

CS341 SYSTEMS BIOLOGY I

LECTURE 12

NUMERICAL METHODS FOR SOLVING DIFFERENTIAL EQUATIONS (CONT)

Effect of step size. The choice of step size determines how accurate the solution is and how long it takes to find the solution (more steps=more time). In the Jupyter notebook, we explored several step sizes that provide the correct qualitative answer, and show this trend. We will return to this topic of accuracy several times over the next few weeks.

We have to accept a trade-off between run-time and accuracy. It would be nice to have fewer steps, but that means bigger steps, and less accuracy. Let's first ask an extreme question: How big is way too big?

Let's pick a step-size that we think may be too big. In the notebook, we ran the code and found the solution went nuts, so the method was considered *unstable*.

We want to know then the forward Euler method will be able to make a qualitatively decent (stable) approximation). We want to know when the approximation won't blow up. I.e. we want to be sure the solution is not increasing over time (actually, we will be even stricter and make sure it is decreasing over time). How can we tell how the time step affects that?

Answer: We examine how the method behaves on the test equation and answer these questions for the test equation. We want to determine under what conditions we will be guaranteed that the solution decreases over time. For the method's solution to decrease over time, we need

$$|\hat{y}(t + \Delta t)| < |\hat{y}(t)|$$

When does that happen with the test equation?

$$\begin{aligned} |\hat{y}(t + \Delta t)| &= |\hat{y}(t) + \Delta t \cdot (-\lambda) \cdot \hat{y}(t)| \\ &= |\hat{y}(t)(1 + \Delta t \cdot (-\lambda))| \end{aligned}$$

Since the ratio of $|\hat{y}(t + \Delta t)|$ to $|\hat{y}(t)|$ must be less than or equal to 1 (the solution is not increasing in magnitude!), we find that

$$|1 + \Delta t \cdot (-\lambda)| < 1$$

which means that $1 + \Delta t \cdot (-\lambda) < 1$ and $1 + \Delta t \cdot (-\lambda) > -1$.

Let's see what that means for Δt in terms of λ .

First, $1 - \lambda\Delta t < 1$ will be true as long as Δt and λ are positive, so that doesn't really constrain us.

$$\begin{aligned} 1 - \lambda\Delta t &> -1 \\ -\lambda\Delta t &> -2 \\ \Delta t &> \frac{-2}{\lambda} \\ \Delta t &< \frac{2}{\lambda} \end{aligned}$$

For the example we have been using in the Jupyter notebook, $\lambda = 2$, which means our steps size must be less than $\frac{2}{2} = 1$.

In our Jupyter Notebook, 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.

$$\begin{aligned}\frac{dy1}{dt} &= -\lambda_1 \cdot y1(t) \\ \frac{dy2}{dt} &= -\lambda_2 \cdot y2(t)\end{aligned}$$

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.