Numerical Simulation

Bowei Xiao

1. Running the simulation as suggested. See Figure 1 as below. With the different choice of the initial growth rate, the population growth rate can either pleateau and stay or die down or flucturate. This also shows how sensitive/chaotic the system is to the growth rate.

```
set.seed(1020)
x0=0.1; sim1=NULL;
r1=0.5; r2=2.8; r3=3.3
sim1=data.frame(t=0,r1=x0,r2=x0,r3=x0,stringsAsFactors = F)
for (i in 1:50){
  sim1=rbind(sim1,data.frame(t=i,r1=r1*sim1$r1[i]*(1-sim1$r1[i]),
                             r2=r2*sim1$r2[i]*(1-sim1$r2[i]),
                             r3=r3*sim1$r3[i]*(1-sim1$r3[i]),stringsAsFactors = F))
}

plot(r1~t,data=sim1,type='l',col='red',ylim=c(0,1),xlab='generations',ylab='Xn')
lines(sim1$t,sim1$r2,col='blue')
lines(sim1$t,sim1$r3,col='black')
legend('topleft',legend=c('r=0.5','r=2.8','r=3.3'),
        col=c('red','blue','black'),lty=1,cex=0.8)
```

2. Coding for Euler method here:

```
euler_for=function(x0,t,h){
  step=data.frame(x=x0,t=0); tmax=as.integer(t/h+0.5)
  for (i in 1:tmax){
    step=rbind(step,data.frame(x=step$x[i]+h*step$x[i],t=h*i))
  }
  return(step)
}
sol1=euler_for(x0=1,t=5,h=0.1);
sol2=euler_for(x0=1,t=5,h=0.01);
sol3=euler_for(x0=1,t=5,h=0.001);

plot(seq(0,5,0.001),exp(seq(0,5,0.001)),type='l',col='red',lwd=2)
lines(sol1$t,sol1$x,col='blue',lty=1)
```

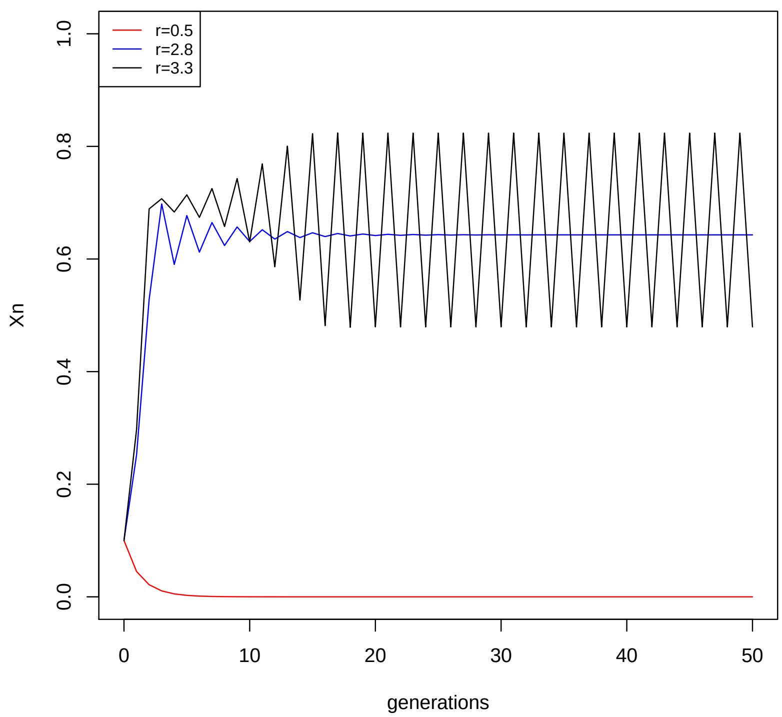

Figure 1: population growth with different r

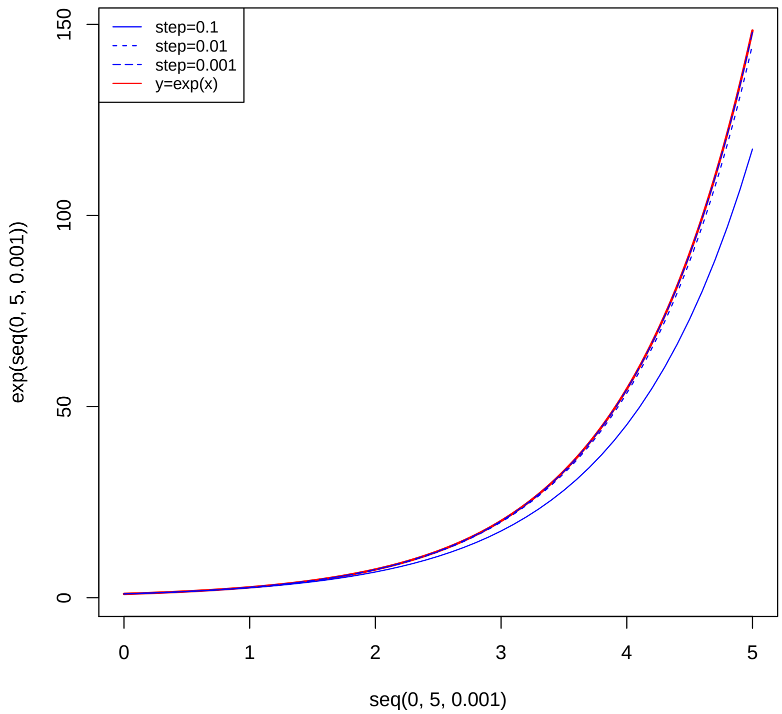

Figure 2: solution wrt h

```
lines(sol2$t,sol2$x,col='blue',lty=2)
lines(sol3$t,sol3$x,col='blue',lty=5)
legend('topleft',legend=c('step=0.1','step=0.01','step=0.001','y=exp(x)'),
        col=c(rep('blue',3),'red'),lty=c(1,2,5,1),cex=0.8)
```

All solutions calculated using such algorithm is close to $y = exp(x)$. Of course, the smaller the step is, the closer the numerical values approaches to the real analytical solutioon.

3. FitzHugh-Nagumo model:

```
fhn=function(t,v0,w0,I,h=0.01){
  tmax=as.integer(t/h+0.5); a=0.7;b=0.8;eps=0.08;
  step=data.frame(v=v0,w=w0,t=0); tmax=as.integer(t/h+0.5)
  for (i in 1:tmax){
```

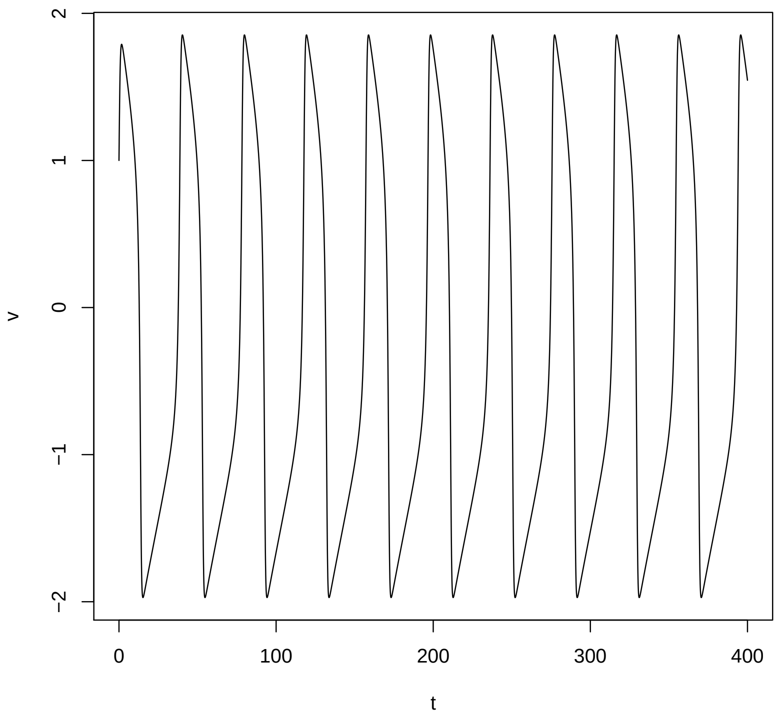

Figure 3: v-t plot

```
    vi=step$v[i]; wi=step$w[i];
    step=rbind(step,data.frame(v=vi+h*(vi-vi^3/3-wi+I),
                              w=wi+h*(eps*(vi+a-b*wi)),t=h*i))
  }
  return(step)
}
sol1=fhn(t=400,v0=1,w0=0.1,I=0.5)
plot(v~t,sol1,type='l')
```

From Figure 3, clearly there is osciallations.
Now look at v vs w with the v-nullcline (in blue) and w-nullcline (in red); See Figure 4

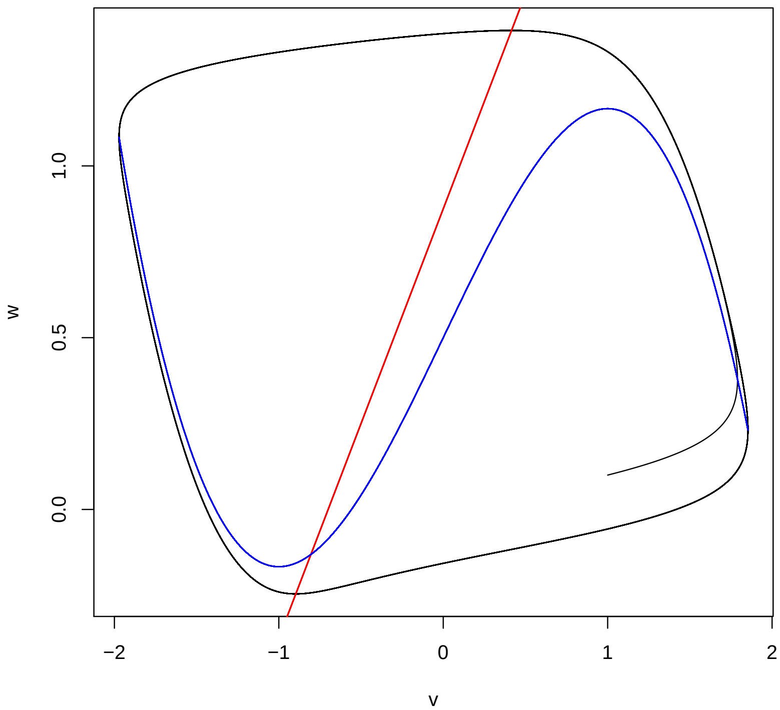

Figure 4: w-v plot

```
plot(w~v,sol1,type='l')
#nullcline
#w-line: v+a-bw=0 -> w=1/b*V+a/b
#v-line v-v^3/3-w+I=0 -> w=-1/3v^3+V+I
lines(sol1$v,(sol1$v+0.7)/0.8,col='red')
lines(sol1$v,-1/3*sol1$v**3+sol1$v+0.5,col='blue')
```

We clearly see the loop pattern.
To calculate fix point. We solved for

$$\begin{aligned} v - v^3/3 - w + I &= 0 \\ \epsilon(v + a - bw) &= 0 \end{aligned}$$

This solves for $v*$, and we can calculate the Jacobians as :
$\begin{pmatrix} 1-\mathrm{v}^2 & -1 \\ \epsilon & \text{-b*}\epsilon \end{pmatrix}$ Thus, a eigendecomposition at fixed point $v*$ will look

at this:

```
a=0.7;b=0.8; I=0.5;eps=0.08;
#-(v+a)/b+I+v-1/3v^3=0 -> (-a/b+I)+(1-1/b)v-1/3v^3=0
vp=polyroot(c(-a/b+I,1-1/b,0,-1/3));
vp

## [1]  0.4024239+1.111681i -0.8048477-0.000000i  0.4024239-1.111681i

# take the real-value one as v*
eigen(matrix(c(1-vp[2]^2,-1,eps,-b*eps),2,2))

## eigen() decomposition
## $values
## [1] 0.1441101-0.1915469i 0.1441101+0.1915469i
##
## $vectors
##                       [,1]                  [,2]
## [1,] -0.2002540+0.1843161i -0.2002540-0.1843161i
## [2,]  0.9622504+0.0000000i  0.9622504+0.0000000i
```

We observed positive real parts for both of the eigenvalues. This is expected because we see from the phase-plane plot that the direction is not stable.
Now, calculate again with $I = 0$. We can see the negative eigenvalues, indicating a stable focues.

```
a=0.7;b=0.8; I=0;eps=0.08;
#-(v+a)/b+I+v-1/3v^3=0 -> (-a/b+I)+(1-1/b)v-1/3v^3=0
vp=polyroot(c(-a/b+I,1-1/b,0,-1/3));
vp

## [1]  0.599704+1.352381i -1.199408+0.000000i  0.599704-1.352381i

# take the real-value one as v*
eigen(matrix(c(1-vp[2]^2,-1,eps,-b*eps),2,2))

## eigen() decomposition
## $values
## [1] -0.2512898+0.2119493i -0.2512898-0.2119493i
##
## $vectors
##                      [,1]                 [,2]
```

```
## [1,] 0.1802197-0.2039484i 0.1802197+0.2039484i
## [2,] 0.9622504+0.0000000i 0.9622504+0.0000000i
```

Now, ranging I from 0 to 0.5, and recreate the bifurcation diagram as in Figure 5.

```
h=0.01; I=0.5; tmax=as.integer(0.5/h+0.5)
a=0.7;b=0.8;eps=0.08;
#when I=0, iot is stable
sim3=data.frame(I=0,v=vp[2],stable=1)
for (i in 1:tmax){
  I=i*h
  vp=polyroot(c(-a/b+I,1-1/b,0,-1/3))
  #find the real-valued v*; tolerance 1e-10
  vn=vp[which(abs(Im(vp))<1e-10)]
  evals=eigen(matrix(c(1-vn^2,-1,eps,-b*eps),2,2))$values
  # if real part of eigevalue is positive -> unstable(0); negative stable(1)
  sim3=rbind(sim3,data.frame(I=I,v=vn,
                 stable=as.numeric(sum(Re(evals)<0)==2)))
}
#create bifurcation diagram
plot(sim3$I,sim3$v,col=c('red','blue')[sim3$stable+1])

## Warning in xy.coords(x, y, xlabel, ylabel, log):  imaginary
parts discarded in coercion
```

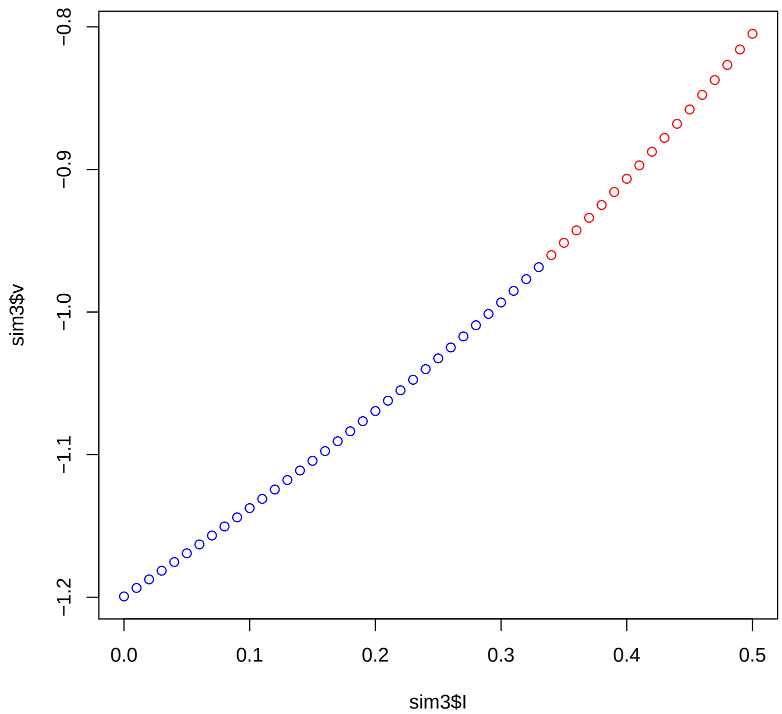

Figure 5: w-v plot