Task2: Parameter estimation

May 13, 2019

In task1 you built a model from a full description of the topology, initial conditions and parameters. This time you are given the topology and some experimental data, but you will have to find the parameters yourself. Its a good idea, when you get time, to read some literature on parameter estimation of ODE models. Its not essential, since mostly Copasi takes care of the details, but it is helpful to have at least a rudimentary understanding of what is going on...

In model 2, A is activated by the presence of S with Hill kinetics. Hill kinetics are a common way of modelling receptor activation. Then we have a small phosphorylation cascade starting at Ap which activates B which in turn activates C. Both forwards and backwards reactions in this chain of phosphorylations follow michaelis-menten kinetics, except for the activation of B which is instead competitively inhibited by I. Note that the competitive inhibition rate law converges to the michaelis menten when the inhibitor is not present. Furthermore, phosphorylated C enhances the dephosphorylation of A, thus acting as a negative feedback loop.

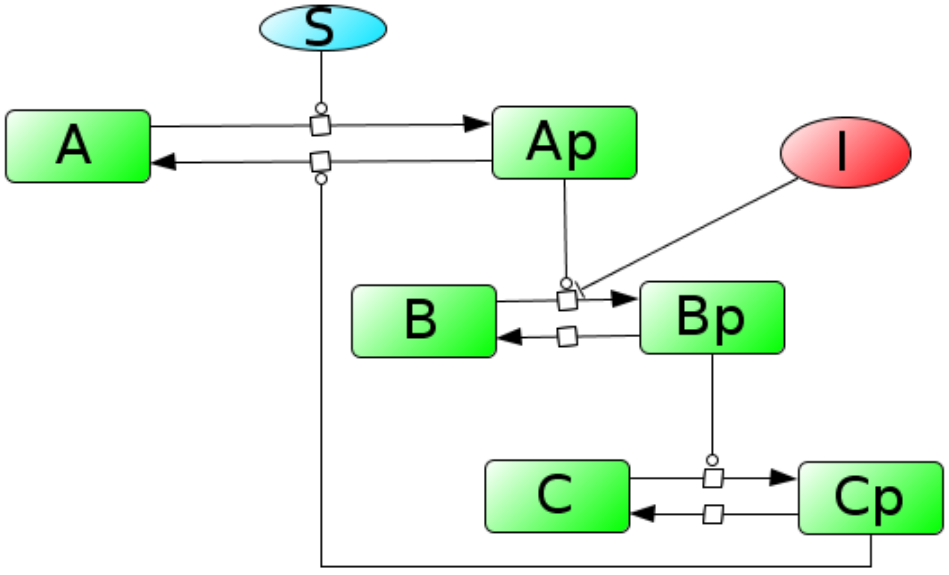

Figure 1: Network topology for model 2.

1. You have a file containing experimental data that will fit this model. Use matplotlib and seaborn Python packages to visualise these data
2. Build the model topology using antimony
3. Use PyCoTools to open the antimony model in Copasi
4. Use Copasi to setup a parameter estimation that estimates all of the parameters of this model
    - Use what you know about the model from the experimental conditions
    - Consider which algorithm to use, as well as the hyperparameters
    - Consider how to weight the objective function.
5. Use PyCoTools to configure and run the same parameter estimation
6. Use PyCoTools to run several parameter estimations simultaneously
7. Use PyCoTools to run many parameter estimations on the cluster
8. Use PyCoTools to configure and run an identifiability analysis (profile likelihoods) on the cluster

Do a small bit of background reading version control and git/github. There's no need to go mad as Git is one of those things that you learn little by little, as and when new functions are needed – most notably when you accidentally destroy your program and want to revert to a previous state. Then:

1. Create a new git repository for the code you have generated so ofar
2. Add your files to the repository and commit your progress
3. Push your repository to the Git servers
4. Change something in your project (literally anything)
5. Use 'git status' in the terminal to see that git has tracked your changes
6. Use the git add, git commit and git push commands to make your changes permanent.