R Practice

Sean Denny

7/8/2019

Questions:

1. If I open RStudio, will all of the Github files already be updated/pulled? Or do I need to pull again?
2. Can I only pull if someone has changed something? Or should I be able to pull regardless?
3. What is the white M in the blue box next to our file name for teamNudi?
4. Confused about the notation for sapply (chunk 10).

Header

Smaller

This is as small as it gets, I think

I can now just type normally and it's the regular font size, but it's not bold. It's equivalent to size ###. If I wanted to bold it though, I could just **do this**. Or, I could italicize by doing *this*. If I wanted to make a list...

1. I
2. Could
3. Do
4. This

To add a new R chunk, use command+option+i, or just go to insert and insert an R script.

```
mean <- 8
```

To reference something in the text in Rmarkdown use backticks and "r {function name}". Example, the mean is **8**. You can also use backticks just to write a function in Rmarkdown text without it applying the function. Example `\pagebreak`.

```
# start by loading packages

library(tidyverse)
library(RColorBrewer)
```

```
#Use warning = FALSE in R chunk header to prevent Rmarkdown including warning messages in HTML or PDF o

#This book has it's own R package

# install.packages('primer')
library(primer)
```

Finally, if you wanted to do a pagebreak you can use this:

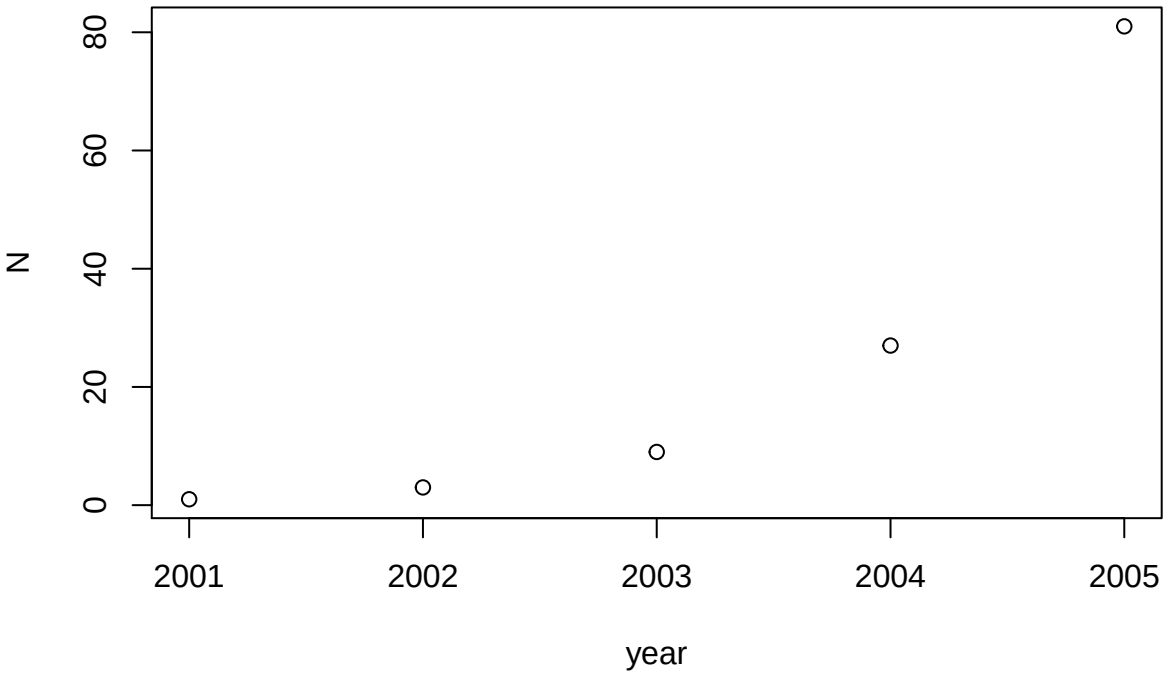
Chapter 1

(By using {.tabset} in Rmarkdown, you can create tabs in the HTML format for the following headers, I believe.)

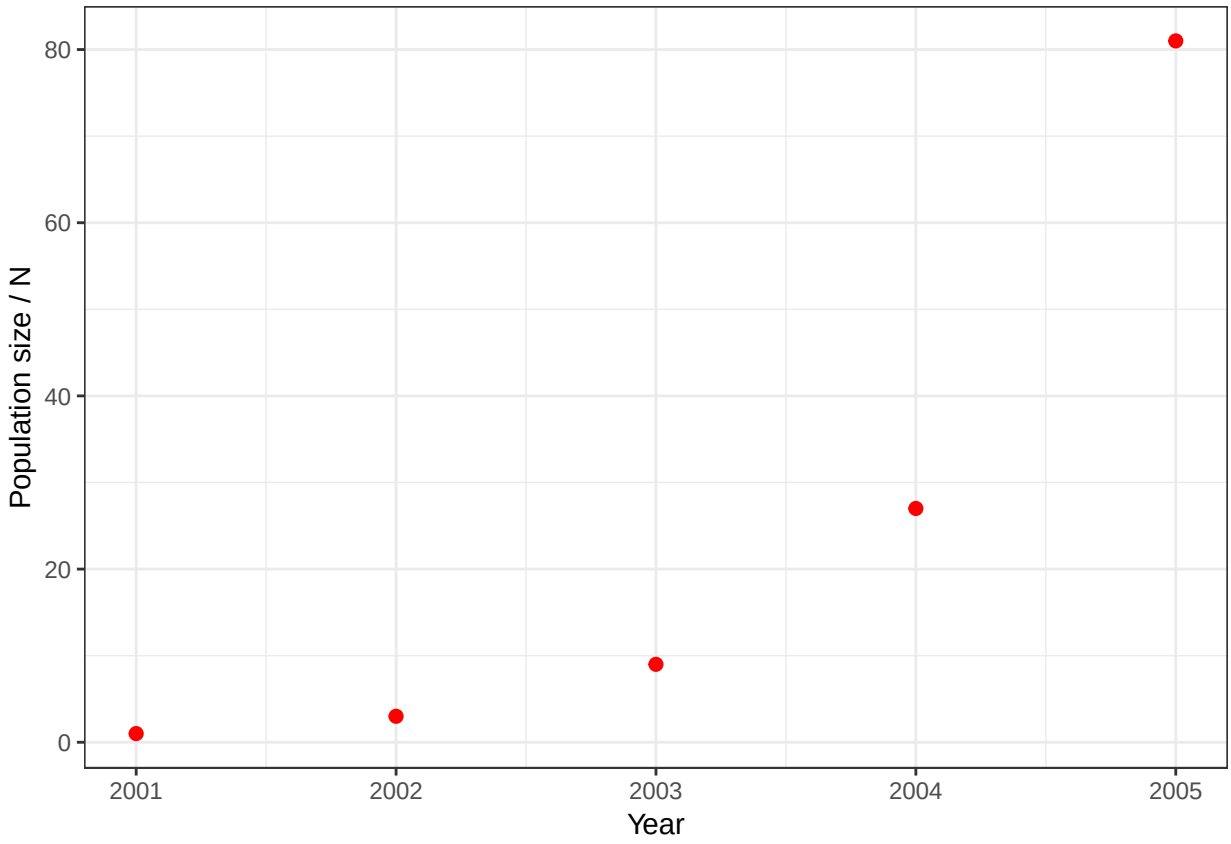
1.3 Exploring Population Growth

Simple graphing of population size (Fig 1.3):

```
# Original version

N <- c(1, 3, 9, 27, 81)
year <- 2001:2005
plot (year, N)
```




```
# Sean's version

exp_growth <- data.frame(N = c(1, 3, 9, 27, 81), year = 2001:2005)

exp_growth_plot <- ggplot(exp_growth, aes(x= year, y = N)) +
   geom_point(color = 'red', size = 2) +
  theme_bw() +
  labs(x = 'Year', y = 'Population size / N')
exp_growth_plot
```



Vectorized math

Here we divide each element of one vector (the second through fifth element of N) by each element of another vector (the first through fourth elements of N).

```
rates = N[2:5/1:4]
rates
```

```
## [1] 3 1 1 1
```

```
rates = N[2:5]/N[1:4]
rates
```

```
## [1] 3 3 3 3
```

```
# [1] 3 3 3 3

# Same rate of change for each population
```

1.3.1 Projecting population size

Here we calculate population sizes for 10 time points beyond the initial. First we assign values for N_0, λ, and time.

```
N0 <- 1
lambda <- 2
time <- 0:10
```

Next we calculate N_t directly using our general formula.

```
Nt <- N0*lambda^time
Nt
```

```
##  [1]    1    2    4    8   16   32   64  128  256  512 1024
```

```
# [1] 1 2 4 8 16 32 64 128 256 5121024
```

1.3.2 Effects of Initial Population Size

We first set up several different initial values, provide a fixed λ, and set times from zero to 4.

```
N0 <- c(10, 20, 30, 100)
lambda <- 2
time <- 0:5
```

We calculate population sizes at once using `sapply` to *apply* a function (`n*lambda^time`) to each element of the first argument (each element of `N0`).

```
Nt.s <- sapply(N0, function(n) n*lambda^time)
Nt.s
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   10   20   30  100
## [2,]   20   40   60  200
## [3,]   40   80  120  400
## [4,]   80  160  240  800
## [5,]  160  320  480 1600
## [6,]  320  640  960 3200
```

```
Nt.s <- sapply(N0, function(n) n * lambda^time)
Nt.s
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   10   20   30  100
## [2,]   20   40   60  200
## [3,]   40   80  120  400
## [4,]   80  160  240  800
## [5,]  160  320  480 1600
## [6,]  320  640  960 3200
```

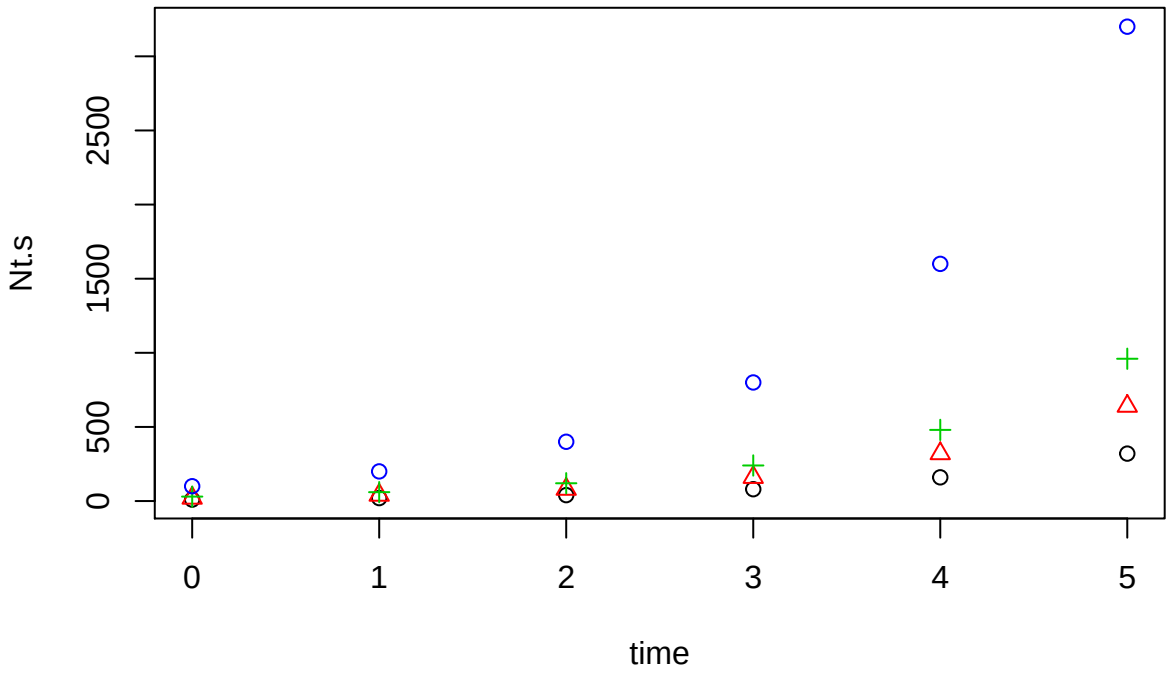
```
#      [,1] [,2] [,3]
# [1,]   10   20   30
# [2,]   20   40   60
# [3,]   40   80  120
# [4,]   80  160  240
# [5,]  160  320  480
```

The result is a matrix, and we see N_0 in the first row, and each population is in its own column. Note that population 2 is always twice as big as population 1.

Graphing a Matrix (Figs. 1.3a, 1.3b)

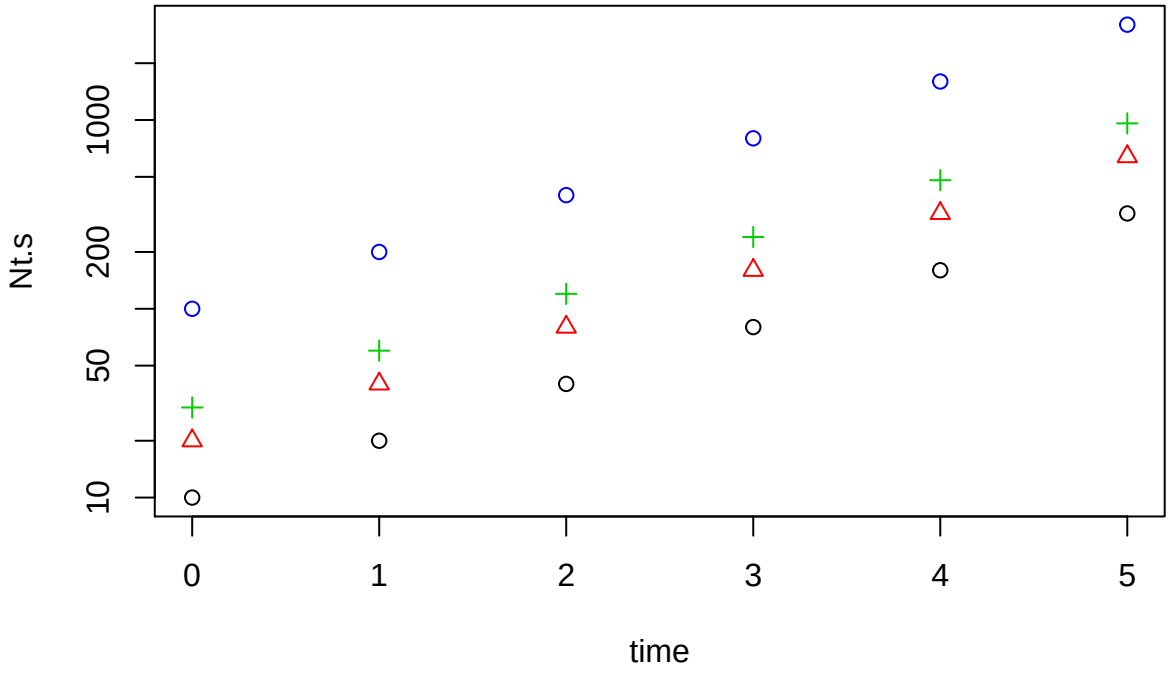
We can use matplot to plot a matrix vs. a single vector on the X-axis. By default it labels the points according to the number of the column

```
matplot(time, Nt.s, pch = 1:3)
```



We can also plot it with a log scale on the y-axis.

```
matplot(time, Nt.s, log = "y", pch = 1:3)
```



1.3.3 Effects of different per capita growth rates

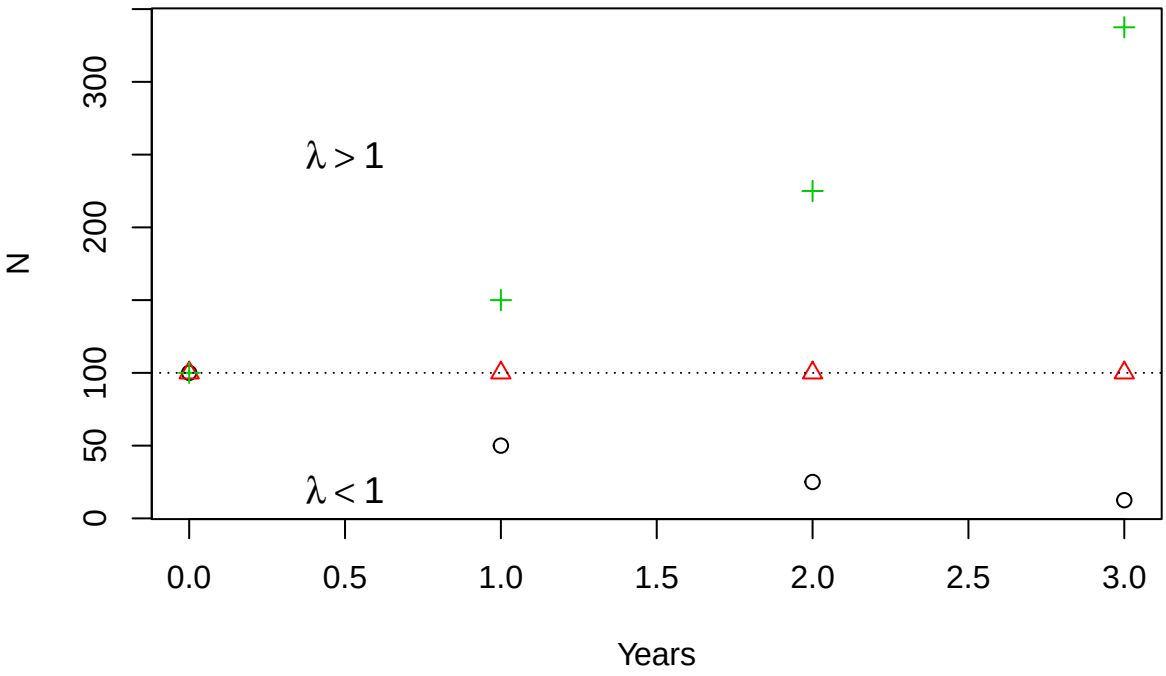
Here we demonstrate the effects on growth of $\lambda > 1$ and $\lambda < 1$. We set $N_0 = 100$, and time, and then pick three different λ.

```
N0 <- 100
time <- 0:3
```

```
lambdas <- c(0.5, 1, 1.5)
```

We use `sapply` again to apply the geometric growth function to each λ. This time, `x` stands for each λ, which our function then uses to calculate population size. We then plot it, and add a reference line and a little text.

```
N.all <- sapply(lambdas, function(x) N0 * x^time)
matplot(time, N.all, xlab = "Years", ylab = "N", pch = 1:3)
abline(h = N0, lty = 3)
text(0.5, 250, expression(lambda > 1), cex = 1.2)
text(0.5, 20, expression(lambda < 1), cex = 1.2)
```



The reference line is a ***h***orizontal line with the ***l***ine ***ty***pe dotted. Our text simply indicates the regions of positive and negative growth.

1.3.4 Average growth rate

Comparing arithmetic and geometric averages (Fig. 1.5)

First we select the number of observed R ($t = 5$); this will require that we use six years of Song Sparrow data.

```
t <- 5
data(sparrows)
SS6 <- sparrows[1:(t + 1), ]
```

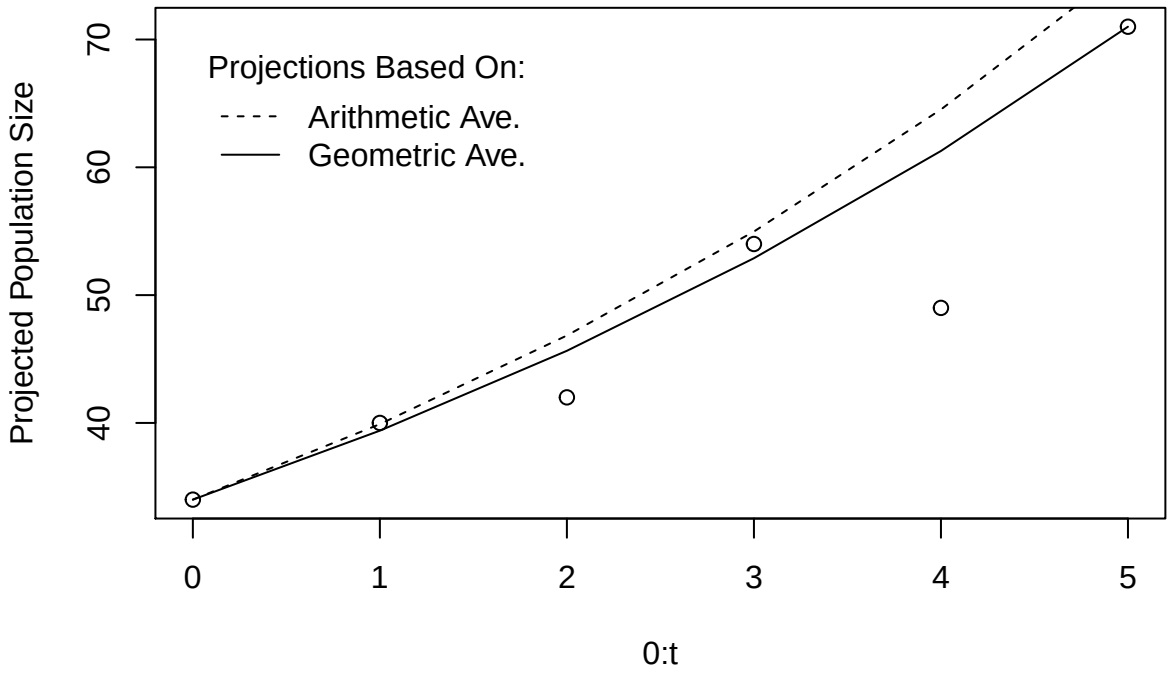
Next we calculate λ for each generation, from t to $t + 1$, and calculate the arithmetic and geometric means.

```
SSgr <- SS6$Count[2:(t + 1)]/SS6$Count[1:t]
lam.A <- sum(SSgr)/t
lam.G <- prod(SSgr)^(1/t)
```

Now we can plot the data, and the projections based on the two averages (Fig. 1.5).

```
N0 <- SS6$Count[1]
plot(0:t, SS6$Count, ylab = "Projected Population Size")
lines(0:t, N0 * lam.A^(0:t), lty = 2)
```

```
lines(0:t, N0 * lam.G^(0:t), lty = 1)
legend(0, 70,
       c("Arithmetic Ave.", "Geometric Ave."),
       title = "Projections Based On:",
       lty = 2:1, bty = "n", xjust = 0)
```



1.4 Continuous Exponential Growth

1.4.1 Motivating continuous exponential growth

Numerical approximation of e

Here we use brute force to try to get an approximate solution to eq. 1.9. We'll let n be the number of divisions within one year. This implies that the finite rate of increase during each of these fractional time steps is r_d/n. Let the $\lambda = 2$ and therefore $r_d = 1$. Note that because $N_0 = 1$, we could ignore it, but let's keep it in for completeness.

```
n <- 0:100
N0 <- 1
rd <- 1
```

Next, we calculate $\left(1 + \frac{r_d}{n}\right)^n$ for ever larger values of n.

```
N1 <- N0 * (1 + rd/n)^n
```

Last, we plot the ratio and add some fancy math text to the plot (see `?plotmath` for details on mathematical typesetting in R).

```
plot(n, N1/N0, type = "l")
text(50, 2, "For n = 100,")
text(50, 1.6,
     bquote((1+frac("r"["d"],"n"))^"n" == .(round(N1[101]/N0, + 3))))
```

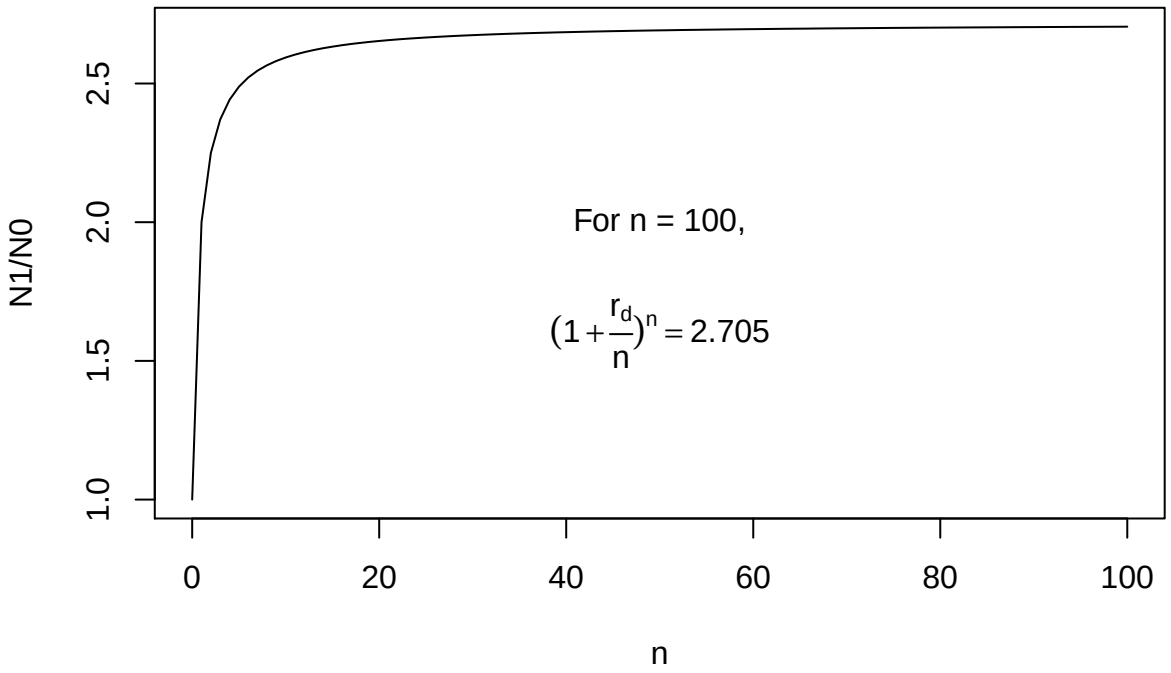


Fig. 1.6: The limit to subdividing reproduction into smaller steps. We can compare this numerical approximation to the true value, $e^1 = 2.718$.

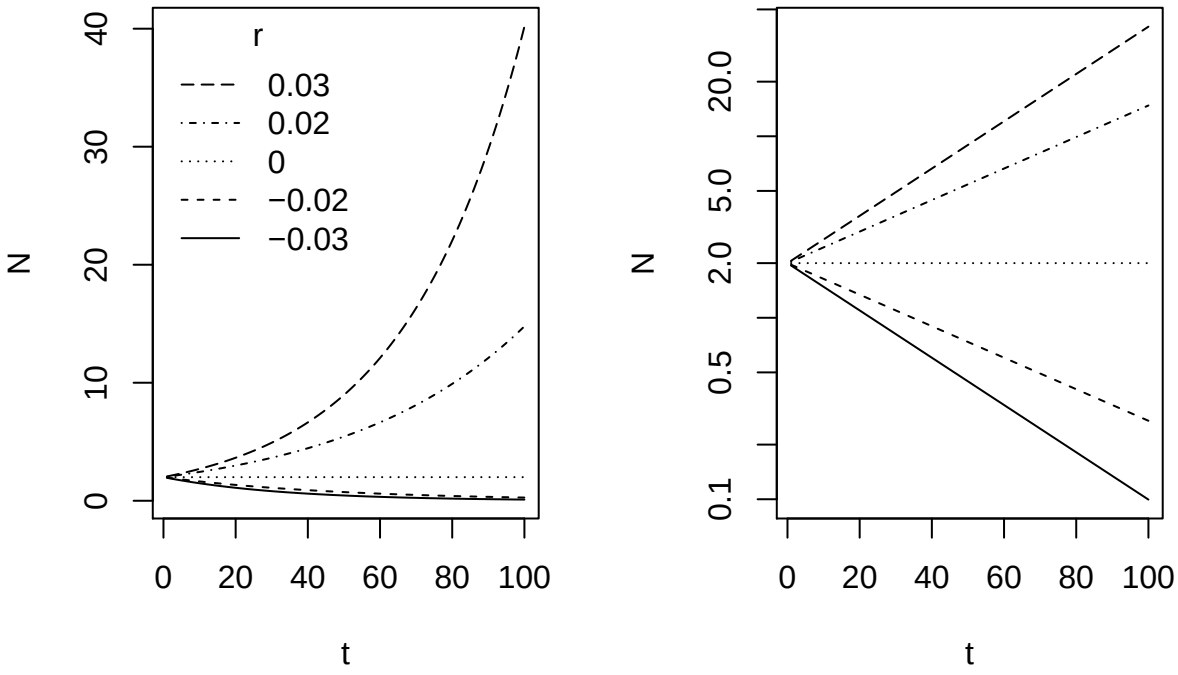
Projecting a continuous population

We select five different values for r: two negative, zero, and two positive. We let t include the integers from 1 to 100. We then use `sapply` to apply our function of continuous exponential growth to each r, across all time steps. This results in a matrix where each row is the population size at each time t, and each column uses a different r.

```
r <- c(-0.03, -0.02, 0, 0.02, 0.03)
N0 <- 2
t <- 1:100
cont.mat <- sapply(r, function(ri) N0 * exp(ri * t))
```

Next we create side-by-side plots, using both arithmetic and logarithmic scales, and add a legend.

```
layout(matrix(1:2, nrow = 1))
matplot(t, cont.mat, type = "l", ylab = "N", col = 1)
legend("topleft", paste(rev(r)),
       lty = 5:1, col = 1, bty = "n",
       title = "r")
matplot(t, cont.mat, type = "l", ylab = "N", log = "y", col = 1)
```



1.4.3 Doubling (and tripling) time

Creating a function for doubling time

We can create a function for this formula, and then evaluate it for different values of m and r. For $m = 2$, we refer to this as "doubling time." When we define the function and include arguments r and m, we also set a default value for `m = 2`. This way, we do not always have to type a value for m; by default the function will return the doubling time.

```
m.time <- function(r, m = 2) {
  log(m)/r
}
```

Now we create a vector of r, and then use `m.time` to generate a vector of doubling times.

```
rs <- c(0, 1, 2)
m.time(rs)
```

```
## [1]       Inf 0.6931472 0.3465736
```

```
# [1]    Inf 0.6931 0.3466
```

Note that R tells us that when $r = 0$, it takes an infinite (`Inf`) amount of time to double. This is what we get when we try to divide by zero!

1.6 Modeling with Data: Simulated Dynamics

1.6.2 Looking at and collecting the data

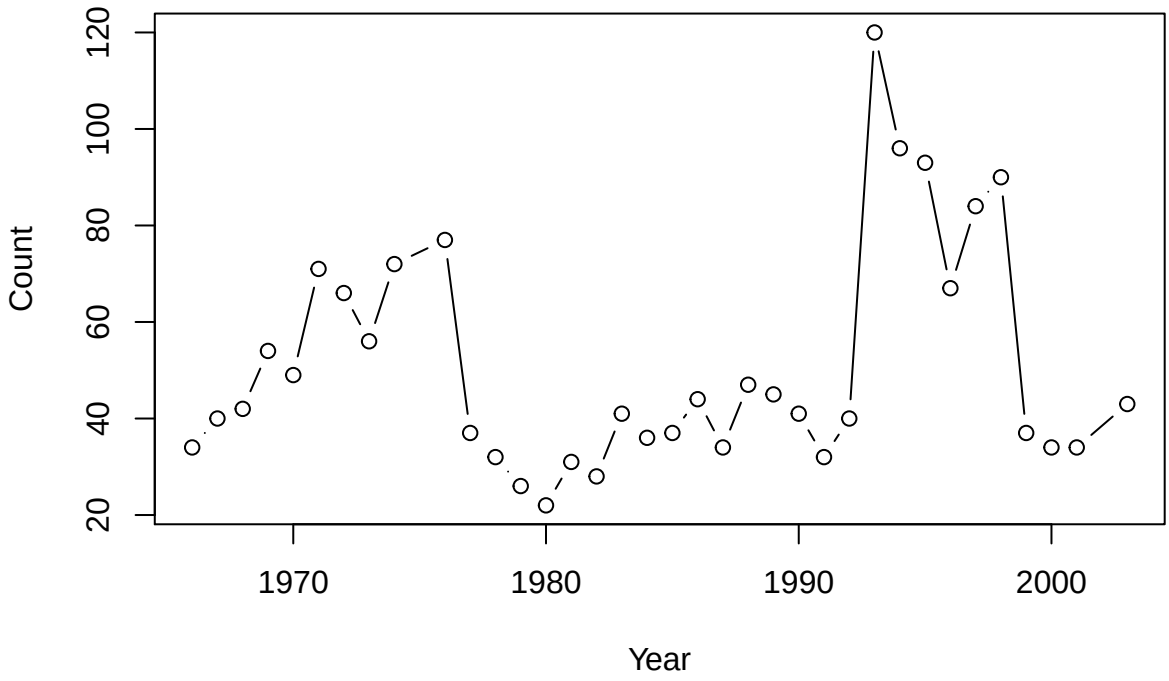
```
names(sparrows)
```

```
## [1] "Year"           "Count"          "ObserverNumber"
```

```
# [1] "Year"           "Count"          "ObserverNumber"
attach(sparrows)
```

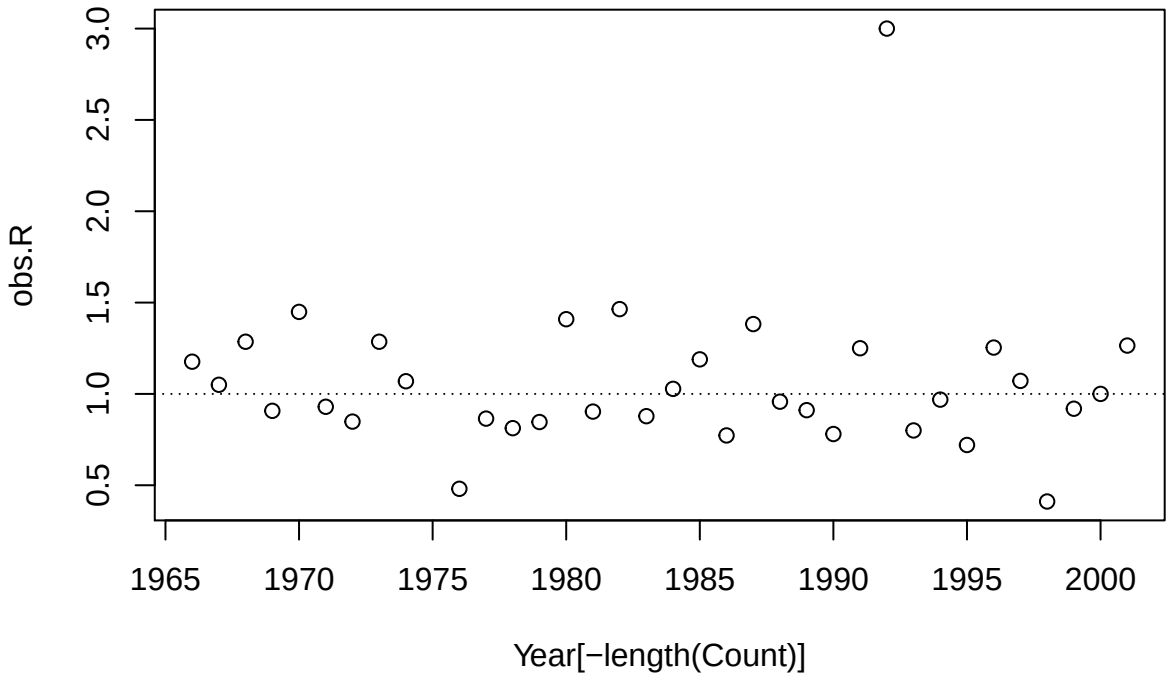
Now we plot these counts through time (Fig. 1.8).

```
plot(Count ~ Year, type = "b")
```



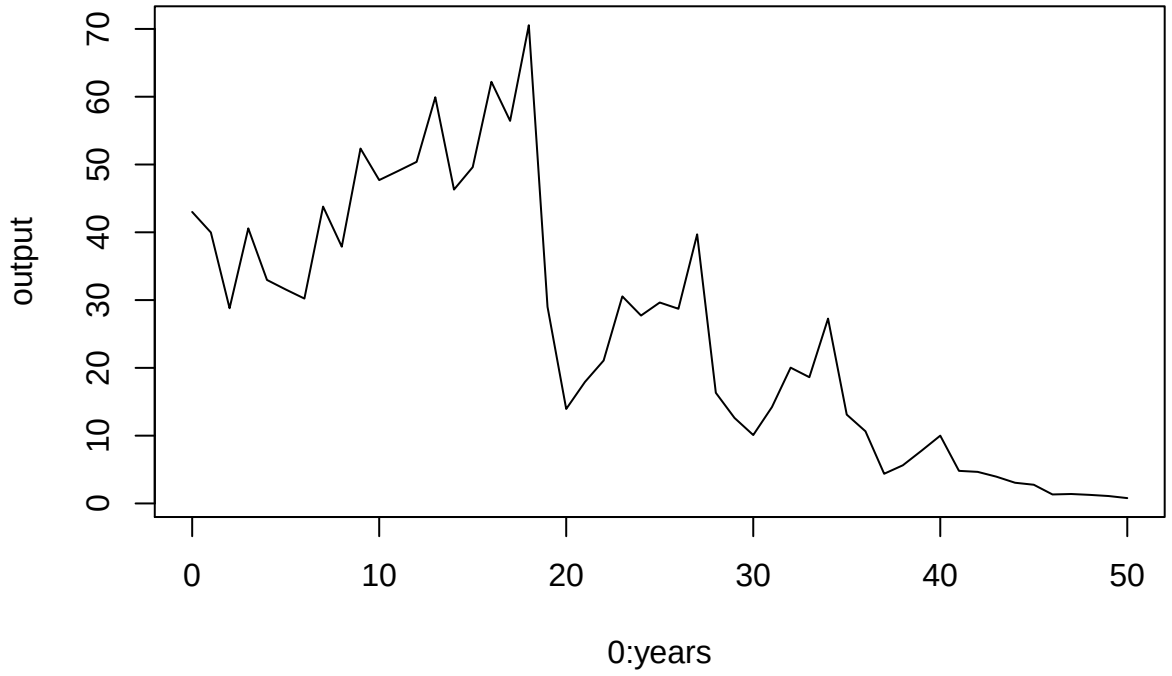

```
obs.R <- Count[-1]/Count[-length(Count)]
```

```
plot(obs.R ~ Year[-length(Count)])
abline(h = 1, lty = 3)
```



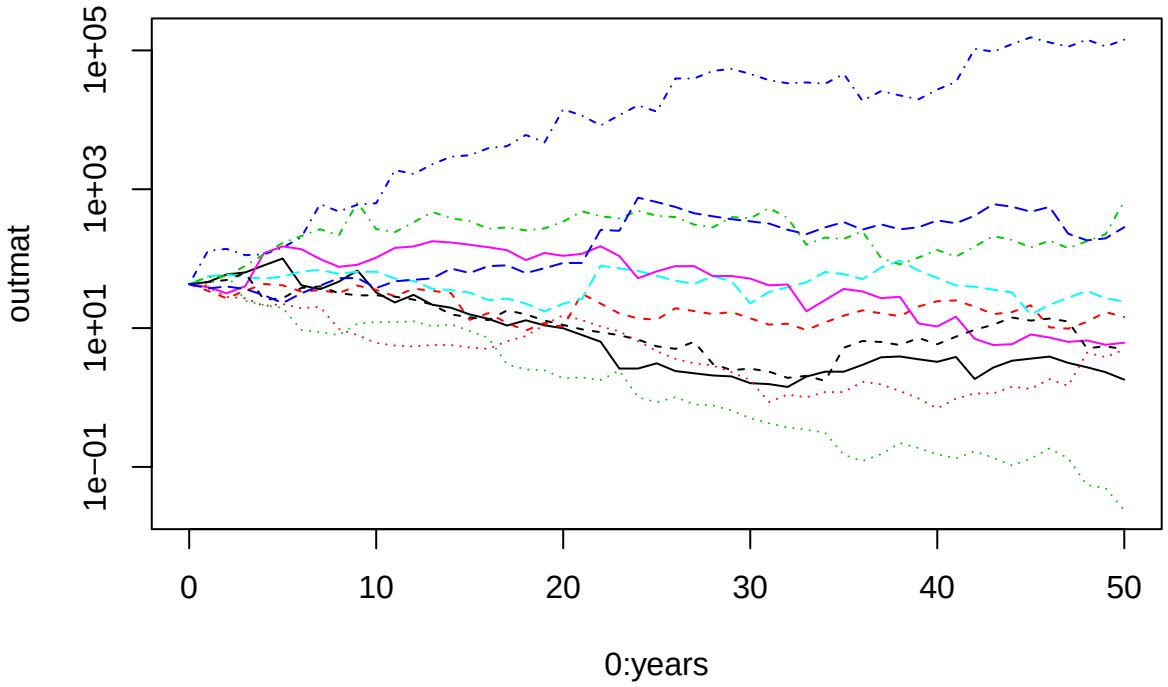
1.6.3 One simulation

```
years <- 50
set.seed(3)
sim.Rs <- sample(x = obs.R, size = years, replace = TRUE)

output <- numeric(years + 1)

output[1] <- Count[Year == max(Year)]

for (t in 1:years) output[t + 1] <- {
  output[t] * sim.Rs[t]
}

plot(0:years, output, type = "l")
```



1.6.4 Multiple simulations

```
sims = 10
sim.RM <- matrix(sample(obs.R, sims * years, replace = TRUE),
                 nrow = years, ncol = sims)

output[1] <- Count[Year == max(Year)]
outmat <- sapply(1:sims, function(i) {
  for (t in 1:years) output[t + 1] <- output[t] * sim.RM[t, i]
  output
  })

matplot(0:years, outmat, type = "l", log = "y")
```



1.6.5 Many simulations, with a function

```
PopSim <- function(Rs, N0, years = 50, sims = 10) {
  sim.RM = matrix(sample(Rs, size = sims * years, replace = TRUE),
                  nrow = years, ncol = sims)
  output <- numeric(years + 1)
  output[1] <- N0
  outmat <- sapply(1:sims, function(i) {
    for (t in 1:years) output[t + 1] <- round(output[t] * sim.RM[t, i], 0)
    output
  })
  return(outmat)
}
```

```
system.time(output <- PopSim(Rs = obs.R, N0 = 43, sims = 1000))
```

```
##    user  system elapsed
##   0.061   0.009   0.070
```

```
#  user  system elapsed
# 0.404   0.004   0.407
```

This tells me that it took less than half a second to complete 1000 simulations. That helps me understand how long 100,000 simulations might take. We also check the dimensions of the output, and they make sense.

```
dim(output)
```

```
## [1]   51 1000
```

```
# [1] 51 1000
```

1.6.6 Analyzing results

```
N.2053 <- output[51, ]
summary(N.2053, digits = 6)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##      0.0     14.0     66.0   1124.6    291.8 332236.0
```

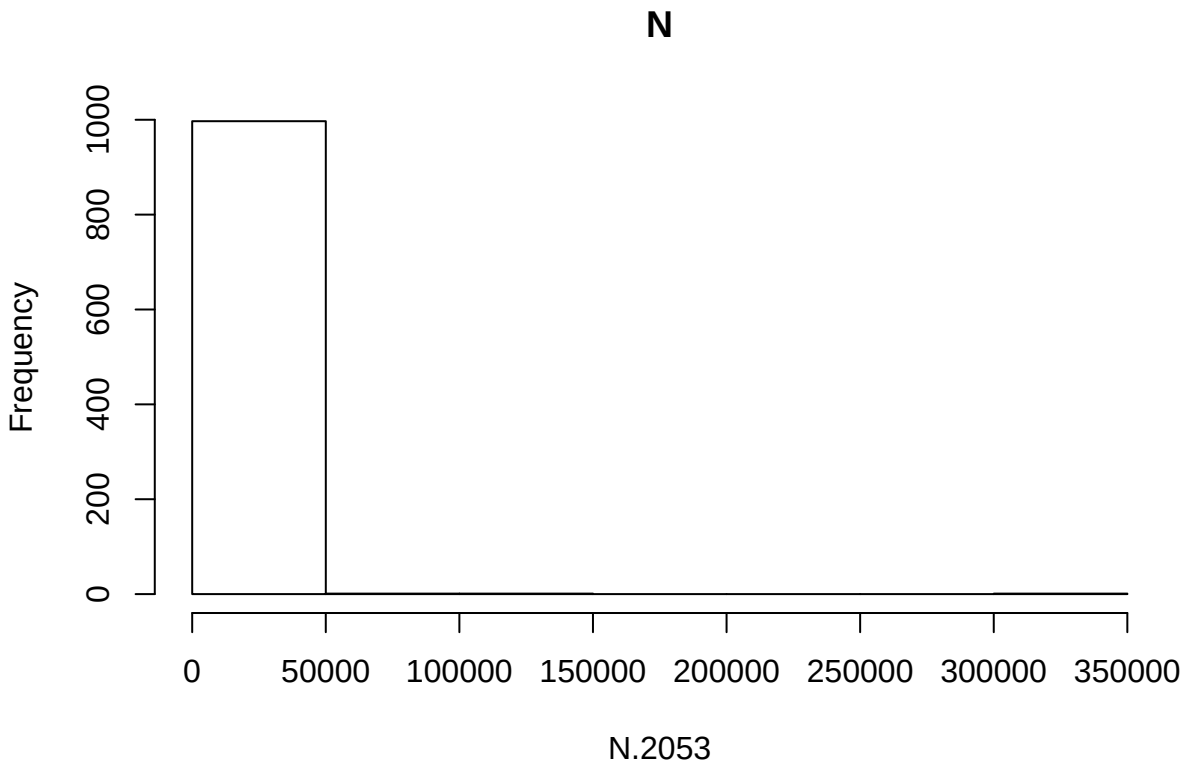
```
# Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#  0.0     14.0     66.0   1124.6    291.8 332236.0
```

```
quantile(N.2053, prob = c(0.0275, 0.975))
```

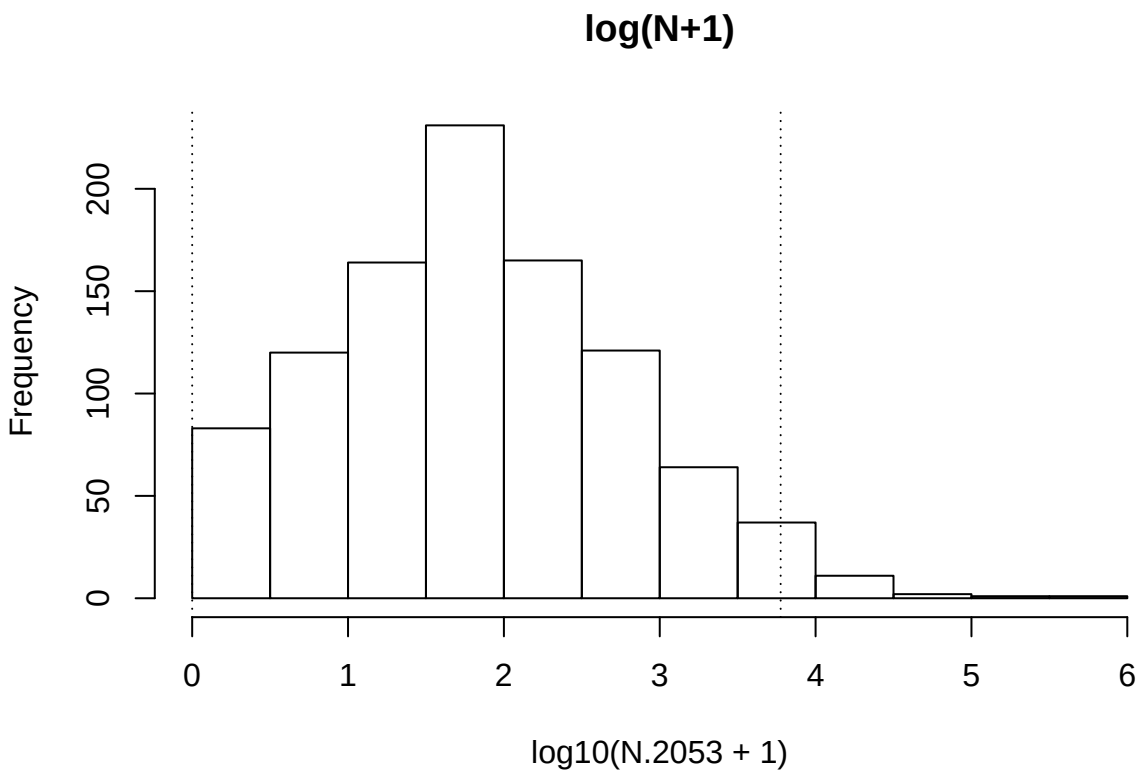
```
##    2.75%    97.5%
##    0.000 5967.375
```

```
# 2.75% 97.5%
#     0  5967
```

```
hist(N.2053, main = "N")
```




```
hist(log10(N.2053 + 1), main = "log(N+1)")
abline(v = log10(quantile(N.2053, prob = c(0.0275, 0.975)) + 1),
       lty = 3)
```




```
logOR <- log(obs.R)
n <- length(logOR)
t.quantiles <- qt(c(0.025, 0.975), df = n - 1)

se <- sqrt(var(logOR)/n)
CLs95 <- mean(logOR) + t.quantiles * se

R.limits <- exp(CLs95)
R.limits
```

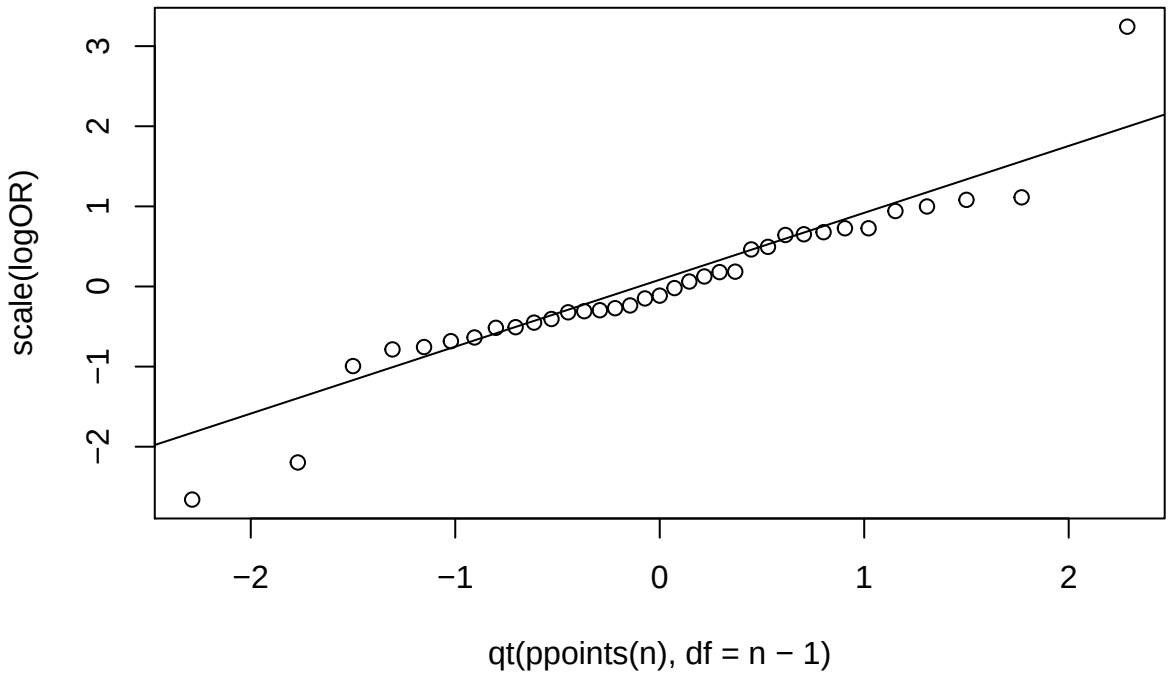
```
## [1] 0.8967762 1.1301703
```

```
# [1] 0.8968 1.1302

N.Final.95 <- Count[Year == max(Year)] * R.limits^50
round(N.Final.95)
```

```
## [1]     0 19528
```

```
# [1] 0 19528
```

```
qqplot(qt(ppoints(n), df = n - 1), scale(logOR))
qqline(scale(logOR))
```



End of chapter questions

1.1. Geometric growth Analyze the following data, relying on selected snippets of previous code.

a. In the years 1996 through 2005, lily population sizes are $N = 150, 100, 125, 200, 225, 150, 100, 175, 100, 150$. Make a graph of population size versus time.
b. Calculate R for each year; graph R vs. time.
c. Calculate arithmetic and geometric average growth rates of this population.
d. Based on the appropriate average growth rate, what would be the expected population size in 2025? What would the estimated population size be if you used the inappropriate mean? Do not use simulation for this.
e. Given these data, develop simulations as above with the user-defined function, `PopSim`. Describe the distribution of projected population sizes for 2010.

1.2. Doubling Time

a. Derive the formula for doubling time in a population with continuous exponential growth.
b. What is the formula for tripling time?
c. If we are modeling humans or *E. coli*, would a model of geometric, or exponential growth be better? Why?
d. If an *E. coli* population grew from 1000 cells to 2×10^9 cells in 6 h, what would its intrinsic rate of increase be? Its doubling time?

1.3. Human Population Growth

a. There were about 630 million people on the planet in 1700, and 6.3 billion in 2003 [33]. What was the intrinsic rate of increase, r?
b. Graph the model of human population size population size from 1700 to 2020.

c. Add points on the graph indicating the population doublings from 1700 onward.
d. What will prevent humans from outweighing the planet by the end of this century? What controls human population growth? Do these controls vary spatially across the planet? See Cohen [33] to get going.

1.4. R functions

Find the R functions in Chapter 1. Demonstrate their uses.