

CS341 SYSTEMS BIOLOGY I

LECTURE 11

NUMERICAL METHODS FOR SOLVING DIFFERENTIAL EQUATIONS (CONT)

In lecture 10, we implemented the forward Euler method and examined the effects of time step on the simulation output. We observed that if the step is smaller, the error is smaller. We ended the lecture with a computation (of the region of absolute stability) that would give us a bound on the time step size. For the test equation, the step size must be smaller than $2/\lambda$, where λ is the degradation rate constant.

Today, we confirm that this bound is appropriate for the test equation. However, we will not do what I really wish we had time to do, and that is talk in detail about how this insight leads to insights for models other than the test equation. What I will say is that λ is the rate of change of f with respect to y and that there is a name for this derivative. It is called the Jacobian matrix. And we can compute the Jacobian matrix for more complicated models, and estimate λ from that by computing the largest eigenvalue. The insight I want you to gain from this example is that the forward Euler method has an explicit region of absolute stability and that bounds on the step size depend on the model. And if the model's values are changing more dramatically, we need a smaller step size.

FORWARD EULER FOR MODELS WITH MORE THAN ONE STATE VARIABLE

Consider the system $\frac{d\vec{y}(t)}{dt} = f(t, \vec{y})$ and $\vec{y}(0) = y_0$

where $\vec{y}$ is the vector of state variables. Then f is a function that returns a vector of derivatives.

The forward Euler method for approximating y is

$$\hat{y}(t + \Delta t) = \hat{y}(t) + \Delta t \cdot f(t, \hat{y}(t))$$

where $\hat{y}(t + \Delta t)$ is a column vector, $\hat{y}(t)$ is a column vector, Δt is a scalar, and $f(t, \hat{y}(t))$ is a column vector.

We implement the multi-state version of forward Euler in the Jupyter notebook and test it with a model that is really just two copies of the degradation model, together.

$$\begin{aligned}\frac{dy_1}{dt} &= -\lambda_1 \cdot y_1(t) \\ \frac{dy_2}{dt} &= -\lambda_2 \cdot y_2(t)\end{aligned}$$

One goal in the next few weeks is not only to simulate models with our own solvers, it is also to understand how well they perform. So we will want a way of computing the error other than plotting it for each state over time. How can we get that down to 1 number? In the Jupyter notebook, we compute the error between the true and approximate solutions using multiple methods.

We need to determine the error at each time step. In a multi-state model, there are multiple viable options for this computation (Euclidean distance, sum of absolute distance, maximum distance).

Then we need to determine how to summarize the error over time. There are also multiple viable options for this computation (mean, max).

Our code uses these approaches:

- Find the error at each time step by computing the Euclidean distance between the true solution and the approximate solution (so we know how far away they are in state space), and then taking the mean across time.
- Find the error at each time step by computing the maximum distance between the true solution and the approximate solution (so we know how far away they are in state space), and then taking the mean across time.