Module 4.3 Exercise

Jelena H. Pantel

2023-12-13 12:14:07.652469

Exercise 1. Our First ABC - estimating parameters of Gaussian distribution given observed data

A. Normal distribution You took some introductory statistics, so I hope you are familiar with a Gaussian distribution (also referred to as a normal distribution). It's useful to try and attribute data to an underlying probability distribution, so we can understand and make inference about that data. For example, many traits that have a genetic basis follow a normal distribution. Body length in animals is a good example of this. Researchers in the early 1900s were quite interested in better understanding genetic inheritance of traits of interest for animal breeding, which is a key reason why population geneticists (RA Fisher) were the ones who developed classical statistical tests such as Analysis of Variance - the goal was to understand how evolution led to shifts in normally distributed traits. The normal distribution itself was discovered by Carl Friedrich Gauss (a German mathematician and physicist!!) in 1809.

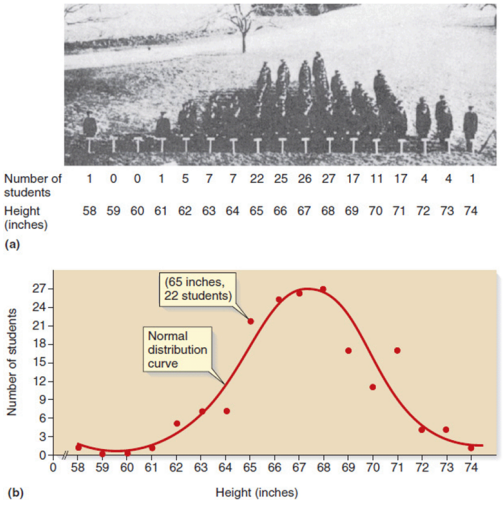


Figure 1: Human height. (a) Height distribution (inches) for 175 students in 1914 attending the Connecticut Agricultural College. (b) Graphical presentation of these student heights showing their close fit to a normal distribution. (a: Reprinted with permission from Albert and Blakeslee: Corn and Man. Journal of Heredity. 1914;5:51. Oxford University Press. b: Reprinted with permission from Brooker RJ: Genetics: Analysis & Principles, 3rd ed. New York: McGrawHill, 2008.)

We can plot data and use our own eyes to consider whether it follows a normal distribution. But I would like you to have an awareness that even claiming / stating "The data x follows a normal distribution" is - you guessed it - formulating a model! So then, given a set of data, we can propose a model, and estimate the paramater values when fitting that data to the model. We will do this with a normal distribution today.

B. The easiest / typical way to estimate parameters of a normal model when fit to observed data The focus of today's exercise will be to use Approximate Bayesian Computation (ABC) for fitting data to models. ABC is most needed with very complicated models where the underlying parameters are often unkown and difficult to estimate. However, its best to show you how ABC works by applying it to a simpler model. So we will learn to estimate the parameters in a normal distribution given a set of data.

The dataset of interest is body weight of a common household insect. Please run the following command to load the dataset:

```
# Dataset with measures of 1000 household insects body mass
# (g)
dat <- read.csv("https://raw.githubusercontent.com/jhpantel/ude-ecomod/main/data/insect.csv",
    header = TRUE, row.names = 1)
```

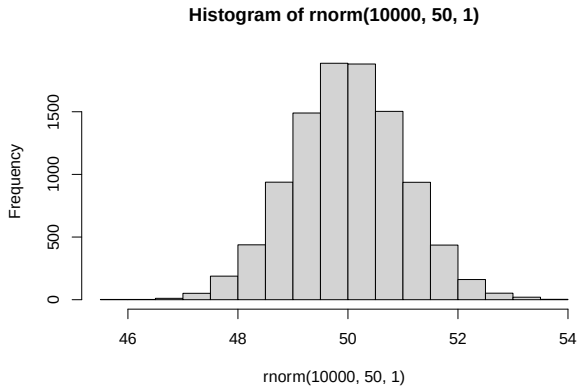
You can visualize and summarize the data - what is the mean (`mean`) and standard deviation (`sd`) of the dataset? What is the *natural logarithm* of the standard deviation of the dataset (`log(sd)`)?

A normal distribution has two parameters to describe its shape - mean (μ) and standard deviation (σ). The formula to generate a normal curve is:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Where p(x) is the probability of observing a value of x given the model. For example, if data is drawn from a normal distribution centered on a mean (μ) = 50 and standard deviation (σ) = 1, the normal distribution looks like this:

```
hist(rnorm(10000, 50, 1))
```



And the probability of observing a value of 60 given this model and these parameters is quite low:

```
# This calculates p(60) using the formula given above
dnorm(60, 50, 1)
```

```
## [1] 7.694599e-23
```

It is of interest to 'fit your data to a normal model'. We can do this very quickly in R - we provide our data `dat` to a command `fitdist` in the R library `fitdistrplus` - it fits your observed data to a normal model and returns estimates of the model parameters (μ and σ):

```
fitdistrplus::fitdist(data = dat$weight_g, distr = "norm")
```

```
## Fitting of the distribution ' norm ' by maximum likelihood
## Parameters:
##       estimate  Std. Error
## mean 3.4242893 0.004553051
## sd   0.1439801 0.003218794
```

How does the `fitdist` model estimates for mean (μ) and standard deviation (σ) compare to the true values used to produce the data? Quite well.

C. Estimating parameters of a normal model using ABC (useful when you have more complex models) Let's learn to use ABC (Approximate Bayesian Computation) to estimate the most likely values of the model parameters given the observe data. This is most useful when (1) you have a very complex model and (2) you don't know the underlying values of the parameters that produced the observed data. ABC uses the following steps:

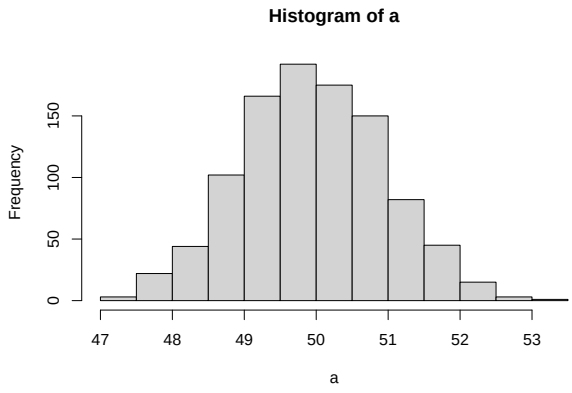
- data is simulated from an underlying model
- summary statistics are calculated from each simulation
- the values of these summary statistics are compared to observed values
- a 'rejection algorithm' is implemented, where simulations with summary statistics that are very far from the observed values are discarded
- the remaining accepted simulations are used to calculate a 'posterior distribution' for underlying model parameters

Let's look at each step more closely:

Step 1. Data is simulated from an underlying model

We stated previously we believe our data is drawn from a normal distribution. We simulate 10000 datasets from a normal model, and we randomly choose parameter values for each simultion from the normal model. We can simulate draws from a normal distribution using the command `rnorm`. Here is a quick example of how to simulate drawing 1000 values from a normal distribution with mean 50, standard deviation = 1:

```
# Randomly draw 1000 values from a normal distribution with
# mean 50, standard deviation 1
a <- rnorm(1000, 50, 1)
hist(a)
```



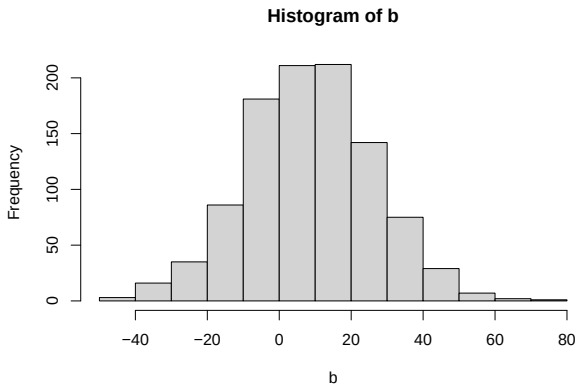
We wish to repeat this process of simulating our insect data (1000 data points) 10,000 times. We do not know the underlying values of the model parameters (μ and σ) that produced our data, so we run our simulations with randomly chosen values for these. We establish a realistic **prior distribution** for each parameter:

```
mu_rand <- runif(10000, -100, 100)  # What is this command doing?
sd_rand <- runif(10000, 0, 20)
```

Create a histogram (`hist`) for each of these - do they look as you would expect?

Now for each set of parameters, we simulate data that's meant to mimic our observed insect data using a normal model. How can we do this? Using the `rnorm` command - we simulate data under a random normal model with parameters μ and σ from each of the 10,000 values we generated above. This works for a single parameter set like this:

```
b <- rnorm(1000, mu_rand[1], sd_rand[1])
hist(b)
```

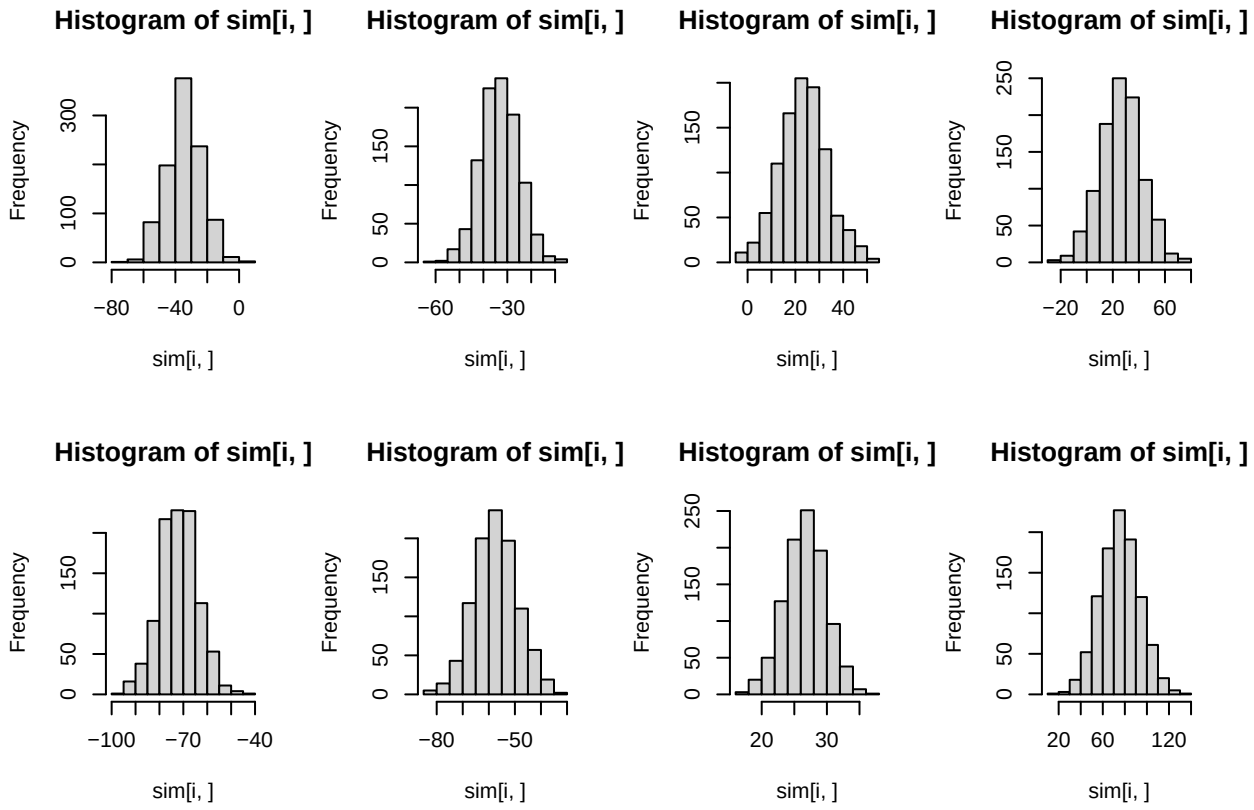


We can do this for all of the parameter sets this way:

```
sim <- array(NA, dim = c(10000, 1000))
for (i in 1:10000) {
    sim[i, ] <- rnorm(1000, mu_rand[i], sd_rand[i])
}
```

Every single dataset is different:

```
d <- sample.int(10000, 8)
par(mfrow = c(2, 4))
for (i in d) {
    hist(sim[i, ])
}
```



Step 2. Summary statistics are calculated from each simulation

Our focal summary statistics are (1) the mean of the dataset and (2) the logarithm of the standard deviation of the dataset. We save those to a new variable called `stat.obs`:

```
stat.obs <- c(mean(dat$weight_g), log(sd(dat$weight_g)))
```

Our simulated summary statistics can be obtained by calculating the mean and log(sd) for each of the 10000 simulations:

```
stat.sim <- array(NA, dim = c(10000, 2), dimnames = list(NULL,
    c("mean", "log_sd")))
for (i in 1:10000) {
    stat.sim[i, 1] <- mean(sim[i, ])
    stat.sim[i, 2] <- log(sd(sim[i, ]))
}
```

Step 3. The values of these summary statistics are compared to observed values (using either a rejection algorithm or a neural network - we use a neural network)

We do this step using the R package `abc`:

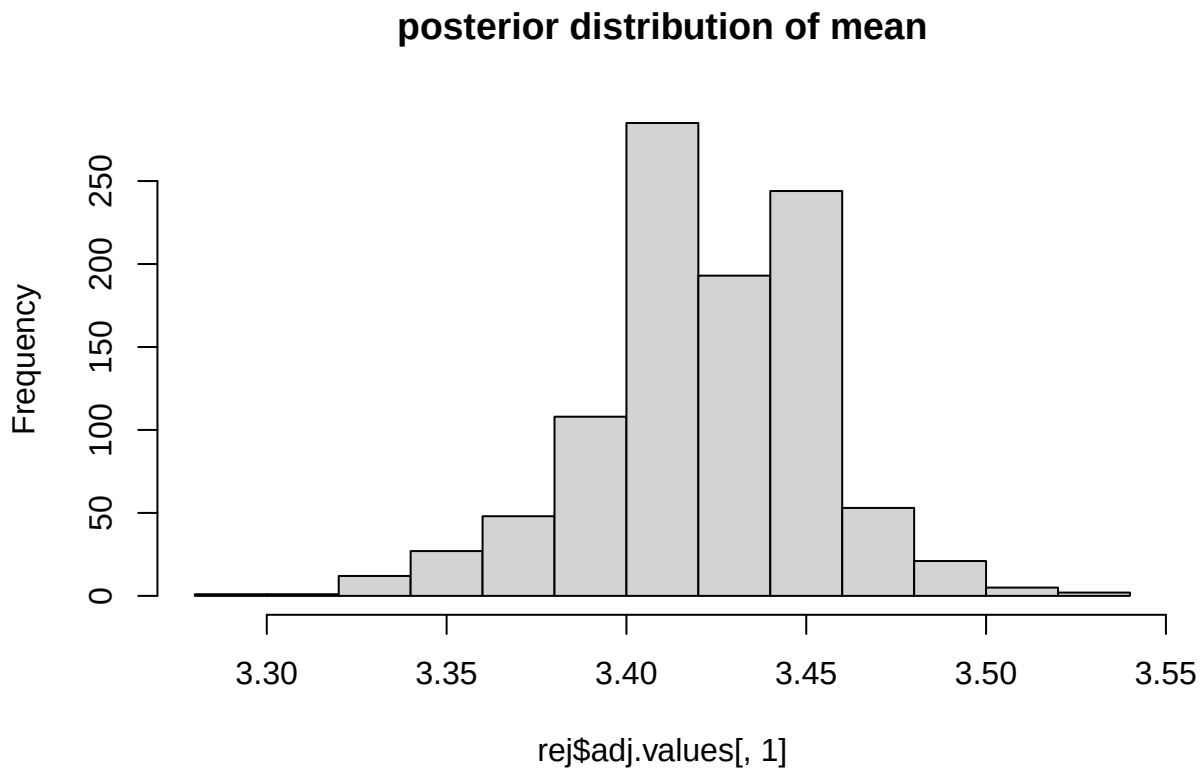
```
par.sim <- cbind(mu_rand, sd_rand)
rej <- abc::abc(target = stat.obs, param = par.sim, sumstat = stat.sim,
    tol = 0.1, method = "neuralnet")
```

```
## 12345678910
## 12345678910
```

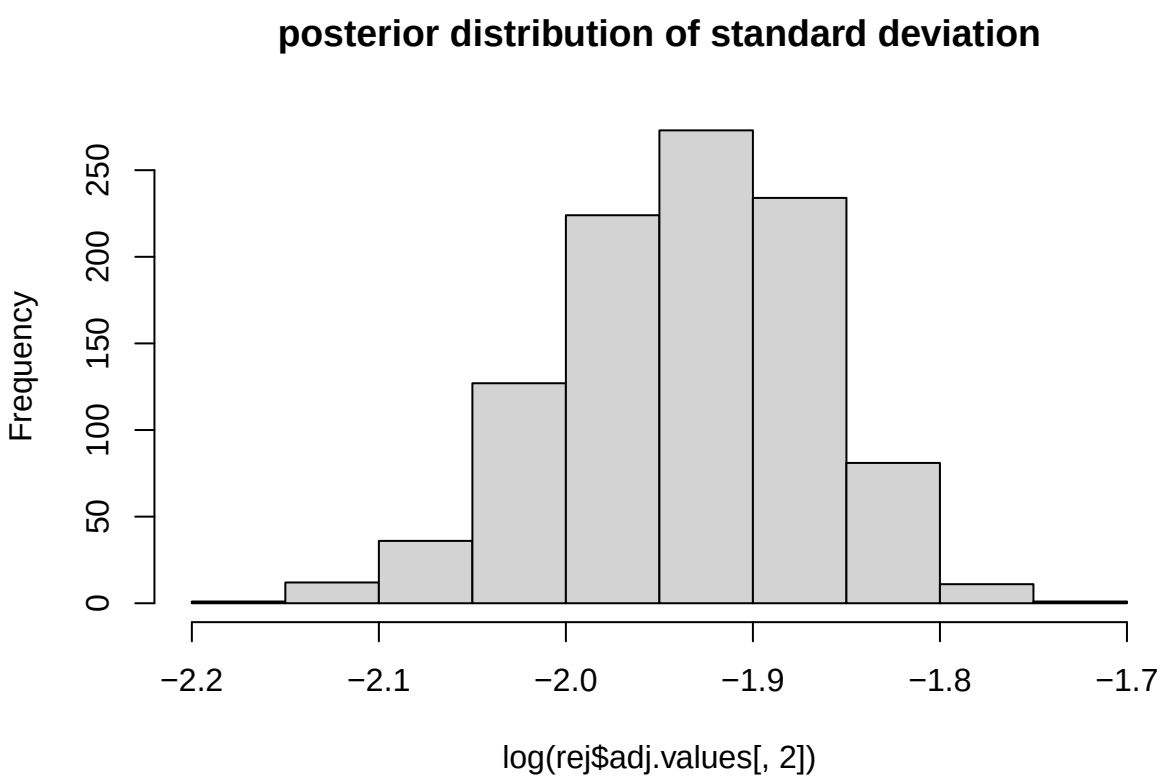
Step 4. The remaining accepted simulations are used to calculate a 'posterior distribution' for underlying model parameters

We can visualize this by plotting the values of the accepted simulations (the ones that are closest to our observed summary statistics):

```
par(mfrow = c(1, 1))
hist(rej$adj.values[, 1], main = "posterior distribution of mean")
```




```
hist(log(rej$adj.values[, 2]), main = "posterior distribution of standard deviation")
```



How well do these compare to the observed values in the original data (`obs.stat`)?

D. Estimating parameters of a discrete-time Lotka Volterra model using ABC Let's repeat using ABC (Approximate Bayesian Computation) to estimate the most likely values of the model parameters given the observe data. We can take Gause's *Paramecium* data and try to estimate the underlying parameters from a discrete-time Lotka Volterra model.

Step 1a. Gather your observed data

```
# load competition data
data("gause_1934_science_f02_03")

# subset out data from species grown in mixture
mixturedat <- gause_1934_science_f02_03[gause_1934_science_f02_03$Treatment ==
    "Mixture", ]

# extract time and species data
time <- mixturedat$Day
species <- data.frame(mixturedat$Volume_Species1, mixturedat$Volume_Species2)
colnames(species) <- c("P_caudatum", "P_aurelia")
```

Step 1b. Arrange your observed data similar to stat.obs above The variable with the observed data, `species`, has dimensions 23, 2. You should arrange the observed data as a vector of length 46. That means, please arrange the data in `species` to be 1 vector of length 46. Save that to a new variable called `stat.obs` Hint: you can try the `reshape2::melt` command, or you can create your own new variable as `stat.obs <- c(species[,1],species[,2])`

```
## SOLUTION ##
stat.obs <- c(species[, 1], species[, 2])
```

Make sure your variable is correct by running this command and getting the same results:

```
length(stat.obs)
```

```
## [1] 46
```

Step 2. Create simulation of data from an underlying model

Our next goal is to use a discrete-time Lotka-Volterra model that can simulate data of the kind observed in Gause's Paramecium. The model should consider the processes - growth and competition - that are important to produce the observed Paramecium time series.

We use the following model:

$$N_{1,t+1} = \frac{N_{1,t}\lambda_1}{1 + \alpha_{11}N_{1,t} + \alpha_{12}N_{2,t}}$$

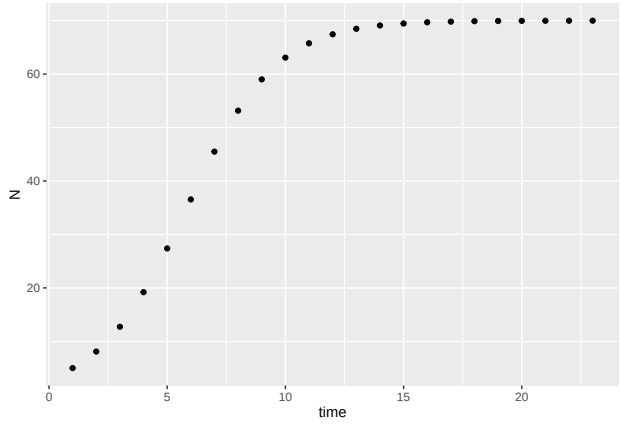
$$N_{2,t+1} = \frac{N_{2,t}\lambda_2}{1 + \alpha_{22}N_{2,t} + \alpha_{21}N_{1,t}}$$

The parameters we need to estimate are:

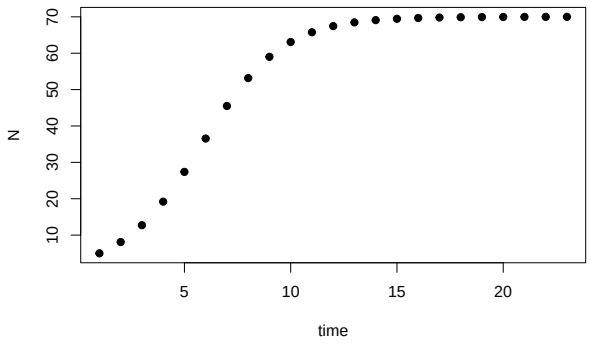
$\lambda_1, \lambda_2, \alpha_{11}, \alpha_{22}, \alpha_{12}, \alpha_{21}$

Please create a simulation of this model. I show here a model for a single species. Please modify that to also simulate growth in a second species. For the missing parameters, you can start by using values of $\lambda_2 = 1.5$, $\alpha_{22} = 0.005$, $\alpha_{12} = 0.03$, $\alpha_{21} = 0.007$, $N_{2,0} = 3$. Change the name of the function to `disc_lv`.

```
# Parameter values to use for simulation
lambda_1 <- 1.7
alpha_11 <- 0.01
N1_0 <- 5
t <- 23
# model function
disc_log <- function(lambda_1, N1_0, alpha_11) {
    Nt1 <- (N1_0 * lambda_1)/(1 + alpha_11 * N1_0)
    return(Nt1)
}
# Simulation of model for t time steps
N <- rep(NA, t)
N[1] <- N1_0
for (i in 2:t) {
    N[i] <- disc_log(lambda_1, N1_0, alpha_11)
    N1_0 <- N[i]
}
# Plot simulation: ggplot
dat <- as.data.frame(N)
dat$time <- as.numeric(rownames(dat))
ggplot2::ggplot(dat, ggplot2::aes(time, N)) + ggplot2::geom_point()
```

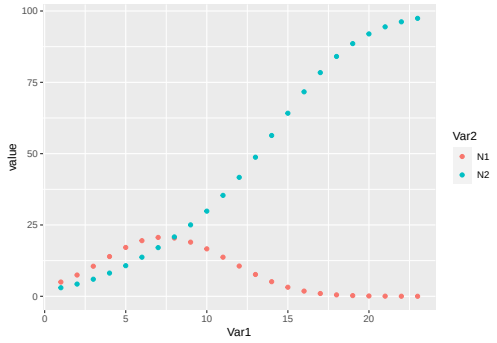



```
# Plot simulation: base R
plot(N, xlab = "time", ylab = "N", pch = 19, col = "black")
```

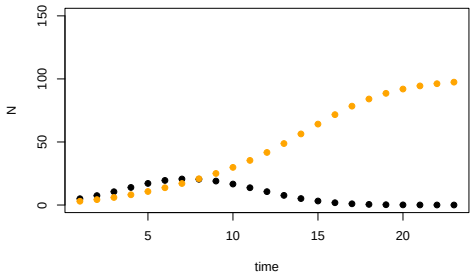



```
## SOLUTION ## Parameter values to use for simulation
lambda_1 <- 1.7
lambda_2 <- 1.5
alpha_11 <- 0.01
alpha_22 <- 0.005
alpha_12 <- 0.03
alpha_21 <- 0.007
N1_0 <- 5
N2_0 <- 3
t <- 23
# model function
disc_lv <- function(lambda_1, lambda_2, N1_0, N2_0, alpha_11,
    alpha_22, alpha_12, alpha_21) {
    Nt1 <- (N1_0 * lambda_1)/(1 + alpha_11 * N1_0 + alpha_12 *
        N2_0)
    Nt2 <- (N2_0 * lambda_2)/(1 + alpha_22 * N2_0 + alpha_21 *
        N1_0)
    return(cbind(Nt1, Nt2))
}
# Simulation of model for t time steps
N <- array(NA, dim = c(t, 2), dimnames = list(NULL, c("N1", "N2")))
```

```r
N[1, 1] <- N1_0
N[1, 2] <- N2_0
for (i in 2:t) {
    N[i, ] <- disc_lv(lambda_1, lambda_2, N1_0, N2_0, alpha_11,
        alpha_22, alpha_12, alpha_21)
    N1_0 <- N[i, 1]
    N2_0 <- N[i, 2]
}
# Plot simulation: ggplot
dat <- reshape2::melt(N)
ggplot2::ggplot(dat, ggplot2::aes(x = Var1, y = value, col = Var2)) +
    ggplot2::geom_point()
```




```r
# Plot simulation: base R
plot(N[, 1], xlab = "time", ylab = "N", pch = 19, col = "black",
    ylim = c(0, 150))
points(N[, 2], pch = 19, col = "orange")
```

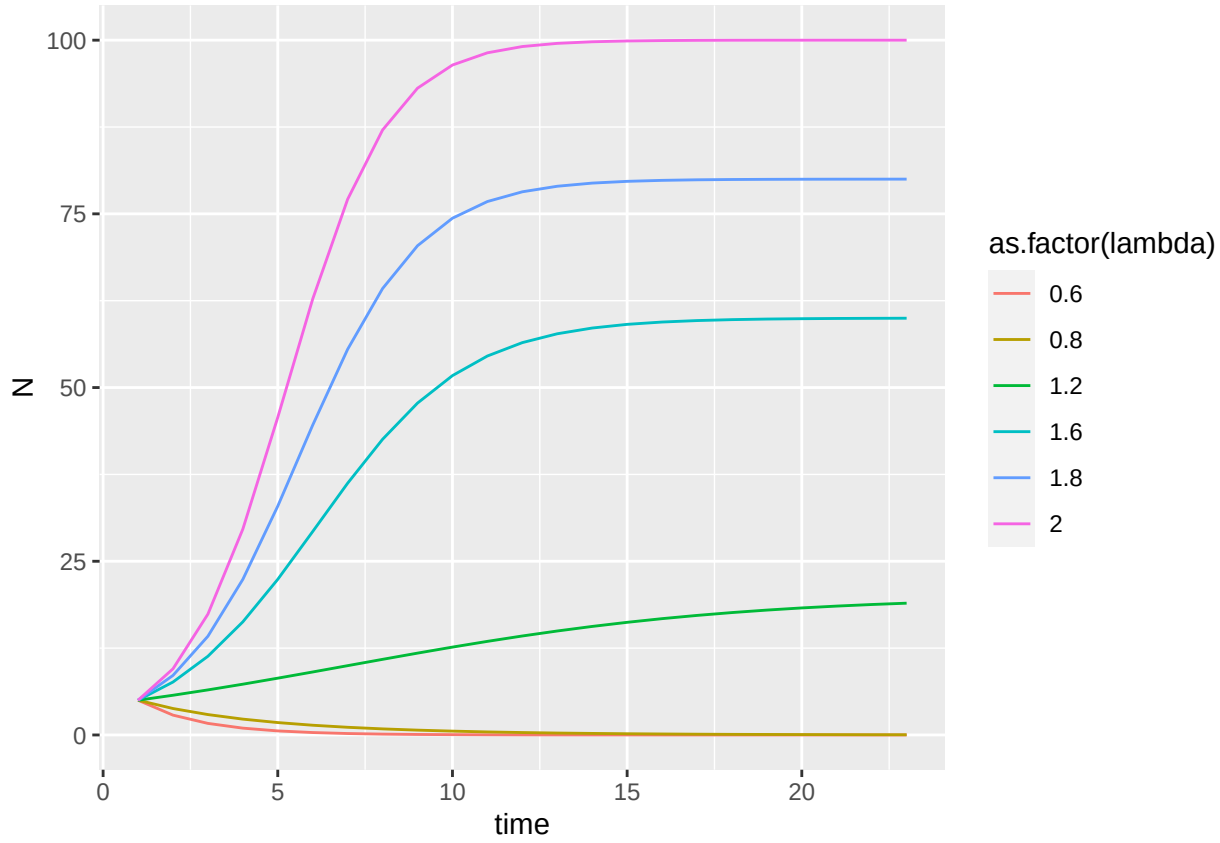


Step 3. Create random simulations of data, using prior distributions for model parameters

Do you recall an exercise where we looked at logistic growth across a range of values for the growth rate λ? We used code that looks like this (using `disc_log` from above):

```r
# Simulation of model for t time steps
lambda_1_range <- c(0.6, 0.8, 1.2, 1.6, 1.8, 2)
N.g <- numeric()
N1_0 <- 5
for (p in 1:length(lambda_1_range)) {
    N <- rep(NA, t)
    N[1] <- N1_0
```

```
    for (i in 2:t) {
        N[i] <- disc_log(lambda_1_range[p], N1_0, alpha_11)
        N1_0 <- N[i]
    }
    dat <- as.data.frame(N)
    dat$time <- as.numeric(rownames(dat))
    dat$lambda <- rep(lambda_1_range[p], t)
    N.g <- rbind(N.g, dat)
    N1_0 <- 5
}
# Plot in ggplot
ggplot2::ggplot(N.g, ggplot2::aes(x = time, y = N, col = as.factor(lambda))) +
    ggplot2::geom_line()
```



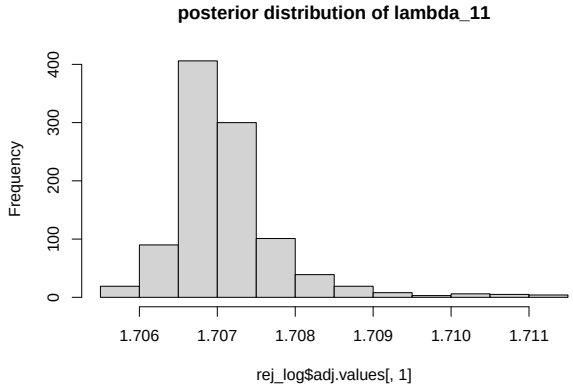
In this exercise, you will choose a *prior distribution* for each of the model parameters to estimate, and you will use a random draw from that prior distribution for each repetition of the simulation. This is how we create the variables `par.sim` and `stat.sim` needed to use ABC, to estimate the unknown parameter values for Gause's Paramecium. I demonstrate how this works for a discrete-time logistic example, using data from Gause's Paramecium.

```
## 1. Get observed data load competition data
data("gause_1934_science_f02_03")
# subset out data from species grown alone (not in mixtire)
alone_dat <- gause_1934_science_f02_03[gause_1934_science_f02_03$Treatment ==
    "Pc", ]
# extract time and species data
```

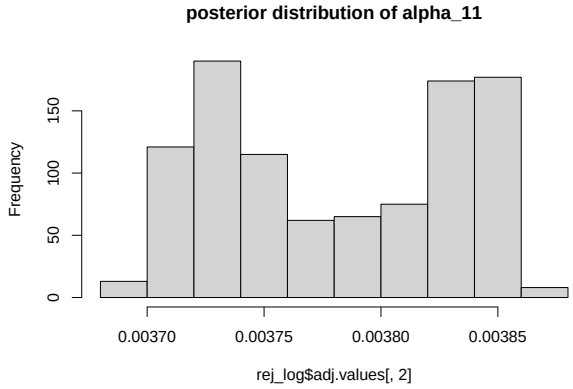
```r
time <- alone_dat$Day
species <- data.frame(alone_dat$Volume_Species1)
colnames(species) <- "P_caudatum"

## 2. Arrange observed data for ABC
stat.obs <- species$P_caudatum

## 3. Create a *random* simulation of disc_log, with draws
## from prior distributions for all model parameters. Save
## the time series values and values for the parameters in
## variables par.sim and stat.sim for ABC.
rand_disc_log <- function(N1_0 = N1_0, disc_log, t) {
    ## Choose random values of lambda_1 from a prior
    ## distribution
    lambda_1r <- runif(1, min = 1.2, max = 3)  # Random uniform distribution between 0-3
    alpha_11r <- runif(1, min = -0.5, max = 0.5)  # Random uniform distribution, bounded -0.5 - 0.5
    # biased towards low values
    N <- rep(NA, t)
    N[1] <- N1_0
    for (i in 2:t) {
        N[i] <- disc_log(lambda_1r, N1_0 = N1_0, alpha_11r)
        N1_0 <- N[i]
    }
    return(list(N = N, lambda_1r = lambda_1r, alpha_11r = alpha_11r))
}

## 4. Run this function 1000 times, and save all of the
## time series and the associated parameter values for
## later ABC
num_iter <- 1e+06
num_par <- 2  # to estimate: lambda_11 and alpha_11
par_name <- c("lambda_11", "alpha_11")
par.sim <- array(NA, dim = c(num_iter, num_par), dimnames = list(NULL,
    par_name))  # to save values of parameters
stat.sim <- array(NA, dim = c(num_iter, t))  # to save values of N over time
colnames(stat.sim) <- paste("S", 1:t, sep = "")
for (k in 1:num_iter) {
    N1_0 <- species$P_caudatum[1]
    simul <- rand_disc_log(N1_0, disc_log, t)
    par.sim[k, 1] <- simul$lambda_1r
    par.sim[k, 2] <- simul$alpha_11r
    stat.sim[k, ] <- simul$N
}

## 5. Use ABC to estimate the parameter values in the
## discrete-time growth model
colnames(stat.sim) <- paste("S", 1:23, sep = "")
rej_log <- abc::abc(target = stat.obs, param = par.sim, sumstat = stat.sim,
    tol = 0.001, method = "neuralnet")
```

```
## 12345678910
## 12345678910
```

```r
## 6. Visualize the posterior distributions of the model
## parameters
par(mfrow = c(1, 1))
hist(rej_log$adj.values[, 1], main = "posterior distribution of lambda_11")
```

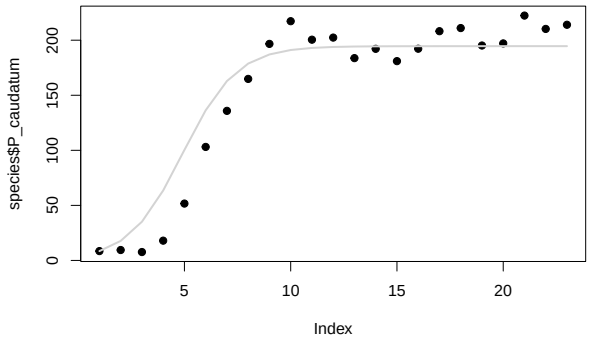



```r
hist(rej_log$adj.values[, 2], main = "posterior distribution of alpha_11")
```




```r
## 7. Add a curve to the observed data that shows the
## ABC-predicted parameter values 7a. Simulate data with
## the average 'accepted' parameter values
lambda_1 <- mean(rej_log$unadj.values[, 1])
alpha_11 <- mean(rej_log$unadj.values[, 2])
N1_0 <- species$P_caudatum[1]
t <- 23
# model function
disc_log <- function(lambda_1, N1_0, alpha_11) {
    Nt1 <- (N1_0 * lambda_1)/(1 + alpha_11 * N1_0)
    return(Nt1)
}
# Simulation of model for t time steps
N <- rep(NA, t)
N[1] <- N1_0
for (i in 2:t) {
    N[i] <- disc_log(lambda_1, N1_0, alpha_11)
```

```
    N1_0 <- N[i]
}
plot(species$P_caudatum, pch = 19, col = "black")
lines(N, lwd = 2, col = "lightgray")
```



Question: In this example, what are the prior distributions for the lambda and alpha parameters?

Question: How do the posterior parameter distributions differ from the prior distributions? Create a plot to show this.

Question: How does the predicted values of P. caudatum N using the posterior mean compare to the observed values?

Question: Can you implement a different prior distribution for alpha, using rnorm(0,0.01) ?

Your goal is to repeat this ABC procedure and estimate the most likely values for the 2 Paramecium species from the dataset in Step 1a. The code above is your 'template' but using only 1 species. Add in the model and parameters for Species #2, and be sure to add in draws of parameter values from the prior distributions for those new parameters.

```
### SOLUTION ### 1. Get observed data load competition data
data("gause_1934_science_f02_03")
# subset out data from species grown in mixture
mixturedat <- gause_1934_science_f02_03[gause_1934_science_f02_03$Treatment ==
    "Mixture", ]
# extract time and species data
time <- mixturedat$Day
species <- data.frame(mixturedat$Volume_Species1, mixturedat$Volume_Species2)
colnames(species) <- c("P_caudatum", "P_aurelia")

## 2. Arrange observed data for ABC
stat.obs <- c(species[, 1], species[, 2])

## 3. Create a *random* simulation of disc_log, with draws
## from prior distributions for all model parameters. Save
## the time series values and values for the parameters in
## variables par.sim and stat.sim for ABC.
rand_disc_lv <- function(N1_0 = N1_0, N2_0 = N2_0, disc_lv, t) {
    ## Choose random values of lambda_1 from a prior
    ## distribution
```

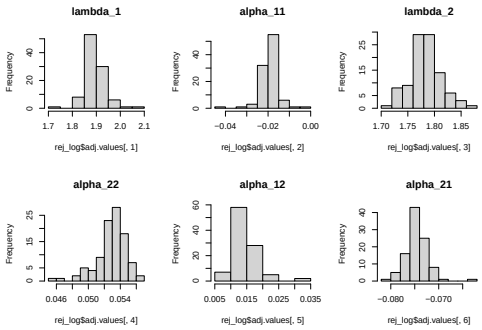
```
    lambda_1r <- runif(1, min = 1, max = 2)  # Random uniform distribution between 1-2
    lambda_2r <- runif(1, min = 1, max = 2)
    z = rbinom(1, 1, 0.5)
    alpha_11r <- z * -rbeta(1, shape1 = 1, shape2 = 50) + (1 -
        z) * rbeta(1, shape1 = 1, shape2 = 50)  # Random beta distribution
    z = rbinom(1, 1, 0.5)
    alpha_12r <- z * -rbeta(1, shape1 = 1, shape2 = 50) + (1 -
        z) * rbeta(1, shape1 = 1, shape2 = 50)  # Random beta distribution
    z = rbinom(1, 1, 0.5)
    alpha_21r <- z * -rbeta(1, shape1 = 1, shape2 = 50) + (1 -
        z) * rbeta(1, shape1 = 1, shape2 = 50)  # Random beta distribution
    z = rbinom(1, 1, 0.5)
    alpha_22r <- z * -rbeta(1, shape1 = 1, shape2 = 50) + (1 -
        z) * rbeta(1, shape1 = 1, shape2 = 50)  # Random beta distribution
    ## Initial population sizes
    N1_0 <- species$P_caudatum[1]
    N2_0 <- species$P_aurelia[1]
    t <- 23
    # Simulation of model for t time steps
    N <- array(NA, dim = c(t, 2), dimnames = list(NULL, c("N1",
        "N2")))
    N[1, 1] <- N1_0
    N[1, 2] <- N2_0
    for (i in 2:t) {
        N[i, ] <- disc_lv(lambda_1r, lambda_2r, N1_0, N2_0, alpha_11r,
            alpha_22r, alpha_12r, alpha_21r)
        N1_0 <- N[i, 1]
        N2_0 <- N[i, 2]
    }
    return(list(N = N, lambda_1r = lambda_1r, alpha_11r = alpha_11r,
        lambda_2r = lambda_2r, alpha_22r = alpha_22r, alpha_12r = alpha_12r,
        alpha_21r = alpha_21r))
}

## 4. Run this function 1000 times, and save all of the
## time series and the associated parameter values for
## later ABC
num_iter <- 1e+05
num_par <- 6  # to estimate: lambda_1, alpha_11, lambda_2, alpha_22, alpha_12, alpha_21
par_name <- c("lambda_11", "alpha_11", "lambda_2", "alpha_22",
    "alpha_12", "alpha_21")
par.sim <- array(NA, dim = c(num_iter, num_par), dimnames = list(NULL,
    par_name))  # to save values of parameters
stat.sim <- array(NA, dim = c(num_iter, (23 * 2)))  # to save values of N over time
colnames(stat.sim) <- c(paste("S1_", 1:23, sep = ""), paste("S2_",
    1:23, sep = ""))
for (k in 1:num_iter) {
    N1_0 <- species$P_caudatum[1]
    N2_0 <- species$P_aurelia[1]
    simul <- rand_disc_lv(N1_0, N2_0, disc_lv, t)
    par.sim[k, 1] <- simul$lambda_1r
    par.sim[k, 2] <- simul$alpha_11r
    par.sim[k, 3] <- simul$lambda_2r
```

```
    par.sim[k, 4] <- simul$alpha_22r
    par.sim[k, 5] <- simul$alpha_12r
    par.sim[k, 6] <- simul$alpha_21r
    # stopped
    stat.sim[k, ] <- c(simul$N[, 1], simul$N[, 2])
}

## 5. Use ABC to estimate the parameter values in the
## discrete-time growth model
colnames(stat.sim) <- c(paste("S1_", 1:23, sep = ""), paste("S2_",
    1:23, sep = ""))
rej_log <- abc::abc(target = stat.obs, param = par.sim, sumstat = stat.sim,
    tol = 0.001, method = "neuralnet")
```

```
## Warning: All parameters are "none" transformed.
```

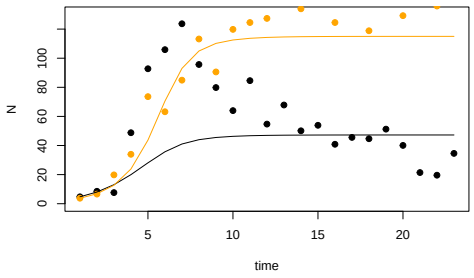
```
## 12345678910
## 12345678910
```

```
## 6. Visualize the posterior distributions of the model
## parameters
par(mfrow = c(2, 3))
hist(rej_log$adj.values[, 1], main = "lambda_1")
hist(rej_log$adj.values[, 2], main = "alpha_11")
hist(rej_log$adj.values[, 3], main = "lambda_2")
hist(rej_log$adj.values[, 4], main = "alpha_22")
hist(rej_log$adj.values[, 5], main = "alpha_12")
hist(rej_log$adj.values[, 6], main = "alpha_21")
```

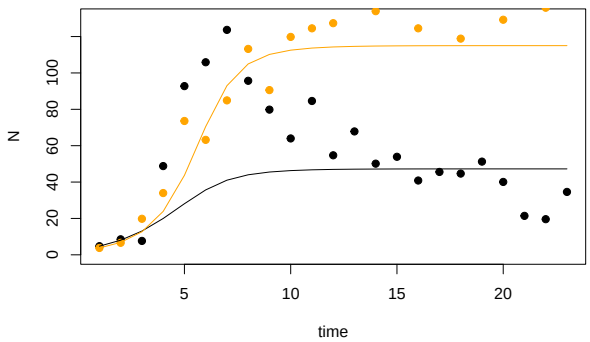



```
## 7. Add a curve to the observed data that shows the
## ABC-predicted parameter values 7a. Simulate data with
## the average 'accepted' parameter values
lambda_1 <- mean(rej_log$unadj.values[, 1])
alpha_11 <- mean(rej_log$unadj.values[, 2])
lambda_2 <- mean(rej_log$unadj.values[, 3])
alpha_22 <- mean(rej_log$unadj.values[, 4])
alpha_12 <- mean(rej_log$unadj.values[, 5])
alpha_21 <- mean(rej_log$unadj.values[, 6])
## Initial population sizes
N1_0 <- species$P_caudatum[1]
```

```r
N2_0 <- species$P_aurelia[1]
t <- 23
# Simulation of model for t time steps
N <- array(NA, dim = c(t, 2), dimnames = list(NULL, c("N1", "N2")))
N[1, 1] <- N1_0
N[1, 2] <- N2_0
for (i in 2:t) {
    N[i, ] <- disc_lv(lambda_1, lambda_2, N1_0, N2_0, alpha_11,
        alpha_22, alpha_12, alpha_21)
    N1_0 <- N[i, 1]
    N2_0 <- N[i, 2]
}
# # Plot simulation: ggplot dat <- reshape2::melt(N)
# ggplot2::ggplot(dat,ggplot2::aes(x=Var1,y=value,col=Var2))
# + ggplot2::geom_point() Plot simulation: base R
par(mfrow = c(1, 1), mar = c(5, 4, 4, 2) + 0.1)
plot(species$P_caudatum, xlab = "time", ylab = "N", pch = 19,
    col = "black", ylim = c(0, 130))
points(species$P_aurelia, pch = 19, col = "orange")
# Add predicted from mean ABC values
# plot(N[,1],xlab='time',ylab='N',pch=19,col='black',ylim=c(0,150))
# points(N[,2],pch=19,col='orange')
lines(y = N[, 1], x = 1:23, col = "black")
lines(y = N[, 2], x = 1:23, col = "orange")
```



I give an example here of what the model-estimated and observed data look like.



If you are struggling to get a good fit of the model to the data, you can try:

- Narrowing the range of the prior distributions
- I used the following prior for my α coefficients:

```
z <- rbinom(1, 1, 0.5)
alpha <- z * -rbeta(1, shape1 = 1, shape2 = 50) + (1 - z) * rbeta(1,
    shape1 = 1, shape2 = 50)
```

Note this gives both negative and positive values drawn from a beta distribution.

- You can try running more iterations of the simulation. I used 100000.
- You can work with either the adjusted or unadjusted values from the ABC. The unadjusted values arise from using a **rejection method** to converge on parameter value estimates that best fit the data. The adjusted values arise from a **neural network** method. Sometimes one method works better than another. You can index those values this way:

```
rej_log <- abc::abc(target = stat.obs, param = par.sim, sumstat = stat.sim,
    tol = 0.001, method = "neuralnet")
# Adjusted values - from the neural network method
rej_log$adj.values
# Unadjusted values - from the rejection method
rej_log$unadj.values
```