ODE Bootcamp assignment

Cristian Groza

January 31, 2020

Github:

Repository available at: `https://github.com/cgroza/QLSC600-ODEs`

Exercise 1

First, lets run the model for all values of r for 50 generations:

```
library(tidyverse)
logistic.map <- function(r){
  x_i <- vector('numeric', length=51)
  x_i[1] <- 0.1

  for(i in 2:51 ){
    x_i[i] <- r * x_i[i - 1] * (1 - x_i[i - 1])
  }
  x_i
}
curves <- lapply(c(0.5, 0.9, 2.8, 3.3), logistic.map)
names(curves) <- c("0.5", "0.9", "2.8", "3.3")
```

Then let's plot the three curves:

We observe that for values of $r \leq 1$, the population decays to zero. For $r = 2.8$, the system reaches a steady state equilibrium. For $r = 3.3$, the system reaches an oscillating equilibrium.

Exercise 2

We implement the Forward Euler algorithm and run it for several step sizes:

```
forward.euler <- function(f.x, x.0, t.max, h){
  iter <- ceiling(t.max/h)
  x_i <- vector("numeric", length = iter)
  x_i[1] <- x.0
  t_i <- vector("numeric", length = iter)
  t_i[1] <- 0

  for(i in 2:(iter+1)) {
```

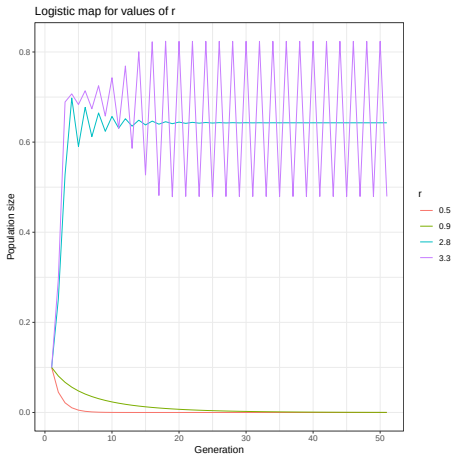

Figure 1: Logistic mapping curves plotted with various parameter values.

```
    x_i[i] <- x_i[i - 1] + h * f.x(x_i[i - 1])
    t_i[i] <- i * h
    }
  tibble(x=x_i, t=t_i, h=as.character(h))
}
solutions <- lapply(c(0.1, 0.01, 0.001), function(h) forward.euler(identity, 1, 6, h))
solutions <- do.call(rbind, solutions)
```

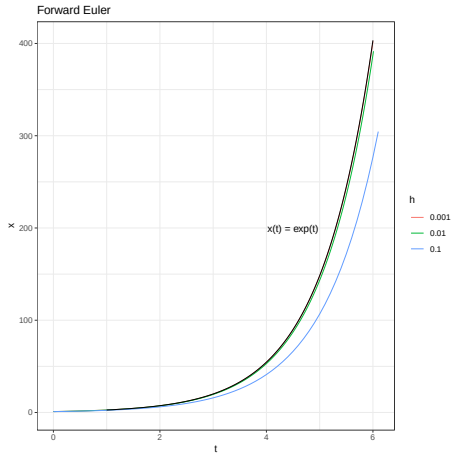

Figure 2: Forward Euler solution with several step sizes.

As we decrease h, the outcome of the simulation becomes closer to the real scenario $x(t) = exp(t)$. The resolution of the simulation increases.

Exercise 3

We implement the Forward Euler algorithm for two dimensions and run the model:

Exercise 3.1

```
forward.euler.2d <- function(dv.dt, dw.dt, v.0 = 1, w.0 = 0.1, I = 0.5, a = 0.7,
                             b = 0.8, epsilon = 0.08, t.max = 400, h = 0.01){
  iter <- ceiling(t.max/h)
  v_i <- vector("numeric", length = iter)
  v_i[1] <- v.0

  w_i <- vector("numeric", length = iter)
  w_i[1] <- w.0

  t_i <- vector("numeric", length = iter)
  t_i[1] <- 0

  for(i in 2:(iter+1)) {
    v_i[i] <- v_i[i - 1] + h * dv.dt(v_i[i - 1], w_i[i - 1], I)
    w_i[i] <- w_i[i - 1] + h * dw.dt(v_i[i - 1], w_i[i - 1], a, b, epsilon)
    t_i[i] <- i * h
  }
  tibble(v = v_i, w = w_i, t=t_i, h=as.character(h))
}
simulation = forward.euler.2d(function(v, w, I) {v - v^3/3 - w + I},
                 function(v, w, a, b, epsilon) {epsilon*(v + a - b*w)}

)
```

We see that both v and w oscillate with t

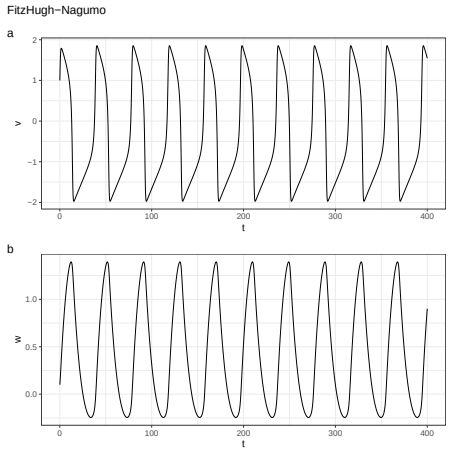

Figure 3: Two dimensional Forward Euler solutions for a) v and b) w.

Exercise 3.2

The $v - nulcline$ is:

$$\frac{dv(t)}{dt} = 0 = v - \frac{v^3}{3} - w + I \implies w = v - \frac{v^3}{3} + I$$

The $w - nulcline$ is:

$$\frac{dw(t)}{dt} = 0 = \epsilon(v + a - bw) \implies v = bw - a$$

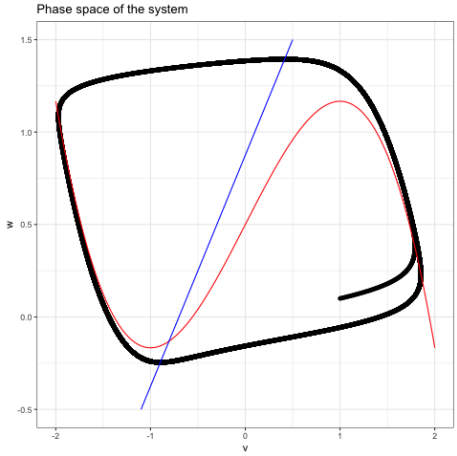

Figure 4: The trajectory of (v, w) in this system. The v-nulcline (red) and w-nulcline (blue) are overlaid on the trajectory.

Exercise 3.3

We need to solve the system:

$$\begin{cases} 0 = \frac{dv}{dt} = v - \frac{v^3}{3} - w + I \\ 0 = \frac{dw}{dt} = \epsilon(v + a - bw) \end{cases}$$

We recognize the equations of the w and v nulcline. The solution can readily be retrieved from the intersection of the v and $w$nulclines in Figure 4.

$$(v^*, w^*) = (-0.80484, -0.30484)$$

Exercise 3.4

The Jacobian matrix is:

$$J = \begin{bmatrix} 1 - v^2 & -1 \\ \epsilon & -b\epsilon \end{bmatrix}_{(v^*, w^*)} = \begin{bmatrix} 0.3522326 & -1 \\ 0.08 & -0.064 \end{bmatrix}$$

Eigen-decomposition

```
eigen <- eigen(matrix(c(0.3522326, -1, 0.08, -0.08 * 0.8), nrow = 2, ncol = 2, byrow=T))
data.frame(Eigenvalue = eigen$values)
```

Indeed, we find positive real parts:

Index	Eigenvalue
1	0.1441163+0.1915401i
2	0.1441163-0.1915401i

We could have guessed this from the fact that the fixed points of the system do not overlap its trajectory in phase space. Therefore, the system cannot reach an equilibrium and is not stable.

Exercise 3.5

Rerun the system with $I = 0$.

```
simulation.newI = forward.euler.2d(function(v, w, I) {v - v^3/3 - w + I},
                 function(v, w, a, b, epsilon) {epsilon*(v + a - b*w)}, I = 0)
```

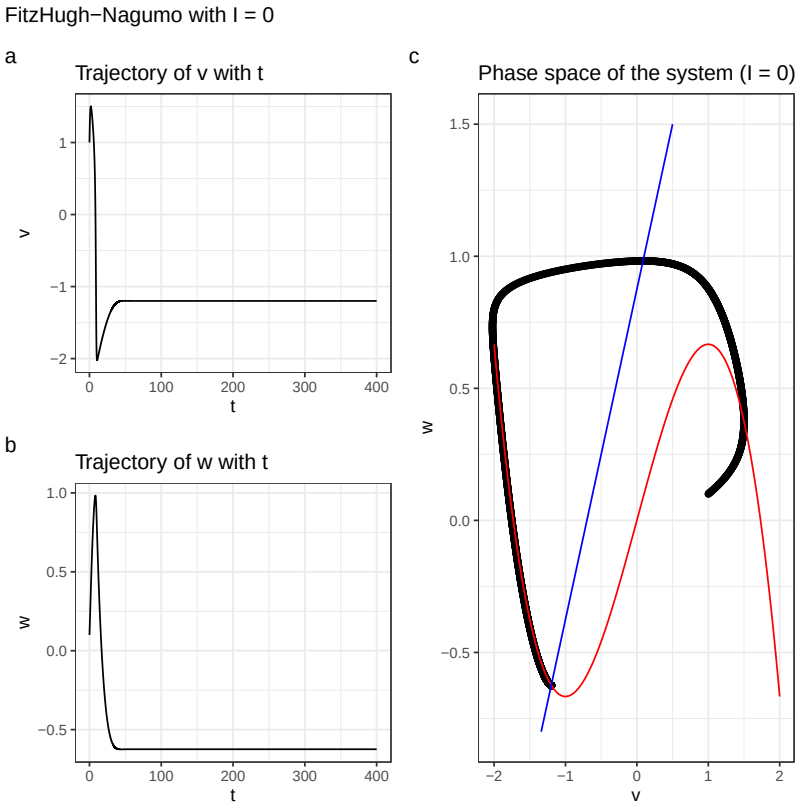

Figure 5: Results with I=0.

We obtain the fixed points for $I = 0$ as before: $(v^{**}, w^{**}) = (-1.1994, -0.6242533)$. The Jacobian matrix is

$$J = \begin{bmatrix} 1 - v^2 & -1 \\ \epsilon & -b\epsilon \end{bmatrix}_{(v^{**}, w^{**})} = \begin{bmatrix} -0.4385604 & -1 \\ 0.08 & -0.064 \end{bmatrix}$$

Eigen-decomposition

```
eigen <- eigen(matrix(c(-0.4385604, -1, 0.08, -0.08 * 0.8), nrow = 2, ncol = 2, byrow=T))
data.frame(Eigenvalue = eigen$values)
```

Index	Eigenvalue
1	-0.2512802+0.2119578i
2	-0.2512802-0.2119578i

We see negative real values, suggesting that both eigenvectors point to stable directions.

Exercise 3.5

First, we run the Jacobian for every value of I:

```
v.fixed_i <- vector("numeric", length = 501)
I_i <- vector("numeric", length = 501)
eigen.value_i <- vector("numeric", length = 501)
i <- 1
for(I in seq(0, 0.5, by = 0.001)){
  roots <- polyroot(c(I - 0.7/0.8, 1 - 1/0.8, 0, -1/3))
  v.fixed <- Re(roots[abs(Im(roots)) < 1e-10])

  v.fixed_i[i] <- v.fixed
  I_i[i] <- I

  eigen <- eigen(matrix(c(1 - v.fixed^2, -1, 0.08, -0.08 * 0.8),
                        nrow = 2, ncol = 2, byrow=T))
  eigen.value_i[i] <- Re(eigen$values)[1]
  i <- i + 1
}
stability <- tibble(v.fixed = v.fixed_i, I = I_i,
                    Stable = ifelse(eigen.value_i > 0, "Unstable", "Stable"))
```

Then we distinguish between negative and positive real parts of the eigenvalues to determine if the fixed point is stable or unstable. The results are plotted below:

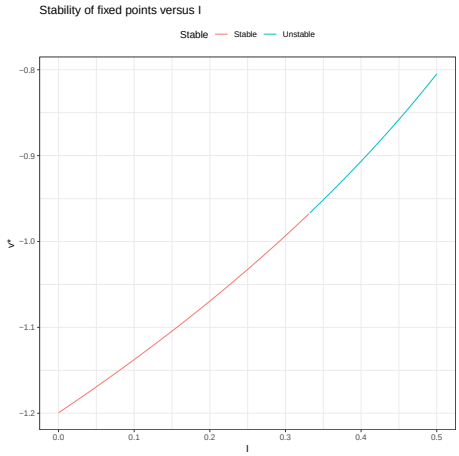

Figure 6: The value of I influences the stability of the fixed point.