

Exponential population growth

Matthew Malishev^{1*}

¹ *Department of Biology, Emory University, 1510 Clifton Road NE, Atlanta, GA, USA, 30322*

Contents

| | |
|---|----------|
| Overview | 3 |
| Install dependencies | 3 |
| Section 1 | 3 |
| Exponential growth equation | 3 |
| Parameters | 3 |
| Example | 4 |
| Instantaneous rate of growth | 4 |
| Example | 4 |
| Simulating population growth | 4 |
| Section 2 | 5 |
| Simulation loop | 5 |
| Event-based conditions | 6 |
| Making reusable R code: functions. | 7 |
| Simulating the continuous population growth model | 8 |

Date: 2018-11-07

R version: 3.5.0

*Corresponding author: matthew.malishev@gmail.com

This document can be found at <https://github.com/darwinanddavis>

R session info

```
params$session
```

R version 3.5.0 (2018-04-23)

Platform: x86_64-apple-darwin15.6.0 (64-bit)

Running under: OS X El Capitan 10.11.6

Matrix products: default

BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib

LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib

locale:

[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:

[1] stats graphics grDevices utils datasets methods base

loaded via a namespace (and not attached):

[1] compiler_3.5.0 backports_1.1.2 magrittr_1.5 rprojroot_1.3-2 tools_3.5.0 htmltools_0.3.6
[7] pillar_1.2.3 tibble_1.4.2 yaml_2.2.0 Rcpp_0.12.19 stringi_1.2.3 rmarkdown_1.10
[13] knitr_1.20 stringr_1.3.1 digest_0.6.15 rlang_0.2.1 evaluate_0.10.1

Overview

Examples of exponential population growth in R.

Install dependencies

```
packages <- c("dplyr", "deSolve", "pdftools")
if (require(packages)) {
  install.packages(packages, dependencies = T)
  require(packages)
}
lapply(packages, library, character.only=T)
```

Section 1

Exponential growth equation

$$N_t = N_0 \cdot e^{rt}$$

N_t = the number of individuals in the population after t units of time

N_0 = the initial population size ($t = 0$)

r = the exponential growth rate

t = time unit (usually in years)

e = the base of the natural logarithms (2.72)

Exponential rate of growth is commonly named the parameter lambda λ

$$\lambda = e^r$$

e^r = lambda. Exponential growth rate parameter.

The natural log (ln) of $e = 1$

$$\ln(e) = 1$$

because $e^1 = e$.

The natural log of $1 = 0$

$$\ln(1) = 0$$

because $e^0 = 1$.

Parameters

```

# parameters
N_t <- 0 # expected pop size
N_0 <- 500 # initial pop size
e <- exp
r <- 0.012 # exponential rate of growth
lambda <- e(1^r)
t <- 10 # time (in years)

# putting the above all together in R
N_t <- N_0 * e(r*t)
N_t

```

Example

A moose population has a growth rate of 0.02. In 2000, the population was 500. What will the population be in 2020?

```

# input your R code here

```

Instantaneous rate of growth

The above growth model equation is discrete growth: population growth happens on discrete time steps. To fill in the gaps between these time steps, we use a continuous growth model. This follows an ordinary differential equation (ODE).

$$\frac{dN}{dt} = rN$$

dN = change in number

dt = change in time

r = the per head maximum potential growth rate

N = number of individuals in a population

```

# in R
N <- 1000
dNdt <- r*N
dNdt

```

Example

A population of 100 individuals. Each individual can on average contribute 1/4 of an individual (new individual) to the population in a given unit of time. Find the rate of population increase.

```

# your r code

```

Simulating population growth

Set your parameters for the population

```

N_0 <- 20; # initial population size

```

Over time

```
N_1 <- N_0 * r; # population size at t = 1
```

What does this look like at each time point?

```
N_2 <- # ??  
N_3 <- # ??  
# etc
```

Population size

```
popsiz = c(N_0, N_1, N_2, N_3, N_4, N_5)  
popsiz  
popsiz[2]
```

Section 2

Simulation loop

```
r <- 2  
N_0 <- 20  
t <- 10  
popsiz <- list() # create empty list to populate in loop  
tt <- 2:t # now make t a vector to make a time period  
  
for(i in tt){ # start loop  
  N_0[i] <- N_0[i-1]*r # middle of loop  
  popsiz <- N_0 # put the pop size into the list we created  
} # end loop  
popsiz
```

Plot

```
# plot it  
plot(popsiz)  
  
# extend plot params  
tta <- c(1,tt) # add the first year onto the time vector  
ttm <- max(tt) # get the maximum value of the time vector  
xlim <- c(1,ttm) # put this in a vector  
ylim <- c(0,max(N_0)) # do the same for growth  
par(las=1,bty="n") # set plotting params  
  
plot(tta,popsiz, # vectors to plot  
     xlim=xlim, # set the x limits  
     ylim=ylim, # y limits  
     xlab="Time",  
     ylab="Population growth",  
     pch=20, # set point type  
     col="pink"  
     )  
  
?plot # help page for plot function
```

Event-based conditions

Changing conditions halfway through a simulation. Using `if` and `else` statements.

```
popsiz<= list() # reset the list

for (i in tt){
  if((i%%5) == 0){ #modify growth rate depending on the year
    r <- 4
  }else{ # or just an ok year
    r <- 2
  }
  N_0[i] <- N_0[i-1]*r # the population growth equation
  popsiz<= N_0
}

popsiz

par(las=1,bty="n") # set plotting params
plot(popsiz,pch=20,col="pink")
```

Another for loop example

```
popsiz<= list() # reset the list

for (i in tt){
  if (i <= 6){ # for the first five years, use r = 2
    r <- 2
  }else if(i >= 10){
    # and also at 10 years
    r <- 2
  }else{
    # but between 5th and 9th year,
    # we grow slower
    r <- 0.5
  } # end of if elseif else statement
  N_0[i] <- N_0[i-1]*r # the population growth equation
  popsiz<= N_0
}

popsiz

par(las=1,bty="n") # set plotting params
plot(popsiz,pch=20,col="pink")
```

How do we figure out at what time point a population has reached a certain growth stage? Using `while` loops.

```
r <- 2 # ?? # set r
N_0 <- 20 # initial size
popsiz<= list() # reset the list
popsiz_d <- 5000 # desired pop size

t <- 0 # set time var
while (N_0[t + 1] <= popsiz_d){ # while we're under the desired pop size
  t = t + 1 ; # advance one time step
```

```

N_0[t + 1] <- N_0[t]*r; # population growth eq
time_to_popsized <- NULL # time to reach popsize_d?
popsize <- N_0
} # close loop

```

How can we use R to get the time to reach popsize_d?

```

# get the length of the population size vector and replace the "NULL" in the above loop with this funct
time_to_popsized <- NULL # ??

popsize[time_to_popsized] # this is the size of the pop at the desired time step

```

Making reusable R code: functions.

Putting all the above into reusable code that replaces the population growth parameters with input variables. Using function.

Functions pass user-defined input variables into an R procedure so that you can package code into a reusable and general program, the function.

Take your simulation you want to put into the function.

```

r <- 2 # set r
popsize <- 20 # initial size
popsize_d <- 5000 # desired size
time_to_popsized <- NULL # ?? # use your answer from above

t <- 0 # set time var
while (popsize[t + 1] <= popsize_d){ # while we're under the desired pop size
  t = t + 1 ; # advance one time step
  popsize[t + 1] <- popsize[t]*r; # population growth eq
  time_to_popsized <- NULL # time to reach popsize_d??
} # close loop

# writing the function. we'll call it 'discrete_growth'.
discrete_growth <- function(r){ # make popsize an input variable
  d <- popsize*r
  return(d) # we're just returning the popsize multiplied by growth rate
}

# now we replace the 'r' parameter with a value for 'r'
discrete_growth(5)

```

What would the function look like if we wanted to figure out the following parameters?

```

r <- NULL ### pop growth
N_0 <- NULL ### initial pop size
t <- NULL
popsize <- list() # create empty list to populate in loop

# write the below function
discrete_growth <- function(??){ # <--- there will be three input variables
  popsize <- ??
  tt <- 2:?? # make t a vector to make a time period
  for (i in tt){

```

```

    N_0[i] <- N_0[i-1]*r # middle of loop
    popsize <- N_0 # put the pop size into the list we created
  } # end loop
d <- popsize
plot(d) # plot popsize with the appropriate time scale
return(d) # get d
}

# now just replace the input variable with the values you want to simulate
discrete_growth()

```

Save the function in the R memory. Call it 'popsize_func'.

```
popsize_func <- ??
```

Now save a separate function with different input parameters, then run both functions. This should give you an idea of how R saves things to memory.

```

# use these values for popsize_func_2
0.1
500
10000

popsize_func_2 <- ?? #
popsize_func # is the output for popsize_func look different to popsize_func_2?

```

Simulating the continuous population growth model

We can use R to simulate the ODE of the population growth model, $dN/dt = rN$, using the `lsoda` function for 'solving' ODEs.

```

?? # first load the 'deSolve' package
?? # then get the help page for the function 'lsoda'

```

We need the following elements for the `lsoda` function:

- a vector of parameters, `par`
- initial population density, `N_0`
- time vector, `time`

```

par <- r # use the population growth rate for the parameter soace
N_0 <- 0.5 # initial population density
# create a time sequence using the following values
# start = 0
# end = 10
# every 0.5 (years)
# you can set these as variables, or just leave them static as integers.
time <- seq() # ??

```

Now we need to create the equation that the `lsoda` ODE solver function will simulate using population growth parameters. These will be time, a previous solution of the ODE representing the state of the population, and the growth rate.

First, we need to use a list vector so that the `lsoda` function can understand the input we feed it.

Here is a toy example of vector notation in a list.


```

# lists can store multiple inputs, such as integers, character strings, and functions.
nums <- 1:10
char <- c("A","B","C","D")
l <- list(nums, char)
# to access each list element, we use vector notation.
l[1] # access the first list element
l[2] # second element
l[[2]] # second element indexed using vector notation
class(l[2]) # what class is this?
class(l[[2]]) # compared to this?
#third item of the second element. notice the use of indexing: '[' vs. '[[']
l[[2]][3]

```

Using list indexing and vector notation, we can provide the right input for the `lsoda` function

```

# t, time
# N, previous solution of the ODE. this will be the population growth eq, dN/dt # dN/dt = rN
# r, growth rate

# this function is the dNdt eq, but in function format so that lsoda can read it.
expgrowth <- function(t,N,r){
  # use vector notation to create a list
  N <- r*N[1] # pop growth eq
  list(N) # return the dN/dt solution as a list so that lsoda can read it
}

```

Now put the population growth equation above into the ODE solver using the variables we have defined above.

```

# expgrowth_solve <- lsoda(initial conditions, time, the population growth model, and the parameter spa
expgrowth_solve <- lsoda(?,?,?,expgrowth,??)
?lsoda # check the help page if stuck

expgrowth_solve
class(expgrowth_solve) # check class
str(expgrowth_solve) # check the structure of the object
# now plot the population growth output from the ODE solver function
xx <- expgrowth_solve[,1] # access the first column of the matrix
yy <- expgrowth_solve[,2] # access the second column of the matrix
expgrowth_solve[20,2] # what does this mean?

par(las=1,bty="n")
plot(xx,yy,
     type="l",col="red",
     xlab="Time (years)",
     ylab="Population size",
     main="Continuous population growth over time \n by simulating the population growth model")

```