equilibrium, until t_f timesteps later, with t_f a reasonably large number of timesteps (e.g. all timesteps between τ_{eq} and the end of the simulation).

This autocorrelation function decreases exponentially like

$$\chi(t) \sim e^{-t/\tau} \tag{17}$$

where τ defines the autocorrelation time of the simulation. This is the timescale of the simulation, and it indicates how much time it takes to arrive at a new uncorrelated configuration of the model.

Questions

- Use the Metropolis algorithm to compute the magnetization per spin defined as

$$m = \frac{1}{N^2} \left| \sum_i s(i) \right| \tag{18}$$

for a temperature $T = 2$ and $N = 50$ as a function of the Monte Carlo time step (you should obtain a plot similar to that shown in Fig. 2 of the lecture notes). Compare the average magnetization with the exact value:

$$m(T) = \left[1 - \sinh^{-4}(2J/T)\right]^{1/8} \tag{19}$$

- From your plot estimate the equilibration time $\tau_{\text{eq.}}$.
- Compute the energy histogram by running the Metropolis algorithm at $T = 2$. To generate the histogram start sampling from times $> \tau_{\text{eq.}}$. Compare your histogram with that obtained from the reweighting method in the previous exercise.
- For $N = 50$, calculate the autocorrelation function as a function of time, if the temperature is taken to be $T = 2.2$. Do this by iterating for a long time, and keep track of the magnetizations at each timestep (sweep). Normalize this autocorrelation function by dividing by $\chi(0)$.
- Determine the correlation time τ from equation (17) from these data.
- **Facultative** - Fix the temperature to the critical value[5] $T = T_c = 2/\log(1+\sqrt{2}) = 2.269\ldots$, and calculate the correlation time τ for lattices of sizes $N \times N$, with $N = 5, 10, 25$. Plot these correlation times as a function of the lattice size. Show that these correlation times grow with N, and that they are much larger than in the case of $T = 2.0$ (this is called critical slowing down). Fit these $T = T_c$ correlation times using the formula

  $$\tau \sim N^z \tag{20}$$

  and estimate the dynamical exponent z.

Some interesting tricks

A Monte Carlo code consists of a central core nucleus which is repeated a large number of times. Therefore we should try to speed up this core to have a very efficient program (with a fast program we can sample a large number of different configurations and thus get an accurate result).

- A common mistake is to compute the exponentials at each Metropolis step. The $\exp(x)$ is an operation which requires a certain amount of polynomial approximation on a computer and it is very slow. Since in the Ising model there are only few discrete values for $\Delta E > 0$ (how many?) it is much faster to calculate these once and store these numbers in a small vector.
- To calculate ΔE we only need to know the 4 spins up, down, right and left of the given spin, because all the other spins are unchanged.
- Often one performs simulations at different temperatures, say $T_1 < T_2 < T_3 \ldots$. To avoid long runs to reach equilibration one can take as begin configuration for the simulation at T_2 the last spin configuration of the run at T_1, and so on.

[5] This is the exact value of the phase transition temperature

3.3 The heat bath algorithm for the Potts model

The Potts model is a generalization of the Ising model, where a spin can have q values: $s_i = 0, 1, 2 \ldots q-1$. The energy of a configuration is given by:

$$E = -J \sum_{\langle ij \rangle} \delta_{s_i s_j} \tag{21}$$

where the sum is extended to the four nearest neighboring spins $J > 0$ is the coupling strength and $\delta_{s_i s_j}$ is a discrete δ-function[6]. The Ising model corresponds to the case $q = 2$ (with a simple variable transformation you can show this). For a general value of $q > 2$ the model has a phase transition from a "ferromagnetic" phase where the majority of the spins assume one of the q possible values, to a configuration at high temperatures, where the spins are equally likely one of the q possible directions.

In this exercise we compare the performance of the standard Metropolis algorithm[7] with that of the heat bath algorithm. The heat bath algorithm is a single spin flip algorithm. The selection probability of two states μ and ν differing for more than a single spin $g(\mu \to \nu) = 0$. If now s_k is the flipping spin we indicate with E_m the energy of the state in which $s_k = m$, which is calculated according to the Hamiltonian (21). Let us consider two configurations μ and ν differing by the spin s_k and let $s_k = n$ in ν then the heat bath algorithm is defined by

$$g(\mu \to \nu) = \frac{1}{qM} \frac{e^{-\beta E_n}}{\sum_{m=0}^{q-1} e^{-\beta E_m}} \tag{22}$$

where M is the total number of spins in the lattice. The heat bath algorithm will thus select a new value for the spin depending on the energy of the new configuration. In this algorithm the acceptance ratio is $A(\mu \to \nu) = 1$ for all states.

Questions

- Show that the transition probabilities defining the heat bath algorithm obey detailed balance and ergodicity.
- For $q = 10$ and starting from a random initial configuration, determine the equilibration time for the heat bath algorithm and the Metropolis algorithm for $T = 0.5$ by plotting the internal energy as a function of the number of iterations. Take the size of the lattice 20×20, and use periodic or helical boundary conditions. You can experiment with other lattice sizes, temperatures and values for q if time allows.
- Which of the algorithms converges the fastest, and how much faster?

We again apply a trick like in the Ising simulations, so that we only need to calculate a few exponentials beforehand, and store them in a small array, thus improving the speed of the algorithm.

[6]The definition is $\delta_{mn} = 1$ if $m = n$ and $\delta_{mn} = 0$ otherwise

[7]The Metropolis algorithm for the q state Potts model is a generalization of that of the Ising model.

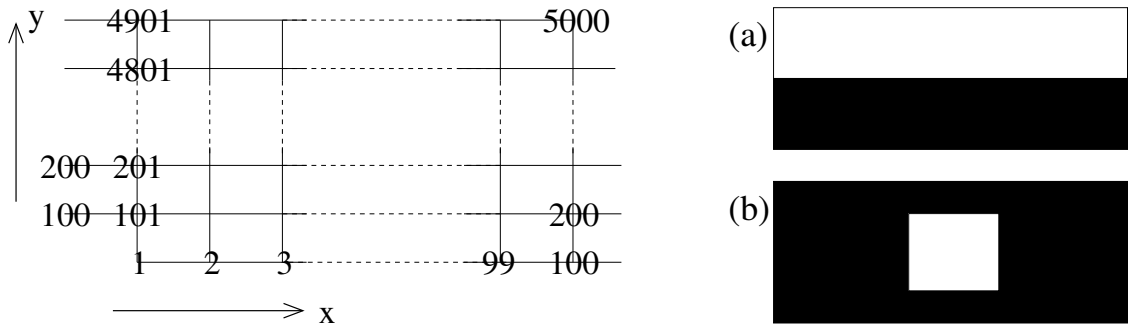

Figure 2: Left: lattice configuration used in the problem. We use helical boundary conditions in the x direction and open boundary conditions in the y direction. Right: start configurations for the simulation (a) half of the lattice filled with spins "plus" and half with spin "minus" separated by a straight horizontal interface and (b) central square of spins "plus" surrounded by spin "minus".

3.4 Kawasaki algorithm: The local version

In this exercise we use the Kawasaki algorithm to study the conserved order parameter Ising model. Differently from the single spin flip Metropolis algorithm, the Kawasaki algorithm conserves the total magnetization of the system:

$$M = \sum_i s_i \tag{23}$$

The idea is very simple: pick up two *neighboring* spins at random. If these are equal do nothing. If they are different try to swap them according to the standard Metropolis rule. We consider an Ising lattice of size $L_x = 200$ and $L_y = 50$. The boundary conditions are fixed in the y-direction[8] and helical in the x-direction as illustrated in Fig. 2.

Questions

- Consider the start configuration of Fig. 2(a) and a temperature equal to 0.9 T_c, where T_c is the critical temperature for the two dimensional Ising model[9]. Run a simulation with the Kawasaki algorithm for N Monte Carlo steps and determine $N_{\rm acc}$ the number of times a swap is actually performed.

- Determine the ratio $N_{\rm acc}/N$ at $T = 0.5T_c, 1.2T_c$ and $2.5T_c$. Does the ratio increases or decreases with temperature? Does this behavior matches your expectations?

- Using the initial configuration of Fig. 2(a), plot the energy per spin as a function of the number of sweeps at $T = 0.9T_c$. From the run determine approximately the equilibration time.

3.5 Kawasaki algorithm: The non-local version

A more efficient simulation algorithm contains non-local moves, in which non-neighboring spins can be swapped. The most efficient way to do this is to create two vectors containing

[8]Fix initially the values of the spins in the top and bottom rows and never select these spins for flipping
[9]The critical temperature is $T_c = 2/\log(1+\sqrt{2}) \approx 2.269$

the list of plus and minus spins, for instance $\vec{u} = (1, 2, 3, 5, 7, 10 \ldots)$ and $\vec{d} = (4, 6, 8, 9 \ldots)$. This means that the spins 1, 2, 3, 5, 7 ..., labeled as in Fig. 2(left) are plus. We also keep the configuration of spins stored in a vector of length $L_x \times L_y$, as done for the Ising model i.e. $\vec{v} = (1, 1, 1, -1, 1, -1, 1, -1, -1, 1 \ldots)$. We select a random spin from the "plus" list and one from the "minus" list, say u_i and d_j, and calculate ΔE, the energy difference between the initial configuration and the one where the spins are swapped. This swap is then accepted according to the Metropolis rule. If the swap is accepted we update the vectors to $u'_i = d_j$ and $d'_j = u_i$, as well as $v'(u_i) = -v(u_i)$ and $v'(d_j) = -v(d_j)$.

Questions

- Using the non-local algorithm estimate once again the acceptance probabilities $N_{\rm acc}/N$ for the three temperatures given above starting from the initialconfiguration of Fig. 2(a). Is this lower or higher compared to the local algorithm?

- What is the equilibration time for $T = 0.9T_c$? Compare it with that obtained in the previous exercise.

We use now as initial condition the configuration in Fig. 2(b).

- Run the Kawasaki algorithm with this initial condition with a square of size 50×50, for $T = 0.25T_c$ and $T = 0.5T_c$. Compare the evolution of the shape of the square at these two temperatures, by plotting the configuration after a few selected numbers of timesteps.

- Consider now a temperature $T = 0.8\ T_c$ and initial squares of sizes 10×10, 20×20 and 30×30. Show that as the simulation is started the smallest square rapidly dissolves, while the largest one persists after a large number of time steps.

4 Continuous systems

4.1 Hard Disks

In this problem we consider a Monte Carlo algorithm for hard spheres in two dimensions (hard disks). The interaction between particles is described by the pair potential

$$V_{\rm HS}(r) = \begin{cases} +\infty & r \leq 2\sigma \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

where r is the distance between two particles and σ is the disk radius. A Metropolis algorithm for this system consists of three steps:

1) Pick up a particle at random, say particle i.

2) Perform a move $\vec{r}_i \to \vec{r}\,'_i = \vec{r}_i + \vec{\Delta}$, where $\vec{\Delta}$ is a random shift.

3) Check that the new position of the particle does not overlap with that of any other particle, i.e. that $|\vec{r}_j - \vec{r}\,'_i| \geq 2\sigma$ for every $j \neq i$. If so accept the move, otherwise reject it.

4) Goto 1).

This is precisely the Metropolis algorithm we have seen for the Ising model. If a move brings two particles to overlap $\Delta E = +\infty$, thus the move is rejected, otherwise $\Delta E = 0$ and the move is always accepted. As a practical implementation we choose the random shift as

$$\Delta_\alpha = (2u - 1)q \tag{25}$$

with $\alpha = \{x, y, z\}$, u a uniform random number in $[0, 1]$ and q a characteristic length. With the choice (25) the algorithm is ergodic.

We consider a $L \times L$ square with periodic boundary conditions. To implement the boundary conditions we perform each move as follows:

$$\vec{r}\,'_i = \vec{r}_i + \vec{\Delta} - L \ \text{int}\left(\frac{\vec{r}_i + \vec{\Delta}}{L}\right) \tag{26}$$

where int(x) denotes the integer part[10] of x (use the same type of approach to compute distances between particles).

The N particles can be placed initially on a regular grid of points or at random, but such that the interparticle distance is always $\geq 2\sigma$.

We choose next the width of a move q (Eq. (25)). If q is very small almost every move is accepted but the update is very slow. We choose q as the typical interparticle distance, thus for N particles in the $L \times L$ square $q \approx \frac{L}{\sqrt{N}}$.

The slowest part of the algorithm is the check that the updated particle position is not overlapping with any of the other $N - 1$ particles. One can alleviate this problem by constructing a "cell list". We divide our L^2 total area in squares of size $r_c \times r_c$. We generate a list which associate to each particle the given cell. If the position update brings the particle in the l-th cell, we check that it is not overlapping with any of the particles in the cell l

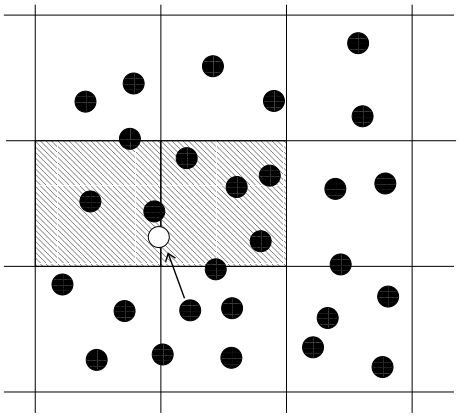

Figure 3: Cell list method: the overlap of the new position is checked against all particles of some neighboring cells.

plus eventually some of its neighboring cells (see Fig. 3). In practice, r_c can be chosen to be somewhat bigger than 2σ, the diameter of the hard disks.

Here our interest in the computation of the pair correlation function defined as

$$g(r) = \frac{L^2}{2\pi r N} \left\langle \sum_i \sum_{j \neq i} \delta(r - r_{ij}) \right\rangle \tag{27}$$

Here r_{ij} is the distance between two pairs of particles. $g(r)$ is the probability of finding a particle at a distance r, given that there is a particle at the origin. Note that for a hard disks fluid $g(r) = 0$ for $r < 2\sigma$. For a low density system (as a gas), $g(r)$ is approximately constant for $r > 2\sigma$. For a fluid at higher densities $g(r)$ has a non-trivial shape, which provides information on the local arrangement of the molecules.

One can compute it by building up an histogram of distances for any two pairs of particles as

$$g(r) \approx \frac{\langle N(r, \Delta r) \rangle}{\rho 2\pi r \Delta r} \tag{28}$$

where Δr is some discrete distance, $\langle N(r, \Delta r) \rangle$ is the average number of pairs with distance in the interval $[r, r + \Delta r]$ and $\rho = N/L^2$ the particle density. It is also possible to use the cell list method to compute this quantity. Given a particle in the l-th cell we only have to check distances with particles in the same in neighboring cells, if we are interested in $g(r)$ up to a certain distance $r_{\max}$.

Questions

- It is customary to define the packing fraction of the fluid as $\eta = N\pi R^2/L^2$, which is the total area occupied by the disks divided by the area of the system. What is the maximal value of η one can have hard disks? (analytical computation)
- Consider a system with $L = 100$ and $R = 2$. Compute the average acceptance ratio of your Monte Carlo moves for some values of η, and some given q.
- Compute $g(r)$ for some values of η. Show that $g(r)$ becomes oscillatory at high η.

[10] In Matlab/Octave use the function **floor**()

- **Facultative** - Setup a program which makes use of the cell list method and compare it with the standard algorithm.

4.2 Monte Carlo in continuous time for a persistent random walk

We consider a one dimensional random walk. In this model a particle can occupy the sites of a one-dimensional infinite lattice $x = \{\ldots -2, -1, 0, 1, 2 \ldots\}$. Initially the particle is placed at position $x = 0$. At each time step (say $\Delta t = 1$) the particle either stays in the same position or it jumps to one of the two neighboring sites. We choose as probabilities $P(x \to x) = 1 - 2\alpha$ and $P(x \to x \pm 1) = \alpha$ with α a parameter. If α becomes small the particle will spend most of its time in a site and will rarely jump to neighboring sites. Let us consider the case $\alpha = 0.01$.

In the case of the continuous time algorithm we estimate first analitically $\Delta t'$, the average time the walker spend on the same lattice point. We choose with equal probability a jump to the right or to the left. At every jump we update the time as $t \to t + \Delta t'$.

Questions

- Given the initial position of the particle at $x = 0$ compute the time it takes to reach $x = 20$ for the first time. Average over many different simulations. Compare the result obtained from the direct (brute force) simulation, with that obtained from the continuous time algorithm explained above.
- What is the difference in run time of the two different codes to reach times $t = 10^6 \Delta t$?

5 Kinetic Monte Carlo

5.1 Birth-annihilation process

Consider a system made of a single particle type, and evolving in time following the two reactions:

$$A \xrightarrow{\lambda} \emptyset \tag{29}$$

$$A \xrightarrow{\mu} AA \tag{30}$$

where λ and μ are the rates. These reaction define the birth-annihilation process.

- Write down and solve the deterministic differential equation which describes the evolution of the average number of particles.
- Simulate the evolution of the system using the Gillespie algorithm, starting from $N = 10,000$ and $N = 100$ particles using $\lambda = 1$, $\mu = 0.2$. Compare the simulations results with the solutions of the deterministic differential equation.
- Starting initially from $N = 100$ particles and using $\lambda = 1$ and $\mu = 0.2$ trace a histogram of extinction times (the extinction time is the time it takes to remove all particles from the system), by repeating sufficiently many times your simulations.

5.2 Lotka-Volterra Model

The Lotka-Volterra model describes the evolution of a predator-prey population. The prey is Y_1 and the predator Y_2. The reactions are:

$$Y_1 \xrightarrow{k_1} 2\ Y_1 \tag{31}$$

$$Y_1 + Y_2 \xrightarrow{k_2} 2\ Y_2 \tag{32}$$

$$Y_2 \xrightarrow{k_3} \emptyset \tag{33}$$

where k_i with $i = 1$, 2 and 3 are the rates. Let us choose the rates $k_1 = 1$, $k_2 = 0.005$ and $k_3 = 0.6$ and start from an initial state with 100 preys and 20 predators.

- Using the Gillespie algorithm produce plots of the trajectory in the plane X_1, X_2 and of $X_1(t)$ together with $X_2(t)$.
- Estimate the average number of oscillations performed in the system, before ending up in one of the two absorbing states (i.e. $X_1 = X_2 = 0$ or $X_1 \to \infty$, $X_2 = 0$)
- How often is the system ending in $X_1 = X_2 = 0$ compared to $X_1 \to \infty$, $X_2 = 0$?
- Determine the distribution of extinction times (here we define as extiction time as the time it takes to reach a state in which either $X_1 = 0$ or $X_2 = 0$).

5.3 Brusselator

We consider here a model of autocatalytic reaction known as the Brusselator. This is characterized by the following reactions[11]

$$\emptyset \xrightarrow{k_1} Y_1 \tag{34}$$

$$Y_1 \xrightarrow{k_2} Y_2 \tag{35}$$

$$2Y_1 + Y_2 \xrightarrow{k_3} 3\ Y_1 \tag{36}$$

$$Y_1 \xrightarrow{k_4} \emptyset \tag{37}$$

where k_i with $i = 1$, 2, 3 and 4 are the rates. We fix them as in the original paper by Gillespie to the values $k_1 = 5 \cdot 10^3$, $k_2 = 50$, $k_3 = 5 \cdot 10^{-5}$ and $k_4 = 5$.

- Using the Gillespie algorithm produce plots of the trajectory in the X_1-X_2 plane and of $X_1(t)$ and $X_2(t)$ using as initial condition $X_1(0) = 1000$ and $X_2(0) = 2000$.
- Repeat the simulations with the three different initial conditions (a) $X_1(0) = 1000$, $X_2(0) = 4000$ (b) $X_1(0) = 5000$, $X_2(0) = 5000$ and (c) $X_1(0) = 500$, $X_2(0) = 500$. Show that the system approaches a "limit cycle" type of oscillation. Compare your results with those reported in Fig. 15 of the Gillespie's paper.

[11] This is a slightly different notation as that used in the original paper by Gillespie, see `http://www.caam.rice.edu/~cox/gillespie.pdf`, Eqs. 44(a-d)