ODEs Bootcamp Day 2

Numerical Simulations

Create a document containing your answers and relevant figures for each exercise and upload this to GitHub along with all of the code you used to generate your answers (fully commented). You may use whichever programming language you are most comfortable in. If you're using R Markdown or Jupyter notebook, feel free to put everything in a single document.

Send the web address of your GitHub repository to niklas.brake@mail.mcgill.ca (make sure your repository is publicly visible). Please send me your solutions before **Tuesday, February 4th**.

Exercise 1. When we model population growth with a differential equation, we are saying that births are occurring continuously. This may be approximately true for a colony of bacteria but not for a species, such as pacific salmon, that gives birth at a specific time each year. In this case, it is more natural to model change in population from one year to the next. Such models are called *difference equations* and are the discrete version of a differential equation.

Consider the following difference equation, called the logistic map:

$$x_{n+1} = rx_n(1 - x_n)$$

Here, $x_n \geq 0$ is a dimensionless measure of the population in the nth generation and $r \geq 0$ is the intrinsic growth rate. Suppose $x_0 = 0.1$.

Using a programming language of your choice, compute and plot the population size over the first 50 generations for $r = 0.5, 2.8, 3.3$. What did you notice with different values of r? Can you give a heuristic explanation for the behaviour you observe?

Exercise 2. Recall that we can write a first-order ODE as follows:

$$x(t + dt) = x(t) + \frac{dx(t)}{dt}dt$$

as $dt \to 0$. We can iterate this equation numerically by approximating dt with a small *step size*, h. This is called the forward Euler method for solving first-order ODEs.

Algorithm 1 Forward Euler

Require: The differential equation, $dx/dt = f(x)$; initial condition, x_0; time to integrate until, t_{max}; step size, h

```
1: function INTEGRATE($f(x), x_0, t_{max}, h$)
2:     $x[0] \leftarrow x_0$
3:     $t[0] \leftarrow 0$
4:     for $i \leftarrow 1$ to $\lceil t_{max}/h \rceil$ do
5:         $x[i] \leftarrow x[i-1] + hf(x[i-1])$        ▷ Approximate differential equation
6:         $t[i] \leftarrow ih$
7:     end for
8:     return $x, t$
9: end function
```

Using the programming language of your choice, implement the Forward Euler algorithm and compute the solution to the following initial value problem:

$$\frac{dx}{dt} = x \qquad x(0) = 1$$

for $t \in [0, 5]$, for the following step sizes: $h = 0.1$, 0.01, 0.001. Plot these solution overlaid with the function $x(t) = \exp(t)$. What do you observe? Is this what you would expect? Why or why not?

Exercise 3. The FitzHugh–Nagumo model is a simplified 2-dimensional model that captures the qualitative behaviour of the Hodkgin-Huxley model (4D) of neuronal excitability:

$$\frac{dv(t)}{dt} = v - v^3/3 - w + I \tag{1}$$

$$\frac{dw(t)}{dt} = \varepsilon\,(v + a - bw) \tag{2}$$

where I represents the applied current. We will use the following parameter values: $a = 0.7$, $b = 0.8$, $\varepsilon = 0.08$.

1. Compute the solution to this system until $t = 400$ for $v(0) = 1$, $w(0) = 0.1$, $I = 0.5$ (a step size of 0.01 should suffice). Plot the variable v against t – you should see oscillations.

2. Plot v against w, that is, look at the trajectory in phase space. Notice that after a small transient, the trajectory stays on a loop in phase space. This is a special type of attractor called a limit cycle. Calculate the v-nullcline and w-nullcline and graph these on the same plot as your trajectory.

3. Calculate the fixed-point of the system. Finding v^* will involve finding the roots of a degree 3 polynomial. You can do this numerically with the following commands: MATLAB, `roots`; python[1], `numpy.roots`; R, `polyroot`.

4. Calculate the Jacobian Matrix of the system. Evaluate it at the fixed-point from 3. Now compute the eigenvalues of this matrix. You can do this with the following commands: MATLAB, `eig`; python, `numpy.linalg.eig`; R, `eigen`. You should find that both eigenvalues are complex numbers, with positive real parts[2]. How could you have guessed this from your phase-plane plot?

5. Repeat the above steps with $I = 0$. You should find that the system does not oscillate and the fixed-point now has negative real parts (it is now a *stable focus*).

6. Write a `for` loop to calculate the steady-state values (v^*,w^*) and its stability for values of I ranging between 0 and 0.5. Graph v^* against I and plot in blue where v^* is stable and red where v^* is unstable. It should look like figure 1.

[1]You must have the package `numpy` installed.

[2]Such a steady-state is called an *unstable focus*.

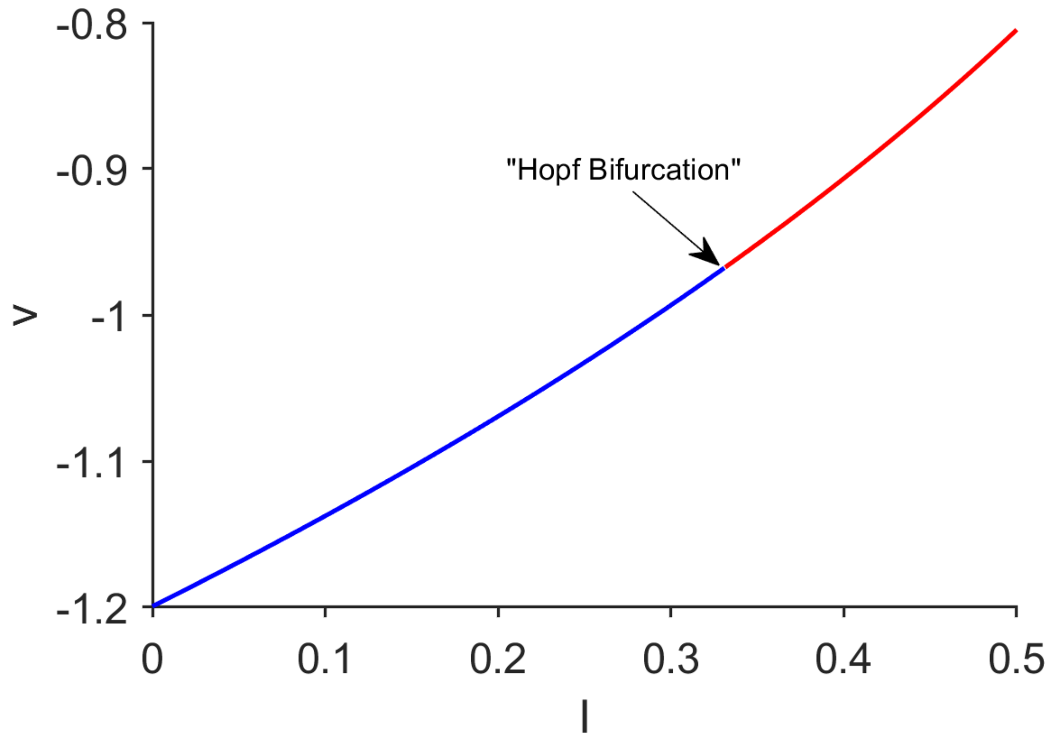

Figure 1: A bifurcation diagram of the fixed-point stability in the FitzHugh-Nagumo model (see eqs. 1-2) with parameters $a = 0.7$, $b = 0.8$, $\varepsilon = 0.08$, as I varies. A Hopf bifurcation occurs when a stable focus switches to an unstable focus and gives birth to a limit cycle. To indicate a limit cycle on a bifurcation diagram, we often plot the maximum and minimum value attained during the cycle (called the upper and lower branch). You can calculate these by computing a solution trajectory for each parameter value, but this can be very computationally intensive, especially for higher dimensional systems. In programs, such as `AUTO`, bifurcation diagrams are computed efficiently using *numerical continuation* methods.