480: Activities 13

Grade: check +

*Online handout: Excerpt from Binder/Heermann*

In this session, we'll extend our study of Monte Carlo methods. We'll compare importance sampling to ordinary random sampling and explore the two-dimensional Ising model. **Please read the background notes for an introduction and then use the Binder/Heermann handout as you go.**

Overview: Simulation of 2D Ising Model [HTML5 version]

To get started, we'll look at an applet that simulates the 2D Ising model. The goal is to visualize before looking at code; it is NOT expected that you will understand all of the physics now!

1. Go to the HTML5 Ising model simulation linked on the course page for today's session and read the short introduction.
2. You will see a grid of blue and yellow squares, which represent up or down spins on a lattice. The magnetization is proportional to the net spin (count blue as +1 and yellow as −1). Below the grid you can read off the temperature and control it with a slide. There are also controls for the size of the grid and the steps per frame. Set the temperature to 10 by moving the slide all the way to the right and press start? *What are the characteristics of the system? Are there any large areas of all blue or all yellow? How large is the magnetization per spin (roughly)? Is it constant?*

**The characteristic of the system is random since there is a good amount of blue and yellow squares mixed in together. There are no large areas of all blue or all yellow.**   
**The magnetization per spin is quite large since the simulation is going at quite a fast rate and it seems roughly constant**

1. Now rapidly bring the temperature to zero and let it settle down. Repeat the cycle of rapid heating and cooling several times. *Does it always end up in a uniform ferromagnetic state? If not, why not? Estimate the magnetization per spin in each case.*

**When the temperature is being brought down to zero, the simulation always ends up in a uniform ferromagnetic state because there is a large area of yellow and a large area of blue. This does not happen in the rapid heating case because the simulation keeps on providing a random pattern. The cooling case has a magnetization per spin of ½ and the heating case has a magnetization per spin of around 2.**

1. Next change the temperature by increments of about 1 (you have to release the slide for the change to take affect). *Characterize the behavior of the system as it heats through the critical temperature to high temperature. What happens at 2.27 (restart the page to get there exactly)? Make a sketch of the magnetization as a function of temperature.*

**The system becomes more and more random as the temperature increases. At 2.27 is seems the one color seems to dominate of large area of the graph, creating a uniform ferromagnetic state.**

Chart, line chart

Description automatically generated

1. Play with changing the cell size (by the Size pull down) to very large and very small. *What differences do you observe in the size of the fluctuations and the behavior of the magnetization?*

**The smaller the size is, the more fluctuating it seems and the magnetization seems more obvious.**

Monte Carlo Sampling

In the sampling\_test.cpp code, three distributions of energy for a one-dimensional Ising model [equation (2.1.1) in the handout] are generated. The first is the exact distribution at temperature kT in the canonical ensemble; that is, the distribution of energies considering every possible configuration of spins weighted by a normalized Boltzmann factor. The second is the energy distribution if a large number of spin configurations are chosen at random. The third is the energy distribution from a Metropolis Markov process.

1. Here are some questions to get you familiar with the model and its implementation in the code sampling\_test.cpp:
   * *How does the "J" in the Ising model determine whether we will have a ferromagnet or an anti-ferromagnet? Which choice does the code start with? What does the ground state (lowest energy state) of a ferromagnet look like?* [Bonus: What is the physical origin of J for a ferromagnetic?]

**J\_ising =1 will be a ferromagnet and J\_ising=-1 will be an anti-ferromagnet. The code starts with 1. The ground state of a ferromagnet has little or no net magnetic field.**

* + *How would you add an external magnetic field H\_ext to the code?* [See Eq.(13.6) in the notes for the Hamiltonian.]

**Based on 13.6, we multiply the Hamiltonian with the spin and put it into a series and subtract it from the product of the -J value and the series sum of the ith and jth components of the spin. The sums can be done using a for loop**

* + *How many total "microstates" are there for a 1-D Ising model with the number of sites in the code? (I.e., how many possible "configurations" of spins are there?) If calculating the energy of each configuration takes the same amount of time, how much longer would it take to calculate a lattice with twice the number of sites?*  
      
    **There are 1048576 total configurations. It would take twice as long**
  + *Given a number of lattice sites, what are the values of the minimum and maximum energies? How many different energy values are possible?* (Hint: You can't get every integer energy between the minimum and maximum.)

const int num\_energies = 2\*num\_sites + 1;  // number of different energie

**There are about 11 energy values possible**

* + *What are the boundary conditions for the line of spins?* Find where it is used in the code. *What other set of boundary conditions could be used?*

1. // now for the last site using the boundary condition
2. nearest = num\_sites - 1;
3. energy += - J\_ising \* double(configuration[nearest] \* configuration[0]);

**We can also use periodic boundary conditions**

* + Find the calculation of the exact partition function in the code. *How is this equal to the same result as (2.1.5) in the Binder/Heerman notes?* [Hint: You can add exp(-Ei/kT) for each spin configuration separately, or group all those with the same energy together and multiply by how many have that energy.]

1. // find the partition function at temperature kT
2. double Z\_exact = 0.;
3. for (int i = 0; i < num\_energies; i++)
4. {   // this assumes the allowed energies are integers
5. Z\_exact += double(energy\_count[i]) \* exp(-energy\_i(i)/kT);
6. }
8. // find the exact canonical energy distribution P(E)
9. for (int i = 0; i < num\_energies; i++)
10. {
11. dist\_exact[i] = double(energy\_count[i]) \* exp(-energy\_i(i)/kT) / Z\_exact;
12. }

**It takes the exponent of the Hamiltonian (-energy\_i(i) in the code) over the temperature**  
  
  
  
*Can you explain how the complete set of configurations are constructed in the code using the "next\_configuration" function?*

 do   // Note the "while" statement below --> check "end"

  {

     int k = 0;

     end = next\_configuration (config\_exact, k); // generate the next config.

**In the do while loop, the next\_configuration function is used a the limit of when the loop ends**

1. Compile and link sampling\_test.cpp (using make\_sampling\_test) and run it to see what the output looks like. *Did you get the correct answer above for the number of configurations and the possible energies?* (If not, rethink!)

**I did**

* + Look through the code and try to understand how each distribution is generated.

1. *Generate and sketch a gnuplot graph of the probability of energy E [called P(E)] vs. the energy (this is what is output to the screen) for kT = 10. and kT = 1.*

*Chart, line chart

Description automatically generated*[You'll need to get the results into data files somehow. There is a plot file sampling\_test.plt to help with the plot, but you have to put the results in files with these names. Cutting and pasting is fine.]

* + *Why is the distribution for ordinary random sampling centered at E=0? Based on this distribution, what kind of temperature does ordinary random sampling correspond to?* [Hint: What Boltzmann factor would result in a distribution like this?]

**The data points end up creating a random distribution. The random sampling method correspond to a higher temperature than the Metropolis algorithm method.**

* + Compare the exact P(E) for a canonical distribution to those of random sampling and the Metropolis algorithm. *Will the Metropolis algorithm be effective in doing importance sampling (which concentrates points where the integrand is large)? Why will the random sampling be a problem for evaluating thermal averages [see figure 2.3 and the discussion after equation (2.1.33) in Biner/Heermann]?*

**It might be since the Metropolis method also produces a normal distribution as of the previous generated graph and as said in the notes for the Metropolis method,** *“(To see that this is the correct procedure, imagine doing it 1000 times. You would expect roughly 300 times to get a number less than 0.3, so keeping those would give the desired probability.)”*.

**It might be a problem because large N will most likely produce 2 distribution curves which will overlap on the “wings”.**

1. Verify that the transition probabilities in (2.1.39a) and (2.1.39b) both satisfy the condition (2.1.38). *Which one is implemented in the code?* Bonus: Switch to the other and check that it works.

**2.1.39b was used**

     // decide whether to accept or reject the new configuration

      random = gsl\_ran\_flat (rng\_ptr, 0., 1.);

      if ( (delta\_energy > 0.) && (random > exp(-delta\_energy/kT))

1. *How do you calculate the average energy at a given temperature? Modify the code to calculate the average energy at kT = 1. and kT = 10. using the Metropolis sampling methods and compare to the exact average energy (according to the canonical Boltzmann distribution).*
2. double sum\_met = 0;
3. double avg;
4. for (int i = 0; i <= num\_energies; i +=4) {
5. sum\_met += dist\_metropolis[i];
6. avg = sum\_met/num\_energies;
7. cout << avg;
8. }

Graphical user interface, text, application, email

Description automatically generated

The Two-D Ising Model

In this section, we explore some aspects of Monte Carlo simulations that are discussed in section 2.2 of the handout. We use the two-dimensional Ising model first with a ferromagnetic interaction (J > 0) and then with an "anti-ferromagnetic" interaction as our example.

1. Take a look at ising\_model.cpp and note that one can switch between the one-d and two-d Ising models by commenting and uncommenting some lines. The default is the two-d model. *What are the differences between the one-d and two-d versions?*  
     
   **The 2-d has more random points than 1-d**
2. **Equilibration.** Compile and link ising\_model.cpp (use make\_ising\_model) and run for several temperatures.
   * Make a gnuplot graph (and sketch it here) of the energy vs. time for kT = 2.0, 1.0, and 0.5 (all on the same graph, so rename your files appropriately). *Run repeatedly at kT = 1.0 until you see that you don't always get the same answer at large times. What does this mean?*

**kT = 0.5**

Graphical user interface

Description automatically generated

**kT = 1.0**

Chart, histogram

Description automatically generated

**kT = 2.0**

Chart, scatter chart

Description automatically generated  
  
  
  
**This means that the energy can also get “random” at large times**

* + Look at the small time region. *How long does it take at each temperature to reach "equilibrium"? If you were going to use the configurations generated here to calculate thermal averages, would you want to use the configurations at the beginning? How can you deal with this?*

**kT = 0.5 = 100 seconds**

**kT = 1.0 = 200 seconds**

**kT = 2.0 = oscillates so frequency**

**As a side note, these were done in 2-D. I would use the same configurations for kT = 0.5 and 1. As for kT = 2, I would switch it to 1-D.**

* + BONUS: Modify the code so that the output file names automatically have the temperature in their names. (Recall filename\_test.cpp from an earlier session.)

1. Modify the code to change it from ferromagnetic to anti-ferromagnetic (only one line needs to be changed!). *What did you change?*
2. const double J\_ising = -1.;

1. **Cooling.** At present, the code starts from a random configuration. Modify the code to implement "cooling" by looping through temperatures kT = 2.0, 1.0, then 0.5 but start the simulation at each successive temperature using the final configuration of the higher temperature as the initial configuration of the lower temperature. *Generate a gnuplot graph of the energy vs. time for each of the temperatures (and sketch it here) and compare to your previous results. Conclusions?*

**kT = 0.5**

Graphical user interface

Description automatically generated

**kT = 1.0**

Chart, histogram

Description automatically generated

**kT = 2.0**

Chart, scatter chart

Description automatically generated  
  
  
**The graphs are similar to the previous ones**

1. **Efficiency.** The code at present has several inefficiencies. Here are some ways to speed it up:
   * The current approach compares energies of new and old configurations by calculating the full energy of each and subtracting. *Can you devise a (much) more efficient approach?* (You don't need to implement it here.)  
       
     **- Using a less random initial configuration probably (?)**
   * During one MCS, we can update each spin sequentially, which saves random number calls but also leads to shorter equilibration times.
   * We can reduce the random number calls when deciding if a spin flips or not.
   * We can use a table of nearest neighbors of each spins to save calculation time.

These optimizations speed up the code by a factor of about six! A new version is ising\_opt.cpp (with make\_ising\_opt). Look at how the optimizations were implemented.

1. **Phase Transition**
   * Modify the optimized code ising\_opt.cpp to calculate the absolute value of the magnetization of the system (with linear size L=5) for various temperatures. Starting at kT=4.0, cool down the system by Delta kT =0.2 until kT=1.0. *What do you observe about the behavior of magnetization?* (Make a plot in gnuplot and sketch it here)

**The energy points slowly disappear as it cools down.**

**kT = 4.0**

A picture containing table

Description automatically generated

**kT = 1.0**

**Graphical user interface, chart

Description automatically generated with medium confidence**

* + *When we calculate the magnetization, we use the absolute value of it, why?* It may help to look at the time dependence of the magnetization around kT=2 with size L=5 or less. [Bonus: Can you explain your observation in terms of "spontaneous symmetry breaking"?]

**In order to get a better result since spin may be -1 and that the magnetization covers the entire lattice.**

Will this problem get better or worse when you increase the system size? [Hint: Would it be possible for an infinite system to change from a state with all up spins to one with all down spins?]

**I predict that it will get worse as the system size increases because more flips might need to be done.**

* + Now change the system size (try L=10, 20, 40). *By changing the system size, can you observe a change in the behavior of magnetization? If so, can you make an argument why this happens?*

**I did and that is that the graph of energy vs. time shows that it will slowly move to “equilibrium as the system size increases. The following graph shows why:**

A picture containing chart

Description automatically generated