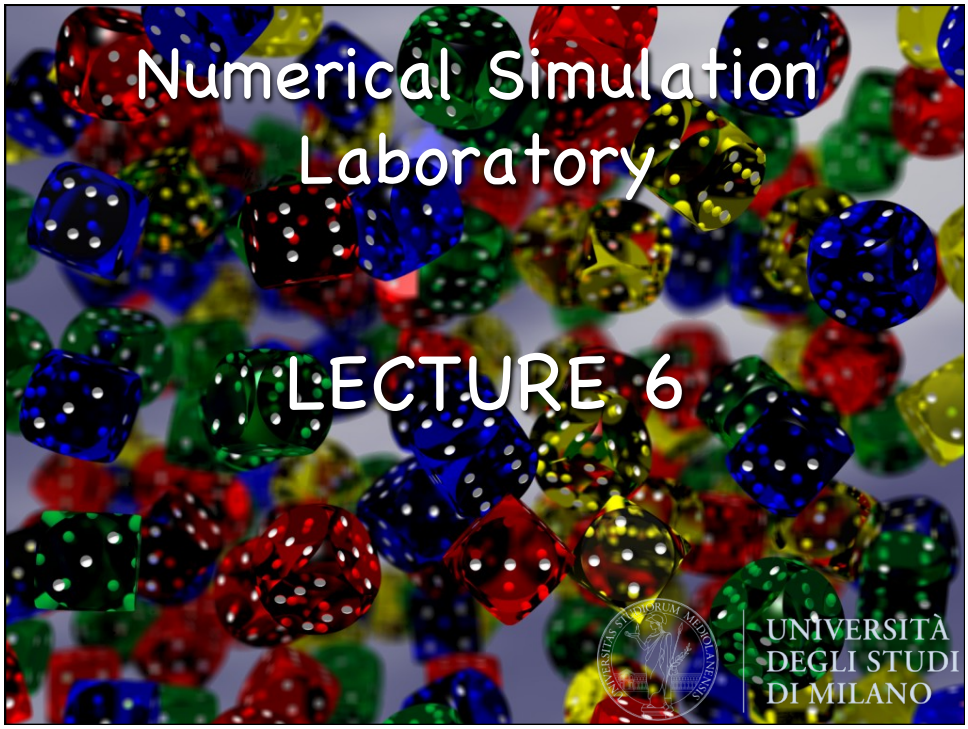


Lecture 6 - outline

- Statistical Mechanics: the canonical ensamble
- Magnetism, the Ising model
- How to use the Metropolis algorithm; a case study: the Ising model
  - Equilibration
  - Measurement
  - Autocorrelation
  - Sparse averaging
  - Data blocking
- Gibbs sampling

The Entropy

- The third/fundamental postulate of statistical mechanics expresses entropy as a function of the accessible volume in phase space, i.e. the volume of the phase space in which the thermodynamic observables have values compatible with a specific thermodynamic state
- Let us assume that the thermodynamic state is determined by the extensive variables (X_1, ..., X_j), e.g. (N,V,E) in the microcanonical ensemble. Each of these variables, X_i for example, is represented (in our hypotheses) by a function $X_i(q,p)$ defined over the space of microscopic states
- The volume of the region of phase space in which these observables have a specific value (within their statistical uncertainty) will be denoted by $\Omega = \Omega(N,V,E)$. Then the fundamental postulate states that: the entropy of a system is

$$S = k_B \ln \Omega(N,V,E)$$

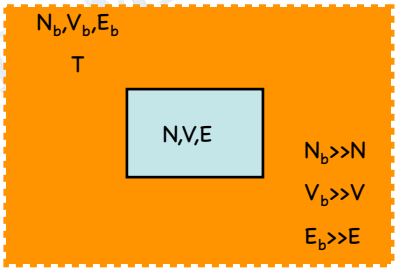
- S is thermodynamic entropy (expressed as a function of extensive variables); k_B=1.3807×10^{-23} J/K is the Boltzmann's constant; The equal sign must be understood as implying "except for non-extensive terms, which become negligible"

Other statistical ensembles

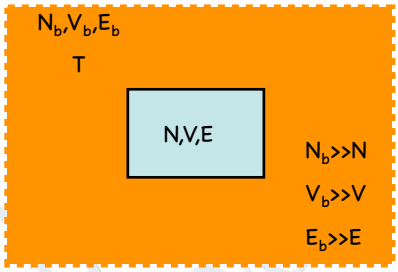
- The microcanonical ensemble, in fact, corresponds to the description of thermodynamic systems, in which the thermodynamic equilibrium states are identified by the values of a collection of fixed extensive variables: the number of particles in the system N, the volume of the system V, the internal energy of the system E
- This microcanonical, i.e. fixed (N,V,E), ensemble corresponds to an experiment in which the thermodynamic system is isolated
- In laboratory you can work at different experimental conditions, e.g. making experiments where the values of some intensive variables as fixed
- In an analogous way, it is possible to define other statistical ensembles, that correspond to these different experimental conditions, and which are able to model the thermodynamic equilibrium for such specific experimental situation
- The most important case is that in which the intensive variable we want to fix is temperature and the corresponding extensive one, which can fluctuate, is the internal energy.

Reservoir: fixed temperature

- To fix the temperature T, we will suppose that we put the system, we are interested in, with N particles in a volume V and Hamiltonian H(q,p) in contact with a much larger system, which we will call reservoir (or heat bath) with $N_b>>N$, $V_b>>V$ and Hamiltonian $H_b(q_b,p_b)$.
- System and reservoir can freely exchange energy. At equilibrium, they will have the same temperature; but since the reservoir is much larger than the system, we can assume that its temperature T does not vary
- In this way, we have imposed a fixed temperature value T on the system, which therefore will be characterized by the triplet (N,V,T)
- Consider a microcanonical ensemble of the composite system with energy $E_{tot}=E+E_b$



- We assume that $E_b >> E$.
- Let $\Omega_b(E_b)$ be the volume occupied by the reservoir in its own phase space.
- The probability of finding our system in a specific microscopic state, i.e. within dq·dp of (q,p), regardless of the state of the reservoir, is proportional to dq·dp $\Omega_b(E_{tot}-E)$ (in fact the composite system is treated in the microcanonical ensemble: every accessible microstate is equiprobable)



- Therefore up to a proportionality constant the density in V space for our system is

$$\rho(q,p) \propto \Omega_b(E_{tot} - E)$$

- Since $\varepsilon_1<<E_{tot}$, we may perform the expansion

$$k_B \ln \Omega_b(E_{tot} - \varepsilon_1) = S_b(E_{tot} - \varepsilon_1) = S_b(E_{tot}) - \varepsilon_1 \left[\frac{\partial S_b(E_b)}{\partial E_b}\right]_{E_b=E_{tot}} + \cdots \approx S_b(E_{tot}) - \frac{\varepsilon_1}{T}$$

where T is the temperature of the reservoir

Canonical ensemble

- Hence $\Omega_b(E_{tot} - E) \approx e^{S_b(E_{tot})/k_B} e^{-E/k_B T}$
- Owing the fact that E=H(q,p) , we may take the ensemble density for our system to be

$$\rho(q,p) = \frac{e^{-H(q,p)/k_B T}}{Q_N(V,T)}$$

- This ensemble density is appropriate for a system whose temperature is determined through contact with a heat reservoir, and is called the canonical ensemble.
- The normalization, i.e. the volume in the phase space occupied by the canonical ensemble is called the partition function:

$$Q_N(V,T) = \int \frac{d^{3N}q d^{3N}p}{h^{3N}N!} e^{-H(q,p)/k_B T}$$

- The average of an observable f=f(q,p) is obtained via

$$\langle f \rangle = \int \frac{d^{3N}q d^{3N}p}{h^{3N}N!} f(q,p) \frac{e^{-H(q,p)/k_B T}}{Q_N(V,T)}$$

7

Partition function-free energy connection

- Strictly speaking we should not integrate over the entire phase space, because $\rho(q,p)$ should vanish if $E>E_{tot}$.
- In the integral for Q_N only few values of the energy H(p,q) contributes and these values will lie in the range where the approximation for Ω_b is valid.
- The thermodynamics of the system is to be obtained from the formula

$$Q_N(V,T) = e^{-A(N,V,T)/k_B T} \quad or \quad A(N,V,T) = -k_B T \ln Q_N(V,T)$$

where A(N,V,T) is the Helmholtz free energy (A=E-TS)

- In fact, we have

$$1 = \frac{Q_N(V,T)}{Q_N(V,T)} = \int \frac{d^{3N}q d^{3N}p}{h^{3N}N!} e^{-\beta[H(q,p)-A(N,V,T)]}$$

8

- Differentiating with respect to β we obtain

$$\int \frac{d^{3N}qd^{3N}p}{h^{3N}N!}\, e^{-\beta[H(q,p)-A(N,V,T)]}\left[A(N,V,T)-H(q,p)+\beta\left(\frac{\partial A}{\partial \beta}\right)_{N,V}\right]=0$$

- This is the same as

$$A = E + T\left(\frac{\partial A}{\partial T}\right)_{N,V}$$

exactly as for the Helmholtz free energy, being $S=-\left(\frac{\partial A}{\partial T}\right)_{N,V}$

- The procedure in the canonical ensemble is:
  - Given the Hamiltonian, compute $Q_N(V,T)$ (most of the time impossible!)
  - Compute the Helmholtz free energy $A(N,V,T)=-k_BT\ \ln[Q_N(V,T)]$
  - The other thermodynamic quantities comes form thermodynamic relations like

$$p(N,V,T)=-\left(\frac{\partial A(N,V,T)}{\partial V}\right)_{N,T} \qquad S(N,V,T)=-\left(\frac{\partial A(N,V,T)}{\partial T}\right)_{N,V}$$

9

The hard task: sampling a multidimensional un-normalized distribution

10

- If we want to exploit the Monte Carlo method to compute stochastically Statistical mechanics observables like

$$\langle f\rangle = \int \frac{d^{3N}qd^{3N}p}{h^{3N}N!}\, f(q,p)\, \frac{e^{-H(q,p)/k_BT}}{Q_N(V,T)}$$

we have to learn how to sample probability distributions like

$$\rho(q,p)=\frac{e^{-H(q,p)/k_BT}}{Q_N(V,T)}$$

without the knowledge of $Q_N(V,T)$! No hope?

- This is not possible via the simple sampling techniques encountered so far …
- But we have see that by using $M(RT)^2$ the normalization goes away in the acceptance step!!!

How-to use M(RT)²

- In the previous lecture we have seen that the Metropolis algorithm, being based on Markov stochastic processes, is powerful but suffers of two fundamental problems:
  - The obtained statistical sampling is correct only asymptotically
  - There is strong correlation between nearby configurations (in Monte Carlo time) visited during the random walk
- We have to learn how to handle with these problems because otherwise we will introduce a bias respectively in the estimator of the average and in the estimator of the error. For example, given

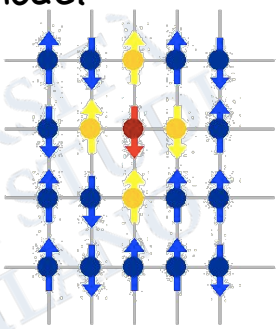
$$I = \int_\Omega g(x)p(x)dx \quad and \quad \sigma_g^2 = \langle g^2 \rangle - \langle g \rangle^2 = \int_\Omega g^2(x)p(x)dx - I^2$$

and N sampled "points" x_i , i=1,...,N, with M(RT)² we will have that

$$I \neq \lim_{N\to\infty}\left[\frac{1}{N}\sum_{i=1}^{N} g(x_i)\right] \quad and \quad err \neq \frac{\sigma_g}{\sqrt{N-1}} \approx \frac{1}{\sqrt{N-1}}\sqrt{\frac{1}{N}\sum_{i=1}^{N} g^2(x_i) - \left[\frac{1}{N}\sum_{i=1}^{N} g(x_i)\right]^2}$$

- Common simple techniques are:
  (1) Equilibration (2) Sparse averaging (3) Data blocking

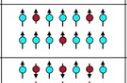
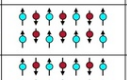
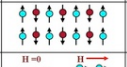
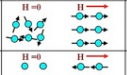
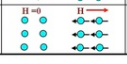
An example: The Ising Model



- To try to make all of this a bit more concrete, we now introduce a particular model which we can try these concepts out on. That model is the Ising model, which is certainly the most thoroughly researched model in the whole of statistical physics.
- Although an exact solution of its properties in 3D still eludes us (it is solved in 1D and 2D), a great deal about it come from MC simulations.
- The Ising model is a model of a magnet. The essential premise behind it is that the magnetism of a bulk material is made up of the combined magnetic dipole moments of many atomic spins within the material. The model postulates a lattice (which can be of any geometry we choose; the simple cubic lattice in three dimensions is a common choice) with a magnetic dipole or spin on each site.
- In the Ising model these spins assume the simplest form possible, which consists of scalar variables s_i which can take only two values ±1, representing up-pointing or down-pointing dipoles of unit magnitude.

Magnets

- In fact, three main types of magnetism are distinguished: diamagnetism, paramagnetism, and ferromagnetism.
- Paramagnetism is a form of magnetism whereby certain materials form internal induced magnetic fields in the direction of the applied magnetic field and are attracted by an externally applied magnetic field. In contrast, diamagnetic materials form induced magnetic fields in the direction opposite to that of the applied magnetic field and are repelled by magnetic fields.
- Ferromagnetism and antiferromagnetism: even in absence of an external magnetic field, a state in which all the magnetic moments μ are aligned in the same direction corresponds to ferromagnetism, if aligned in antiparallel directions, with zero resulting macroscopic magnetization is zero, is antiferromagnetism .

type	spin alignment	spin in simplified plot	examples
ferromagnetic	all spins align parallel to one another: spontaneous magnetization- $M = a + b$		Fe, Co, Ni, Gd, Dy, $SmCo_5$, Sm_2Co_{17}, $Nd_2Fe_{14}B$
ferrimagnetic	most spins parallel to one another, some spins antiparallel: spontaneous magnetization- $M = a - b > 0$		magnetite (Fe_3O_4), yttrium iron garnet (YIG), $GdCo_5$
antiferromagnetic	periodic parallel-antiparallel spin distribution: $M = a - b = 0$		chromium, FeMn, NiO
paramagnetic	spins tend to align parallel to an external magnetic field: $M=0$ @ $H=0$, $M>0$ @ $H>0$		oxygen, sodium, aluminum, calcium, uranium
diamagnetic	spins tend to align antiparallel to an external magnetic field $M=0$ @ $H=0$, $M<0$ @ $H>0$		superconductors, nitrogen, copper, silver, gold, water, organic compounds

13

14

- Given the quantum origin of the magnetism in matter, i.e. orbital motion and intrinsic angular momentum of electrons in atoms/ molecules (spin), the magnetic dipole moment **μ** and its component μ_z in the direction of the applied field cannot have arbitrary values
- Quite generally, we have a direct relationship between the magnetic moment **μ** of a given particle/dipole and its total angular momentum **j** :

$$\vec{\mu} = g\frac{e}{2mc}\vec{j} \qquad j^2 = J(J+1)\hbar^2 \qquad J = \frac{1}{2}, \frac{3}{2}, \cdots \text{ or } J = 0, 1, 2, \cdots$$

- g is known as Lande's g-factor. If the net angular momentum of the dipole is due solely to electron spins, then g = 2; on the other hand, if it is due solely to orbital motions, then g = 1. In general, however, its origin is mixed; g is then given by the formula

$$g = \frac{3}{2} + \frac{S(S+1) - L(L+1)}{2J(J+1)}$$

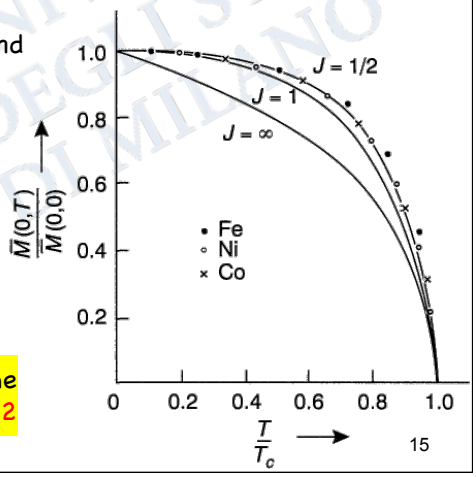
- S and L being, respectively, the spin and the orbital quantum numbers of the particle/dipole
- We thus have $\mu^2 = \left(g\frac{e}{2mc}\right)^2 J(J+1)\hbar^2 = g^2\mu_B^2 J(J+1)$

where $\mu_B = e\hbar/2mc$ is the Bohr magneton

- The component μ_z of the magnetic moment along a specific direction (say z) is, on the other hand, given by

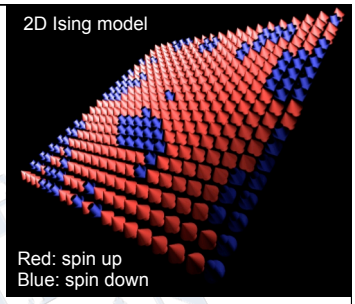
$$\mu_z = g\mu_B m \qquad m = -J, -J+1, \cdots, J-1, J$$

- The presence of a spontaneous magnetization at temperatures below a certain (critical) temperature T_C and its absence above that temperature will then be interpreted as a ferromagnetic phase transition in the system at $T=T_C$
- Detailed studies, both theoretical and experimental, have shown that, for many ferromagnetic materials, data on the temperature dependence of the spontaneous magnetization fit best with the value J=1/2
- Therefore, in discussing the problem of ferromagnetism, we may specifically start by taking: $\mu^2=g^2\mu_B{}^2 s(s+1)$, where s is the quantum number associated with the electron spin, and μ_z can take only 2 values, corresponding to $m=\pm J=\pm 1/2$

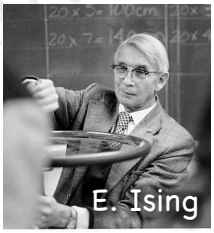


The Ising model

- With s=1/2, only two orientations are possible for each lattice site, namely s_z = +1/2 (with $\mu_z=+\mu_B$) and s_z = -1/2 (with $\mu_z=-\mu_B$). The whole lattice is then capable of 2^N configurations; one such configuration is shown in this picture for a 2D model



- This is the classical Ising model, the most famous model of statistical mechanics. It was invented by W. Lenz (1920), who proposed it as a thesis to one of his students (E. Ising).



- The energy of a configuration is measured by :

$$E = -\tfrac{1}{2}\sum_{i,j=1}^{N} J_{ij}\sigma_i\sigma_j - B\sum_{i=1}^{N}\sigma_i$$

- Where: $\sigma_i \equiv$ component z of the magnetic moment of an electron; $B \equiv$ applied magnetic field; $J_{ij} \equiv$ tendency of electrons to align themselves parallel (or anti-parallel) because of the exchange interaction

- In fact, in a real, magnetic material the spins interact, and the Ising model mimics this by including terms in the Hamiltonian proportional to products $s_i s_j$ of the spins.
- In the simplest case, the interactions are all of the same strength, denoted by J which has the dimensions of an energy, and are only between spins on sites which are nearest neighbours on the lattice.
- We can also introduce an external magnetic field B coupling to the spins. The Hamiltonian then takes the form:

$$H = -J\sum_{\langle ij\rangle} s_i s_j - B\sum_i s_i$$

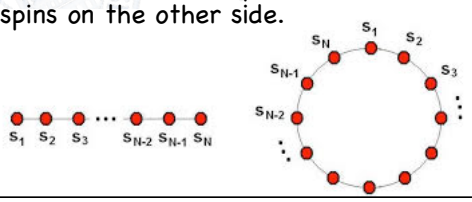
where the notation ⟨ij⟩ indicates that the sites i and j appearing in the sum are nearest neighbours.

- With the signs as they are here, a positive value of J makes the spins want to line up with one another, a ferromagnetic model as opposed to an anti-ferromagnetic one which is what we get if J is negative; the spins also want to line up in the same direction as the external field- they want to be positive if B > 0 and negative if B < 0.

- The states of the Ising system are the different sets of values that the spins can take. Since each spin can take two values, there are a total of 2^N states for a lattice with N spins on it. The partition function of the model is the sum

$$Z = \sum_{s_1=\pm 1}\sum_{s_2=\pm 1}\cdots\sum_{s_N=\pm 1}\exp\left[\beta J\sum_{\langle ij\rangle} s_i s_j + \beta B\sum_i s_i\right] \quad or \quad Z = \sum_{\{s_i\}} e^{-\beta H}$$

- If we can perform this sum then we can apply all the standard results of statistical mechanics to find the free energy, the entropy, the internal energy, the specific heat, and so forth.
- We are interested in simulating an Ising system of finite size using Monte Carlo methods, so that we can estimate the values of quantities at any given temperature $k_B T=1/\beta$.
- By convention we apply periodic boundary conditions (p.b.c), so that there are interactions between spins on the border of the array and the opposing spins on the other side.



19

- Most of the interesting questions concerning the Ising model can be answered by performing simulations in zero magnetic field B = 0, so for the moment at least we will concentrate on this case.
- The Boltzmann statistical weight for a generic state μ is given by

$$p(\mu) = \frac{e^{-\beta H_\mu}}{Z} = \frac{e^{-E_\mu / k_B T}}{\sum_{\nu=1}^{2^N} e^{-E_\nu / k_B T}}$$

- The average value (estimator) for a generic quantity G=G(μ) is given by

$$\langle G \rangle = \sum_{\mu} G(\mu) p(\mu) = \sum_{\mu=1}^{2^N} G(\mu) \frac{e^{-E_\mu / k_B T}}{Z}$$

- Let us look now at how we would actually go about writing a computer program to perform a simulation of the Ising model using the Metropolis algorithm.
- First, we need an actual lattice of spins to work with, so we would define a set of N variables (an array) which can take the values ±1 , so it would be an integer array.

20

- The usage of p.b.c. ensures that all spins have the same number of neighbours and local geometry, and that there are no special edge spins which have different properties from the others; all the spins are equivalent and the system is completely translationally invariant. In practice this simulate an infinite system and considerably improves the quality of the results from our simulation.
- Next we need to decide at what temperature we want to perform our simulation, and we need to choose some starting value for each of the spins: the initial state of the system, say μ.
- In a lot of cases, the initial state we choose is not particularly important, though sometimes a judicious choice can reduce the time taken to come to equilibrium.
- The two most commonly used initial states are the zero-temperature state and the infinite temperature state.
- At T = 0 the Ising model will be in its ground state. When the interaction energy J is greater than zero and the external field B is zero there are actually two ground states.
- These are the states in which the spins are all up or all down. In any other state there will be pairs of spins which contribute +J to the Hamiltonian, so that its overall value will be higher. (If B≠0 then there will only be one ground state, the field ensures that one of the two is favoured over the other)

- The other commonly used initial state is the T = ∞ state. When T = ∞ the thermal energy $k_B T$ available to flip the spins is infinitely larger than the energy due to the spin-spin interaction J, so the spins are just oriented randomly up or down in an uncorrelated fashion.
- Now we start our simulation. The first step is to generate a new state, say ν, by choosing a trial transition probability T(ν|μ).
- For example, the new state could differ from the present one by the flip of just one spin, and every such state should be exactly as likely as every other to be generated. This is an easy task to perform. We just pick a single spin k at random from the lattice to be flipped. Note that, with this choice, T(ν|μ)= T(μ|ν).
- Next, we need to calculate the difference in energy $E_\nu - E_\mu$ between the new state and the old one, in order to apply the Metropolis algorithm

$$A(\nu|\mu) = \min\left[1, \frac{T(\mu|\nu)p(\nu)}{T(\nu|\mu)p(\mu)}\right] = \min\left[1, \frac{p(\nu)}{p(\mu)}\right] = \min\left[1, e^{-\beta(E_\nu - E_\mu)}\right]$$

- The change in energy between the two states is

$$E_\nu - E_\mu = -J\sum_{\langle ij\rangle} s_i^\nu s_j^\nu + J\sum_{\langle ij\rangle} s_i^\mu s_j^\mu = \cdots = -J \sum_{i\ n.n.to\ k} s_i^\mu (s_k^\nu - s_k^\mu) = \cdots$$

in fact the only contributions to this change in energy come from those term that involve the flipped spin. The others stay the same and so cancel out when we take the difference $E_\nu - E_\mu$.

- Now

$$\left.\begin{array}{lll} if & s_k^\nu = -1 & then \quad s_k^\nu - s_k^\mu = -2 \\ if & s_k^\nu = +1 & then \quad s_k^\nu - s_k^\mu = +2 \end{array}\right\} \Rightarrow \quad s_k^\nu - s_k^\mu = -2s_k^\mu$$

therefore an efficient calculation of the difference $E_\nu - E_\mu$ is

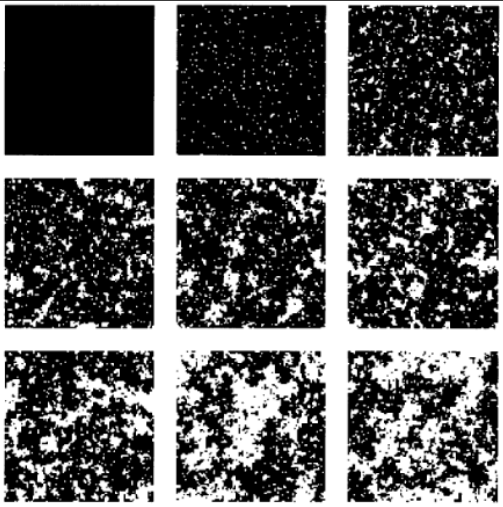
$$E_\nu - E_\mu = \cdots = -J \sum_{i\ n.n.\ to\ k} s_i^\mu (s_k^\nu - s_k^\mu) = 2Js_k^\mu \sum_{i\ n.n.\ to\ k} s_i^\mu$$

- If $E_\nu - E_\mu < 0$ then A(ν|μ)=1 and we definitely accept the move and flip the spin. If $E_\nu - E_\mu > 0$ we still may want to flip the spin. The Metropolis algorithm tells us to flip it with probability $A(\nu|\mu) = \exp[-\beta(E_\nu - E_\mu)]$.
- We can do this as follows. We evaluate the acceptance ratio A(ν|μ) using our value of $E_\nu - E_\mu$, and then we choose a random number r between 0 and 1. If that number is less than our acceptance ratio, r < A(ν|μ), then we flip the spin. If it isn't, we leave the spin alone.
- And that is our complete algorithm. Now we just keep on repeating the same calculations over and over again, choosing a spin, calculating the energy change we would get if we flipped it, then deciding whether to flip it.

Equilibration

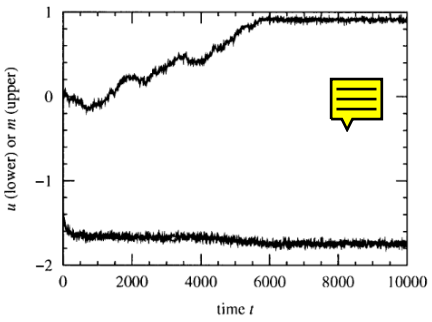
- So what do we do with our Monte Carlo program for the Ising model, once we have written it? Well, we probably want to know the answer to some questions like "What is the magnetization at such-and-such a temperature?", or "How does the internal energy behave with temperature over such-and- such a range?"
- To answer these questions we have to do two things. First we have to run our simulation for a suitably long period of Monte Carlo time until it has come to equilibrium at the temperature we are interested in. Then we have to measure the quantity, $G(\mu)$, we are interested in over another suitably long period of MC-time and average it.
- This leads us to several other questions. What exactly do we mean by "allowing the system to come to equilibrium"? And how long is a "suitably long" time for it to happen? How do we go about measuring our quantity of interest, and how long do we have to average over to get a result of a desired degree of accuracy?
- These are very general questions which we need to consider every time we do a Monte Carlo calculation with a Metropolis algorithm. Although we will be discussing them here using the Ising model simulation as an example, the conclusions we will draw are applicable to all equilibrium Monte Carlo calculations.

- "Equilibrium" means that the average probability of finding our system in any particular state μ is proportional to the Boltzmann weight $p(\mu)$ of that state, i.e., we are correctly sampling $p(\mu)$.
- If we start our system off in a state such as the T=0 or T=∞ states and we want to perform a simulation at some finite non-zero temperature, it will take some MC-time before we reach equilibrium.
- In the version of $M(RT)^2$ which we have described here, we can only flip one spin at a time, and since we are choosing the spins we flip at random, it could take quite a while before we hit on the correct sequence.



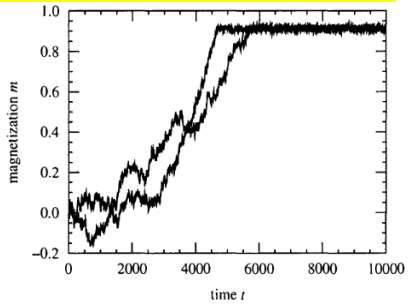
Nine snapshots of a 100×100 Ising model on a square lattice with $J = 1$ coming to equilibrium at a temperature $T = 2.4$ using the Metropolis algorithm. In these pictures the up-spins ($s_i = +1$) are represented by black squares and the down-spins ($s_i = -1$) by white ones. The starting configuration is one in which all the spins are pointing up. The progression of the figures is horizontally across the top row, then the middle row, then the bottom one. They show the lattice after 0, 1, 2, 4, 6, 10, 20, 40 and 100 times 100 000 steps of the simulation. In the last frame the system has reached equilibrium

- However looking at pictures of the lattice is not a reliable way of gauging when the system has come to equilibrium.
- A better way, which takes very little extra effort, is to plot a graph of some quantity of interest, like the magnetization per spin m of the system or the energy of the system E, as a function of MC-time from the start of the simulation.
- Judging the equilibration of a system by eye from a plot is a reasonable method, provided we know that the system will come to equilibrium in a smooth and predictable fashion as it does in the case of the figure beside.



The magnetization (upper curve) and internal energy (lower curve) per site of a two-dimensional Ising model on a square lattice of 100×100 sites with $J = 1$ simulated using the Metropolis algorithm of Section 3.1. The simulation was started at $T = \infty$ (i.e., the initial states of the spins were chosen completely at random) and "cooled" to equilibrium at $T = 2.0$. Time is measured in Monte Carlo steps per lattice site, and equilibrium is reached after about 6000 steps per site (in other words, 6×10^7 steps altogether).

- In some cases it is possible for the system to get stuck in some metastable region of its state space for a while, giving roughly constant values for all the quantities we are observing and so appearing to have reached equilibrium.
- To avoid this potential pitfall, we commonly adopt a different strategy for determining the equilibration MC-time, in which we perform two different simulations of the same system, starting them in different initial states.
- In the case of the Ising model we might, for example, start one in the T=0 state with all spins aligned, and one in the T=∞ state with random spins. Or we could choose two different T=∞ random-spin states.
- We should also run the two simulations with different "seeds" for the random number generator. Then we watch the value of some quantity in the two systems to see when it reaches the same value.



The magnetization of our 100×100 Ising model as a function of time (measured in Monte Carlo steps per lattice site) for two different simulations using the Metropolis algorithm. The two simulations were started off in two different $T = \infty$ (random-spin) states. By about time $t = 6000$ the two simulations have converged to the same value of the mean magnetization, within the statistical errors due to fluctuations, and so we conclude that both have equilibrated.

Measurement

- Once we are sure the system has reached equilibrium, we need to measure whatever quantity G(μ) it is that we are interested in.
- Being N_{eq} the number of Monte Carlo steps we have found necessary to reach the correct sampling of p(μ), we will estimate G via

$$\langle G \rangle \cong G_N = \frac{1}{N - N_{eq}} \sum_{i=N_{eq}+1}^{N} G(\mu_i)$$

- From now on, we will assume that we start from an equilibrated configuration in order to be able to write more simply: $G_N = \frac{1}{N} \sum_{i=1}^{N} G(\mu_i)$
- In order to average quantities, we need to know how long a run we have to average them over to get a good estimate of their expectation values.
- What we really need is a measure of the correlation MC-time t_c of the simulation. The correlation time is a measure of how long it takes the system to get from one state to another one which is significantly different from the first, i.e., a state in which the number of spins which are the same as in the initial state is no more than what you would expect to find just by chance.

Auto-correlation functions

- Usually the equilibration time is considerably longer than the correlation time, because two states close to equilibrium are qualitatively more similar than a state far from equilibrium and one close to equilibrium.
- The most direct of these is to calculate the "time-displaced autocorrelation function" of some property of the model.
- Let us take the example of the magnetization m of our Ising model. The time-displaced autocorrelation χ(t) of the magnetization is given by

$$\chi(t) = \frac{\langle m(t')m(t'+t) \rangle_{t'} - \langle m \rangle^2}{\sigma_m^2} =: Ac_{[m]}(t)$$

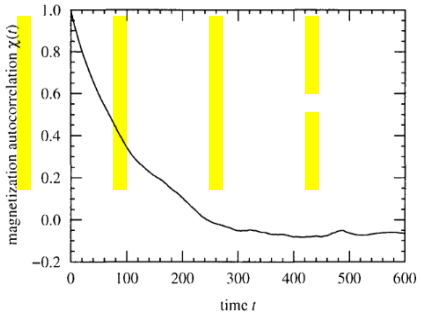
where m(t) is the istantaneous value of the magnetization at (Monte Carlo) time t and ⟨m⟩ is its average value.

- For our Metropolis simulation of the Ising model it is clear that if we measure the magnetization at two times just a single Monte Carlo step apart, the values we get will be very similar, so we will have a large positive autocorrelation.

29

- On the other hand, for two times a long way apart the magnetizations will probably be totally unrelated, and their autocorrelation will be close to zero.
- As we can see in the figure, the autocorrelation does indeed drop from a significant non-zero value at short times t towards 0 at very long times.
- The correlation time is the typical time-scale on which the autocorrelation drops off; the autocorrelation is expected to fall off exponentially at long times thus

$$\chi(t) \approx e^{-t/t_c}$$



The magnetization autocorrelation function $\chi(t)$ for a two-dimensional Ising model at temperature $T = 2.4$ on a square lattice of 100×100 sites with $J = 1$ simulated using the Metropolis algorithm

30

- With this definition, we see that in fact there is still a significant correlation between two samples taken a correlation time apart: at time $t = t_c$ the autocorrelation function, which is a measure of the similarity of the two states, is only a factor of 1/e down from its maximum value at t=0.
- If we want truly independent samples then, we may want to draw them at intervals of greater than one correlation time. In fact, the most natural definition of statistical independence turns out to be samples drawn at intervals of $2t_c$.
- In general, if a run lasts a time t_{max} , then the number of independent measurements we can get out of the run, after waiting a time t_{eq} for equilibration, is on the order of

$$n = t_{max}/2t_c$$

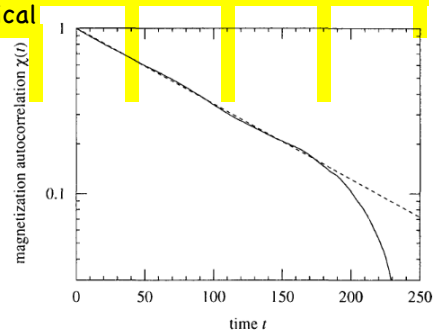


FIGURE 3.6 The autocorrelation function of Figure 3.5 replotted on semi-logarithmic axes. The dashed line is a straight line fit which yields a figure of $\tau = 95 \pm 5$ in this case. Note that the horizontal scale is not the same as in Figure 3.5.

- In Markov chain MC time is discrete; therefore $\chi(t)$ can be calculated directly only for a finite set of MC-times. If we have a set of samples of the magnetization m(t) measured at evenly-spaced times up to some maximum time t_{max} , then the correct formula for the autocorrelation function is

$$\chi(t) = \frac{\frac{1}{t_{max}-t}\sum_{t'=0}^{t_{max}-t} m(t')m(t'+t) - \frac{1}{t_{max}-t}\sum_{t'=0}^{t_{max}-t} m(t') \times \frac{1}{t_{max}-t}\sum_{t'=0}^{t_{max}-t} m(t'+t)}{\frac{1}{t_{max}}\sum_{t'=0}^{t_{max}} m^2(t') - \left(\frac{1}{t_{max}}\sum_{t'=0}^{t_{max}} m(t')\right)^2}$$

- Notice how we have evaluated all the terms using the same subsets of the data. This is not strictly speaking necessary, but it makes $\chi(t)$ a little better behaved.
- When t gets close to t_{max} , the upper limit of the sums becomes small. This means that the statistical errors in $\chi(t)$ due to the random nature of the fluctuations in m(t) may become large.
- Anyway, a typical simulation would always run for many correlation times, in which case we will probably not be interested in the very tails of $\chi(t)$ since the correlations will have died away by then, by definition.

Sparse averaging

- When the evaluation of a quantity $G(\mu)$ on a configuration, μ, produced with a Metropolis random walk, is computationally much more expansive than producing a new configuration, it is smart to use only uncorrelated configurations which carry really new information. This is the sparse averaging technique: one uses only configurations with a "distance" in MC-time around the correlation time t_c or $2t_c$.
- The bad news are that we waste our MC-time in producing configurations that we will never use (this is even worst than what happens with the inefficiency of simple rejection techniques).
- The good news are that, being the configurations uncorrelated, the Central Limit Theorem is satisfied and therefore we already know how to estimate the uncertainty of our calculation via the standard deviation of the mean: $\sigma_G/\sqrt{N}$, being N the number of used configurations.
- However, it is normal practice in a Monte Carlo simulation to make measurements at intervals of less than the correlation time. There are a number of reasons why we do it this way.

33

- First of all, we usually don't know what the correlation time is until after the simulation has finished, or at least until after it has run for a certain amount of Monte Carlo time and we want to be sure of having at least one measurement every two correlation times.
- Another reason is that we want to be able to calculate the autocorrelation function for times less than a correlation time, so that we can use it to make an accurate estimate of the correlation time. If we only had one measurement every $2t_c$, we wouldn't be able to calculate t_c with any accuracy at all.
- But the main reason is, as we shall soon see in the following, that we can use the blocking method or other more sophisticated methods (resampling techniques) to avoid biases in the calculation of the statistical uncertainties
- For this reason one tipically avoid using sparce averaging if the computation of $G(\mu)$ on a configuration μ, is numerically inexpensive with respect to the numerical cost of obtaining the configuration μ; on the contrary one exploits sparce averaging and discard a reasonable amount of configurations before to measure $G(\mu)$.

34

Calculation of errors

- Normally, as well as measuring expectation values, we also want to calculate the errors on those values, so that we have an idea of how accurate they are.
- As with experiments, the errors on Monte Carlo results divide into two classes: statistical errors and systematic errors.
- Statistical errors are errors which arise as a result of random changes in the simulated system from measurement to measurement (thermal fluctuations, for example) and they can he estimated simply by taking many measurements of the quantity we are interested in and calculating the spread of the values.
- In a Monte Carlo calculation the principal source of statistical error in the measured value of a quantity is usually the fluctuation of that quantity from one time step to the next. This error is inherent, in the Monte Carlo method.
- Systematic errors on the other hand, are errors due to the procedure we have used to make the measurements, and they affect the whole simulation.

Systematic errors

35

- Just as in experiments, systematic errors are much harder to handle than statistical ones: they do not show up in the fluctuations of the individual measurements of a quantity. (That's why they are systematic errors.)
- The main source of systematic errors (apart from subtle bugs we have been able to introduce in our simulation code!) in the usage of the Metropolis algorithm is the fact that we wait only a finite amount of time for the system to equilibrate.
- There is no good general method for estimating systematic errors: each source of error has to be considered separately and a strategy for estimating it evolved.
- This is essentially what we were doing when we discussed ways of estimating the equilibration time for the Metropolis algorithm.
- Another possible source of systematic error would be not running the simulation for a long enough time after equilibration to make good independent measurements of the quantities of interest. When we discussed methods for estimating the correlation time t, we were dealing with this problem.

Statistical errors

36

- As we have learned, it is often straightforward to estimate the statistical error in a measured quantity, since the assumption that the error is statistical (i.e., that it arises through random deviations in the measured value of the quantity) implies (Central Limit Theorem) that we can estimate the true value by taking the mean of several different measurements, and that, the error on that estimate is simply the error on the mean.
- Thus, if we are performing the Ising model simulation described in the previous slides, and we make N measurements of the magnetization of the system during a particular run, then our best estimate of the true thermal average of the magnetization is the mean of those N measurements and our best estimate of the standard deviation on the mean is given by

$$\sigma = \sqrt{\frac{1}{N-1}\left(\langle m^2 \rangle - \langle m \rangle^2\right)}$$

- This expression assumes that our samples of the magnetization (or better the N configurations of the spins in which we have found the system in our N measurements) are statistically independent, which in general they won't be.

- As we pointed out, it is normal to sample at intervals of less than a correlation time, which means that successive samples will in general be correlated.
- A simple and usually adequate solution to this problem is to use the value given by $n=t_{max}/2t_c$ (the number of statistically independent samples in t_{max}) rather than the actual number of samples taken:

$$\sigma = \sqrt{\frac{1}{n-1}\left(\langle m^2\rangle - \langle m\rangle^2\right)} \approx \sqrt{\frac{2t_c}{t_{\max}}\left(\langle m^2\rangle - \langle m\rangle^2\right)}$$

- There are some cases where it is either not possible or not straightforward to estimate the error in a quantity using the direct method described here.
- This happens when the result we want is not merely the average of some measurement repeated many times over the course of the simulation, as, for example, the magnetization is, but is instead derived in some more complex way from measurements we make during the run.
- In such cases, there are other more general methods of error estimation which lend themselves to this problem, though it should be clear that they are applicable to almost any quantity that can be measured in a Monte Carlo simulation.

Data blocking

- The simplest of our general-purpose error estimation methods is the blocking method. The idea is that we take the measurements of a quantity that we made during the simulation and divide them into several groups, or blocks. We then calculate its average separately for each block, and the spread of values from one block to another gives us an estimate of the error.
- To see how this works, suppose we make M measurements of a quantity, g, during our Ising model simulation, and then split those into N groups of L=M/N measurements. We can evaluate a specific average, g_i , for each group and then find the mean of those n results exactly as we did for the magnetization above:

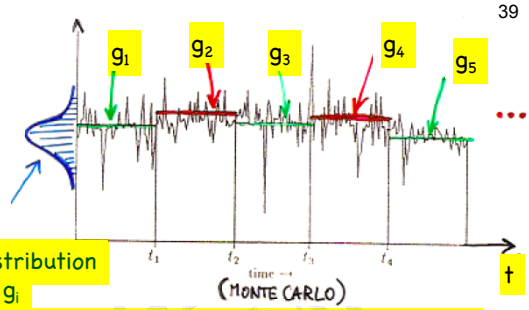
$$\langle g\rangle = \frac{1}{N}\sum_{i=1}^{N} g_i = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{L}\sum_{j=(i-1)L+1}^{(i)L} g(\mu_j)\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{N}{M}\sum_{j=(i-1)M/N+1}^{(i)M/N} g(\mu_j)\right) = \frac{1}{M}\sum_{i=1}^{M} g(\mu_i)$$

- The error on the mean is given again by

$$\sigma_{\langle g\rangle} = \sqrt{\frac{1}{N-1}\left(\langle g^2\rangle - \langle g\rangle^2\right)} = \sqrt{\frac{1}{N-1}\left[\frac{1}{N}\sum_{i=1}^{N} g_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} g_i\right)^2\right]}$$

except that M is now replaced by the number N, of blocks.

- The method works because even if the M measurements are correlated, the average in the blocks is uncorrelated if the blocks are large enough
- One should find the length of the blocks such that averages between blocks are statistically independent; when this is the case, the Central Limit Theorem can be applied and this gives the standard formula



Effect of data blocking during a simulation. Note that the block averages fluctuate much less then the individual estimates of g(μ)

$$\sigma_{\langle g\rangle} = \sqrt{\frac{1}{N-1}\left(\langle g^2\rangle - \langle g\rangle^2\right)} = \sqrt{\frac{1}{N-1}\left[\frac{1}{N}\sum_{i=1}^{N} g_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} g_i\right)^2\right]}$$

- Note that by increasing the number of estimates in the blocks L=M/N (with M fixed), one approaches the true statistical error due to the usage of a Markov chain; once this limit is reached, by increasing furthermore L one should affect the error no more; in fact The Central Limit Theorem is satisfied and one is changing the distribution of g_i (which is narrower) but not $\sigma_{\langle g\rangle}$ because N becomes smaller.

The Gibbs sampling (Heat-bath)

- The Gibbs sampling, also known as the heath-bath algorithm, is conceptually the simplest of the Markov chain sampling methods. It is widely applicable to problems where the variables take on values from a small finite set, or have conditional probability distributions of a form that can easily be sampled from
- In practice it can be seen as a particular case of the Metropolis algorithm where the proposed move is always accepted
- The Gibbs sampler does this by repeatedly replacing each component with a value picked from its distribution conditional on the current values of all other components
- The process can be seen as generating a realization of a Markov chain that is built from a set of base transition probabilities T_k , for k=1,...,n with

$$T_k(\vec{x}|\vec{y}) = p_{1|n-1}(x_k|\{y_i : i \neq k\}) \cdot \prod_{i(\neq k)} \delta(x_i - y_i)$$

i.e., T_k leaves all the components except y_k unchanged and draws a new $y_k \rightarrow x_k$ from its distribution conditional on the current values of all other components. This is assumed to be a feasible operation.

- These base transitions are usually applied in sequence, though at each step we could instead pick T_k at random. For simplicity we will start to discuss it concretely in two dimensions: Suppose we wish to sample from the joint distribution for $\{x_1,x_2\}$ given by $p(x_1,x_2)$, where the range of the x_i may be either continuous or discrete

- Recall that, for example, the conditional probability distribution of x_1 given x_2 is given by the ratio $p_{1|1}(x_1|x_2)=p(x_1,x_2)/p_1(x_2)$ where

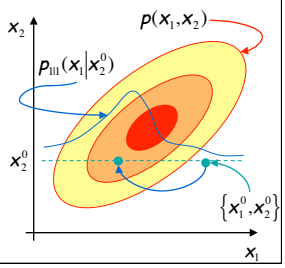
$$p_1(x_2) = \int dx_1\, p(x_1,x_2)$$

is the marginal distribution for the x_2 variable. Recall also that $p_{1|1}(x_1|x_2)$ is a probability distribution once x_2 has been fixed.

- The first step of the Gibbs sampling algorithm consists in choosing a starting position as it is usually done also for the M(RT)^2 : $\{x_1^0, x_2^0\}$

in keeping fixed the variable x_2: $x_2 = x_2^0$

and in sampling x_1 from: $p_{1|1}(x_1|x_2^0)$

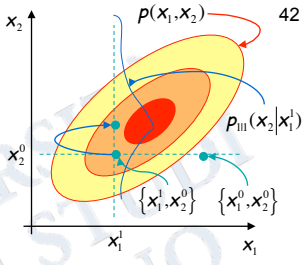


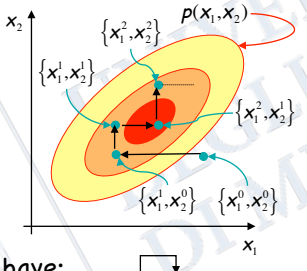
- The second step consists in keeping fixed the variable x_1 and in sampling x_2 from the probability distribution

$$p_{1|1}(x_2|x_1^1)$$

- After that we have moved all the variables the algorithm restart, realizing the sampling of $p(x_1,x_2)$:

the algorithm will works if we are able to sample from the T_k's





- Generalizing in $\mathbb{R}^n$ we have:

(Note that the new value for x_{i-1} is used immediately when picking the next value for x_i)

repeat

$$\begin{aligned}
x_1^{t+1} &\leftarrow p_{1|n-1}(x_1|x_2^t,x_3^t,\cdots,x_n^t)\\
x_2^{t+1} &\leftarrow p_{1|n-1}(x_2|x_1^{t+1},x_3^t,\cdots,x_n^t)\\
x_3^{t+1} &\leftarrow p_{1|n-1}(x_3|x_1^{t+1},x_2^{t+1},x_4^t,\cdots,x_n^t)\\
&\vdots \quad\quad \vdots \quad\quad \vdots\\
x_n^{t+1} &\leftarrow p_{1|n-1}(x_n|x_1^{t+1},x_2^{t+1},\cdots,x_{n-1}^{t+1})
\end{aligned}$$

The proof of the convergence of the Gibbs' sampling is present in the supplementary material

Gibbs sampling and the Ising model

- The Ising model provides a simple example of Gibbs sampling. To simulate this system, as for the Metropolis algorithm, we first select starting values for the spins, s_i , according to some initial distribution, a starting configuration that we must equilibrate
- We then visit the various spins repeatedly, either in some predefined order, or by picking a spin at random each time.
- When spin s_i is visited, a new value for it is chosen, independently form its actual value, from the conditional distribution defined by the other n.n. spins (heat-bath):

$$p(s'_k \,|\, \{s_j : j \neq k\}) := \frac{p(s_1,\cdots,s'_k,\cdots,s_N)}{p(s_1,\cdots,s_k=+1,\cdots,s_N)+p(s_1,\cdots,s_k=-1,\cdots,s_N)} = \frac{e^{\beta J \sum_{\langle ij\rangle} s_i s_j}\Big|_{s_k=s'_k}}{e^{\beta J \sum_{\langle ij\rangle} s_i s_j}\Big|_{s_k=+1} + e^{\beta J \sum_{\langle ij\rangle} s_i s_j}\Big|_{s_k=-1}}$$

joint (numerator); conditional (left of $|$); marginal (denominator)

- Thus

$$p(s'_k = \pm 1 \,|\, \{s_j : j \neq k\}) = \frac{1}{1+\exp\left(-\beta \Delta E_{s_k=\mp 1 \to s_k=\pm 1}\right)} = \frac{1}{1+\exp\left(\mp 2\beta J \sum_{i(n.n.\,to\,k)} s_i\right)}$$

- Such transition probabilities, once computed, let us to sample s_k always accepting the "proposed" move! And so on for the other spins.

A remark

- In fact, by discussing the Gibbs sampling in relation with the Ising model we have seen that the conditional probability distribution used to propose the move is of the form

$$T(x|y) = \frac{p(x)}{p(x)+p(y)}$$

we can see that such transition probability corresponds to a "trial" stochastic nucleus that proposes a move which is always accepted with probability 1

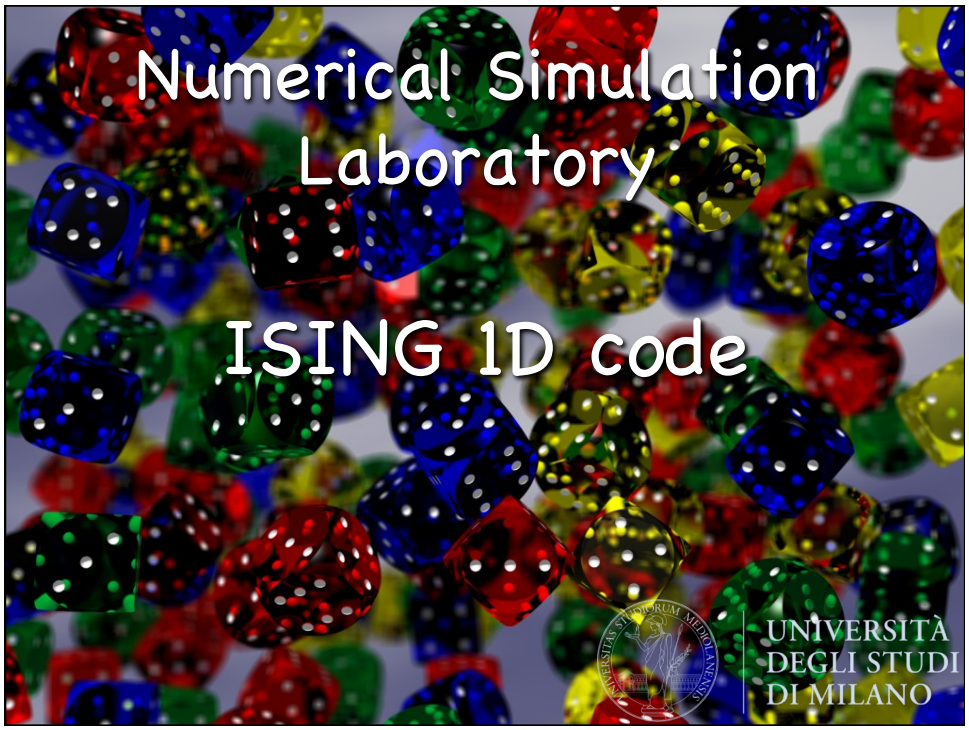
- Note that this functional form has been already proposed as part of a non-Metropolis acceptance probability as a different possibility to build a stochastic nucleus satisfying the detailed balance condition
- Now we can try to build a Metropolis algorithm (not necessarily connected with the Ising model, i.e. more generic) with such trial transition probability; the Metropolis acceptance turns out to be:

$$A(x|y) = \min\left[1, \frac{T(y|x)p(x)}{T(x|y)p(y)}\right] = \min\left[1, \frac{\frac{p(y)}{p(y)+p(x)}p(x)}{\frac{p(x)}{p(x)+p(y)}p(y)}\right] = \min[1,1] = 1 \qquad \text{c.v.d.}$$

thus it should be always accepted! We can use this sampling only if we are able to sample from such T(x|y) (and not from p(x) ...)

Lecture 6: Suggested books

- Newman, Barkema *Monte Carlo Methods in Statistical Physics* – Oxford 2001
- Kalos, Whitlock *Monte Carlo Methods* – Wiley 1986
- R. K. Pathria, "Statistical mechanics" – Oxford
- L. Peliti, "Appunti di meccanica statistica" – Bollati Boringhieri
- R. Piazza "Note di Fisica Statistica" – Springer



Ising 1D code: main

```
#include <iostream>       // cin, cout: Standard Input/Output Streams Library
#include <fstream>        // File stream classes
#include <ostream>        // Standard output stream class
#include <cmath>          // pow, sqrt, … etc.
#include <iomanip>        // setw
#include "Monte_Carlo_ISING_1D.h"

using namespace std;

int main()
{
  Input();                                  //Inizialization
  for(int iblk=1; iblk <= nblk; ++iblk)     //Simulation
  {
    Reset(iblk);                            //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move(metro);                          //Metropolis algorithm
      Measure();                            //Measure properties
      Accumulate();                         //Update block averages
    }
    Averages(iblk);                         //Print results for current block
  }
  ConfFinal();                              //Write final configuration

  return 0;
}
```

47

Ising 1D code: Input()

```
#include <iostream>       // cin, cout: Standard Input/Output Streams Library
#include <fstream>        // Stream class to both read and write from/to files
#include <ostream>        // Stream class to both read and write from/to files
#include <cmath>          // pow, … etc.
#include <iomanip>
#include "Monte_Carlo_ISING_1D.h"

using namespace std;

int main()
{
  Input();                                  //Inizialization
  for(int iblk=1; iblk <= nblk; ++iblk)     //Simulation
  {
    Reset(iblk);                            //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move(metro);                          //Metropolis algorithm
      Measure();                            //Measure properties
      Accumulate();                         //Update block averages
    }
    Averages(iblk);                         //Print results for current block
  }
  ConfFinal();                              //Write final configuration

  return 0;
}
```

48

Ising 1D code: Input()

```
void Input(void)
{
  ifstream ReadInput;

  cout << "Classic 1D Ising model             " << endl;
  cout << "Monte Carlo simulation             " << endl << endl;
  cout << "Nearest neighbour interaction      " << endl << endl;
  cout << "Boltzmann weight exp(- beta * H ), beta = 1/T " << endl << endl;
  cout << "The program uses k_B=1 and mu_B=1 units " << endl;

//Read seed for random numbers
   int p1, p2;
   ifstream Primes("Primes");
   Primes >> p1 >> p2 ;
   Primes.close();

   ifstream input("seed.in");
   input >> seed[0] >> seed[1] >> seed[2] >> seed[3];
   rnd.SetRandom(seed,p1,p2);
   input.close();

//Read input informations
  ReadInput.open("input.dat");

  ReadInput >> temp;
  beta = 1.0/temp;
  cout << "Temperature = " << temp << endl;

  ReadInput >> nspin;
  cout << "Number of spins = " << nspin << endl;

  ReadInput >> J;
  cout << "Exchange interaction = " << J << endl;
```


Ising 1D code: Input() …continue…

```
  ReadInput >> h;
  cout << "External field = " << h << endl << endl;

  ReadInput >> metro; // if=1 Metropolis else Gibbs
  ReadInput >> nblk;
  ReadInput >> nstep;

  if(metro==1) cout << "The program perform Metropolis moves" << endl;
  else cout << "The program perform Gibbs moves" << endl;
  cout << "Number of blocks = " << nblk << endl;
  cout << "Number of steps in one block = " << nstep << endl << endl;
  ReadInput.close();

//Prepare arrays for measurements
  iu = 0; //Energy
  ic = 1; //Heat capacity
  im = 2; //Magnetization
  ix = 3; //Magnetic susceptibility

  n_props = 4; //Number of observables

//initial configuration
  for (int i=0; i<nspin; ++i)
  {
    if(rnd.Rannyu() >= 0.5) s[i] = 1;
    else s[i] = -1;
  }

//Evaluate energy etc. of the initial configuration
  Measure();
//Print initial values for the potential energy and virial
  cout << "Initial energy = " << walker[iu]/(double)nspin << endl;
}
```

Ising 1D code: Reset


```
#include <iostream>       // cin, cout: Standard Input/Output Streams Library
#include <fstream>        // Stream class to both read and write from/to files
#include <ostream>        // Stream class to both read and write from/to files
#include <cmath>          // pow, … etc.
#include <iomanip>
#include "Monte_Carlo_ISING_1D.h"

using namespace std;

int main()
{
  Input();
  for(int iblk=1; iblk <= nblk; ++iblk)
  {
    Reset(iblk);
    for(int istep=1; istep <= nstep; ++i
    {
      Move(metro);
      Measure();
      Accumulate();
    }
    Averages(iblk);
  }
  ConfFinal();

  return 0;
}
```

```
void Reset(int iblk) //Reset block averages
{

   if(iblk == 1)
   {
       for(int i=0; i<n_props; ++i)
       {
           glob_av[i] = 0;
           glob_av2[i] = 0;
       }
   }

   for(int i=0; i<n_props; ++i)
   {
     blk_av[i] = 0;
   }
   blk_norm = 0;
   attempted = 0;
   accepted = 0;
}
```

Ising 1D code: Move

```
#include <iostream>
#include <fstream>
#include <ostream>
#include <cmath>
#include <iomanip>
#include "Monte_Carlo_

using namespace std;

int main()
{
  Input();
  for(int iblk=1; iblk
  {
    Reset(iblk);
    for(int istep=1; i
    {
      Move(metro);
      Measure();
      Accumulate();
    }
    Averages(iblk);
  }
  ConfFinal();

  return 0;
}
```

```
void Move(int metro)
{
  int o;
  double p, energy_old, energy_new, sm;
  double energy_up, energy_down;

  for(int i=0; i<nspin; ++i){
  //Select randomly a particle (for C++ syntax, 0 <= o <= nspin-1)
    o = (int)(rnd.Rannyu()*nspin);

    if(metro==1){ //Metropolis
// INCLUDE YOUR CODE HERE
    }
    else //Gibbs sampling
    {
// INCLUDE YOUR CODE HERE
    }
  }
}

double Boltzmann(int sm, int ip)
{
  double ene = -J * sm * ( s[Pbc(ip-1)] + s[Pbc(ip+1)] ) - h * sm;
  return ene;
}

int Pbc(int i)  //Algorithm for periodic boundary conditions
{
    if(i >= nspin) i = i - nspin;
    else if(i < 0) i = i + nspin;
    return i;
}
```

Ising 1D code: Measure


```
#include <iostream>       // cin, cout: Standard Input/Output Streams Library
#include <fstream>        // Stream class to both read and write from/to files
#include <ostream>        // Stream class to both read and write from/to files
#include <cmath>          // pow, … etc.
#include <iomanip>
#include "Monte_Carlo_ISING_1D.h"

using namespace std;

int main()
{
  Input();
  for(int iblk=1; ibl
  {
    Reset(iblk);
    for(int istep=1;
    {
      Move(metro);
      Measure();
      Accumulate();
    }
    Averages(iblk);
  }
  ConfFinal();

  return 0;
}
```

```
void Measure()
{
  int bin;
  double u = 0.0, m = 0.0;

//cycle over spins
  for (int i=0; i<nspin; ++i)
  {
     u += -J * s[i] * s[Pbc(i+1)] - 0.5 * h * (s[i] + s[Pbc(i+1)]);
// INCLUDE YOUR CODE HERE
  }
  walker[iu] = u;
// INCLUDE YOUR CODE HERE
}

int Pbc(int i)  //Algorithm for periodic boundary conditions
{
    if(i >= nspin) i = i - nspin;
    else if(i < 0) i = i + nspin;
    return i;
}
```

Ising 1D code: Accumulate


```
#include <iostream>       // cin, cout: Standard Input/Output Streams Library
#include <fstream>        // Stream class to both read and write from/to files
#include <ostream>        // Stream class to both read and write from/to files
#include <cmath>          // pow, … etc.
#include <iomanip>
#include "Monte_Carlo_ISING_1D.h"

using namespace std;

int main()
{
  Input();                             //Inizialization
  for(int iblk=1; iblk <= nblk; ++iblk)   //Simulation
  {
    Reset(iblk);                       //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move(metro);                     //Metropolis algorithm
      Measure();                       //Measure properties
      Accumulate();                    //Update block averages
    }
    Averages(iblk);
  }
  ConfFinal();

  return 0;
}
```

```
void Accumulate(void) //Update block averages
{

   for(int i=0; i<n_props; ++i)
   {
     blk_av[i] = blk_av[i] + walker[i];
   }
   blk_norm = blk_norm + 1.0;
}
```

Ising 1D code: Averages

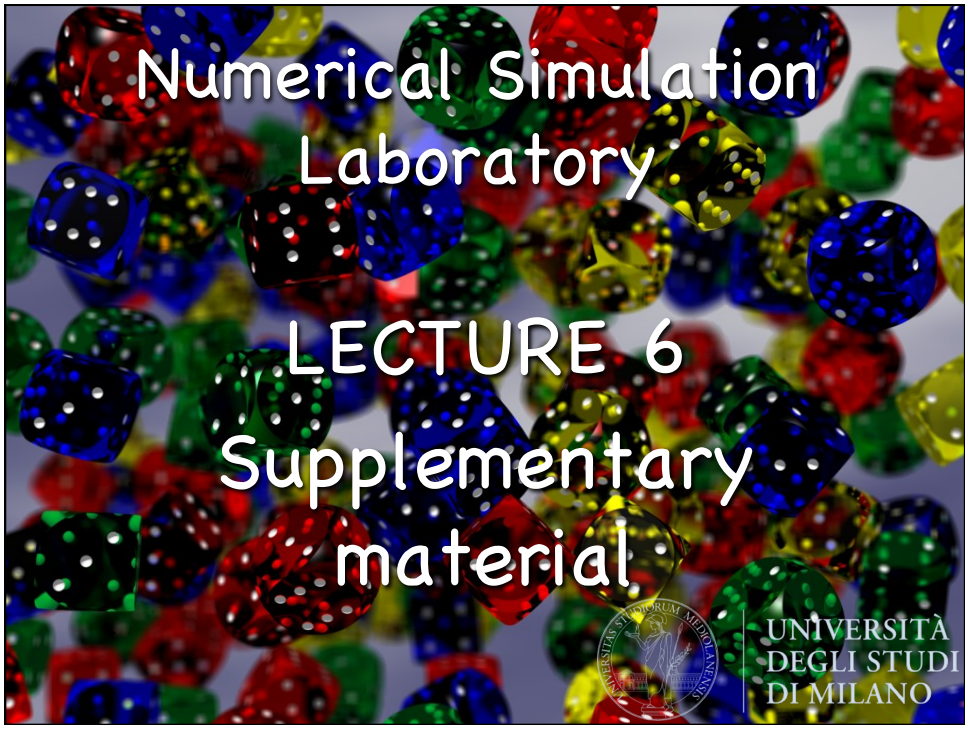

```
#include <iostream>
#include <fstream>
#include <ostream>
#include <cmath>
#include <iomanip>
#include "Monte_Carlo_

using namespace std;

int main()
{
  Input();
  for(int iblk=1; iblk
  {
    Reset(iblk);
    for(int istep=1; is
    {
      Move(metro);
      Measure();
      Accumulate();
    }
    Averages(iblk);
  }
  ConfFinal();

  return 0;
}
```

```
void Averages(int iblk) //Print results for current block
{

   ofstream Ene, Heat, Mag, Chi;
   const int wd=12;

    cout << "Block number " << iblk << endl;
    cout << "Acceptance rate " << accepted/attempted
         << endl << endl;

    Ene.open("output.ene.0",ios::app);
    stima_u = blk_av[iu]/blk_norm/(double)nspin; //Energy
    glob_av[iu]  += stima_u;
    glob_av2[iu] += stima_u*stima_u;
    err_u=Error(glob_av[iu],glob_av2[iu],iblk);
    Ene << setw(wd) << iblk <<  setw(wd) << stima_u << setw(wd)
        << glob_av[iu]/(double)iblk << setw(wd) << err_u << endl;
    Ene.close();

// INCLUDE YOUR CODE HERE

    cout << "----------------------------" << endl << endl;
}

double Error(double sum, double sum2, int iblk)
{
    if(iblk == 1) return 0.0;
    else return sqrt((sum2/(double)iblk -
                pow(sum/(double)iblk,2))/(double)(iblk-1));
}
```

Ising 1D code: ConfFinal


```
#include <iostream>       // cin, cout: Standard Input/Output Streams Library
#include <fstream>        // Stream class to both read and write from/to files
#include <ostream>        // Stream class to both read and write from/to files
#include <cmath>          // pow, … etc.
#include <iomanip>
#include "Monte_Carlo_ISING_1D.h"

using namespace std;

int main()
{
  Input();                          //Inizialization
  for(int iblk=1; iblk <= nblk; ++iblk)   //Simulation
  {
    Reset(iblk);
    for(int istep=1; istep
    {
      Move(metro);
      Measure();
      Accumulate();
    }
    Averages(iblk);
  }
  ConfFinal();

  return 0;
}
```

```
void ConfFinal(void)
{
  ofstream WriteConf;

  cout << "Print final configuration to file config.final "
       << endl << endl;
  WriteConf.open("config.final");
  for (int i=0; i<nspin; ++i)
  {
    WriteConf << s[i] << endl;
  }
  WriteConf.close();

  rnd.SaveSeed();
}
```



- We can now prove the convergence of the Gibbs sampler by showing again that this algorithm is nothing more that a particular case of the Metropolis algorithm with acceptance acc=1
- A trial stochastic nucleus T_k , that cyclically runs over all the components, k=1,…,n , and that propose a move for the k-th component of a vector $\{x_1,x_2,\ldots,x_n\}$ as in the Gibbs algorithm is:

$$T_k(\vec{x}'|\vec{x}) = T_k(x_1,\cdots,x'_k,\cdots,x_n|x_1,\cdots,x_k,\cdots,x_n) = p_{1|n-1}(x'_k|\{x_j : j\neq k\})$$

  x and x′ differ only for the k-th component.
- It is therefore important to note that

$$T_k(\vec{x}'|\vec{x}) = p_{1|n-1}(x'_k|\{x_j : j\neq k\}) = p_{1|n-1}(x'_k|\{x'_j : j\neq k\})$$

  because $x'_j = x_j \quad \forall j\neq k$
- In order to compute the Metropolis acceptance probability it is useful to express the joint probability distributions in terms of the conditional probabilities:

  where $p(x_1,\cdots,x_n) = p_{1|n-1}(x_k|\{x_j : j\neq k\})p_{n-1}(\{x_j : j\neq k\})$

$$p_{n-1}(\{x_j : j\neq k\}) = \int dx_k\, p(x_1,\cdots,x_n)$$

  is the marginal distribution of all the x_j for j≠k

- Due to the fact that by using T_k only the k-th component is involved in the move, one has

$$p(\vec{x}') = p(x_1,\cdots,x'_k,\cdots,x_n) = p_{1|n-1}(x'_k|\{x'_j : j \neq k\})p_{n-1}(\{x'_j : j \neq k\})$$
$$= p_{1|n-1}(x'_k|\{x_j : j \neq k\})p_{n-1}(\{x_j : j \neq k\})$$

- Now we are able to compute the Metropolis acceptation of a move proposed with the stochastic nucleus T_k:

$$A_k(\vec{x}'|\vec{x}) = \min\left[1,\frac{T_k(\vec{x}|\vec{x}')p(\vec{x}')}{T_k(\vec{x}'|\vec{x})p(\vec{x})}\right] = \min\left[1,\frac{p_{1|n-1}(x_k|\{x'_j : j \neq k\})p(\vec{x}')}{p_{1|n-1}(x'_k|\{x_j : j \neq k\})p(\vec{x})}\right] =$$

$$= \min\left[1,\frac{p_{1|n-1}(x_k|\{x_j : j \neq k\})p(\vec{x}')}{p_{1|n-1}(x'_k|\{x_j : j \neq k\})p(\vec{x})}\right] =$$

$$= \min\left[1,\frac{p_{1|n-1}(x_k|\{x_j : j \neq k\})p_{1|n-1}(x'_k|\{x_j : j \neq k\})p_{n-1}(\{x_j : j \neq k\})}{p_{1|n-1}(x'_k|\{x_j : j \neq k\})p_{1|n-1}(x_k|\{x_j : j \neq k\})p_{n-1}(\{x_j : j \neq k\})}\right] =$$

$$= \min[1,1] = 1$$

thus Gibbs is a Metropolis where moves are always accepted, and this guarantees the convergence of the sampling to p(x).