Lab exercise 2: Entropy - Biophysical Chemistry

March 26nd, 2018

1 Requirements

In each section there are tasks listed. All such tasks need to be answered in writing (submit via mondo assignments) no later than the start of the next exercise. Please number submitted responses as "exercise.section.task" as in previous assignments.

Note that brevity is encouraged; shorter is better, but answers need to be correct. As such, you decide the length of the "report". Be clear, concise and correct, and you will pass.

2 Introduction

In the previous lab you where introduced to a simulation method, which mimics the dynamics of real system through random jumps to and from different states according to some rule. This is the Monte Carlo method, named after the famous casino in Monaco for no apparent reason other than the chance based analogy.

In the the previous assignment,each state was associated with an enthalpy and thus had a probability to be occupied according to its Boltzmann factor. In this exercise we will expand on this topic by looking into groups of such states, and the implications this has for the corresponding physical properties in real systems. A group of states will be called a *macrostate* and, to avoid misunderstandings, each single state inside such a macrostate will be called a *microstates*. The reason for this will be clarified shortly. Also note that the quantity referred to as energies in the previous assignment will here be referred to as *enthalpies*, since we will need to start distinguishing between different types of energies.

But before we dive into the biophysical aspect we will introduce a simple tool written in Python that you will be using. The tool consists of a single Python application that uses input and output data *files*, along with a few parameters that you can set manually. Under the hood, however, it performs exactly the same operations as the Python script in the previous assignment. The command-line interface of the application, and its use of data-files, is meant to be very similar to scientific applications that you are likely to encounter in the future. It also takes the focus away from coding the actual simulation engine, permitting you to focus on the results.

Download the two supplementary files for this assignment and place it into a path of your choosing. The file named *mcrun.py* is the application that will be used throughout this assignment. **Open up a terminal, *cd* into the directory you placed the application in** and type

```
./mcrun.py -h
```

Here, the "h-flag" that you passed into the command, stands for "help". Hence it should output a description of how you can interact with the application. The help-syntax is a standard used in most Linux applications, but also some Mac and Windows applications. In this case the output clarifies that for this particular tool, it is mandatory to input a state-file at start up. All of the other options, however, have default values which we will leave alone for now. The mandatory input-file will be called the *enthalpy-file* and it should contain the enthalpies of all the microstates that you have in your system, one per row. The number of microstates is thus defined by the number of rows in this file.

3 Run the First Simulation

Create a text file named "enth.dat" and put only 4 rows in there containing a single number between 0 and 10. This will define four microstates in the system and assign each an enthalpy equal to the value you set at the corresponding row. **To use this file as an input and execute the simulation run the following command**

```
./mcrun.py enth.dat
```

If everything went fine this will generate a couple of lines as output in the terminal stating what the program did. It also produced a single output-file named "traj.dat" in your working directory. If you open the traj-file you should find exactly one thousand lines of numbers between 0 and 3. These are the indices of the microstates that the system occupied at a certain step in the simulation. The first line of the file should be 0, which is the index of the initial state of the system. As the system evolved it will have visited other microstates with higher indices, shown in the subsequent rows of the file. "traj" is short for *trajectory*.

The reason the output-file contain a thousand lines is that the step-limit is set to this default value. You can manually modify this with the n-flag. Try

```
./mcrun.py enth.dat -n 10000
```

This overwrites the old traj-file with a larger file containing 10-times as many lines. We will not go further into the usage of this application for now, but you are free to play around with other flags before you continue, since we will return to this application in later assignments.

4 Units

The values that you specify in the enthalpy-file are used by the application to calculate the enthalpy difference between two state (i.e. $\Delta E = E_{to} - E_{from}$) and is used to calculate the Boltzmann factor as

$$e^{-\Delta E_i/T} \tag{1}$$

where T is the temperature and i the index of the state.

Tasks

1. What is the unit of E or ΔE in Eq. (1)?
2. What is the conversion formula between this unit of energy and the regular unit in Joules?

Hints

1. Notice the absence of the Boltzmann constant k.

5 Plot Some Results

Go back to the enthalpy-file and change the enthalpies of the first three microstates to -1 and the last to -200 (arbitrary unit).

Run the simulation for the number of steps you think is required. Visually examine the traj-file with any editor just to check that it's not completely off. Does it contain the right indices and number of rows?

Now, let's plot this data. **Use the provided "plot.py" file to look at the distribution of the microstates.** It plots the simulated population by counting the number of entries in the traj-file of each of the four microstates. Compare with the enthalpies that you assigned to the corresponding microstates. Does it look as expected?

Tasks

Open up the plot-file and uncomment line 26 and replace "???" with what you think should be there instead. Save and run the script again.

1. Play around until the predictions approximately reproduce the simulation results and write down how you did it and why it makes sense.

2. Why is there a tiny discrepancy between the predicted and simulated distribution?

6 Multiplicity

To make our simple model more similar to real biophysical systems, we will now add another level of complexity. We need to handle groups of microstates, which as mentioned before will be called macrostates. We will emphasise the distinction between the two types of states. Macrostates will be referred to by alphabetic indices and the microstates by numerical indices. **The multiplicity of a macrostate is the number of microstates it contains.** The simple set of four microstates can for instance be divided into the two macrostates A and B.

If we assign the microstate 1,2 and 3 to macrostate A and 4 to B, we get a system that is shown in figure 1. We won't have to rerun the simulation, but simply reconsider our interpretation of the data. We'll do this by counting all the hits in the traj-file for microstates 1-3 as belonging to macrostate A, and hits on microstate 4 as macrostate B. Here's todays most crucial idea, which we learnt last time; **the energy and population of a state are directly related, via the Boltzmann distribution.** This means that there must be an "energy" for the macrostate A, dependent on it's population relative to B. Clearly, the population for a macrostate is the sum of the population of its microstates, so it must be that

$$\langle N_A \rangle \propto e^{-E_A/T} = e^{-E_1/T} + e^{-E_2/T} + e^{-E_3/T}. \tag{2}$$

Where E_A is the "effective" energy of macrostate A.

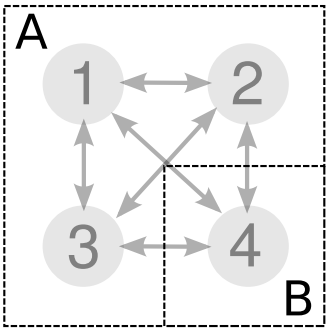

Figure 1: Showing the assignment of microstate 1,2 and 3 to the macrostate A and 4 to B. The arrows show the possible transition between states, which here is all to all.

Tasks

1. Rewrite the "plot.py" file to count microstates 1,2 and 3 as macrostate A and microstate 4 as macrostate B, both for the simulated and predicted distribution (see figure 1). You are free to do this your way, however, a tip is to simply reuse the lists "simulation_dist" and "prediction_dist" as holders for the two macrostates and add the contribution from the microstates to each of them. **Attach a plot of the results.**

2. Instead of summing together the Boltzmann factors for the microstates, rewrite plot.py so that it calculates the predicted macrostate-distribution in the more compact form $e^{-E_A/T}$ and $e^{-E_B/T}$. What should E_A and E_B be set to? Try different values for calculating the predicted distribution till it matches the simulated one. **Present the values.**

3. If we assume that all enthalpies for the microstates within the macrostate are equal (i.e. $E_1 = E_2 = E_3 = E$), we can write $E_A = E - T \times C$. Derive this formula from equation (2). What is C dependent on and what physical quantity does it represent? What would its unit be if we assume that E_A is in joules?

4. Rewrite the code to calculate the predicted distribution using this form of the Boltzmann factor. **Attach the code for this calculation.**

5. What is the physical quantity E_A called? Can it still be considered an enthalpy?

6. Can you imagine a real physical system that displays states with multiplicity? How would this manifest itself?

Hints

1. You can access individual list elements in python with square brackets: `simulation_dist[0]`

2. You can easily sum a set of list elements using the sum-function: `sum(simulation_dist[0:3])`. This will return the sum of the first three elements in the list.

3. Can all microstates be distinguished in real systems?

4. The constant C is a very important physical quantity. It vanishes for a macrostate with no multiplicity.