PHY407 Computational Lab 10
Monte Carlo Simulations and Simulated Annealing

Recommended reading for this lab: Newman Sections 10.3-10.4.

Computational and Physics Background

- **Monte Carlo Simulation for Statistical Mechanics:** Section 10.3.1 of the text discusses Monte Carlo simulations for statistical mechanics.

  We want to evaluate an expectation (or average) value for some quantity for a system in thermal equilibrium at temperature T. The system will pass through a succession of states such that at any particular moment, the probability of it occupying state i with energy E_i is given by the Boltzmann formula:

$$P(E_i) = \frac{\exp(-\beta E_i)}{Z}, \qquad Z = \sum_i \exp(-\beta E_i) \tag{1}$$

  where $\beta = 1/k_B T$, k_B is Boltzmann's constant, Z is called the 'partition function' and the sum is over all possible states at that temperature.

  The expectation value of a quantity X that takes the value X_i in the ith state is:

$$< X >= \sum_i X_i P(E_i) \tag{2}$$

  Essentially, this is a weighted average of the property X of each state over the sum of states.

  Normally we can't calculate the sum over all states because there are way too many. Instead, we use a Monte Carlo approach where we randomly sample terms in the sum to approximate it. So we approximate equation (2) with:

$$< X >\approx \frac{\sum_{k=1}^{N} X_i P(E_k)}{\sum_{k=1}^{N} P(E_k)} \tag{3}$$

  The denominator is there to ensure the average is normalized correctly. Importance sampling (which you saw last week for evaluating an integral) is used to evaluate the sum so as to choose more states that contribute significantly to the sum (since those with $E_i >> k_B T$ contribute very little to the sum). Using a weighted average, the sum can be written

$$< X >\approx \left\langle \frac{X_i P(E_i)}{w_i} \right\rangle_w \sum_i w_i, \tag{4}$$

  where the angular brackets on the right denote a weighted sum. This is identical to equation 10.38 that we used for the integral last week if you set $f(x) = X_i P(E_i)$ and do a sum

instead of an integral.

We evaluate this sum by selecting a set of N sample states randomly but non-uniformly, such that the probability of choosing state i is

$$p_i \approx \frac{w_i}{\sum_j w_j} \tag{5}$$

Combining this with equation (4) results in:

$$< X > \approx \frac{1}{N} \sum_{k=1}^{N} \frac{X_k P(E_k)}{w_k} \sum_i w_i \tag{6}$$

Note that the first sum is over only those states k that we sample, but the second is over all states i. This second sum is usually evaluated analytically.

The goal is to choose the weights w_i so that most of the samples are in the region where $P(E_i)$ is big and such that we can analytically do the second sum. We choose $w_i = P(E_i)$ in which case $\sum_i w_i = \sum_i P(E_i) = 1$ and we are left with:

$$< X > \approx \frac{1}{N} \sum_{k=1}^{N} X_k \tag{7}$$

So basically, we just choose N states in proportion to their Boltzmann probabilities and take the average of the quantity X over all of them.

- **The Markov Chain Method** The problem we still have to deal with is how to calculate $P(E_i)$ from equation (1) (since we want to choose states with this probability). Notice P needs the partition function which is a sum over all states (and again, if we could do that, we wouldn't need a Monte Carlo simulation). We can find $P(E_i)$ without the partition function using the Markov Chain Method (see text for more details).

  The Metropolis algorithm allows us to choose values of the transition probabilities T_{ij}. The total Markov chain Monte Carlo simulation involves the following steps:

  1. Choose a random starting state.
  2. Choose a move uniformly at random from an allowed set of moves, such as changing a single molecule to a new state.
  3. Calculate the value of the acceptance probability P_a:

$$P_a = \begin{cases} 1 & \text{if } E_j \leq E_i \\ \exp(-\beta(E_j - E_i)) & \text{if } E_j > E_i \end{cases} \tag{8}$$

  4. With probability P_a, accept the move, meaning the state of the system changes to the new state; otherwise reject it, meaning the system stays in its current state.
  5. Measure the value of the quantity of interest X in the current state and add it to a running sum of such measurements.

6. Repeat from step 2.

- **Simulated Annealing** This is a Monte Carlo method for finding GLOBAL maxima/minima of functions. Remember that in Chapter 6 we studied various methods for finding local maxima/minima, but sometimes we need the global value. Simulated Annealing can do this.

  For a physical system in equilibrium at temperature T, the probability that at any moment the system is in a state i is given by the Boltzmann probability (equation 1 above). Assume the system has a single unique ground state and choose the energy scale so that $E_i = 0$ in the ground state and $E_i > 0$ for all other states. If we cool down the system to $T = 0$, $\beta \to \infty \Rightarrow \exp(-\beta E_i) \to 0$ except for the ground state where $\exp(-\beta E_i) = 1$. Thus, in this limit, $Z = 1$ and

  $$P_a = \begin{cases} 1 & \text{for } E_i = 0, \\ 0 & \text{for } E_i > 0 \end{cases} \tag{9}$$

  This is just a way of saying that at absolute 0, the system will definitely be in the ground state.

  This suggests a computational strategy for finding the ground state: simulate the system at temperature T, using the Markov chain Monte Carlo method, then lower the temperature to 0 and the system should find the ground state. This approach can be used to find the minimum of ANY function f by treating the independent variables as defining a 'state' of the system and f as being the energy of that system.

  There is one issue that needs to be dealt with: If the system finds itself in a local minimum of the energy, then all proposed Monte Carlo moves will be to states with higher energy and if we then set $T = 0$ the acceptance probability becomes 0 for every move so the system will never escape the local minimum. To get around this, we need to cool the system slowly by gradually lowering the temperature rather than setting it directly to 0.

  To implement simulated annealing: Perform a Monte Carlo simulation of the system and slowly lower the temperature until the state stops changing. The final state that the system comes to rest in is our estimate of the global minimum. For efficiency, pick the initial temperature such that $\beta(E_j - E_i) << 1$ meaning that most moves will be accepted and the state of the system will be rapidly randomized no matter what the starting state. Then choose a cooling rate (typically exponential):

  $$T = T_0 \exp(-t/\tau) \tag{10}$$

  where T_0 is the initial temperature and τ is a time constant.

  Some trial and error is needed in picking τ. The larger the value, the better the results (because of slower cooling) but also the longer it takes the system to reach the ground state.

Lab Instructions

For grading purposes, you only need to hand in solutions to the following parts:

- Q1: This is a warm-up exercise with the code already given, so just hand in a couple of plots of the equilibration procedure. Show us how you calculate the heat capacity and provide a plot of $E(T)$ and $\overline{n}(T)$.
- Q2: Hand in your code and a snapshot of your final magnetization visualizations at T=1.0, T=2.0, T=3.0, and a brief discussion to answer part (d).
- Q3: For part a) do the analysis required and show a couple of screenshots but do not hand in the code, since it is provided. For Part b) hand in the code and plots/analysis required.

1. **A first Monte Carlo simulation, 20% of the lab):** This is meant to be a warm-up exercise to give you a feel for how to approach Monte-Carlo modelling.

   Take a look at Example 10.3 of the textbook and make sure you understand each step of the code. Ask Paul or Oliver about it if you have any questions. The code provided allows you to simulate an non-interacting quantum ideal gas in a box. Run the code for `T=10.0` and create plots of the energy as a function of time (as provided in the script) and the frequency distribution of energy E (see equation 10.65) as a function of $n = \sqrt{n_x^2 + n_y^2 + n_z^2}$. The way I accomplished this was to create a 50-point frequency distribution as follows. I neglect multiplicative factors and define the energy $e_n \equiv n^2$. Then $n = \sqrt{e_n}$. The `hist` function returns the number $f(e_n)$ of particles with value near e_n. The array `n_vals` below represents the approximate values of n associated with each bin.

   ```
   #This calculates the energy of each particle,
   # neglecting constant factors
   energy_n = n[:,0]**2+n[:,1]**2+n[:,2]**2
   #This calculates the frequency distribution and creates a plot
   figure(2)
   clf()
   hist_output = hist(energy_n,50)
   #This is the frequency distribution
   energy_frequency = hist_output[0]
   #This is what the x-axis of the plot should look like
   # if we plot the energy distribution as a function of n
   energy_vals = 0.5*(hist_output[1][:-1]+hist_output[1][1:])
   n_vals = energy_vals**0.5
   #Create the desired plot
   figure(3)
   clf()
   bar(n_vals,energy_frequency, width = 0.1)
   ```

   Using the code above as a starting point, you can calculate the average value of n for this system, which can be approximated as

   $$\overline{n} \approx \frac{\sum_n f(e_n) n}{\sum_n f(e_n)}.$$

With this background, calculate and plot the total energy $E(T)$ and $\overline{n}(T)$ for T over the range `T=10.0` to `T=1600.0` for 1000 particles. This is just a warm-up exercise so only take a few values of T in this range, for example `T=10.0, 40.0,100.0, 400.0, 1200.0, 1600.0` is sufficient. You will find that different numbers of steps are required to come to equilibrium in each case; check that you are running to equilibrium and tell us briefly what you needed to do. Estimate the heat capacity $\Delta E/\Delta T$ over this range. You do not need to show us all the plots for each case.

2. **Ising Model (40% of the lab):** Do Exercise 10.9. To get started, read the question and the file `example_1Dising.pdf` provided with this lab which sketches how to implement the Ising chain in one dimension. We will start the 1D version of the program in lecture. Hint: use "sum" from numpy to sum over all elements in an array (this works for 2D arrays).

3. **Examples of simulated annealing optimization (40% of the lab):**

   (a) Example 10.4 in the book shows you how to implement simulated annealing optimization in the travelling salesman problem. In this warm up exercise, I would like you to test the sensitivity of the script to the cooling schedule time constant τ. To do this carefully, you need to pick a particular single set of points and find optimal paths for that set of points. To pick the same set of points each time, import `seed` from `random` and insert a line like `seed(10)` before the first set of calls to `random` (e.g. just before the line beginning `r = empty(...)`). If you don't like the set of points generated, just change the seed number.

   Now the program will run exactly the same way each time on your computer. To get the annealing procedure to take a different optimization path, you will need to change the seed number by inserting a second `seed(n)` call with a different `n` just before the `while` loop.

   With this background, do the following. First, choose a set of points that you like with an initial seed. Then carry out simulated annealing optimization on this set of points a few times (by varying the second seed before the `while` loop each time), and tell us how much the final value of the distance `D` tends to vary as a result of different paths taken. Next, vary the scheduling time constant τ by making it shorter and then longer than the default value. I suggest varying the second seed each time to increase the number of paths. Provide a couple of screenshots of the different paths taken. What is the impact on `D` as you allow the system to cool more quickly or more slowly? There is no expected "correct" answer for this problem.

   (b) **Global minimum of a function (see Newman Exercise 10.10)** Consider the function $f(x,y) = x^2 - \cos 4\pi x + (y-1)^2$. The profile of f at $y = 1$, $f(x,1) = x^2 - \cos 4\pi x$, can be found plottd on p.497. The global minimum of this function is at $(x,y) = (0,1)$. Write a program to confirm this fact using simulated annealing starting at, say, $(x,y) = (2,2)$, with Monte Carlo moves of the form $(x,y) \to (x+\delta x, y+\delta y)$ where δx and δy are random numbers drawn from a Gaussian distribution with mean zero and standard deviation one. (See Section 10.1.6 for a reminder of how to generate Gaussian random numbers.) Use an exponential cooling schedule and adjust the start and end temperatures, as well as the exponential constant, until you

find values that give good answers in reasonable time. Have your program make a plot of the values of (x, y) as a function of time during the run and have it print out the final value of (x, y) at the end. You will find the plot easier to interpret if you make it using dots rather than lines.

Now adapt your program to find the minimum of the more complicated function $f(x, y) = \cos x + \cos\sqrt{2}x + \cos\sqrt{3}x + (y-1)^2$ in the range $0 < x < 50$, $-20 < y < 20$. (This means you should reject (x, y) values outside these ranges.) The correct answer is around $x \approx 16$ and $y = 1$, but there are competing minima for $y = 1$ and $x \approx 2$ and $x \approx 42$, so if the program settles on these other solutions it is not necessarily a bad thing.