3rd problem set on Computer Simulations, 2024

Hand-in: see schedules for your group, linked in the TeachCenter.
Please upload/submit your program files in the TeachCenter *before* the conversation with the tutor.

This exercise is an example for a typical application of computer simulations. There are several phases, namely (after initial problem definition, model building and choice of method) it encompasses writing and verifying a program, simulations, error analysis, and physics analysis. The main emphasis is on producing reliable *results, with reliable error estimates, and on how to find the limits of applicability of a given method.*

When one performs calculations carefully in this way (and often with better MC update methods), Monte Carlo simulations are the method which can produce reliable *results for the largest systems, by far, (e.g. millions of sites) even in very difficult cases like Quantum Spin systems.*

Simulation of the Ising model

Write and apply a program to simulate the two-dimensional Ising model on a square lattice. The Hamilton function is

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

The "critical temperature" of the two-dimensional Ising model is known exactly. One finds $\tanh^2(2J\beta_c) = \frac{1}{2} \;\Rightarrow\; J\beta_c = 0.44068\ldots$, i.e. $\frac{k_B T_c}{J} \simeq 2.269\ldots$.
At $T < T_c$ the Ising model is in a ferromagnetic phase, at $T > T_c$ it is paramagnetic. You should perform simulations in the range $0 \leq \beta J \leq 0.6$. (The unit of energy „$J = 1$" is often not written.) Read the whole problem set before beginning.

1 Program

(1.5 points) Use the Metropolis algorithm, as explained in the lecture notes. *In order to facilitate writing an efficient program, the following guide is quite detailed.* The program should consist of the following major parts, called from a main program.

1. *Lattice geometry:* The lattice has periodic boundary conditions, i.e. the topology of a torus, and size $L \times L$. First you should generate an array *neighbor(i,nu)*, where $i = 1, 2, \ldots, L^2$ enumerates the lattice sites, and $nu = 1, 2, 3, 4$ denotes the directions to the nearest neighbors of site i. Then $j = neighbor(i, nu)$ is the respective nearest neighbor site. You can use the *matlab*–script *get_neighbor()* supplied on the web site of this class: *neighbor = get_neighbor(L)*. This way one *separates* the lattice geometry from the update algorithm ! On the one hand, this approach saves time, since the location of neighbors does not need to be calculated every time. On the other hand, it permits to simulate other lattice geometries very easily, basically by changing the array *neighbor(i,nu)*.

2. *Update routine:* Metropolis-Hastings algorithm, as described in the lecture notes.

   Write a subroutine "sweep", which repeats L^2 times: select a site at random, and call a subprogram „Metropolis-Single-Spin-Flip" for that site.

   Note: since only a single spin s_i is proposed to be flipped, in „Metropolis-Single-Spin-Flip" you do not need the whole sum $\sum_{\langle ij \rangle} s_i s_j$, but only the sum $\sum s_j$ over the spins of nearest neighbors of site i, which you can obtain easily by using the array "neighbor". This sum can only take a few discrete values, for which you can tabulate the corresponding Metropolis probabilities beforehand. (The expression $\sum_{\langle ij \rangle}$ means that every geometric edge of the lattice appears *once* in the sum. For a periodic $L \times L$ lattice, the sum then contains $2L^2$ terms.)

3. *Measurements:* Measure energy E and magnetization M of the whole lattice after each sweep, then normalize by $1/L^2$, i.e. per lattice site. During measurement, also calculate *averages* and the "naive" *variance* of averages, i.e. the variance which would be correct in the absence of autocorrelations. This is the variance of individual measurements, divided by the number of measurements. Its square root is the "naive" error. Note that in order to calculate this variance on the fly, you only need a sum of measured values and a sum of their squares, both of which can be updated cheaply.

4. *Visualisation:* If possible, display the spin configurations graphically during simulation, initially after each complete sweep for illustration. More rarely for longer runs, in order to save time.

(plus 0.5 points) Note that the CPU time per sweep for updates and measurements naturally grows only *linearly* with system size for large systems. *You must make sure* that you do not accidentally write a program with a quadratic (or worse) behavior.

Examples of actual run-times (without graphics): On an 8×8 lattice the run-time for one million sweeps (of $L^2 = 64$ single spin update attempts each) in some matlab program was about 90 seconds, and in a compiled fortran program about 8 seconds. The run-time for the autocorrelation analysis was less than a second.

2 Verification

(1 point) Calculate the expectation value of the energy *by hand* for a 2×2 lattice, by adding the contributions from all spin configurations. Note the periodic boundary conditions. On the 2×2 lattice they imply that all nearest neighbor pairs of sites are *doubly* connected. Compare to the average energy from simulations, for several values of βJ. The energy should agree to about three decimal digits (relative error of about 10^{-3}), within suitably small error bars. One value as a reference: the energy per site at $\beta J = 0.4$ is about -1.602168.

3 Statistical tools and their application

1. *Time series: (0.5 points)* Generate graphical *time series* of energy and magnetization for each simulation, and analyze them *by eye*. For each simulation: what is roughly the longest time scale on which you can recognize autocorrelations ? (You may have to zoom into the time series to see correlations).

   (0.5 points) Note that the length of the simulation *must* be much longer (rule of thumb: 100 times) than the largest time scale visible in an observable. You will likely encounter (and you must recognize!) cases in which the run cannot be made long enough. Then the average value of that observable is not converged well, i.e. it is *wrong*. The autocorrelation coefficients are then also incorrect, and the variance *cannot* be calculated correctly. Such a "short" run (however expensive) can *not* be used to calculated reliable averages. In an actual application it would have to be extended or *thrown away*.

2. *Autocorrelation function: (0.5 points)* Provide a program which calculates the autocorrelation coefficient (=autocorrelation function) for E and M, and which plots it in semilogarithmic fashion as a function of distance t. You may be able to use your program from the previous exercise.

   For each simulation and observable (E and M): determine approximately the time scale τ of the autocorrelation coefficient (inverse slope, when plotted semilogarithmically). Since you need to do this step often, you may want to automate it, for example by a fit. As an approximation, it is ok for this exercise to just take the inverse slope between two times in a sensible range (determined separately for each case), instead of a fit. To determine the range, note that the actual autocorrelation coefficient $\rho(t)$ îs positive and decays monotonically. You can set a time t^* just before $\rho(t)$ violates these conditions, and then calculate the inverse slope τ of ρ between $0.3t^*$ and $0.6t^*$.

   How does τ compare to the time scale which you can see in the times series ?

3. *Error estimate: (1 point)* In order to calculate an estimate of the true error of your observables (energy, magnetization), you can choose one of the following approaches:

   a) In the present example you will see more or less a single exponential, and you can therefore take the inverse slope τ as an estimate for the integrated autocorrelation time, and use it to calculate approximately the true variance of your observables (see lecture notes).

   b) Alternatively you can integrate over the autocorrelation function according to the definition of τ_{int} (but do not integrate too far into noise!),

   c) or use the binning procedure to obtain error estimates (and thus also estimates of τ_{int}).

   (plus 0.5 points) Make sure that you specify average values always together with an estimate of their error!

4 Exploring the limits of applicability

In order to produce reliable results, calculations *must* begin on very small lattices, and be extended *slowly* to larger lattices, while monitoring autocorrelations (at least by eye). After verification for the 2×2 system, begin by simulating on a 4×4 lattice.

1. *Freezing at low temperatures: (1 point)* Look at the time series (including thermalization) of measurements of M and E at $\beta = 0.2$, $\beta = 0.4$, and $\beta = 0.6$. You will find that the magnetization at the larger values of β exhibits long autocorrelations already on the 4×4 system. *Attention:* This is about time series of individual measurements, *not* about time series of running averages, because in those, autocorrelations are more and more hidden at later times.

   Repeat the calculations for a 6×6 and an 8×8 lattice. Leave out initial measurements for thermalization now. Calculate the autocorrelation function, the autocorrelation time, and an error estimate of the averages of E and M.

   At the larger values of β and larger lattices, the magnetization may not converge anymore, depending on the actual length of your simulations. (Reminder: the *expectation* value of M at vanishing magnetic field is *zero*). The largest time scale can even become too long to be recognized.

   N.B. At low temperatures the magnetization "freezes" and autocorrelation times are particularly large, growing even exponentially in β and lattice volume. In order to simulate such cases, one needs better Monte Carlo methods than we are using here.

2. *Autocorrelations at the critical temperature: (1 point)* Autocorrelations are also important around the critical temperature, which is the physically most interesting region.

   Perform calculations on lattices $L = 4, 8, 16, 32$ at $\beta = \beta_c$. Look at the time series and autocorrelations of E and M and perform error estimates.

   You can estimate the largest autocorrelation time to be expected here beforehand: The correlation length is bounded by L, thus the maximum autocorrelation time is roughly L^2 (and thus much smaller than at low temperatures). The time scales visible in time series are roughly of this size; the actual integrated autocorrelation time of the energy is lower by about a factor of 10. Using the expected autocorrelation time you can estimate how many sweeps you need to perform in order to obtain reliable averages.

   (N.B. Away from the critical temperature, the correlation length becomes smaller in both directions. The autocorrelation time becomes smaller at higher temperatures).

5 Phase transition

1. (0.5 points) Look at a few configurations for systems of size $L = 8$ and $L = 32$ at $\beta = 0.2$, $\beta = \beta_c$ $\beta = 0.6$. What are typical features at low and at high temperatures ? How can one interprete this physically ?

2. (1 point) Plot the average of E and of the modulus $|M|$ (modulus taken for each measured spin configuration) as a function of β for lattices of sizes $L = 4, 8, 16$ (and perhaps 32), in the range $0 < \beta < 0.6$. For each L, choose sensible values for β, such that one can recognize the occurence of a phase transition. It should become sharper in β on larger lattices.

   N.B. Such data, and even the precise dependence on lattice size ("Finite Size Scaling") can be used to obtain detailed information about phase transitions in infinitely large systems, i.e. in the thermodynamic limit. More on this in the lectures on "Phase transition and critical phenomena".

6 Handing in

You can work on this problem set in groups of two students and produce a joint solution. Each student still needs to hand in the solution individually and fulfill the usual requirements.
Some parts of this problem set require long Monte Carlo calculations; you might run your program up to a few hours for individual simulations. Prepare the presentation of corresponding results in advance. For simple simulations, the program should also run during the presentation.
Upload: If you work in a team, then please upload the files only for one of the participants. The other participant should only upload a file called "See_nnnnn", where nnnnn is the name of the first participant.