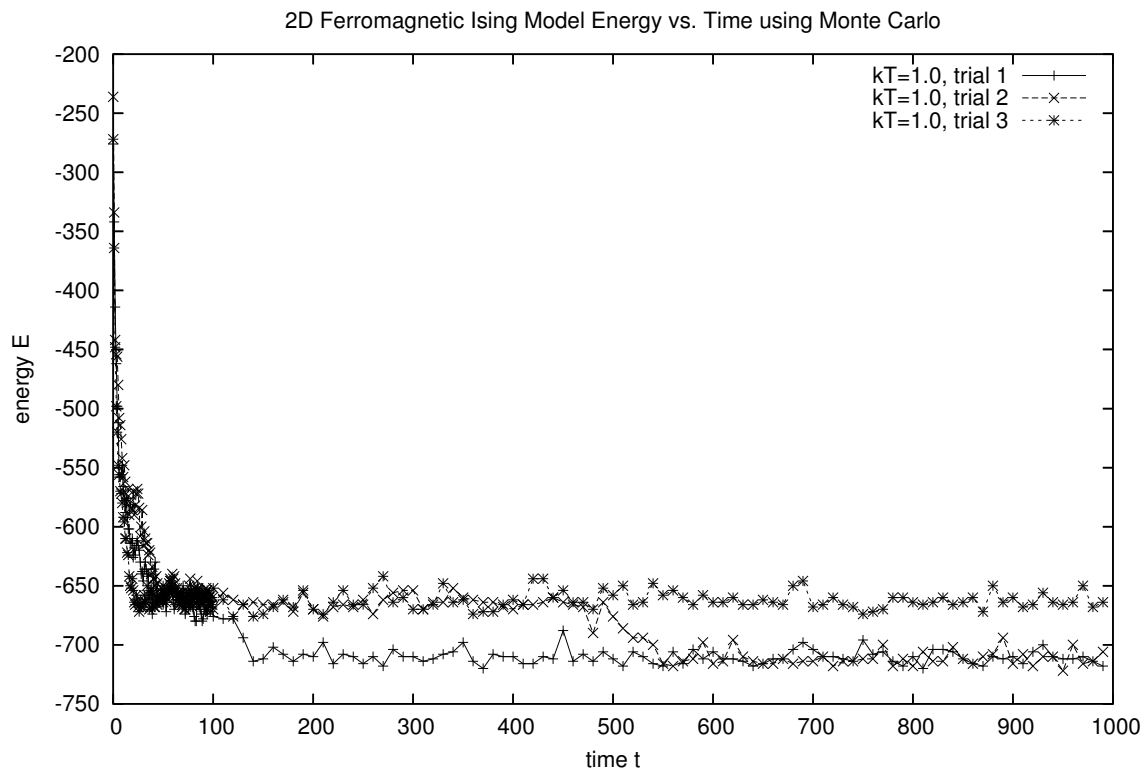


14. 780.20 Session 14

a. Follow-ups from Sessions 12 and 13

- **Equilibration and Cooling.** Consider the 2D Ising model with ferromagnetic J . Here is an example of what you might find from running the `ising_model.cpp` code with $kT = 1$ three times:



We can see a rapid decrease in energy at small times t , which reaches a plateau by $t = 50$ about which successive configurations fluctuate. This is the equilibration or thermalization time (so these are the configurations we should throw out). But we see later that in one trial the energy drops to a lower plateau starting around $t = 100$ and another drops to the same lower plateau starting around $t = 500$. We might imagine the third trial will eventually drop down to the lower energy, but we can't predict when. This problem of equilibrating to a higher energy (and there may be many other possibilities if we run for a long time) is *not* fixed by skipping a larger number of configurations at the beginning. (What would you pick? Up to $t = 100, 500, 2000$?) Rather, we give the system a chance to settle into the lowest energy by *cooling*. That is, we start the simulation at a higher temperature with a random configuration and equilibrate at that temperature. Then we use the final configuration as the starting configuration (rather than starting again with a random configuration) for a somewhat lower temperature until it equilibrates. And so on, until we get to the desired

temperature. How much to lower the temperature at each stage depends on the problem. (We’ll experiment with this rate in a slightly different context in Session 14 with simulated annealing.)

- **Calculating the Magnetization.** In the “Phase Transition” section of Session 12, you are asked to calculate the absolute value of the magnetization, which is most usefully defined for this problem as the average spin *per site* (i.e., $M = \frac{1}{N} \sum_{i=0}^{N-1} S_i$ with $S_i = \pm 1$ and N is the number of lattice sites). To study the phase transition, you should look at the time *average* of this quantity, which means you average the result over all the Monte Carlo steps (after equilibration). If you are well below the phase transition, the spins will be either all up or all down, so you should find $M = \pm 1$. The reason to take the absolute value is that, for small systems (like the ones with $L \leq 20$), in the course of a trial you could go from $M = +1$ to $M = -1$. In this case, the average would be less than 1, which does not correctly reflect the physics. This problem is fixed by taking the absolute value. As a function of temperature, you should see the transition from disorder (small average magnetization) to order ($\langle |M| \rangle = 1$) as the temperature is lowered.
- **Why the Autocorrelation Function Goes to Zero.** In Session 13 you calculate the autocorrelation function $C(l)$, where

$$C(l) = \frac{\langle A_{n+l}A_n \rangle - \langle A_n \rangle^2}{\langle A_n^2 \rangle - \langle A_n \rangle^2} \quad \text{with} \quad \langle A_{n+l}A_n \rangle = \frac{1}{M} \sum_{n=1}^M A_{n+l}A_n . \quad (14.1)$$

Thus, we can use $C(l)$ to see whether the value at $n+l$ is related to the value at n , averaged over $n = 1$ to M (strictly speaking, to $M-l$). It is clear that $C(0) = 1$, but what do we expect as n increases? If quantities A and B are *uncorrelated*, then the average of their product is the product of their averages: $\langle AB \rangle = \langle A \rangle \langle B \rangle$. So if the configurations become uncorrelated, then $\langle A_{n+l}A_n \rangle \approx \langle A_{n+l} \rangle \langle A_n \rangle \approx \langle A_n \rangle^2$ (since it shouldn’t matter whether we average starting at 0 or l) and $C(l) \rightarrow 0$. Note that it will not go exactly to zero, but fluctuate with each Monte Carlo step about zero (see Fig. 9.1 on page 256 of the Pang excerpt). The correlation time (how many configurations to skip when averaging) is roughly how long it takes to go from 1 to the fluctuation region.

- **Variational Monte Carlo code.** The program `variational_SH0.cpp` in Session 13 uses the `VariationalMC` class to implement variational Monte Carlo for a toy one-dimensional harmonic oscillator example. This is a new code, which seems to work more-or-less, but which probably still has bugs. You’re invited to help find them! One sign that something is wrong comes from using a trial wave function that includes as a special case the exact solution. In that case, the variance (and thus the error) of the Monte Carlo estimate should be zero for the exact value of the variational parameter a , but it is not zero in practice. Let me know if you identify the problem!
- **More Explanation of Metropolis, etc.** You are strongly encouraged to look at the notes by Morten Hjorth-Jensen [1], which are linked on the 780.20 webpage under “Supplementary Readings.”
 - Chapter 9 is “Random walks and the Metropolis algorithm.” Hjorth-Jensen explains how a Markov process (which is the basis of the Metropolis algorithm) is actually a discretized

version of the diffusion equation. There is a discussion of entropy and equilibrium and ergodicity in Sect. 9.4 and a clear and concise discussion of the Metropolis algorithm in Sect. 9.5.

- Chapter 10 is “Monte Carlo methods in statistical physics,” which has a review of the important thermodynamics as well as an explicit and detailed discussion of the Ising model. (See Fig. 10.2 for a graph of the absolute value of the average magnetization per spin vs. temperature for different size lattices.)
- Chapter 11 is “Quantum Monte Carlo methods,” which includes a thorough discussion of variational Monte Carlo.

b. Simulated Annealing

Standard optimization methods are very good at finding local minima near where the minimization was started, but not usually good at finding the *global* minimum. Finding the global minimum of a function (such as the energy) is often (but not always) the goal. One strategy using conventional minimizers is to run multiple trials with the minimization started at different places in the parameter space (perhaps chosen at random) and then to keep the best minimum found of all the trials.

An alternative approach is to adapt the Metropolis Monte Carlo algorithm for generating a canonical Boltzmann distribution of configurations at a temperature T to mimic how physical systems find their ground states (i.e., the energy minimum at $T = 0$). At high temperature (which means kT large compared to characteristic energy spacings), the equilibrium distribution will include many states. If the system is cooled *slowly*, then it will have the opportunity to explore many states and then settle into the lowest energy state as T goes to zero. This is called *annealing*. If the system is cooled quickly, it can get stuck in a state that is not the minimum (“quenching”); this is analogous to the routines we looked at in Session 10, which rapidly go “downhill” but only to local minima.

The strategy of *simulated annealing* is to mimic the annealing process by treating the function to be minimized as an energy (it might actually be an energy!), introducing an artificial temperature T , and generating a sequence of states in a canonical distribution via the Metropolis algorithm. Then we lower the temperature according to a “schedule” (this just means according to a definite pattern) and let the system settle into (hopefully!) a configuration that minimizes the energy.

In practice this is not so easy:

- The problem needs to be cast into a form appropriate for this technique. This means we need to have a description of possible configurations of the system and then a way to change the configuration randomly (i.e., the analogs to specifying all the spins on a lattice and generating a new configuration by randomly flipping a spin). At the same time we need to identify an appropriate energy function to be minimized; this may be immediate, if the problem *is* to minimize a function, or less obvious, if the problem is to find a solution to a problem such as the Traveling Salesman [2].

- We need to devise an appropriate annealing schedule for the control parameter T . For example, do we change T after 10 or 100 or 1000 or ? random changes in the configuration? And how much do we change it each time? These are critical questions to the success of the procedure. If there is a physical connection to the problem, we may be able to use physics insight to determine the appropriate scales. Often it is more a trial-and-error procedure.
- For continuous control parameters, as opposed to combinatoric problems, the (common) possibility of long, narrow valleys in parameter space is a serious problem. If one takes steps at random, the most likely step will be uphill rather than along the valley. So one needs to modify the basic strategy, as discussed in Ref. [2].

We'll apply simulated annealing to one artificial and one real (i.e., physical) problem. The artificial problem is just the global minimization of a one-dimensional function:

$$f(x) = e^{-(x-1)^2} \sin(8x) , \quad (14.2)$$

which has multiple local minima. We'll compare a standard minimization routine (from GSL) to simulated annealing. You'll need to adjust the simulated annealing control parameters to make it work effectively. The second problem is the shape of molecules built from sodium (Na) and chlorine (Cl) atoms. This problem is described in a Session 10 handout from the book *An Introduction to Computational Physics* by T. Pang, in a section entitled "Geometric structures of multicharge clusters." The idea is that one can write a potential energy function that depends on the relative positions of the elements of the clusters (here Na and Cl atoms). The parameters of the function are taken from experiment or theoretical calculations. The kinetic energy can be ignored, so the arrangement of the cluster is determined by minimizing the energy. There are many local minima corresponding to configurations that might be metastable but do not have the very lowest energy.

c. References

- [1] M. Hjorth-Jensen, *Computational Physics*. These are notes from a course offered at the University of Oslo. See the 780.20 webpage for links.
- [2] W. Press *et al.*, *Numerical Recipes in C* (Cambridge, 1992). Individual chapters are available online from <http://lib-www.lanl.gov/numerical/bookcpdf.html>. There are also versions for Fortran and C++.