Jacobian and Leslie Simulations

Mikaela Provost

August 6, 2019

I. Load functions

A) sim_model: simulates Leslie and Jacaobian matrices without density dependence

```
sim_model <- function(A,timesteps,initial_eggs,maxage) {
  age_at_mat <- min(which(!A[1,] ==0)) # set inequality to min threshold for maturity, for cod it is al
  ages = length(seq(1,maxage,by=1))
  NO = c(initial_eggs, rep(0,ages-1)) #vector, length=num of ages, first age is initial eggs
  set.seed(2) #Set the seed so that every simulation uses same random sequence

  Nt = matrix(0,ages,timesteps) #empty matrix to store output: rows=ages, cols=time

  Nt[,1] = NO #Put in initial values
  Nt[,2] = A %*% Nt[,1]  # multiply initial age vector with Jacobian to get 2nd age vector
  eggs = c(initial_eggs, rep(NA,timesteps-2)) #will save egg production here, this will be the input in
  eggs[2] = Nt[1,2]

  for(t in 1:(timesteps-2)){ #step through time
    #Save egg production, this is new number of age 1 individuals
    eggs[t+1] = Nt[1,t+1]

    #perform population projection for one time step
    Nt[,t+2] = A %*% Nt[,t+1]
  }
  Nsize = colSums(Nt[age_at_mat:maxage,]) #sum rows that correspond to spawning adults
  eggs = eggs[1:(timesteps-1)] #egg production
  return(list(eggs=eggs, Nsize=Nsize))
}
```

B) sim_model_dd: simulates Leslie and Jacaobian matrices with density dependence and noise

```
sim_model_dd <- function(A,timesteps,alpha,beta,sig_r,initial_eggs) {
  age_at_mat <- min(which(!A[1,] ==0)) # set inequality to min threshold for maturity, for cod it is al

  ages = length(seq(1,maxage,by=1))
  NO = c(initial_eggs, rep(0,ages-1)) #vector, length=num of ages, first age is initial eggs
  set.seed(2) #Set the seed so that every simulation uses same random sequence

  Nt = matrix(0,ages,timesteps) #empty matrix to store output: rows=ages, cols=time

  Nt[,1] = NO #Put in initial values
```

```
  Nt[,2] = A %*% Nt[,1]  # multiply initial age vector with Jacobian to get 2nd age vector
  eggs = c(initial_eggs, rep(NA,timesteps-2)) #will save egg production here, this will be the input in
  eggs[2] = Nt[1,2]
  recruits1 = eggs[1]/( (1/alpha) + (eggs[1]/beta) )
  recruits = c(0,rep(NA, timesteps-2)) #will save recruits here (output from BH)
  recruits[2] = recruits1 #we are ignoring recorinding recruits at time 0

  for(t in 1:(timesteps-2)){ #step through time
    #Save egg production, this is new number of age 1 individuals
    eggs[t+1] = Nt[1,t+1]
    #save recruits, treat egg production as spawning biomass in BH
    recruits[t+1] = eggs[t]/( (1/alpha) + (eggs[t]/beta) ) #((alpha*eggs[t+1])/(1+beta*eggs[t+1]))
    #replace age 1 in numbers-at-age matrix with recruits from BH, add noise
    Nt[1,t+1] = (recruits[t+1])*exp(sig_r*rnorm(1,mean=0,sd=1))
    #perform population projection for one time step
    Nt[,t+2] = A %*% Nt[,t+1]
  }
  Nsize = colSums(Nt[age_at_mat:maxage,]) #sum rows that correspond to spawning adults
  eggs = eggs[1:(timesteps-1)] #egg production
  recruits = recruits[1:(timesteps-1)] #recruits produced via BH, before influence of environmental noi
  return(list(eggs=eggs, recruits=recruits, Nsize=Nsize))
}
```

C) load the function: extract_first_eigen_value()

```
source("C:/Users/provo/Documents/GitHub/popdy/cod_code/2_cod_functions.r")
```

II. Generate Leslie matrix

I use North Sea cod population as a test case. First I define all the North Sea parameters for assembling the Leslie matrix.

```
#source(file = paste('C:/Users/provo/Documents/GitHub/popdy/cod_pops/','Northsea', '.r', sep=''))
# North Sea example
maxage = 10
K = 0.217
L_inf = 126
F.halfmax=0
B0 = -6.42
B1 = 1.72
tknot=0
MG = 0.427
```

I generate a Leslie matrix (A) and life table (NEAR) using the function assemble_Leslie().

```
Leslieout = assemble_Leslie(maxage=maxage, K=K, L_inf=L_inf,F.halfmax=0, B0=B0, B1=B1, tknot=0)
NEAR = Leslieout$NEAR #Life table
LEP = sum(NEAR$egg_production) #sum egg production (not adjusted)
A = Leslieout$A #Leslie matrix
A
```

```
##              [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
```

```
##  [1,] 0.001338342 0.04220362 0.4842093 2.3951743 5.2163595 7.5652866
##  [2,] 0.652463552 0.00000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [3,] 0.000000000 0.65246355 0.0000000 0.0000000 0.0000000 0.0000000
##  [4,] 0.000000000 0.00000000 0.6524636 0.0000000 0.0000000 0.0000000
##  [5,] 0.000000000 0.00000000 0.0000000 0.6524636 0.0000000 0.0000000
##  [6,] 0.000000000 0.00000000 0.0000000 0.0000000 0.6524636 0.0000000
##  [7,] 0.000000000 0.00000000 0.0000000 0.0000000 0.0000000 0.6524636
##  [8,] 0.000000000 0.00000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [9,] 0.000000000 0.00000000 0.0000000 0.0000000 0.0000000 0.0000000
## [10,] 0.000000000 0.00000000 0.0000000 0.0000000 0.0000000 0.0000000
##            [,7]       [,8]       [,9]    [,10]
##  [1,] 9.4974936 11.1752547 12.6402013 13.90409
##  [2,] 0.0000000  0.0000000  0.0000000  0.00000
##  [3,] 0.0000000  0.0000000  0.0000000  0.00000
##  [4,] 0.0000000  0.0000000  0.0000000  0.00000
##  [5,] 0.0000000  0.0000000  0.0000000  0.00000
##  [6,] 0.0000000  0.0000000  0.0000000  0.00000
##  [7,] 0.0000000  0.0000000  0.0000000  0.00000
##  [8,] 0.6524636  0.0000000  0.0000000  0.00000
##  [9,] 0.0000000  0.6524636  0.0000000  0.00000
## [10,] 0.0000000  0.0000000  0.6524636  0.00000
```

III. Simulate Leslie and Jacobian without density dependence

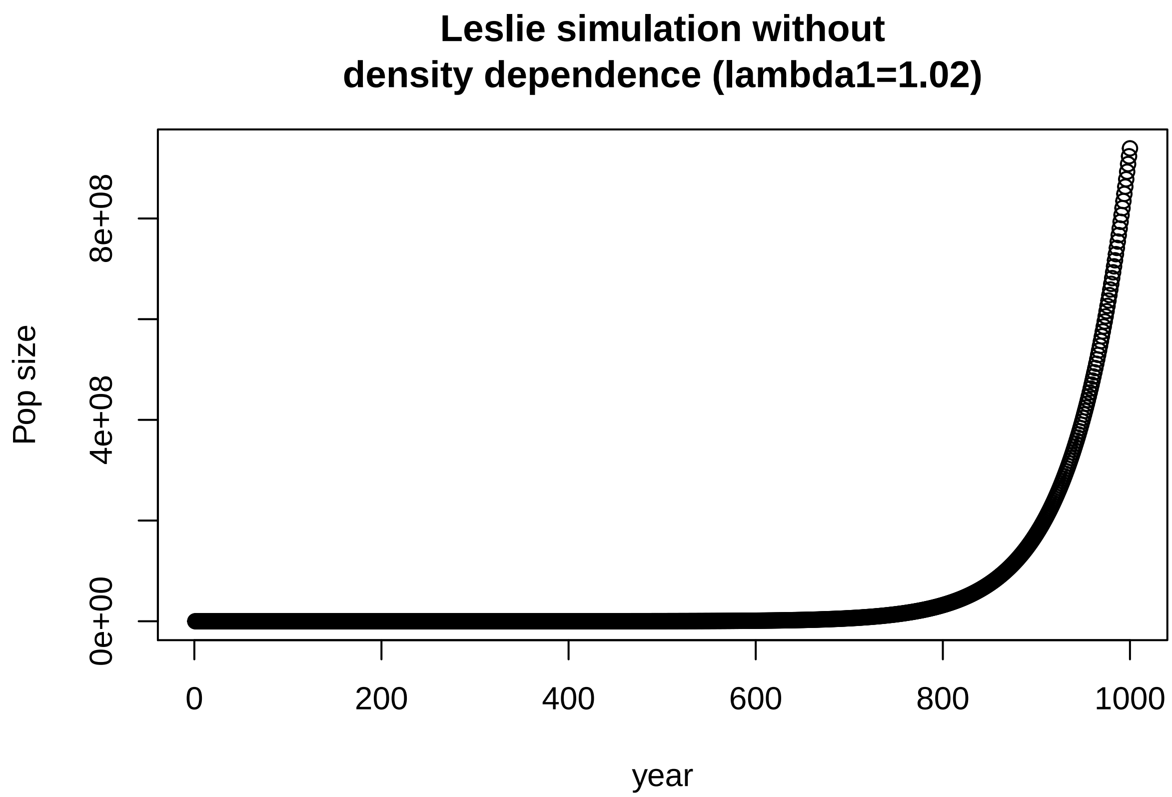
A) Leslie simulation without density dependence

Before we simulate the Leslie matrix, we first have to standardize LEP so that LEP is constant among populations. Here I set LEP=1.1. To do this we adjust fecundities, fecundity-at-age should be: f-at-age/(LEP*0.9)

```r
conLEP = 1.1 #constant LEP
adjFec = round(1/conLEP,digits=1) #1/LEP is 1.1, which means alpha must be greater than this
A[1,] <- A[1,]/(LEP*adjFec) #adjusted Leslie matrix
```

Since the leading eigenvalue of the fecundity-adjusted-Leslie matrix is 1.0170278, we expect the population to exponentially grow because lambda1 > 1. The plot below shows population size increases exponentially.

```r
Leslie_output_withoutDD_lambda_small <- sim_model(A=A,timesteps=1000,initial_eggs=100,maxage=10)
plot(Leslie_output_withoutDD_lambda_small$Nsize,xlab="year",ylab="Pop size",main="Leslie simulation wit
```



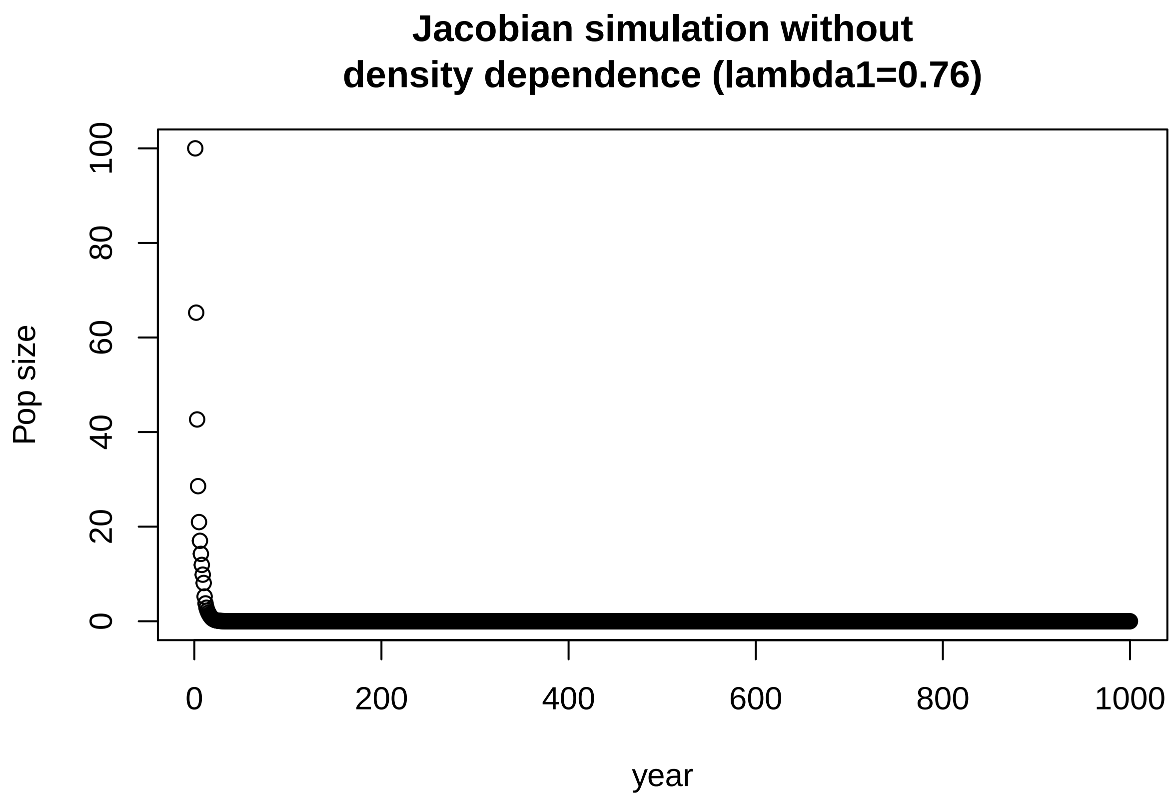
B) Jacobian simulation without density dependence (alpha=5.51)

Second, convert Leslie to Jacobian: multiply top row of Leslie by 1/(alpha x LEP^2). The slope at equilibrium is 1/(alpha x LEP^2) (see ch 4 in Loo's book or my appendix for derivation). Since we adjusted LEP to 1.1, we multiply to top row by: 1/(alpha*1.1^2) (we call this value 'k'). If alpha is 5.51 then k is 0.15.

```
alpha = 5.51 #with this value of alpha, slope at equilibrium is 0.15
slope_at_equilibrium = 1/(alpha*conLEP^2) #we call the slope 'k'
A[1,] <- A[1,]*slope_at_equilibrium #convert to Jacobian
```

Lambda1 of the Jacobian is 0.7670967. Since lambda1 <1 we expect a simulation of the Jacobian to converge to zero. The plot below shows the population is perturbed initially, but quickly goes back to zero.

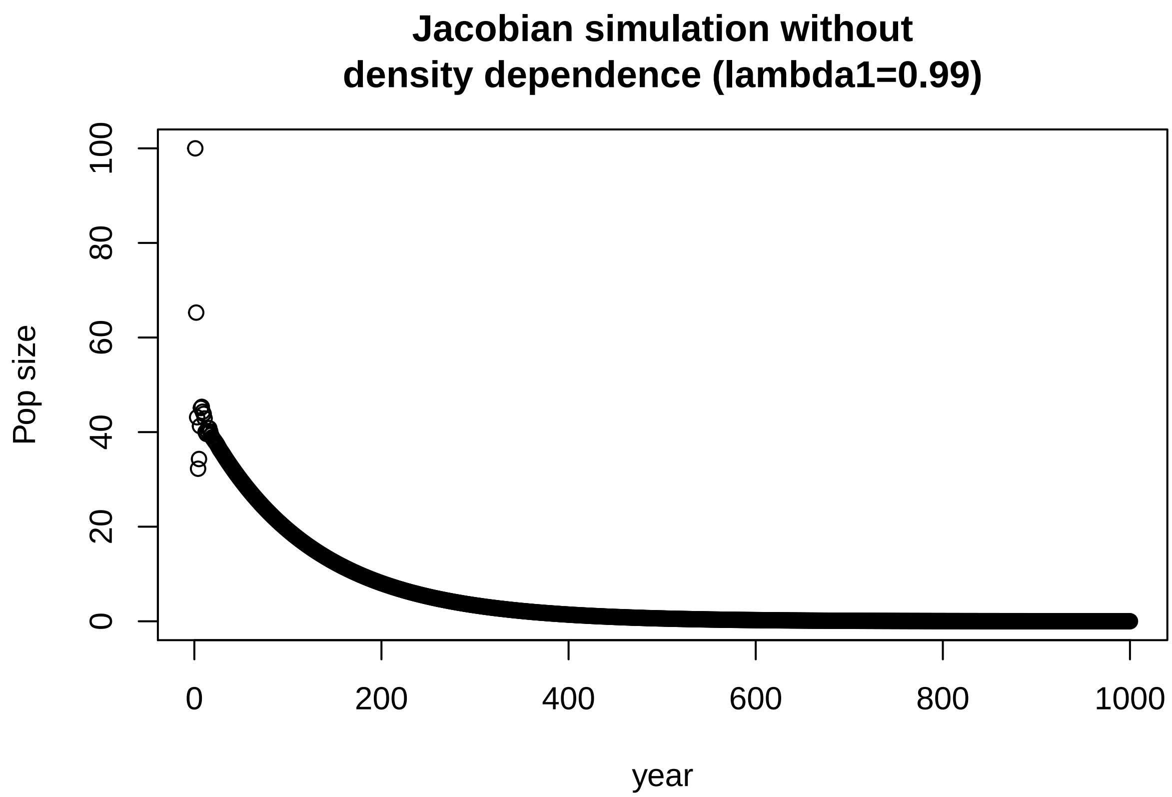
```
Jacobian_output_withoutDD_lambda_small <- sim_model(A=A,timesteps=1000, initial_eggs=100,maxage=10)
plot(Jacobian_output_withoutDD_lambda_small$Nsize,xlab="year",ylab="Pop size",main="Jacobian simulation
```



C) Jacobian simulation without density dependence (alpha=0.97)

If alpha is 0.97 then k is 0.8520065. With this new Jacobian, the leading eigenvalue is 0.7670967 which means we expect the population to approach zero but at a slower rate than before because lambda1 is closer to 1.

```
alpha0.97 = 0.97
A = Leslieout$A #reset Leslie matrix to original form without adjustments
A[1,] <- A[1,]/(LEP*adjFec) #adjust LEP in Leslie matrix to make LEP=1.1
slope_at_equilibrium = 1/(alpha0.97*conLEP^2) #we call the slope at equilibrium 'k'
A[1,] <- A[1,]*slope_at_equilibrium #convert Leslie to the Jacobian by multiplying top row 1/(alpha*LEP
# Simulate Jacobian:
Jacobian_output_withoutDD_lambda_big <- sim_model(A=A,timesteps=1000, initial_eggs=100,maxage=10)
plot(Jacobian_output_withoutDD_lambda_big$Nsize,xlab="year",ylab="Pop size",main="Jacobian simulation wi
```



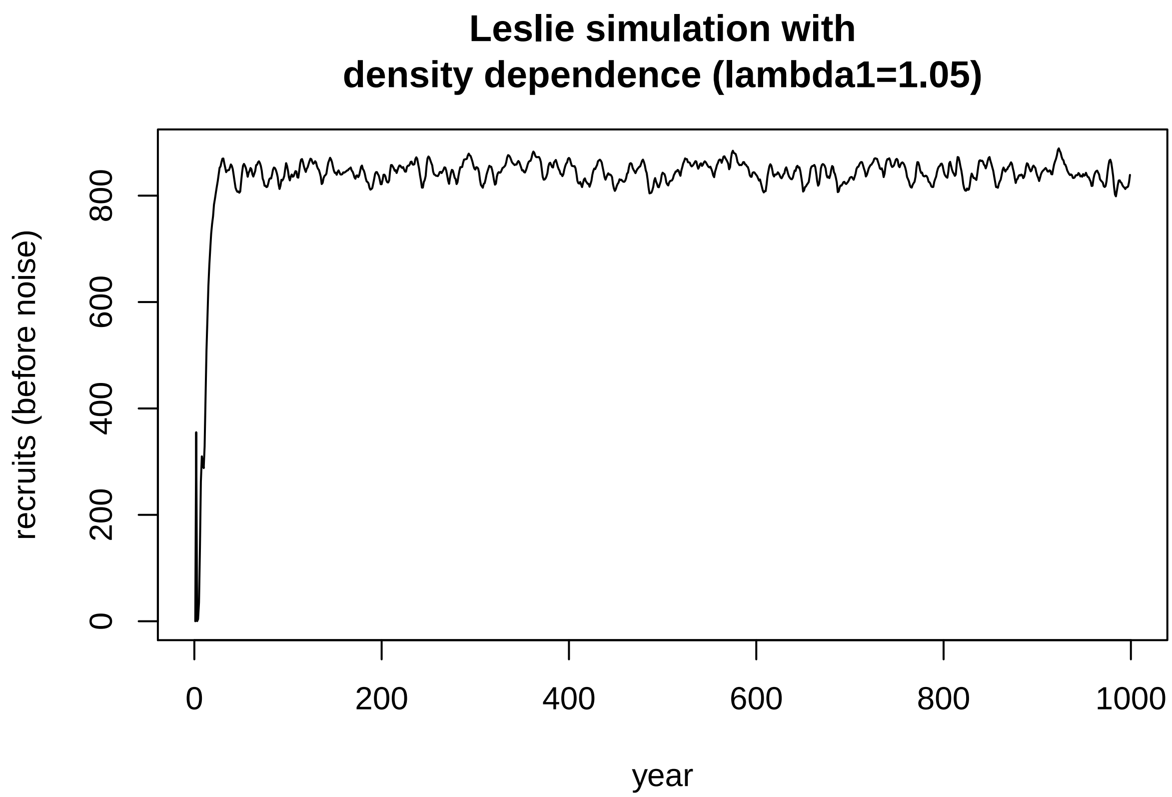
IV. Simulate Leslie and Jacobian with density dependence and noise

A) Simulate Leslie with density dependence (alpha = 5.51):

```
# First, set up the fecundity-adjusted Leslie matrix:
A = Leslieout$A #reset Leslie matrix to original form without adjustments
A[1,] <- A[1,]/(LEP*adjFec) #adjust LEP in Leslie matrix to make LEP=1.1
```

Lambda1 of the Leslie matrix is 1.0170278. A plot of annual recruitment before noise is below.

```
alpha5.51 <- 5.51 # define alpha for BH in DD
Leslie_output_withDD_lambda_small <- sim_model_dd(A=A,timesteps=1000,alpha=5.51,beta=1000,sig_r=0.3,ini
plot(Leslie_output_withDD_lambda_small$recruits,xlab="year",ylab="recruits (before noise)",main="Leslie
```

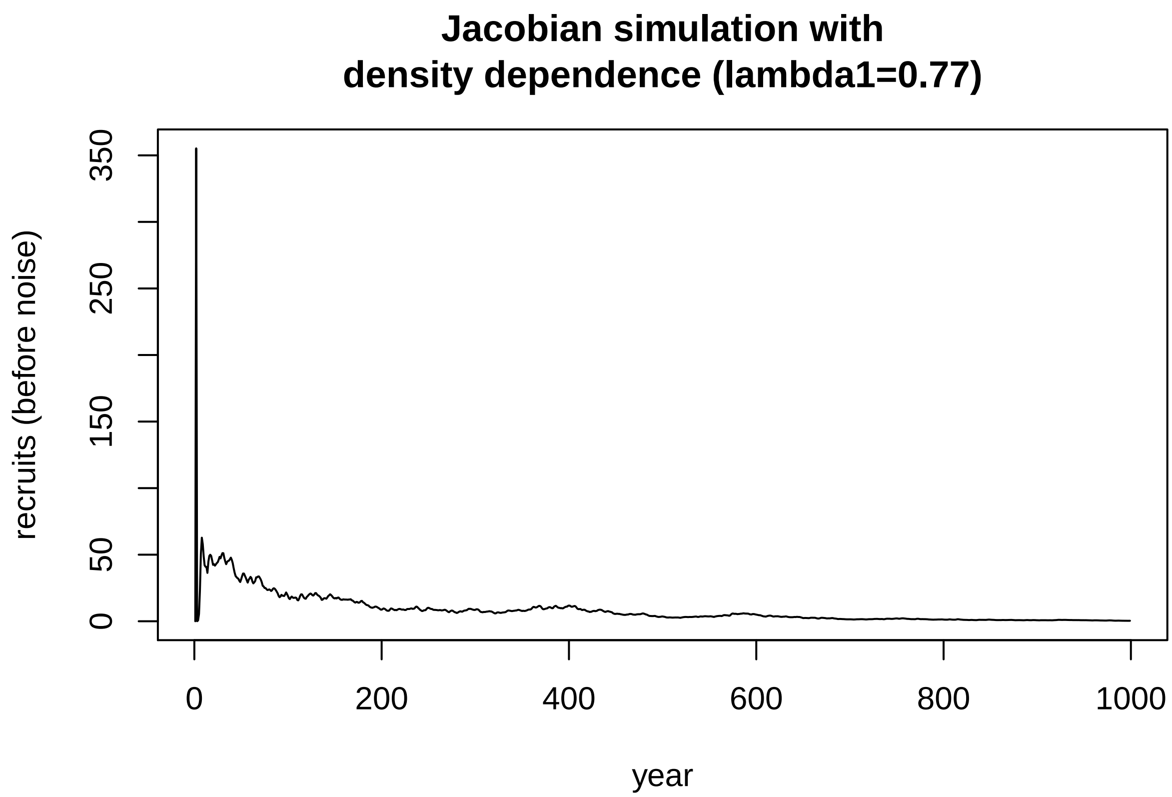


B) Simulate Jacobian with density dependence (alpha = 5.51):

```
# Second, set up the Jacobian:
slope_at_equilibrium = 1/(alpha5.51*conLEP^2) #we call the slope at equilibrium 'k'
A[1,] <- A[1,]*slope_at_equilibrium #convert Leslie to the Jacobian by multiplying top row 1/(alpha*LEP
# Let's check the leading eigenvalue of this Jacobian:
```

Lambda1 of the Jacobian is 0.7670967. Below is a plot showing a simulation of annual recruits (before noise) using the Jacobian maxtric.

```
Jacobian_output_withDD_lambda1_small <- sim_model_dd(A=A,timesteps=1000,alpha=5.51,beta=1000,sig_r=0.3,
plot(Jacobian_output_withDD_lambda1_small$recruits,xlab="year",ylab="recruits (before noise)",main="Jac
```



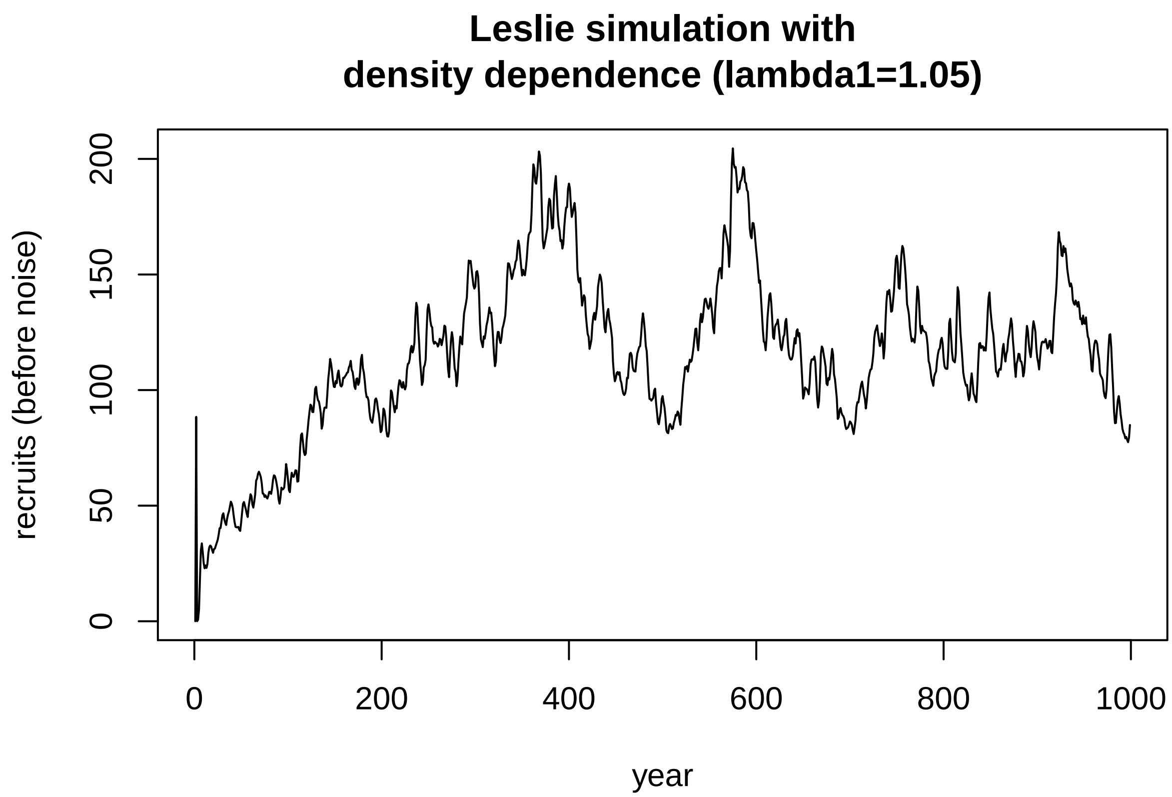
C) Simulate Leslie with density dependence (alpha = 0.97)

Set up the Leslie matrix:

```
alpha0.97 <- 0.97
A = Leslieout$A #reset Leslie matrix to original form without adjustments
A[1,] <- A[1,]/(LEP*adjFec) #adjust LEP in Leslie matrix to make LEP=1.1
```

Lambda1 of the Leslie is 1.0170278. Below is a plot showing a simulation of annual recruits (before noise) using the Leslie matrix.

```
# Simulate adjusted Leslie with density dependence:
Leslie_output_withDD_lambda_big <- sim_model_dd(A=A,timesteps=1000,alpha=0.97,beta=1000,sig_r=0.3,initia
plot(Leslie_output_withDD_lambda_big$recruits,xlab="year",ylab="recruits (before noise)",main="Leslie si
```

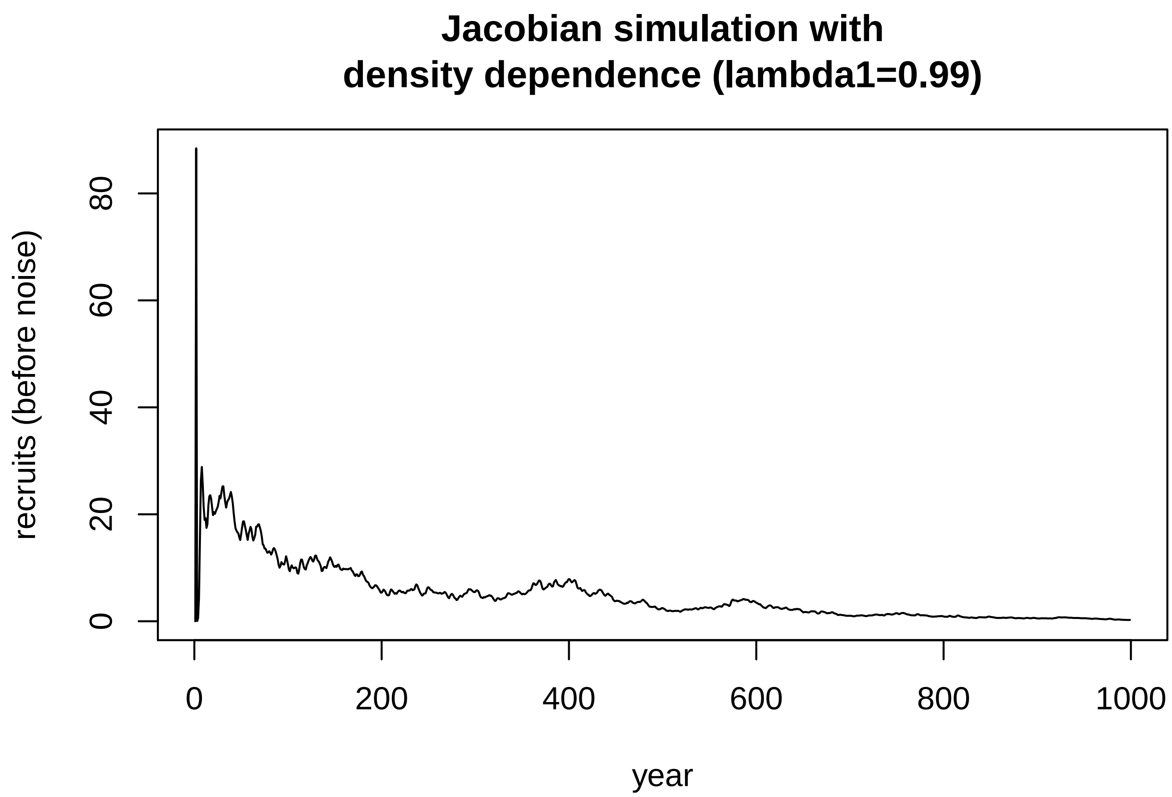


D) Simulate Jacobian with density dependence (alpha = 0.97)

```
# Next, set up the Jacobian:
slope_at_equilibrium = 1/(alpha0.97*conLEP^2) #we call the slope at equilibrium 'k'
A[1,] <- A[1,]*slope_at_equilibrium #convert Leslie to the Jacobian
```

Lambda1 of the Jacobian is 0.9913201. Below is a plot showing a simulation of annual recruits (before noise) using the Jacobian matrix.

```
# Simulate model w/dd:
Jacobian_output_withDD_lambda1_big <- sim_model_dd(A=A,timesteps=1000,alpha=0.97,beta=1000,sig_r=0.3,ini
plot(Jacobian_output_withDD_lambda1_big$recruits,xlab="year",ylab="recruits (before noise)",main="Jacobi
```



Jacobian simulation with
density dependence (lambda1=0.99)
recruits (before noise)
0
20
40
60
80
0
200
400
600
800
1000
year