CS341 SYSTEMS BIOLOGY I

PROJECT 3 (DUE THURS. NOV. 2, 2017)

Your goal is to assess the trade-offs of stability, accuracy, and run-time of a least two methods applied to two ODE systems.

The first ODE system is two proteins degrading. The ODEs are:

$$\begin{aligned} \frac{dy_1}{dt} &= -\alpha y_1 \\ \frac{dy_2}{dt} &= -\beta y_2 \end{aligned}$$

where $\alpha = 0.1$ and $\beta = 0.2$.

The second ODE system is the Lotka-Volterra model of predator-prey populations and their interactions. You can check out the Wikipedia entry to learn more, but what you need to know here is that it is a 2-ODE system and that, with the right parameter values, it leads to oscillations. The ODEs are:

$$\begin{aligned} \frac{dx}{dt} &= \alpha x - \beta xy \\ \frac{dy}{dt} &= -\gamma y + \delta xy \end{aligned}$$

where $\alpha = 0.25$, $\beta = 0.01$, $\gamma = 1$, and $\delta = 0.01$.

The Project

Consider three methods – the forward Euler method, the explicit trapezoidal method, and an adaptive step-size forward Euler method (that uses the trapezoidal method to estimate the local error).

Compare the performance of each of your three methods on the degradation and predator-prey systems. In your analysis, you will want to consider both efficiency (how long does it take the method to run?) and accuracy (how close is the method to the "true" solution?). For the first two methods, be sure to consider the effect of step-size on accuracy and speed. For the third, be sure to analyze the step-sizes chosen and the effect of error tolerance on the step size.

Extensions

This project will be more interesting if you tackle one or more of these extensions. Feel free to develop your own extension. And, as always, a particularly thorough analysis is smiled upon.

- Implement the Backward Euler method and compare its results to those of your other methods. (Note that backward euler is an implicit method, which means the solution at the new time step appears on both sides of the equation and, therefore, requires additional computation. For example, you will need to use Newton's method to approximately solve a nonlinear system of algebraic equations.)
- Analyze the van der Pol oscillator in addition to the models above. The equations for it are:

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

  When $\mu = 50$, the period of oscillation is long (nearly 80) and the system is stiff. When $\mu = 1$, the period is short (approximately 6) and the system is not stiff. We expect implicit methods to perform better on the stiff system. Demonstrate that it is better to use the implicit method when $\mu = 50$, but that explicit methods are perfectly adequate when $\mu = 1$.

- Implement additional explicit Runge-Kutta methods. In particular, it would be nice to see the results of a fourth order method.

Notes

To time your method, use Matlab's **tic** and **toc** statements like this:

```
tic;
[t,yfe] = forwardEuler(my_model,t,y0);
toc;
```

To compute the accuracy of your method, compare its output to the output of Matlab's ode15s (which will stand in for the "true solution"). Set its relative tolerance so that it is "tight" (i.e. you will tolerate very little error). Error calculations should be performed by comparing the solutions to y at the same time steps, but you will likely be comparing the true solution to outputs from methods using lots of different time steps. The best way to handle this is to call ode15s in a new way. By providing just the first and last times (t0 and tfinal) in the input, and asking for just one output, you can instruct odes15s to provide you a record of its internal time steps and additional information.

```
sol = ode15s(my_model,[t0 tfinal],y0,odeset('RelTol',1e-9));
```

So you call ode15s just once for each model. Then you can use that record to compute the values of y for any series of time steps between t0 and tfinal, using the function deval like this:

```
y = deval( sol, t0:0.1:tfinal );
```

To compute the error, you may want to use the maximum absolute difference between ym and yfe over time.

Another measure, which captures the error over time, is the vector 2-norm. The following code demonstrates how to compute the absolute error for the first state (using the time steps used for the forward euler call above).

```
ym = deval( sol, t );
errFE1 = norm(yfe(:,1)-ym(:,1));
```

For your analysis, you should compute the error for both states, so code like this is appropriate:

```
errFE2 = norm(yfe(:,2)-ym(:,2));
errFE = mean([FE1 FE2]);
```

One thing you will need to consider is whether an absolute error measure or a relative error measure is more informative.