Lab exercise 1: Enthalpy - Biophysical Chemistry

March 27nd, 2017

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", so that todays (exercise 1) tasks are for instance:

1.3.1, 1.3.2
1.4.1
1.5.1, 1.5.2, 1.5.3

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 course, we've discussed how the difference in energy between two states determine how likely it is to "be" in one or the other state. States with low energy are more likely to be populated, and thus, states with low energy will also be observed more frequently if we *sample* a system of states over time. A sample in this case is just an observation. In fact; **to repeat observations and count events is something we typically call by the word "sampling"**. In the course you have also been shown that the mathematical rule which dictates this population (or probability to be observed) is that of Boltzmann statistics;

$$\langle N_i \rangle = N \cdot \frac{e^{-E_i/kT}}{Z} \tag{1}$$

where Z is the partition function (or simply total probability), N is the total number of observations made, and $\langle N_i \rangle$ is the expected number of observations in state i, a state which has energy E_i. In this exercise you will examine this by making a simple model or two of systems obeying this formula. The aim is to;

Construct and understand a simple program capable of demonstrating a system populated according to Boltzmann statistics, if that system is described as being held at a given temperature and having a number of states, each of those states having a predefined "state-energy".

One of the most important aspects of science is the ability to construct such models to quantify observations objectively within a framework which can be extended to predict other aspects of the

system. While the initial steps might seem trivial, the labs in this course will attempt to add complexity to show how simple models can reveal fundamental properties shared with real systems. We will of course show you the approach we had in mind, and provide you with the necessary tools.

The central aim of many computational methods is to construct a system which behaves just like reality, and then study this artificial system and its aspects, which are then inferred back onto "real" systems. This means that you must take care to construct model systems which behave in the expected way. This is what this exercise is about. We want to create such a model.

Sp how could one use a computer to emulate a number of states and the distribution between them? If you feel like you could do this without any more guidance, we encourage you to try it for yourself. Just notify us (tell one of the instructors), and start programming. If you would rather follow our scripts and instructions, simply continue with the next section.

3 The predicted Boltzmann distribution

Start an interactive python-session by opening a terminal and starting the program `ipython` or `python`. Then import some handy tools by adding the numpy module under the alias np, like so;

```
import numpy as np
```

Anytime we want to use a special feature in the numpy-module, we can now just write

```
np.theFunctionWeWant()
```

We have provided two files. Begin by looking in Lab1A_1.py, and consider only sections 1 and 3 (ignoring section 2 for now). You will alter which function is being used in Lab1A_1.py (line 141), and then use it to plot the distribution of states, depending on the energies of a few states. It contains extensive instructions and clarifications, so if you are in doubt, read it carefully. When you want to run it, simply write

```
run /path/to/Lab1A_1.py
```

in your ipython-session. This should plot bars indicating the population of states described by the energies specified within the file. When you have managed to plot this, complete the following tasks:

Tasks

1. Identify the correct conversion function (section 1), by changing the function used on line 141.
2. Argue why the others are incorrect, if they are.

Hints

1. Change the energy so that all states are equal. Is the resulting distribution reasonable?
2. Increase the temperature to 100'000K. What happened to the distribution, and is this also reasonable?
3. Decrease the temperature to 0K.

4 Random sampling

Let us now start to think about how to *model* the Boltzmann probability-distribution with simple computational tools, in contrast to just calculating what they *should* be according to equation (1).

In your ipython-session, try np.random.rand(). Call it a couple of times, what does it do? Could you make a computational model to reproduce Boltzmann statistics using it? Again, we encourage you to think and try yourself, but you are provided with instructions and tools to guide you in case you prefer this.

Let's try to connect Equation (1) to the rand()-function. Equation (1) returns the probability of any possible state. We also know that the sum of all those probabilities must be 1 (the probability of being in ANY state is 100%). In very simple terms, the probability of a state is represented by the length of a stick. If we line up the sticks of all states, one after another, they stretch from 0 to 1.

Since the rand()-function randomize numbers on the 0-to-1 interval, we can use this to randomly select states according to which of the sticks the random point coincides with. If we have made all the sticks to have the correct length according to Boltzmann statistics (i.e. set the probability of the states correctly), then a process of generating random numbers should result in a population of states which compare well to the predicted Boltzmann distribution.

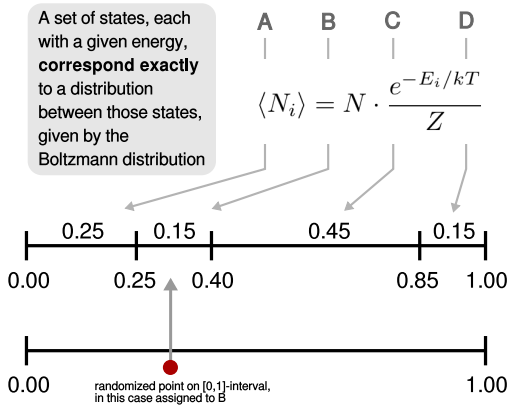


Figure 1: By arranging the probabilities calculated through Boltzmann statistics on a line, we can assign a randomized point to one of the possible states. Repeating this randomization many times should show a distribution similar to the expected distribution. This may appear trivial, but the practicality is that we can later extend this model without doing any more math.

You may again use the provided template-file Lab1A_1.py, now using section 2 as well. This section performs a random sampling as described above, called "sampleStates()". Change the conversion formula used by it to the correct one (which you found in the last exercise), and uncomment line 164 which calls this function. Now save and run the file as before.

You should find plotted the same prediction as before, but now also bar-plots indicating the population of states described by the random sampling. Does the analytical and random-generated distributions agree? How can you tell (How well do they agree)? How about at 10K?

Task

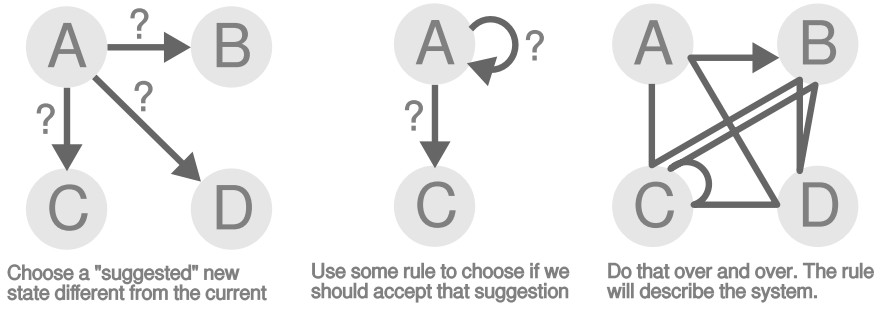
1. Explain when the random sampler does not *exactly* agree with the predicted results, and why.

5 Simulation

Now we will move on to create a similar method to arrive at the Boltzmann distribution, but which shows us something new about the way the systems works, and which will allow us to extend the capability and complexity of the model when we have learnt more about statistical physics.

What we will construct is a so-called Monte-Carlo simulation. Is such a simulation, we sample transition between states, rather than the states themselves. We do this by randomizing *trial moves* between states, which are accepted or rejected according to some simple rule. For instance;

> There are four states; A, B, C and D. You are in state A.
> All states are **equally likely** to be picked, and we randomly pick C.
> You now either move to the randomly picked state (C) or stay in A.
> We can now create a model by choosing the probability to move/stay,
> but we have to do it for all possible transitions, not just from A to C.



This is also summarized in the figure, and the important thing to keep in mind is that **how we define the rule of accept-or-reject will define behavior of the system we simulate**. This is where we attempt to model reality, because we want our later inferences to be valid. We are in fact completely free to define any rule we like. One rule might be

> **PrimeRule:**
> **If the new state is a prime number, then move to it. If it is NOT prime, then toss a die, and move to the new state only if it lands on 6.**

This is a clear-cut, simple rule, and you can repeat it for as many times over as you like. With it we can run a simulation of "state-hopping". Of course, the above PrimeRule does not have very much to do with the physics we are interested in.

The point is that as the master of this universe of states, you can choose ANY rule you want. You will find that new computational methods may simply invent a new way to examine a system

by stating a new, perhaps better or more efficient systematic rule like this.

The challenge in unsign computational methods is to justify that the rule or method is relevant to study anything, and that the predictive power of the model is real. The PrimeRule (although useless for our purposes) is implemented as an example in a new template file Lab1A_2.py.

Task

1. Define the correct conversion formula into Lab1A_2.py to get the correct Boltzmann-prediction, to be used as comparison. Then try running the template-file using the PrimeRule, and give a reason for why it does not agree with the Boltzmann prediction.

2. Create your own accept/reject rule. We simply want to see you make a rule which runs a simulation of a system, whatever that rule may mean or what system it might represent. It does not have to make physical sense or be related to the Boltzmann distribution. In your report, give your rule as a python function.

3. Create a more physically correct accept/reject (stay/move) rule, which results in sampling according to the Boltzmann distribution. In your report, give your rule as a python function.

Hints

1. For the last exercise, think about what the probability of moving should depend on, and how you could create a relative proportion which indicates how likely you are to stay or go.

6 Digression

The major point of this entire exercise is that you can easily create reasonable behavior in a simulated system which are nonetheless wrong or incorrect. This is because you feed the computer instructions, and it simply executes them. You should therefore be careful to question not just the results but the instructions which produced those results. In short; The simulated system will simply behave as you asked it to, NOT according to physics, unless you manage to specify physics *correctly*. But these methods it also affords you the freedom to set arbitrary values and observe very detailed behavior. We cannot make an experiment at 100'000K, but we CAN make a simple model and just ask what would happen. This is a fantastic thing, and this freedom will be explored even more in the next exercise.