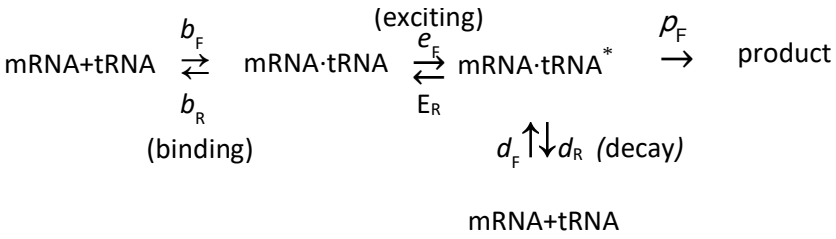
EE194/BIO196: Modeling Biological Systems
HW #4: Kinetic proofreading

Overview

In this homework, we will learn three things. First, we will use the course's chemical-simulation framework. Second, we'll learn one of the simplest of optimization techniques – try every reasonable solution and see what works best. Finally, we will get to use a bit of a time-honored and extremely useful skill: programming by undocumented example ☺.

We will be working with *kinetic proofreading*, and simulating the process of mRNA binding to tRNA as part of translation. In theory, each mRNA molecule is supposed to find the matching tRNA molecule for its codon and never bind to a tRNA that is carrying the incorrect amino acid. In practice, however, an mRNA molecule might have an affinity constant to the wrong tRNA molecule that is only 100x or so lower than its affinity to the correct tRNA molecule. Kinetic proofreading is the process by which this 100x difference is amplified enough to make the translation process quite reliable.

We will look at the main chemical reactions for mRNA-tRNA binding, and try to find combinations of the forward and reverse reaction rates that explain translation's reliability. The reactions are as follows:



In *binding*, an mRNA molecule binds to a tRNA molecule. Hopefully it binds to the correct one, but that depends on how b_F and b_R for the correct tRNA molecule compare to b_F and b_R for other tRNA molecules. A high b_F and low b_R would indicate tight binding; in practice b_F for many tRNA molecules is similar, and b_R for "close-match" tRNA molecules may be only 100x higher than for the desired tRNA molecules.

In *exciting*, the bound complex mRNA·tRNA takes energy (from the decay of GTP) and proceeds to an excited state mRNA·tRNA*. This excited complex can then go through either of two paths.

In *decay*, mRNA·tRNA* decays back to individual mRNA and tRNA molecules (note the convention that d_F is the rate of the combination rather than of the decay). In *product*, mRNA*·tRNA goes through translation, and produces "product;" i.e., transfers an amino acid to the end-product protein.

The product reaction, like many such reactions in the body, is essentially irreversible, since the resultant proteins are quite stable. For this homework, we will ignore the product reaction, and measure success by the concentration of the bound, excited mRNA·tRNA*.

The simulation framework

We've discussed the chemical-simulation framework in class. For the most part, calling the framework is already done for you in this homework, in a function called *sim*(). However, you will have to fix this function slightly. So let's hit the highlights of the framework, to help you understand how *sim*() works. On a high level, you must do several things to simulate a system of reactions.

1. Declare all of your metabolites using *add_metab* (*metabolite*, *initial_concentration*). At this point, we are not differentiating between reactants and products (and indeed most metabolites will be both).
2. Declare all of your reactions using *add_reaction* (*model*, *name*, *reactants*, *products*, *parameters*). *Model* is the name of a function that you must supply, and that models the reaction. The infrastructure will repeatedly call *model* ([*reactants*], [*products*], *parameters*); i.e., as the simulation progresses and metabolite concentrations change, it will call *model*() and give it the current reactant and product concentrations, as well as the reaction parameters (which are typically the rate constants). The *model*() is then responsible for returning the consumption rate of all reactants, and the generation rate of all products. Each reaction instance has a name also. This can be useful for debugging your simulation.
3. Run a simulation using, in our case, *steady_state_sim*(), which simulates the pathways until steady state is reached. Then retrieve the final metabolite concentrations with *final_val*().

Your task

You get to start with a skeleton of the code you will write. It will give you all of the functions. You must still fill some of the code in. Specifically…

We've provided a top-level function *kinetic_proofreading*(). You must fill it in with code. At a high level, your code should look as follows:

```
for every reasonable value of b_F, b_R, e_F, e_R, d_F and d_R:
        call sim() with those reaction rates and get the final [EB]
        call sim() with 100x higher d_R and get the final [EB]
        if both sims worked, and the ratio of (good [EB])/(bad [EB]) is over 9900:
                print out the reaction rates and the ratio
                if this is the best set of rates so far:
                        save it
```

Your code should try every reasonable value of the reaction rates b_F, b_R, e_F, e_R, d_F and d_R (you may assume that b_R=d_R). For each such combination, you must call *sim*() twice. The first time, use the current values of the reaction rates b_F, b_R, e_F, e_R, d_F and d_R. The second time, use values of b_R and d_R that are 100x higher, to model mRNA binding to the improper tRNA, and hence the bound mRNA·tRNA complex decaying much more quickly (whether or not it is excited).

What constitutes "every reasonable combination?"

- You may assume that b_F is always 1. Reaction rates are relative, and so we may always choose one of them to be whatever we like.
- b_R can have values of .0001, .001, .01, .1 and 1. Ditto for e_F.
- e_R can have values of 0, .001, .01, .1 and 1. Ditto for d_F.

- d_R should always have the same value as b_R, since they both represent essentially the same dissociation.

Our goal is to find a combination of reaction rates such that our cells do a very good job of rejecting improper bindings; i.e., one where the final [EB] is much higher for a correct binding than for an incorrect one. Thus, you should calculate the ratio of the final [EB] for the correctly-bound case / the incorrectly-bound case. Print out all of the cases where the ratio is better than 9900. Furthermore, keep track of the absolute best ratio you get and print that out (along with the reaction rates that generated it).

Your next task is to finish writing the function *sim*(). Most of it is there. However, I have somehow forgotten to add any of the metabolites (mRNA, tRNA, B and EB). You should fix this mistake, giving all of them an initial concentration of 1. Furthermore, I have forgotten to import the necessary packages to be able to call *add_metabolite*() and *add_reaction*(); you should fix that as well.

Debugging

You may want to proceed in two steps. While you will eventually try every reasonable combination of reaction rates, at first you may want to debug your code by just hand-coding one combination. Once you have verified that this produces reasonable results, you can go ahead and automate the large-scale simulation runs.

Discussion questions:

1. What did you notice about the reaction-rate combinations that worked well? Did this pretty much match what we discussed in class? Which reaction reates were always bigger than others? Which reactions tended to be irreversible?
2. Roughly how long did the entire program take to run? Roughly how much time did each combination of reaction rates take to model (i.e., for both of the calls to *sim*() together)? We simulated 5 different values of 4 different reaction rates, for a total of 625 trials. If the system had been more complex, and we had needed to try 5 values of 10 reaction rates rather than just 4, how long might it have taken (assuming that the time to model one simulation run stayed the same)? Would our exhaustive-simulation technique have still been feasible?
3. One of the functions is called Edecay. Luckily, you don't have to write it. However, you do have to figure out what it does (since reverse-engineering other people's code is often a fact of life). What do you think it does? Explain the equation *dF*mRNA*tRNA - dR*EB*.

Extra credit: The first discussion question asked for a hypothesis about why some reaction rates work quite well. For extra credit, analyze this mathematically. I.e., do the algebra to compute the discrimination based on the reaction rates. You will probably find it easier to work in two parts; first looking only at the reactions for binding and excitation, compute the final [*B*] (i.e., when [*B*]'=0) for the good and bad cases, and state a condition that is necessary for effective discrimination. Second, assuming that [*B*] is constant, and looking only at the reactions for excitation, decay and product formation, compute the final value of [*EB*] (i.e., when [*EB*]'=0), and state a condition that is necessary for effective discrimination. To simplify the analysis slightly, you may assume in both cases that excitation is irreversible (but do not assume that decay is irreversible).

Logistics:

- Use any lab PC system to write your code. You may use your own laptop if you prefer.
- The due date for this assignment is on the class calendar
- Submit your project at https://www.ece.tufts.edu/ee/194MSO/provide.cgi, which is also accessible from the course web page. You should turn in two files: kinetic_proofreading.py and discussion.pdf (or whatever other format you use).