PHY407-Lab11

due Dec 6 2022

Physics Background

Protein folding: The material provided here is based on Section 12.1 of the textbook "Computational Physics" by Giordano and Nakanishi.

Proteins are chains or sequences of amino acids, which themselves are smaller molecules typically containing on the order of 10's of atoms. There are 20 different amino acids found in nature, but these can be combined in an enormous number of different ways to form proteins, which typically consist of something between 50 to several thousand amino acids. Furthermore, given the particular sequence of amino acids that make up a protein, there are many different ways in which the amino acids can be oriented with respect to each other (in terms of the angles of the bonds between each pair of amino acids) and each of these orientations will lead to a different energy state for the protein. In practice, the proteins will "fold" themselves into their lowest energy state. In this lab, we will use Monte Carlo simulations to explore the physics of this behaviour. Note that the amino acids and proteins are sometimes referred to as monomers and polymers, respectively.

The order of monomers that make up the chain is referred to as the primary structure of the protein. The particular orientation or folding pattern that the protein takes is known as its tertiary structure. To capture the essential physics of protein folding in the simplest possible way, we will assume that the amino acids are arranged on a rectangular 2D lattice. The protein will have N individual monomers, and we will assume there is only one type of amino acid (or monomer).

An example of such a simplified protein is given in Figure 1. The energy of the particular orientation of a protein sequence is defined by the adjacent monomers that are not directly connected to each other. Each not directly connected pair of adjacent monomers is assumed to have an interaction energy of ϵ, which we will always take to be negative here. Thus, the polymer in Figure 1 has an energy of $E = 3\epsilon$ since there are three pairs of adjacent monomers that are not directly connected.

Cooling (Annealing) of Systems: 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

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

with $Z = \sum_i \exp\left(-\beta E_i\right)$ and $\beta = \frac{1}{k_B T}$. 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. As the system cools down, $T \to 0$, $\beta \to \infty$,$\exp(-\beta E_i) \to 0$ except for the ground state where $\exp(-\beta E_i) = 1$. Thus in this limit $Z = 1$ and

$$P_a = \begin{cases} 0, & E_i = 0 \\ 1, & E_i > 0 \end{cases} \tag{2}$$

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

Computational Background

Markov Chain/Metropolis method: Given the setup presented in the Physics Background, the procedure for Monte Carlo simulation of protein folding goes as follows.

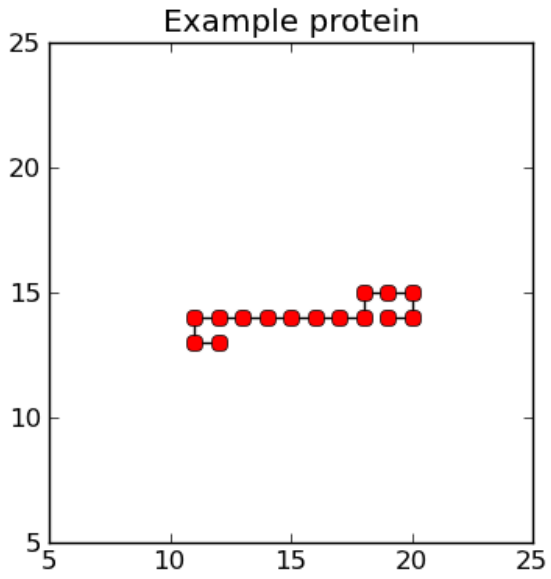

Figure 1: An example polymer of length $N = 15$ whose energy is given by $E = 3\epsilon$.

1. Choose a random monomer on the chain.
2. Then, choose one of the four possible diagonal nearest neighbour positions next to that monomer.
3. Check if it would be possible to move the given monomer to that position without "stretching" the chain (i.e., ensuring that the same distance between connected monomers is always maintained).
4. If the move is possible, calculate the difference in energy, ΔE, between the original and new states.
   - If the difference is negative and the new state would be at a lower energy, make the move.
   - If the difference is positive, then make the move only if the Boltzmann factor $\exp\left(-\frac{\Delta E}{T}\right)$ is greater than a random number between 0 and 1.
5. Repeat.

After many moves, you can compute averages over many steps to find the typical energy of a given protein at some temperature.

The code provided with this lab, `L11-protein-start.py`, implements the Monte Carlo algorithm described above. The initial condition for the protein is taken to be a flat (unfolded) horizontal line. The most important parameters in the code that you can adjust are N, the length of the chain; T, the temperature; ϵ, the interaction energy; and n, the number of Monte Carlo steps that the simulation will take.

Simulated annealing: This is a Monte Carlo method for finding *global* maxima/minima of functions. (Remember that Chapter 6 of Newman discusses various methods for finding *local* maxima/minima.)

The discussion of annealing of systems in the Physics Background 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 $\beta(E_j - E_i) \ll 1$ 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, such as

$$T = T_0 \exp\left(-\frac{t}{\tau}\right) \tag{3}$$

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.

Questions

1. Simulated Annealing Optimization [50%]

(a) Example 10.4 in the book shows you how to implement simulated annealing optimization in the travelling salesman problem. In this exercise, you will test the sensitivity of the method 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, you should seed the random number generator with a known value, for example:

```
from random import seed
seed(10)
...
```

before the first set of calls to `random`. (If you don't like the particular 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 can change the seed number by inserting a second `seed(n)` call with a different `n` before the `while` loop.

With this background, do the following.

- Choose a set of points that you like with an initial seed.
- Carry out simulated annealing optimization on this set of points a few times (by varying the second seed before the while loop each time). Assess how much the final value of the distance D tends to vary as a result of different paths taken.
- Vary the time constant τ by making it shorter and then longer than the default value. Vary the seed each time to increase the number of paths taken. What is the impact on D as you allow the system to cool more quickly or more slowly?

Submit your code, plot(s) showing a few different paths taken, and brief written answers.

(b) Consider the function

$$f(x, y) = x^2 - \cos(4\pi x) + (y - 1)^2. \tag{4}$$

The profile of f at $y = 1$ can be found 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.

- Start at $(x, y) = (2, 2)$

- Use 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 0 and standard deviation 1. (See Section 10.1.6 for a discussion 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 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.)

Submit code, plot, and printout of the final value of (x, y).

(c) Now adapt your program in the previous part to find the minimum of the more complicated function

$$f(x, y) = \cos x + \cos(\sqrt{2}x) + \cos(\sqrt{3}x) + (y - 1)^2 \tag{5}$$

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 wrong. As in the previous part, your program should make a plot of the values of (x, y) as a function of time during the run, and print out the final value of (x, y) at the end.

Submit code, plot, and printout of the final value of (x, y).

2. Protein Folding [50%]

(a) Run the script `L11-Qprotein-start.py` using the default parameters ($N = 30$, $T = 1.5$, $\epsilon = -5$, $n = 10^5$.) Note this make take up to 20–30 seconds to run, depending on your computer. The script will create two figures, one which shows the final structure of the protein, and the other which shows the energy as a function of the Monte Carlo step. Briefly describe what you see in the energy plot. Now change the temperature to $T = 0.5$ and $T = 5$ and describe what you see for each of those cases.

Submit the plot and written answers.

(b) For the $T = 0.5$ and $T = 1.5$ cases, run the simulation for $n = 1,000,000$ steps. This may take up to a couple minutes for each simulation. In which case is the typical energy of the protein (over say, the second half of the simulation) lower? Does this agree with what you would expect? Looking at the final structure of the protein, and knowing that the initial condition for the protein is just a straight horizontal line, what can you say about what is happening in the $T = 0.5$ case?

Submit the plot and written answers.

(c) **Optional** Make some modifications to the code to speed it up.

(d) As you should have found in the previous parts, when using a low temperature like $T = 0.5$, the protein does not change substantially from its initial conditions. To get around this, we can start with a higher temperature and steadily decrease the temperature over the course of the simulation in order to more quickly reach equilibrium for the given final temperature. Implement this as follows: create an array of temperatures of length n, then (given the final temperature T_f) have the temperature decrease by 1 over T_{steps} until it reaches T_f. Explicitly, this can be done as follows.

```
import numpy as np
T_f = 0.5
T_steps = 4
T_i = T_f + T_steps - 1
T_array = np.zeros(n)
for step in range(T_steps):
    T_array[step*n//T_steps:(step+1)*n//T_steps] = \
        (T_i-T_f)*(1-step/(T_steps-1)) + T_f
```

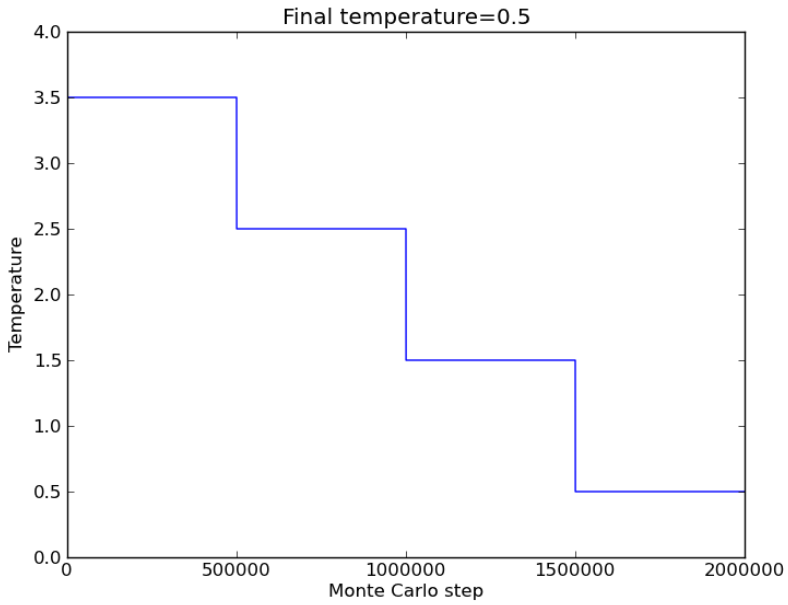

Figure 2: Temperature steps.

Now, for the i^{th} Monte Carlo step, use the i^{th} temperature from T_{array}. See Figure 2 for a plot of the temperature as a function of Monte Carlo step for $T_f = 0.5$, $T_{\text{steps}} = 4$ and $n = 2 \times 10^6$.

Implement this change to the code, and run it with $T_f = 0.5$, $T_{\text{steps}} = 4$ and $n = 2 \times 10^6$. What is the approximate energy the protein has over the last quarter of the simulation (i.e., when $T = 0.5$)? How does this compare to the $T = 0.5$ simulation from part (b)?

Submit any changed code, and your brief written answers.

(e) Quantitatively explore the temperature dependence of the energy of a particular protein, by simulating the protein folding starting at a temperature of $T = 10$, stepping by $\delta T = 0.5$ until you reach $T = 0.5$. At each temperature simulate 500,000 steps and calculate the mean energy and standard deviation at that temperature. At the end of the simulation you should have values of temperature, mean energy, and standard deviation in energy. Make a plot of energy versus temperature (you may find `matplotlib.pyplot.errorbar` helpful). What do you find? Do you think there is evidence for a phase transition (i.e., a sharp jump in energies over a small temperature range)?

Note: running the simulation for this many steps will take a while (up to 30 minutes), so you should test the code with a smaller n. It will also help if you did the previous Optional part, to speed up the code.

Submit code, plot(s), and written answers to the questions.