ODE_bootcamp

Shael Brown

2/1/2020

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

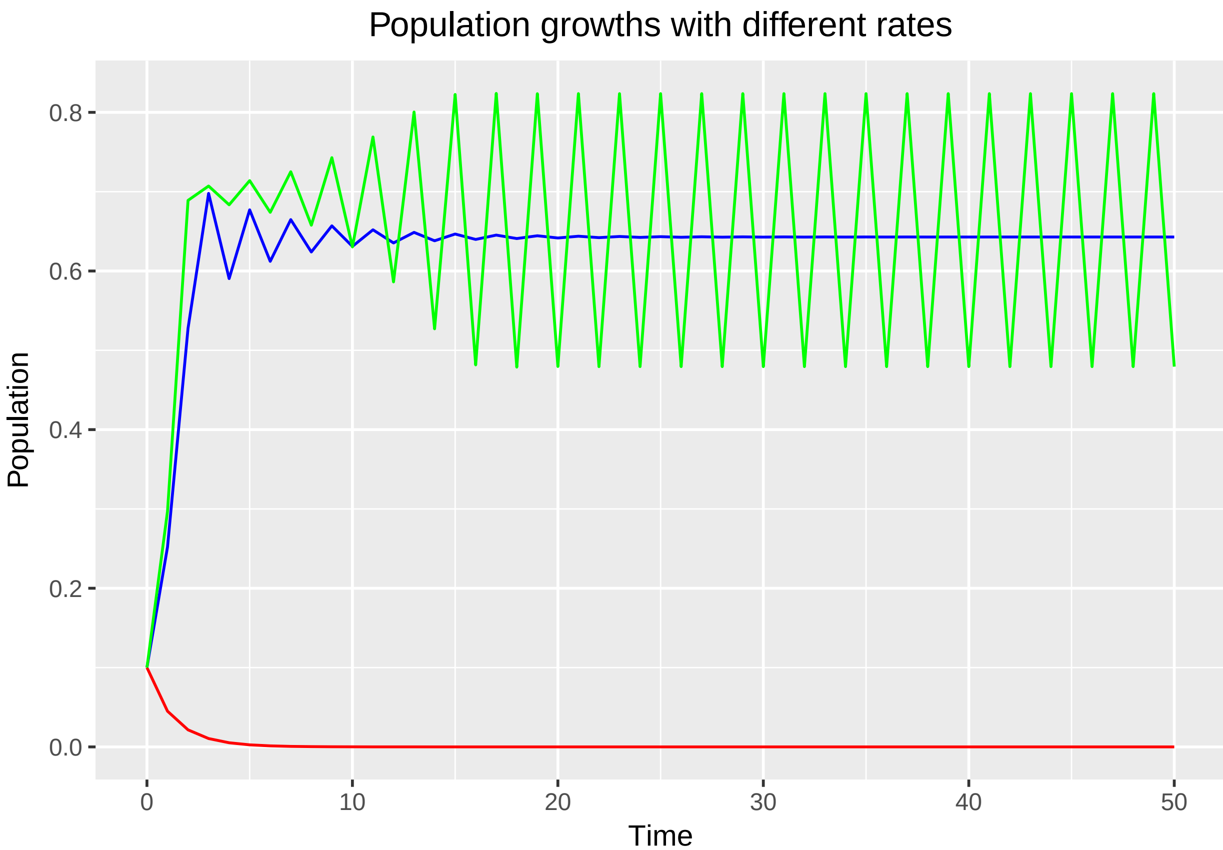
Question 1

We consider a continuous population growth model given by the equation $x_{n+1} = rx_n(1 - x_n)$. To explore the behavior of this model we can first plot the first 50 populations values under various r parameters - 0.5, 2.8 and 3.3. The following code sets up the three tables.

```
rate_0.5 <- data.table(n = c(0),x_n = c(0.1))
rate_2.8 <- data.table(n = c(0),x_n = c(0.1))
rate_3.3 <- data.table(n = c(0),x_n = c(0.1))

for(i in 1:50)
{
  rate_0.5 <- rbind(rate_0.5,
                    data.table(n = c(i),
                               x_n = c(0.5*rate_0.5[nrow(rate_0.5),x_n]*(1-rate_0.5[nrow(rate_0.5),x_n]
  rate_2.8 <- rbind(rate_2.8,
                    data.table(n = c(i),
                               x_n = c(2.8*rate_2.8[nrow(rate_2.8),x_n]*(1-rate_2.8[nrow(rate_2.8),x_n]
  rate_3.3 <- rbind(rate_3.3,
                    data.table(n = c(i),
                               x_n = c(3.3*rate_3.3[nrow(rate_3.3),x_n]*(1-rate_3.3[nrow(rate_3.3),x_n]
}
```

If we plot the results together, with $r = 0.5$ in red, $r = 2.8$ in blue and $r = 3.3$ in green we can see that for large enough values of r the population does not seem to converge however for smaller values it does.



With some elementary calculus, we notice that if the sequence $\{x_n\}_{n=1}^{\infty}$ has a limit, L, then $L = rL(1-L)$ which implies that either $L = 0$ or $L = 1 - \frac{1}{r}$. If $r \leq 1$ and $x_0 < 1$ then $x_{n+1} = rx_n(1-x_n) \leq x_n(1-x_n) < x_n$. Therefore, since the sequence is positive and monotone decreasing it has a limit of 0. However, if $r > 1$ then there is no guarantee of a decreasing sequence, as seen in the plots.

Question 2

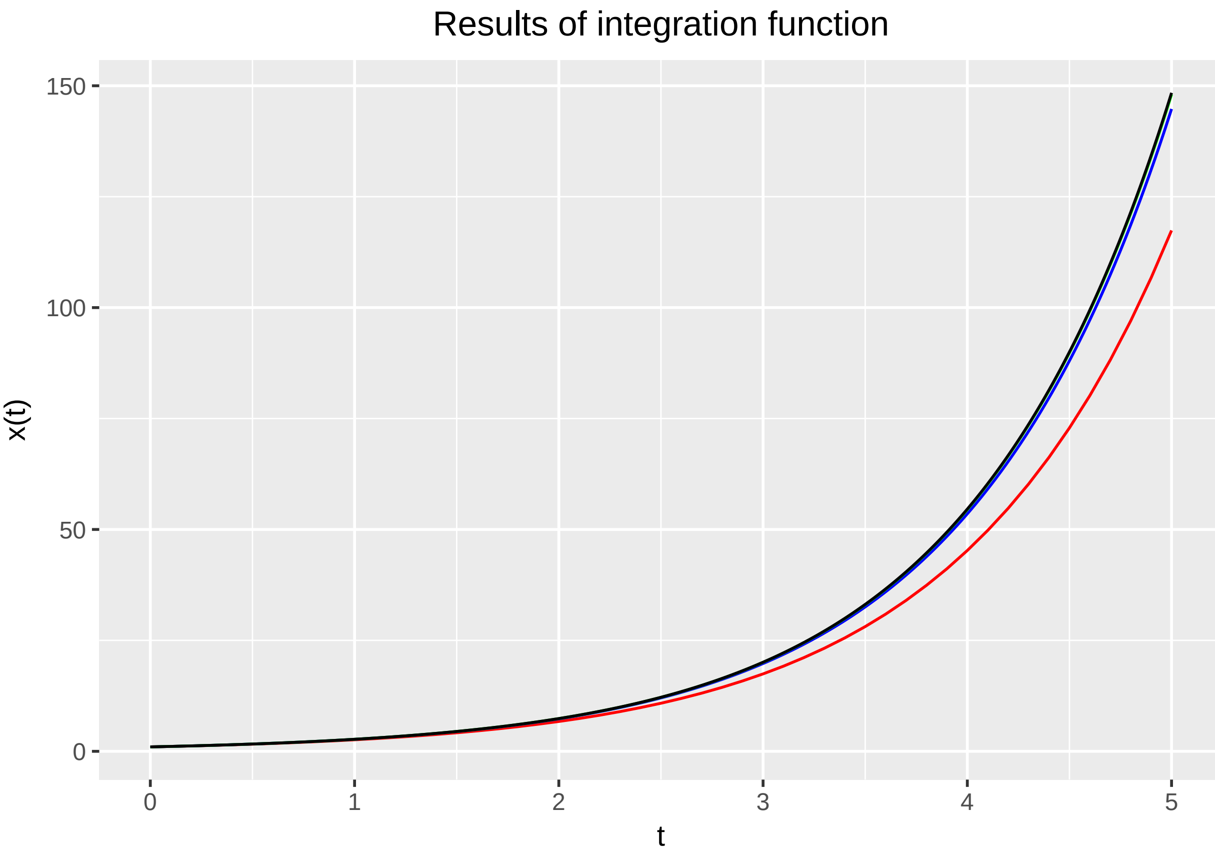
We can implement the forward Euler algorithm as follows:

```
integrate <- function(f,x_0,t_max,h){
  x <- c(x_0)
  t <- c(0)
  for(i in 1:ceiling(t_max/h))
  {
    x = c(x,x[[length(x)]] + h*f(x[[length(x)]]))
    t = c(t,i*h)
  }
  return(data.table(t = t,x = x))
}
```

As an example, we can run this algorithm with $f(x) = x$, $t_{max} = 5$, $x_0 = 1$ and h taking the values 0.1, 0.01 and 0.001 to get better and better approximations to the solution.

```
h_0.1 <- integrate(f = function(x){return(x)},x_0 = 1,t_max = 5,h = 0.1)
h_0.01 <- integrate(f = function(x){return(x)},x_0 = 1,t_max = 5,h = 0.01)
h_0.001 <- integrate(f = function(x){return(x)},x_0 = 1,t_max = 5,h = 0.001)
```

We can plot these approximations with the curve $x(t) = exp(t)$ in black:



This is what we would expect since as the step size gets closer to 0 we approximate the derivative better and therefore the solution better as well.

Question 3

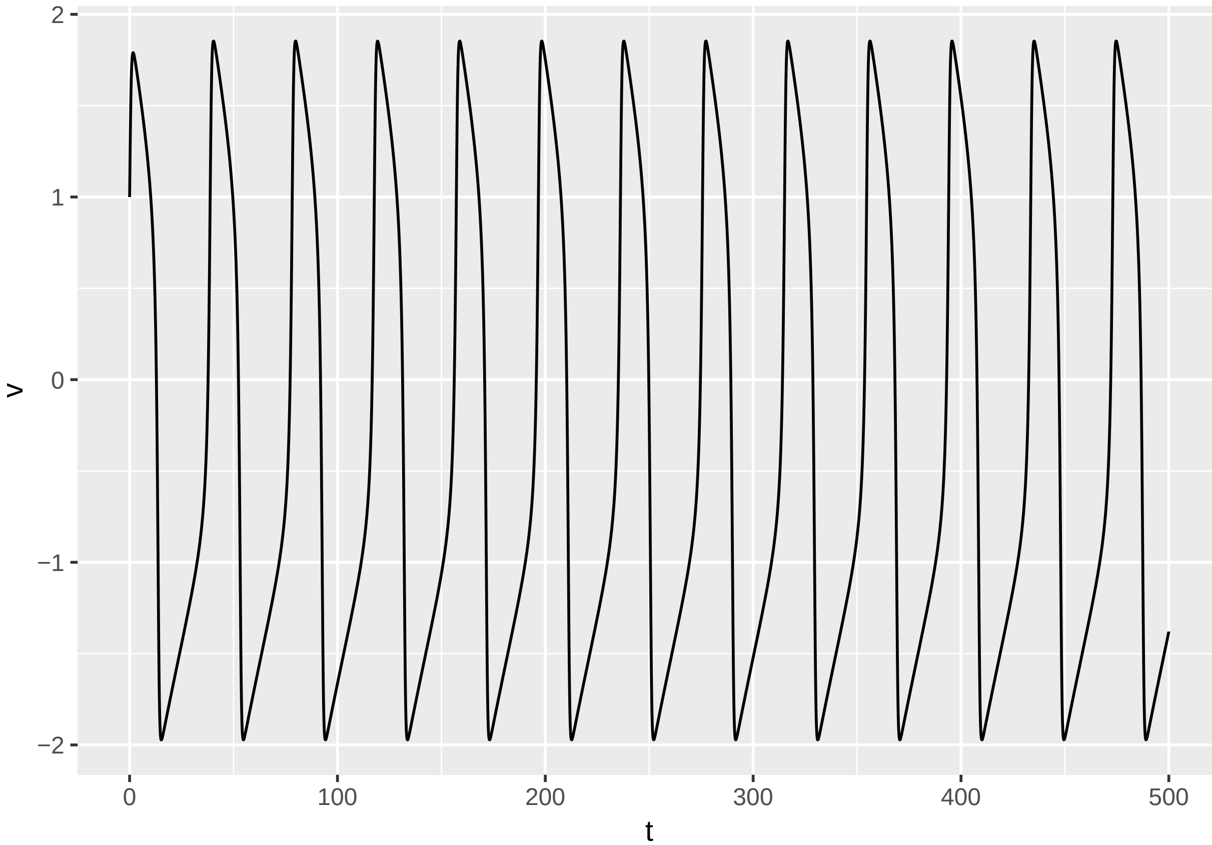
Part 1

Let $\frac{dv(t)}{dt} = v - \frac{v^3}{3} - w + I$ and $\frac{dw(t)}{dt} = \epsilon(v + a - bw)$ represent the FitzHugh-Nagamo model, where $a = 0.7$, $b = 0.8$ and $\epsilon = 0.08$. We can compute the solution to this system up to $t = 400$ with $v(0) = 1$, $w(0) = 0.1$, $I = 0.5$ and $h = 0.01$ using an adapted integrate function:

```
integrate_2D <- function(f,g,v_0,w_0,h,t_max){
  v <- c(v_0)
  w <- c(w_0)
  t <- c(0)
  for(i in 1:ceiling(t_max/h))
  {
    v = c(v,v[[length(v)]] + h*f(v = v[[length(v)]],w = w[[length(w)]]))
    w = c(w,w[[length(w)]] + h*g(v = v[[length(v) - 1]],w = w[[length(w)]]))
    t = c(t,i*h)
  }
  return(data.table(t = t,v = v,w = w))
}
FN <- integrate_2D(f = function(v,w)
```

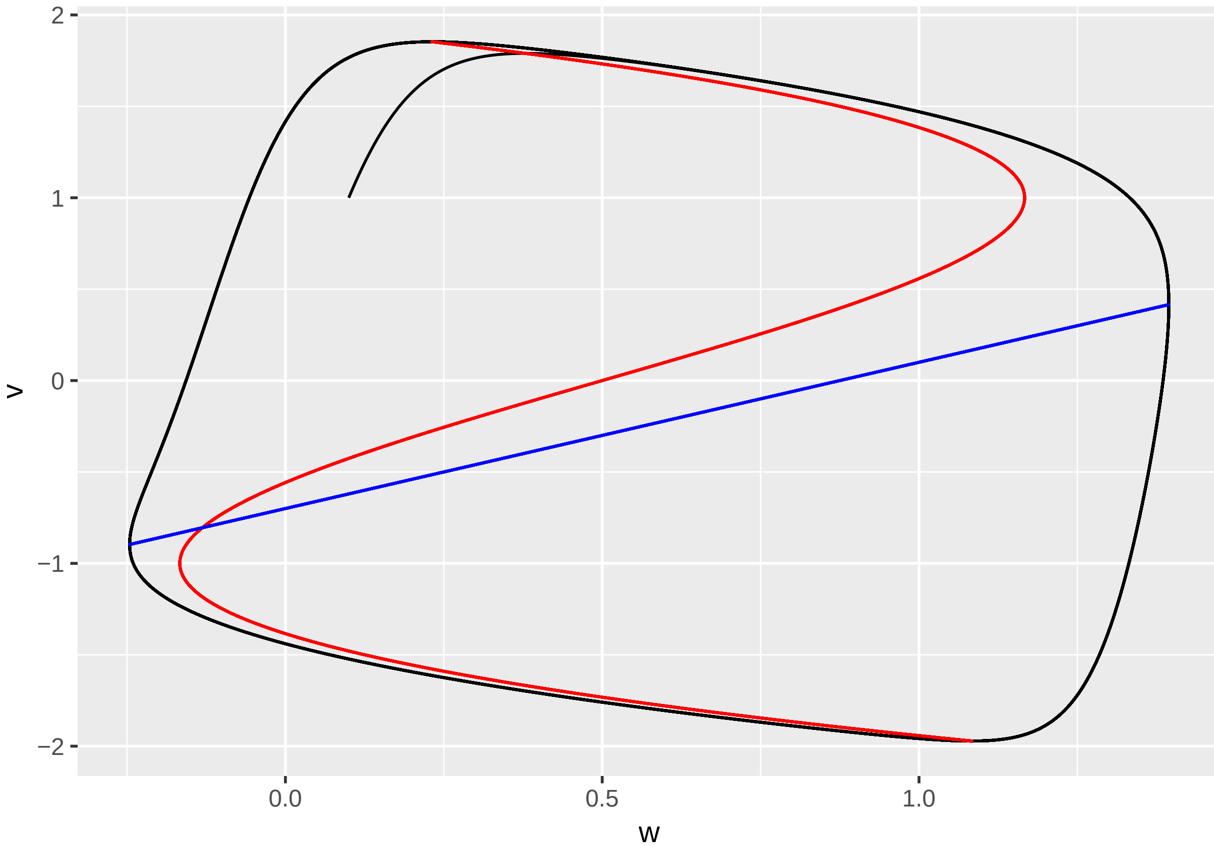
```
{return(v-(v^3)/3-w+0.5)},g = function(v,w){return(0.08*(v+0.7-0.8*w))},
v_0 = 1,w_0 = 0.1,t_max = 500,h = 0.01)
```

We can then plot v vs. t to see that the system oscillates:



Part 2

The v-nullcline is given by the equation $v - \frac{v^3}{3} - w + 0.5 = 0$, or $w = v - \frac{v^3}{3} + I$, and the w nullcline is given by the equation $\epsilon(v + a - bw) = 0$, or $v = bw - a = 0.8w - 0.7$. When we overlay these curves with the phase space plot we get:



Part 3

We can now find the fixed point to this system by substituting the solution for the w-nullcline into the v-nullcline, i.e. solving $0.8w - 0.7 - (0.8w - 0.7)^3/3 - w + 0.5 = 0$. We can do this in R by the command

```
roots = polyroot(z = c(- 0.585667 + 0.5,- 0.592,0.448,-0.170667))
```

which finds one real root at $w* = -0.131$ with corresponding $v* = -0.8048$.

Part 4

The Jacobian of this system is given by the matrix

$$\begin{pmatrix} 1 - v^2 & -1 \\ 0.08 & -0.064 \end{pmatrix}$$

Therefore, at the fixed point of the system we get a Jacobian of

$$\begin{pmatrix} 0.352297 & -1 \\ 0.08 & -0.064 \end{pmatrix}$$

Calculating the eigenvalues using the command eigen, we obtain the values

```
## [1] 0.1441485+0.1915051i 0.1441485-0.1915051i
```

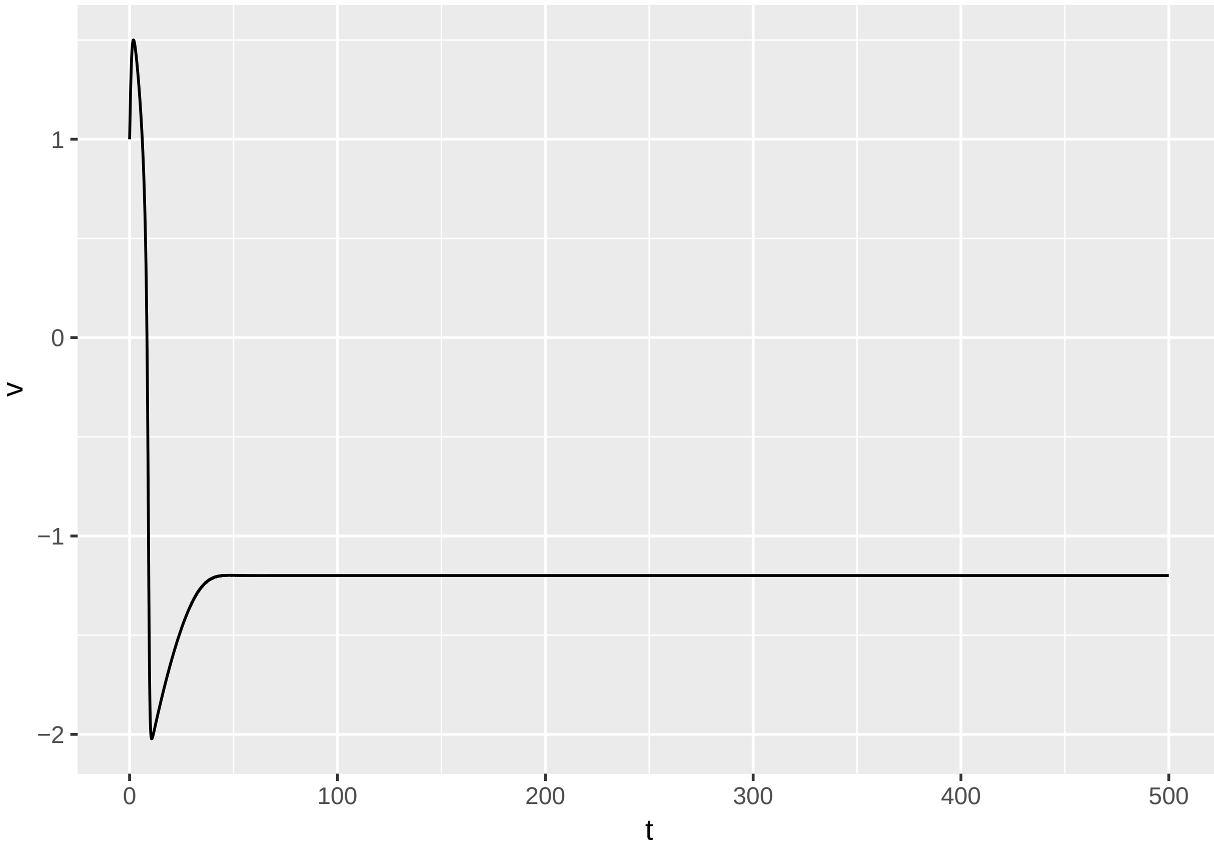
which are two complex numbers with positive real parts. This is not surprising since the trajectory in the phase space spirals (i.e. unstable).

Part 5

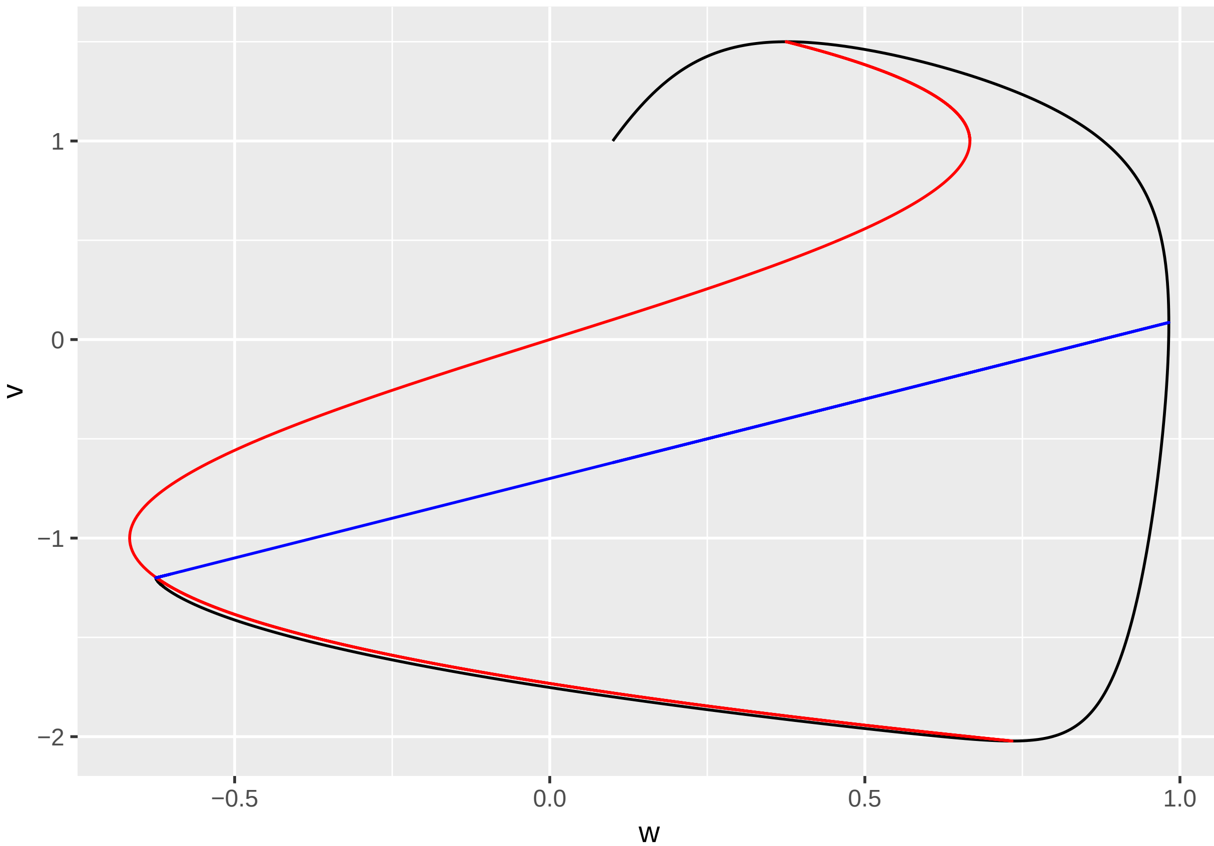
We can now repeat the above calculations with $I = 0$:

```
FN2 <- integrate_2D(f = function(v,w)
  {return(v-(v^3)/3-w)},g = function(v,w)
    {return(0.08*(v+0.7-0.8*w))},
  v_0 = 1,w_0 = 0.1,t_max = 500,h = 0.01)
```

Plot of v vs t:



The phase plot with nullclines shows no oscillations:



The solution of the fixed point in this case is now $w* = -0.624$ and $v* = -1.1992$. Solving for the eigenvalues of the corresponding Jacobian we get

```
## [1] -0.2510403+0.2121696i -0.2510403-0.2121696i
```

which are both complex numbers with negative real parts - as expected since the system doesn't spiral (the solution is stable).

Part 6

We can now loop through values of I between 0 and 0.5 (I did this in parallel):

```
cl = makeCluster(detectCores() - 1)
registerDoParallel(cl)

output <- foreach(i = 0:1000,.combine = rbind) %dopar%
{
  library(data.table)
  I = 0.5*i/1000
  roots_temp = polyroot(z = c(- 0.585667 + I,- 0.592,0.448,-0.170667))
  real_roots <- Re(roots_temp[[which(round(Im(roots_temp),10)==0)]])
  fixed_points <- data.table(w = real_roots,v = 0.8*real_roots - 0.7)
  fixed_points[,stable:=0]
  for(j in 1:nrow(fixed_points))
  {
    J = matrix(c(1-(fixed_points[j,v])^2,0.08,-1,-0.064),nrow = 2,ncol = 2)
    eigens <- eigen(x = J,symmetric = F)$values
    if(Re(eigens[[1]]) < 0 & Re(eigens[[2]]) < 0)
```

```
    {
      set(fixed_points,i = j,j = 3L,value = 1)
    }
  }
  return(fixed_points)
}

stopImplicitCluster()
```

If we now plot the results, we get the intended plot where stable steady-states are light blue and unstable steady-states are dark blue: