Module 10: Modeling Population Growth

*Dai Shizuka*

*updated 02/21/17*

* [9.1 Modeling the basic exponential/geometric population growth model](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#modeling-the-basic-exponentialgeometric-population-growth-model)
* [9.2 Varying the growth rate](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#varying-the-growth-rate)
* [9.3 Simulating stochasticity](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#simulating-stochasticity)
* [9.4 Population Viability Analysis: An over-simplified example](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#population-viability-analysis-an-over-simplified-example)
  + [9.4.1 The Song Sparrow Dataset](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#the-song-sparrow-dataset)
  + [9.4.2 Calculating lambda from counts](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#calculating-lambda-from-counts)
  + [9.4.3 Projecting the population over 20 years](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#projecting-the-population-over-20-years)
  + [9.4.4 The PVA: What is the extinction risk?](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#the-pva-what-is-the-extinction-risk)

In this module, we will be demonstrating some basic concepts in population ecology using R. This will help us get practice with using a variety of R tricks such as apply functions, sampling random numbers, and using for-loops.

Set up:

To begin this module, I will show you how to set up the .Rmd file so that code does not go off the page when rendering in pdf. To do this, you will use the opts\_chunk$set() function in the knitr package:

**library**(knitr)

opts\_chunk$set(tidy.opts=list(width.cutoff=60),tidy=TRUE)

Let’s also install and load packages that we’ll use.

*# if you don't have these packages on your computer:*

install.packages("primer")

install.packages("RColorBrewer")

**library**(primer)

**library**(RColorBrewer)

9.1 Modeling the basic exponential/geometric population growth model

Let’s think of a hypothetical population that you have observed over two years. Say there were 100 individuals in the first year, and 150 individuals in the second year. Then, we can say that the growth rate of the population over those two years is N2N1=1.5N2N1=1.5.  
More generally, we can think of the annual growth rate, λλ as the number of individuals in one year compared to the year before:

λ=Nt+1Ntλ=Nt+1Nt

If this rate of growth is constant over tt number of years and the initial population size is N0N0, then the population size at year tt is:

Nt=N0λtNt=N0λt

This is the model of ***geometric population growth***, i.e., a population growing in discrete time steps with constant growth rate.

***Exponential population growth*** occurs when a population is growing in continuous time with a constant growth rate, rr:

Nt=N0ertNt=N0ert

Here, rr is the *instantaneous rate of growth* which is the birth rate (births per individual) minus the death rate (deaths per individual) at any given time scale.

Geometric and exponential growth models are very similar, except that they differ in several mathematical properties. But given a certain time unit, we can go between the two models pretty easily. You just have to remember that λ=erλ=er and lnλ=rln⁡λ=r

Let’s now project population growth over 10 years with starting population of 100 and λ=1.5λ=1.5. What we need to do is assign the constants (N0N0and λλ) and then project the population sizes at NtNt when tt ranges from 0 (intial population) to 10

N0 = 100

lambda = 1.5

t = 0:10

N = N0 \* lambda^t

round(N, 0)

## [1] 100 150 225 338 506 759 1139 1709 2563 3844 5767

*# I'm rounding this here because it doesn't make sense to*

*# have a fraction of an individual.*

Let’s plot this on a linear scale and log scale.

plot(t, N, type = "o", pch = 19, las = 1)

plot(t, N, type = "o", log = "y", pch = 19, las = 1)

**Note:** *Compare this with this alternative method to plot y on the log-scale (base 10 log). Clue: pay attention to the y-axis tick marks.*

*# output not shown*

plot(t, log10(N), type = "o", pch = 19, las = 1)

These two plots describe exponential (or geometric) population growth: i.e., population under a constant growth rate.

**A digression**: Let’s use this opportunity to demonstrate a solution to a question I’ve had a number of times in assignments: how can you plot a two-panel plot and customize the margins on each panel separately? You can do this by setting up the plotting using a layout() function (instead of par(mfrow=)), which will allow us to set the margins separately for each panel. Here’s a demo:

layout(matrix(c(1, 2), nrow = 2))

par(mar = c(0.5, 4, 4, 4))

plot(t, N, type = "o", pch = 19, las = 1, xaxt = "n", main = "Population Growth on Two Scales")

axis(side = 1, labels = F)

par(mar = c(4, 4, 0.5, 4))

plot(t, N, type = "o", log = "y", pch = 19, las = 1)

9.2 Varying the growth rate

Now, let’s try projecting the population with different growth rates, λλ. Let’s think about how to do this. We will now have two parameters that vary (ttand λλ). This means that we want an output that is a *matrix*, with one parameter (say tt) on the rows and the other parameter (λλ) on the columns. We will do this by employing the sapply() function.  
Now, we will define a sequence numbers for λλ (here, it will be 0.6, 0.8, 1, 1.2 and 1.4) and then use sapply() to run multiple versions of the population growth rate equation. This function will return a vector or matrix, depending on what is the simplest form.

N0 = 100

lambda = seq(0.6, 1.4, 0.2)

t = 0:10

N = sapply(lambda, **function**(lambda) N0 \* lambda^t)

N

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

## [1,] 100.0000000 100.00000 100 100.0000 100.0000

## [2,] 60.0000000 80.00000 100 120.0000 140.0000

## [3,] 36.0000000 64.00000 100 144.0000 196.0000

## [4,] 21.6000000 51.20000 100 172.8000 274.4000

## [5,] 12.9600000 40.96000 100 207.3600 384.1600

## [6,] 7.7760000 32.76800 100 248.8320 537.8240

## [7,] 4.6656000 26.21440 100 298.5984 752.9536

## [8,] 2.7993600 20.97152 100 358.3181 1054.1350

## [9,] 1.6796160 16.77722 100 429.9817 1475.7891

## [10,] 1.0077696 13.42177 100 515.9780 2066.1047

## [11,] 0.6046618 10.73742 100 619.1736 2892.5465

What we end up with is a matrix with 11 rows corresponding to t=t= 0 through 10 and 5 columns corresponding to the five values of λλ (0.6, 0.8, 1, 1.2, 1.4). We can plot this output nicely using the matplot() function:

matplot(t, N, las = 1)

Let’s make it prettier by creating a color palette in RColorBrewer and using different point types and lines to show the different trajectories.

colors = brewer.pal(5, "Set1")

matplot(t, N, type = "o", las = 1, pch = 1:5, col = colors, lty = 1:5)

legend("topleft", legend = c("0.6", "0.8", "1", "1.2", "1.4"),

title = "lambda", pch = 1:5, lty = 1:5, col = colors, cex = 0.8)

Ok, we can see that there are multiple trajectories here, but it’s kind of hard to see what is goin on at the bottom of the figure. Let’s try plotting the population sizes on a log scale:

colors = brewer.pal(5, "Set1")

matplot(t, N, type = "o", las = 1, pch = 1:5, col = colors, lty = 1:5,

log = "y", yaxt = "n")

marks = c(1, 5, 10, 50, 100, 500, 1000)

axis(side = 2, at = marks, labels = format(marks, drop0trailing = TRUE),

las = 1)

legend("bottomleft", legend = c("0.6", "0.8", "1", "1.2", "1.4"),

title = "lambda", pch = 1:5, lty = 1:5, col = colors, cex = 0.8)

You can see that two lines go up (λ>1λ>1), two lines go down (λ<1λ<1) and one line stays flat (λ=1λ=1). That is, populations increase when λ>1λ>1and vice versa.

Since the instantaneous rate of growth, r=lnλr=ln⁡λ:

* Population does not change when λ=1λ=1 or r=0r=0
* Population increases when λ>1λ>1 or r>0r>0
* Population decreases when λ<1λ<1 or r<0r<0

9.3 Simulating stochasticity

Thus far, we have assumed that population growth rate is constant through time. This is clearly not the case in most natural populations. Even disregarding density-dependent mechanisms, populationg growth rates fluctuate due to different sources of stochasticity (i.e., randomness). One major source of randomness in population growth rate is ***environmental stochasticity:*** i.e., random fluctuations in population growth rate due to good and bad years.

Let’s say that a population has a stable population growth rate–i.e., average instantaneous growth rate, r¯=0r¯=0. But the actual growth rate fluctuates around this average with some variance, σ2r=0.01σr2=0.01. Hence, for each time step (say, 1,000), we could randomly draw a number from a normal distribution with mean of 0 and variance of 0.01 (or sd=var−−−√=0.1sd=var=0.1), like so:

set.seed(2)

rs = rnorm(1000, mean = 0, sd = 0.1)

hist(rs, xlab = "r")

For consistency of modeling purposes, we want to convert this distribution of growth rates to the discrete population growth rate, λλ, which can also be expressed as erer. So, the distribution of λλ in this case is:

hist(exp(rs), xlab = "lambda", main = "Histogram of lambdas")

The two distributions look similar, but the top (distribution of rr) is a **normal distribution** centered around 0, and bottom (distribution of λλ) is **log-normal distribution**–i.e., the log of the value is a normal distribution–centered around 1.

For simplicity, we can also just take the distribution of lambdas from a log-normal distribution directly (output not shown):

set.seed(2)

lambdas = rlnorm(100, meanlog = 0, sdlog = 0.1)

round(lambdas, 2)

Ok, now let’s use this method of generating stochasticity in population growth rates to simulate population growth. Let’s remind ourselves what the population projection over 20 years looks like under geometric population growth with λ=1.2λ=1.2 and N0=100N0=100.

Here, let’s try generating this plot slightly differently–we will use a for-loop to calculate NtNt for each time step tt and store it as a vector.

N0 = 100 *#initial population size*

times = 20 *#number of years into the future*

N = vector(length = times) *#empty vector to store pop. sizes*

N[1] = N0 *#initial population size should be the first N*

lambda = 1.2 *#growth rate*

*# start loop: Take previous year's N and multiply by lambda*

**for** (t **in** 2:times) {

N[t] = N[t - 1] \* lambda

}

plot(1:times, N, type = "o", las = 1)

This is a bit more round-about than before, but this lays the groundwork for our stochasticity simulation. Now, let’s simulate a population with the same mean growth rate[1](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#fn1) but with standard deviation of 0.1.

set.seed(2)

N0 = 100 *#initial population size*

times = 20 *#number of years into the future*

N = vector(length = times) *#empty vector to store pop. sizes*

N[1] = N0 *#initial population size should be the first N*

*# lambdas--we only need 19 numbers because growth only*

*# happens between 2 years.*

lambda = rlnorm(times - 1, meanlog = 0, sdlog = 0.1)

*# start loop: Take previous year's N and multiply by lambda*

**for** (t **in** 2:times) {

N[t] = N[t - 1] \* lambda[t - 1]

}

plot(1:times, N, type = "o", las = 1)

The main take away is that the population grows much more slowly when there is stochasticity in the growth rate. Whereas the exponential growth model predicted the population to be >3,000 in 20 years, this simulation shows the population to be about 140 individuals in 20 years.

Exercise: Remove the set.seed() part of the routine above and try running the script several times to see how the dynamics can change.

Now, let’s try running this simulation multiple times (10x) and plotting the results in one figure. To do this, we will employ the sapply() function and run the for-loop sequence inside it. The nice thing about this is that the sapply() function will automatically organize the output as a matrix for us, so that we can directly use the matplot() function to plot the results.

*# multiple simulations*

set.seed(2)

sims = 10

outmat = sapply(1:sims, **function**(x) {

times = 20

N0 = 100

N = vector(length = times)

N[1] = N0

lambda = rlnorm(times - 1, meanlog = 0, sdlog = 0.1)

**for** (t **in** 2:times) {

N[t] = N[t - 1] \* lambda[t - 1]

}

N

})

matplot(1:times, outmat, type = "l", las = 1, ylab = "Population Size",

xlab = "Years")

abline(h = 100, lty = 2)

This figure shows the population trajectories of this population under 10 runs of the simulation. Remember that in all of these cases, the mean population growth rate[2](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#fn2) is 1.2–i.e., the population is supposed to be growing at a pretty quick pace. Yet, none of the simulations show the population growing much over 20 years. In fact, the population size decreased in 4 out of the 10 simulations. This demonstrates a central concept in population ecology (and conservation biology)–*variation in population growth rates tend to slow down population growth*.

9.4 Population Viability Analysis: An over-simplified example

Population Viability Analysis (PVA) is a population modeling method to assess extinction risk of a population or species. It is the mathematical foundation of conservation biology, and it is used worldwide to guide management decisions for threatened species. Here, I will walk through a simplified example of PVA using an example dataset.

9.4.1 The Song Sparrow Dataset

Here, we will be using the sparrows dataset from the R package primer. You should have already installed and loaded the package at the beginning of the module. If you haven’t yet, run library(primer) to load it.  
The primer package is a package that is associated with a textbook called “A Primer of Ecology with R”, written by Henry Stevens[3](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#fn3). The sparrowdataset is based on counts of Song Sparrows (*Melospiza melodia*) in Darrtown, OH from 1966 to 2003.

Song Sparrow

Let’s load the dataset and look at it:

data(sparrows)

head(sparrows)

## Year Count ObserverNumber

## 1 1966 34 1

## 2 1967 40 1

## 3 1968 42 1

## 4 1969 54 1

## 5 1970 49 1

## 6 1971 71 1

We can plot the data:

plot(Count ~ Year, data = sparrows, type = "o", las = 1)

9.4.2 Calculating lambda from counts

The first thing we can do with this data is to extract the annual population growth rates λλ for each pair of years. That is, we need to calculate N2N1N2N1, N3N2N3N2, N4N3N4N3, etc.

counts = sparrows$Count

l = counts[-1]/counts[-length(counts)]

round(l, 2)

## [1] 1.18 1.05 1.29 0.91 1.45 0.93 0.85 1.29 1.07 0.48 0.86 0.81 0.85 1.41

## [15] 0.90 1.46 0.88 1.03 1.19 0.77 1.38 0.96 0.91 0.78 1.25 3.00 0.80 0.97

## [29] 0.72 1.25 1.07 0.41 0.92 1.00 1.26

hist(l, breaks = 20, main = "Histogram of lambdas")

Let’s extract the mean and standard deviation of lnλln⁡λ:

mean(log(l))

## [1] 0.006709703

sd(log(l))

## [1] 0.3366946

You’ll notice that the average lnλ=rln⁡λ=r is really close to 0, but slightly above it. This means that, the population is pretty stable at the moment. However, the variance is fairly big.

Let’s just assume that we can estimate the distribution of annual population growth rates λλ as a log-normal distribution with those mean-logs and sd-logs. With this, we can generate 50 projected λλ values:

set.seed(2)

sim.l = rlnorm(50, meanlog = mean(log(l)), sdlog = sd(log(l)))

round(sim.l, 2)

## [1] 0.74 1.07 1.72 0.69 0.98 1.05 1.28 0.93 1.96 0.96 1.16 1.40 0.88 0.71

## [15] 1.83 0.46 1.35 1.02 1.42 1.16 2.04 0.67 1.72 1.94 1.01 0.44 1.18 0.82

## [29] 1.31 1.11 1.29 1.12 1.45 0.91 0.78 0.82 0.56 0.74 0.83 0.93 0.88 0.52

## [43] 0.76 1.91 1.24 1.97 0.91 0.98 0.95 0.67

9.4.3 Projecting the population over 20 years

Let’ use this to simulate this Song Sparrow population over the next 20 years (or rather, from 2003 to 2023).

*# Single simulation*

set.seed(2)

time = 21

N0 = 43

N = vector(length = time)

N[1] = N0

sim.l = rlnorm(time, meanlog = mean(log(l)), sdlog = sd(log(l)))

**for** (t **in** 2:time) {

N[t] = N[t - 1] \* sim.l[t - 1]

}

par(mar = c(4, 4, 1, 4))

plot(1:(time), N, type = "o", las = 1, xaxt = "n")

axis(side = 1, at = c(1, 6, 11, 16, 20), labels = c(2003, 2008,

2013, 2018, 2023))

Using the same principles as the stochastic simulations in Section 9.3, we can also conduct multiple simulations and plot the results

set.seed(2)

sims = 5

outmat = sapply(1:sims, **function**(x) {

time = 21

N0 = 43

N = vector(length = time)

N[1] = N0

sim.l = rlnorm(time, meanlog = mean(log(l)), sdlog = sd(log(l)))

**for** (t **in** 2:time) {

N[t] = N[t - 1] \* sim.l[t - 1]

}

N

})

par(mar = c(4, 4, 1, 4))

matplot(1:time, outmat, type = "l", las = 1, lty = 5, ylim = c(0,

200), ylab = "N", xaxt = "n", xlab = "Year")

axis(side = 1, at = c(1, 6, 11, 16, 20), labels = c(2003, 2008,

2013, 2018, 2023))

9.4.4 The PVA: What is the extinction risk?

Now, let’s say we want to ask a question about extinction risk of this population. For example, we might ask what are the chances that this population goes extinct within 100 years, given the observed pattern in population sizes? To figure this out, we need to conduct a large number of simulations (in this case, let’s do 1,000x) projecting the population for 100 years (or 101, if starting from year 0). Then, we want to ask: in how many iterations of this model does the population size dip below 2 (the minimum number of individuals needed to reproduce)?

Let’s start by doing the simulations in the same way as above, but changing sims=1000 and time=101.

set.seed(2)

sims = 1000

outmat = sapply(1:sims, **function**(x) {

time = 101

N0 = 43

N = vector(length = time)

N[1] = N0

sim.l = rlnorm(time, meanlog = mean(log(l)), sdlog = sd(log(l)))

**for** (t **in** 2:time) {

N[t] = N[t - 1] \* sim.l[t - 1]

}

N

})

dim(outmat)

## [1] 101 1000

So, what we have now is a matrix with 101 rows (year 0 to year 100), and 1,000 columns, each representing a single iteration of the simulation. Thus, we could figure out how many columns has at least one time point in which N < 2 by using the apply() function to ask whether the minimum of each column is less than 2.

*# which columns have at least one value less than 2?*

minpop = apply(outmat, 2, **function**(x) min(x) < 2)

sum(minpop + 0)/sims *#proportion of columns with TRUE*

## [1] 0.279

Thus, there is approximately a 28% chance that the population will go extinct within 100 years due purely to stochasticity.

Now, this is a really simplified version of PVA–it does not account for uncertainty in the estimates of mean and sd of lnλln⁡λ, nor does it take into account any other realistic factors such as temporal autocorrelation of growth rates, density-dependent factors, habitat change/loss, etc. etc. But I hope that you got a sense for how to use R for simulating complex dynamics.

1. Technically, the *geometric mean* of λλ would be 1.2 here, because lambda is going to have a log-normal distribution.[↩](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#fnref1)
2. Again, technically speaking, I mean the *geometric mean*.[↩](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#fnref2)
3. Stevens, M.H.H. (2009) *A Primer of Ecology with R*, Springer, 2nd Printing.[↩](https://rstudio-pubs-static.s3.amazonaws.com/252603_9c6a71110dc74cc7832f154449235f7f.html#fnref3)