Dynamic Models in Biology, Computer Lab:

Numerical integration of ODEs in Matlab

In this computer lab, you will numerically integrate ODEs using Matlab's built-in `ode45` solver. This solver is based on a so-called Runge-Kutta algorithm and is more robust and efficient than the simple Euler method for numerical integration.

1. Simulating exponential decay

Matlab example file

The file `lab1.m` is a Matlab file that uses the `ode45` solver to numerically integrate a model whose dynamics are exponential decay, e.g., a bathtub drain model. The governing differential equation is

$$\dot{x} = -kx,$$

where the parameter `k` gives the decay rate.

The differential equation is implemented via a function that gives $\dot{x}$. Open the `lab1.m` file in Matlab and you'll find this function in lines 17-23. Note that Matlab uses % to comment text, so that a line of %'s is simply used to visually separate the function from the main program.

In addition to the actual differential equation, Matlab needs a few more pieces of information to run the integration. For one, it needs an initial condition (`x0`, given in line 2, and fed into the solver in line 8). Matlab also needs the duration of the simulation, defined by `t_end` in line 5. This end time is passed to the solver (in addition to the start time of zero) as the second argument (line 8).

It is often helpful, and sometimes necessary, to specify an error tolerance. We're not going to worry about the details of this tolerance in this course - the relative tolerance of 10^{-8} used in this example (line 7) will almost certainly suffice in your Dynamic Models in Biology modeling work.

Finally, the `lab1.m` file provides some plotting instructions. In line 11, we tell Matlab to plot `x` as a function of `t` using a blue dot for each data point and solid blue lines connecting the data points. To get a better sense of the options available for plotting, check out the documentation on the `plot` function by typing `doc plot` at the Matlab prompt. Note that the `hold on` call (line 12) allows subsequent plotting commands to be added to this figure.

Running the simulation
You are now ready to run your simulation. Type `lab1` at the Matlab prompt. Examine the output.

To save figures of your output (to put in your lab report!), use the `print` command. For example,
```
print -dpdf exp_decay_figure.pdf
```
will save your figure as a pdf file. Use `doc print` to get further information on saving to specific file formats.

Messing with the output
The `ode45` algorithm uses variable integration time steps, Δ`t`. In general, Δ`t` will vary over the course of a simulation, being small when variables change quickly and larger when variables change more slowly. This makes the algorithm more efficient than the Euler method. Because of the variable Δ`t`, the `x` values returned by the `ode45` in the routine are not evenly spaced and you must use the corresponding `t` vector in conjunction with it to interpret the results.

To see this, try plotting the `x` values as a function of an evenly-spaced `t` vector on the same interval. To create this `t`-vector, you may enter:
```
dt = t_end/(length(t)-1);
t_even = 0:dt:t_end;
```
at the Matlab prompt.

Now plot this result in the same figure as the proper result:
```
plot(t_even, x, 'r.-')
```

Compare the two traces. And never forget to plot your simulation result as a function of the correct `t` vector!

2. Simulating the lactose switch model

Lactose model
Recall from the lecture the simple lactose switch model, given by the ODE:

$$\dot{x} = \frac{a + x^2}{1 + x^2} - kx,$$

where `a` and `k` are parameters.

Implementation
Using the `lab1.m` file as a template, make a Matlab program that numerically integrates

the lactose switch model. Use `a=0.05`, `k=0.3`, and an initial condition of `x0=5`. Run the simulation. Does `x` reach a steady-state? If not, increase `t_end` until it does.

Effect of changing the initial condition
Now, run simulations in which you vary the initial condition. Try several different values for `x0` in the range from 0 to 5 to convince yourself that there is only one fixed point.

Investigation of model dynamics
Increase `k` to 0.6. What type of dynamics do you see? Again, vary the initial condition to check that there is only one fixed point.

Bistability
Use Matlab to plot the two components of the $f(x) = \frac{a+x^2}{1+x^2} - kx$ function, i.e., $g(x) = \frac{a+x^2}{1+x^2}$ and $h(x) = kx$. You may try something like:

```
x = 0:.01:5;
```

to generate a vector of `x` values over which to evaluate the functions. Then use:

```
g = (a+x.^2)./(1+x.^2);
h = k*x;
```

to generate the functions, and finally:

```
plot(x, g, 'r-'), hold on, plot(x, h, 'g-')
```

to plot the curves. (Please ask if you don't get the syntax.)

Recall from the lecture that there are fixed points where `g` and `h` intersect. Based on these curves, find a value of `k` for which there is bistability.

Use this value of `k` in your model to run simulations and verify the presence of the two fixed points. Varying the initial condition `x0`, can you determine the value of `x0` that separates the two basins of attraction? Compare this value to the value of the separating point (i.e., the unstable fixed point) on the graph of `g` and `h`.

Bifurcations
Use numerical integration or graphing of `g` and `h` to determine the values of `k` for which the two saddle-node bifurcations occur.