

CS341 SYSTEMS BIOLOGY I

LECTURE 17

PARAMETER-FITTING

Today we begin the 4th of topics in the course. The first was kinetics (i.e. the components of models), the second was models (examples of common motifs and of circadian clocks), the third was numerical solution of ODE's (i.e. simulating models). Today, we talk about how to find values for model parameters that will lead to simulations with biologically realistic output. The fifth topic will be sensitivity analysis (what can models tell us about points of sensitivity/robustness in biological systems?).

Most parameters (e.g. rates of dimerization) are not directly measurable. So it is typical when developing a model to use numeric optimization to find parameter values that lead to the desired output. We saw in Forger & Peskin's paper that one strategy for finding parameters was to match time-series from experimental data to simulation time-series. We will see in a paper by Locke et al (in HW11) an alternative approach, which aims to achieve a set of important features (e.g. the period of oscillation and the difference in peak times). We will be taking the latter approach, though everything I say today applies to the former as well.

To find a set of parameters, we define a cost function:

- input: a set of candidate parameters
- output: a single number (≥ 0) representing the cost of those parameters. A perfect parameter set will have cost=0.
- what it does: simulate the model with the given parameters and compute the cost according to simulation output.

We then search for a set of parameter values that cause the cost function to be as small as possible. In other words, this is a numerical minimization problem. We also call it optimization (we are looking for an optimal set of parameters).

How do we find this optimal parameter set?

One option is to go to the `scipy.optimize` library and find a method that will work. And that is certainly feasible. But not as satisfactory as learning how to implement our own method. Today, I am going to basically just present set-up for the approach we will be taking over the next couple of weeks. That approach will be developing model-specific cost

functions that can be used to find parameter sets for clock models. We will be implementing a genetic algorithm to carry out the optimization.

But before we get to genetic algorithms, let's start with a simpler algorithm. One that aims to find a cost function minimum by descending a hill.

1D Hill-Descending Approach. Since this is not the focus of this topic, I am not asking you to write code for this. I coded it up. But I want you to follow the intuition for how it works and when it doesn't. The idea is that we begin with a parameter value. Then, we compute the cost function for a parameter that is a little bigger and for a parameter that is a little small. Whichever has the smaller cost determines the direction we head. We update our parameter value to be the one that led to the lower cost. We repeat, possibly adjusting how we compute the values of the nearby parameter (how big of a step do we take). I coded up the Hill-climbing algorithm presented in Wikipedia: https://en.wikipedia.org/wiki/Hill_climbing

We find that it works well for our trial functions - a parabola and a V.

What is the drawback to this approach? It could get stuck in a local minimum..

Parameter-Fitting. Now, let's use that method in the context of parameter-fitting.

First, we need a cost function. Let's use the van der Pol oscillator because it has just one parameter. It is easier to think about optimization when only one thing needs to change. Later we scale up to more than one parameter.

The van der Pol oscillator has 2 state variables and one parameter μ :

$$\begin{aligned}\frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= \mu(1 - y_1^2)y_2 - y_1.\end{aligned}$$

What do we want the van der Pol oscillator to do? I would like to say "oscillate with a period of 24 h so that we can use it as a clock model." Unfortunately, when we use just one parameter for the van der Pol oscillator the same parameter that controls the period also controls the stiffness of the model. And if we make the parameter big enough to make the period circadian, then the model becomes super spiky and hard to solve. So I am going to arbitrarily declare that our goal is to make the van der Pol oscillate with a period of 10 h (assuming the units of time is hours).

This means our cost function needs to

- Run the simulation with a tight tolerance for a long time (e.g. 500 h) so that it reaches its limit cycle.
- Rerun the simulation with a tight tolerance and small time step for several cycles. Then estimate the period (recall `get_period` from HW2!).
- Then return the error in the period. What should that be? It needs to be 0 when perfect and > 0 as it gets further from perfect. That means it could be the absolute value of the difference between the simulation period and 10. Or it could be the square of that difference. It is traditional to use the square of the difference (partly to smooth out the cost function, partly to make outliers stand out... but that last part only comes into play where there are lots of costs of different aspects of the oscillation that need to be summed together). We will choose the traditional route and return $(\text{period}-10)^2$.

We implement this cost function and run the hill-descending algorithm on it. Depending on the initial condition, it sometimes succeeds and sometimes fails to produce a low enough cost.

Scipy's Local Optimizers. There are many strategies for optimizing functions. The hill-descent method that we just implemented falls into a category of methods called *local methods* because they make decisions about what value to test next based on local information (values near the current point).

Scipy has an optimization library:

<https://docs.scipy.org/doc/scipy/reference/optimize.html>.

In that library are several local methods. First is a scalar method (like the one we implemented) - a scalar method optimizes a function of a single variable. Then, are the multivariate methods - they optimize a function with a vector of variables. The hill-descent method we implemented can be augmented to be a multivariate method (and is described as such in the Wikipedia page). But there are many more methods that can be considered as well. We do not have time to talk about all of the algorithms, but we can run some of them and compare their performance to that of the one we implemented. This also allows us to write cost functions with multiple variables. Let's try finding a set of parameters for the Goodwin oscillator that allow it to oscillate with a period of 24 h.

In the notebook we have mixed success. If we start with a set of parameters close enough to the minimum to be in the same "well" in the cost function landscape, then they work. If not, they don't.

So what do we need? We need a global method that will not get trapped in local minima.