Walkthrough of Pedigree-Informed Abundance Estimation

Timothy R. Frasier

May 5, 2021

1 Getting Started

All of these analyses are conducted in R (via RStudio), and using rstan (for the Bayesian parts) and ggplot2 (for visualization). Therefore, all of these need to be installed prior to conducting these analyses. Appropriate links are below.

1. R
2. RStudio
3. R Packages (can be installed by opening RStudio, and then selecting **Tools→Install Packages**, and then typing in the package names):
   (a) rstan (see documentation here)
   (b) ggplot2

For all of these commands to work (as-is), you need to set R's working directory to the **code** folder here!

2 Simulations & Validation

To test the performance of this approach we developed a simulation script in R called `abundanceSIM.R`. This is located in the **code** folder. This script has one function, called `abundanceSim` that requires three arguments to run:

1. `N` - the true population size
2. `propReproductive` - the proportion of N that are reproductive (e.g., 0.25, 0.5, etc.)
3. `propSampled` - the proportion of N that are sampled (e.g., 0.25, 0.5, etc.)

The function then goes through three main steps.

Step 1: Generate individuals and families. First, the function generates offspring. The number of offspring generated is $((N \times propReproductive)/2)$. Therefore, it makes the assumption that each mating pair has just one offspring at a time. Then, moms and dads are generated such that each offspring has one mom and one dad. Then, non-reproductive individuals are created so that the sum of all offspring, moms, dads, and non-reproductives equals N.

Step 2: Randomly sample this population. Next, the function randomizes these individuals, and then randomly samples them based on the `propSampled` argument provided by the user.

Step 3: Collect data on sampled individuals. Next, the function identifies which of the sampled individuals are offspring, moms, dads, and non-reproductives. These are used to generate the quantities needed for pedigree-informed abundance estimation. Specifically:

1. N_s (the number of sampled individuals) is just counted.
2. The number of breeders sampled (B_s) is calculated based on: (2 $\times$ the number of mother-offspring-father triads sampled) + (the number of mother-offspring dyads sampled) + (the number of father-offspring dyads sampled).
3. The number of inferred individuals (N_{in}) is calculated as: (the number of mother-offspring dyads sampled—because the father can be inferred) + (the number of father-offspring dyads sampled—because the mother can be inferred) + (2 $\times$ the number of sampled offspring without any sampled parents—because both parents can be inferred)

The function then returns these three values: N_s, N_{in}, and B_s.

2.1 Simple Example

Let's perform a simple example. Let's run the function setting `N` to 500 individuals, `propReproductive` to 0.4, and `propSampled` to 0.6. The command would look like this:

Before we run any commands we must "source" the script so that the functions within are available to R. You only need to do this when you start a new R session, and do not need to do this each time you want to run the simulation function.

```
source("abundanceSIM.R")
```

Then we can run the function.

```
test = abundanceSim(N = 500, propReproductive = 0.4, propSampled = 0.6)
```

If we look at `test`, we will see what the resulting values are for the parameters needed for pedigree-informed abundance estimation (N_s, N_{in}, and B_s, respectively). The values that you get may be slightly different, because of the stochastic nature of the simulations.

```
test
[1] 300 50  70
```

We can then see how the new methods performs by comparing the estimated values for these parameters to the true ones.

First, "source" the code with the `pedigreeAbundance.R` script, which contains all of the necessary code.

```
source("pedigreeAbundance.R")
```

Then, run the scripts. This function requires three arguments, and has an optional fourth one, as follows:

1. `Ns` - the number of sampled individuals
2. `Nin` - the number of individuals inferred
3. `Bs` - the number of breeders sampled
4. `mode` (optional) - the default is `"singular"`, which means that you are analyzing just one data set at a time. With this option the scripts will print to the screen the posterior distributions for all of the parameters:
   (a) P_s - the probability of being sampled
   (b) P_{ns} - the probability of not being sampled
   (c) P_{bd} - the probability of breeding
   (d) P_{nb} - the probability of not breeding
   (e) N_{ns} - the number of individuals that are not sampled
   (f) N_{nsnb} - the number of non-sampled non-breeders
   (g) N_{bns} - the number of breeders that are not sampled
   (h) N_{bnsni} - the number of breeders that are not sampled nor inferred
   (i) N - the total number of individuals in the population
   The output when run under `mode = "singular"` will be a list where each element is the entire posterior distribution for each of the following parameters, in this order: P_{ns}, P_{bd}, N_{ns}, N_{nsnb}, N_{bns}, N_{bnsni}, and N. This way, users can do what they want with these posterior distributions. However, sometimes it is desirable to analyze multiple data sets at once, and therefore it is not desirable to plot all results to the screen, or to save the entire posterior distributions. In these situations you can set `mode = "multiple"` and this will change two things. First, the posterior distributions will not be printed to the screen. Second, instead of being the entire posterior distributions for the parameters listed above (in the in-line list, not parameters (a) though (i)), the output will be one row of data containing the mode, lower 95% highest density interval (HDI) value, and upper 95% HDI value for each of these parameters. This way, the results from multiple runs can be combined into a single data frame (see section 2.2 below).

We will run the script for this single data set. Note that this will take a while, because the Stan code will need to be compiled, and also run. So... please me patient!

```
test = pedigreeAbundance(Ns = 300, Nin = 50, Bs = 70)
```

The method performed very well! Below are the plots of the estimates for each of these three parameters, and we can see that they capture the true values of $N = 500$, $P_{ns} = 0.4$, and $P_{bd} = 0.4$.

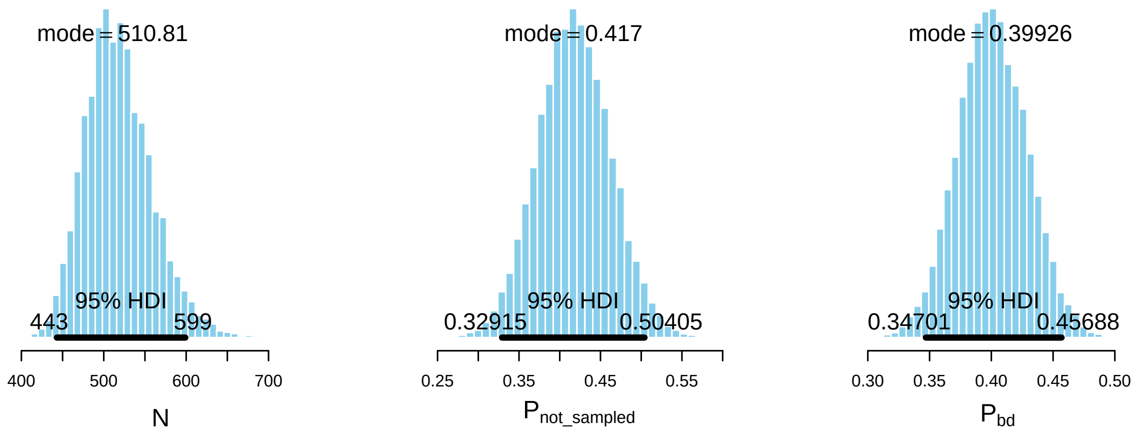


Figure 1: Posterior distributions for the estimates for the known (user-determined) parameters that were used in the simulation function.

2.2 Effects of Sampling

To create one of the figures from the paper (looking at the effects of sampling a different proportion of individuals) we need to generate many data sets. Specifically, for this analysis we want to keep `N` constant (here at 1,000 individuals), and `propReproductive` constant (here at 0.4), but choose different values for `propSampled`). Specifically, we will use 0.2, 0.4, 0.6, and 0.8. Moreover, we want to conduct multiple (100) iterations of each scenario. We can do this with the code below. What we are doing is conducting 100 simulations under each scenario (each value for `propSampled`), and then combining them all together.

```
iterations = 100
simResults_Sampled1 = matrix(NA, nrow = iterations, ncol = 3)
for (i in 1:iterations) {
    simResults_Sampled1[i, ] = abundanceSim(N = 1000, propReproductive = 0.4,
        propSampled = 0.2)
}

simResults_Sampled2 = matrix(NA, nrow = iterations, ncol = 3)
for (i in 1:iterations) {
    simResults_Sampled2[i, ] = abundanceSim(N = 1000, propReproductive = 0.4,
        propSampled = 0.4)
}

simResults_Sampled3 = matrix(NA, nrow = iterations, ncol = 3)
for (i in 1:iterations) {
    simResults_Sampled3[i, ] = abundanceSim(N = 1000, propReproductive = 0.4,
        propSampled = 0.6)
}

simResults_Sampled4 = matrix(NA, nrow = iterations, ncol = 3)
for (i in 1:iterations) {
    simResults_Sampled4[i, ] = abundanceSim(N = 1000, propReproductive = 0.4,
        propSampled = 0.8)
}

simResults_Sampled = data.frame(rbind(simResults_Sampled1, simResults_Sampled2,
    simResults_Sampled3, simResults_Sampled4))
colnames(simResults_Sampled) = c("Ns", "Nin", "Bs")
```

The results from when we ran these simulations are in the "results" folder, in a file named "`simulationResults.csv`".

Now, we want to analyze all of these data sets together. Here, we will take advantage of the `mode = "multiple"` argument in the **pedigreeAbundance** function, as described in section 2.1. This will take quite a while: Stan is estimating each parameter for each of the 400 data sets, so you should go get a coffee and perhaps check your email, twitter, or news feed while this is running.

```
nSteps = length(simResults_Sampled[, 1])
simSampled_Estimates = matrix(NA, nrow = nSteps, ncol = 21)
for (i in 1:nSteps) {
    simSampled_Estimates[i, ] = pedigreeAbundance(Ns = simResults_Sampled[i, 1], Nin =
         simResults_Sampled[i, 2], Bs = simResults_Sampled[i, 3], mode = "multiple")
}
```

Let's first look at the uncertainty around abundance estimates (N) as the proportion of sampled individuals changes. To do this, let's first summarize the mode, low HDI, and HDI for each value of `propSampled` (we know that there are 100 of each, and what order they are in). First, let's make a vector containing the different values for the proportion of individuals sampled.

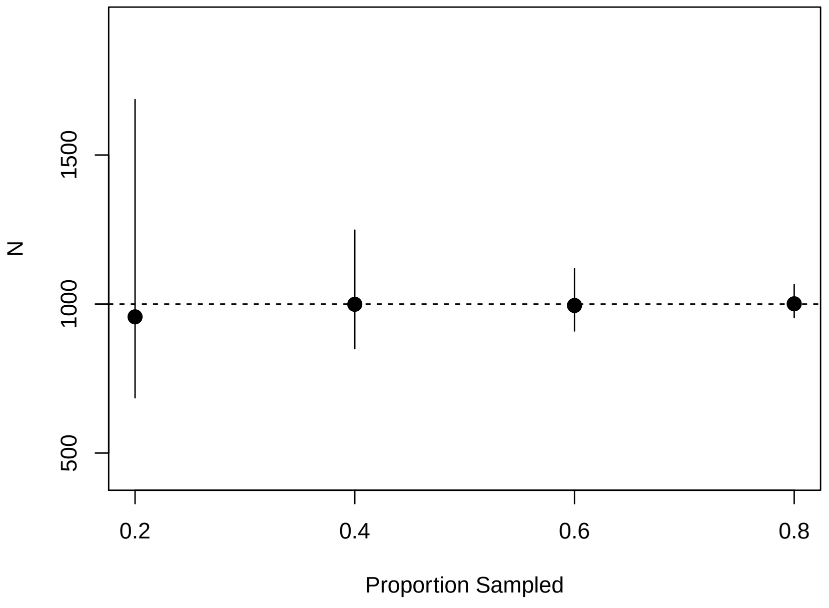
```
propSampled = c(0.2, 0.4, 0.6, 0.8)
```

Then, we'll get the mean values for the the parameters we're interested in.

```
N = c(mean(simSampled_Estimates[1:100, 19]), mean(simSampled_Estimates[101:200, 19]),
    mean(simSampled_Estimates[201:300, 19]), mean(simSampled_Estimates[301:400, 19]))

lowHDI = c(mean(simSampled_Estimates[1:100, 20]), mean(simSampled_Estimates[101:200,
    20]), mean(simSampled_Estimates[201:300, 20]), mean(simSampled_Estimates[301:400,
    20]))

highHDI = c(mean(simSampled_Estimates[1:100, 21]), mean(simSampled_Estimates[101:200,
    21]), mean(simSampled_Estimates[201:300, 21]), mean(simSampled_Estimates[301:400,
    21]))
```

Now we can plot these value in a way that provides information on the precision and accuracy of our abundance estimation based on this analysis.

```
#---------------------------------#
# Plot the data, customizing:
# 1. The plot character & size (pch = 16 and cex = 1.5)
# 2. The upper and lower limits for the y-axis (ylim = c(min(lowHDI) - 250, max(
    highHDI) + 250))
# 3. The x-axis. Do this by not plotting here, and customizing it in the second line
#---------------------------------#
plot(N ~ propSampled, pch = 16, cex = 1.5, ylim = c(min(lowHDI) - 250, max(highHDI) +
    250), xaxt = "n", xlab = "Proportion Sampled")
axis(1, at = propSampled, labels = propSampled)

#------------------------------------------------------#
# Add a dashed line indicating the true population size #
#------------------------------------------------------#
abline(h = 1000, lty = "dashed")

#------------------------------------------------------------------------#
# Add line segments to each point indicating the low and high HDI values #
#------------------------------------------------------------------------#
for (i in 1:4) {
    segments(x0 = propSampled[i], y0 = lowHDI[i], x1 = propSampled[i], y1 = highHDI[i
        ])
}
```



From this, we can see that this method results in accurate (i.e., not biased) estimates of abundance, regardless of the proportion of individuals that are sampled. However, as expected the precision of these estimates increases as the proportion of sampled individuals increases.

We can also use these data to see how this approach performs in estimating other aspects of the population, such as the probability of being sampled P_s and the probability of being a breeder (P_{bd}). These are the two main parameters estimates from the data, with the rest being obtained based on algebraic treatment of these in association with direct counts (e.g., N_s, B_s).

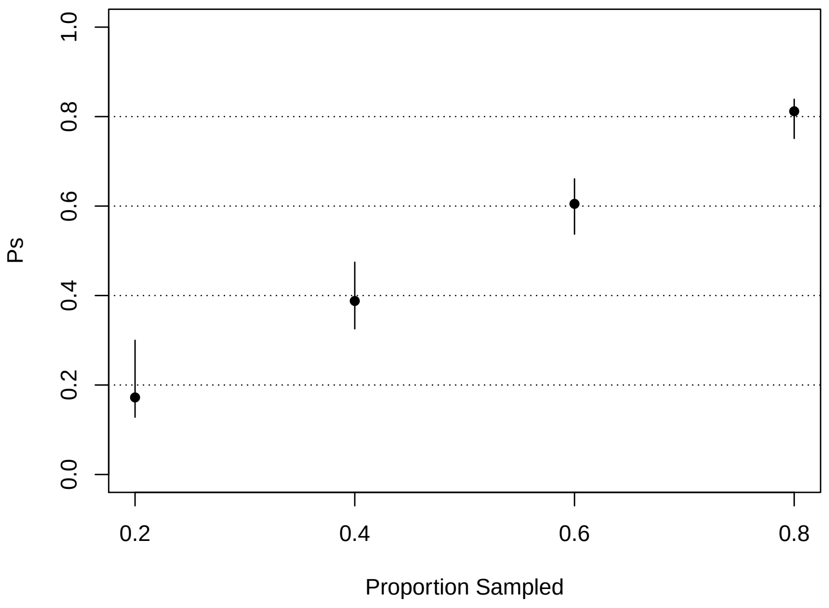
Similar to our estimates of N, we can see how our estimates of the probability of being sampled (P_s) vary in association with the proportion of individuals sampled. Note that the code returns the probability of not being sampled, which can be converted to the probability of being sampled by subtracting it from 1.

First, we can organize the data, as we did for N, just selecting the appropriate columns from the returned data (in this case it is columns 1-3 for the mode, lower HDI, and upper HDI).

```
Ps = c(1 - mean(simSampled_Estimates[1:100, 1]), 1 - mean(simSampled_Estimates
    [101:200, 1]), 1 - mean(simSampled_Estimates[201:300, 1]), 1 - mean(simSampled_
    Estimates[301:400, 1]))

highHDI = c(1 - mean(simSampled_Estimates[1:100, 2]), 1 - mean(simSampled_Estimates
    [101:200, 2]), 1 - mean(simSampled_Estimates[201:300, 2]), 1 - mean(simSampled_
    Estimates[301:400, 2]))

lowHDI = c(1 - mean(simSampled_Estimates[1:100, 3]), 1 - mean(simSampled_Estimates
    [101:200, 3]), 1 - mean(simSampled_Estimates[201:300, 3]), 1 - mean(simSampled_
    Estimates[301:400, 3]))
```

Then, we can plot these results.

```
#———————————————————————————————#
# Plot the data, customizing:
# 1. The plot character (pch = 16)
# 2. The upper and lower limits for the y—axis (ylim = c(0, 1))
# 3. The x—axis. Do this by not plotting here, and customizing it in the second line
#———————————————————————————————#
plot(Ps ~ propSampled, pch = 16, ylim = c(0, 1), xaxt = "n", xlab = "Proportion
    Sampled")
axis(1, at = propSampled, labels = propSampled)

#————————————————————————————————————————————————————————#
# Add dotted lines for true proportions sampled          #
#————————————————————————————————————————————————————————#
for (i in 1:4) {
    abline(h = propSampled[i], lty = "dotted")
}

#——————————————————————————————————————————————————————————————————————————#
# Add line segments to each point indicating the low and high HDI values #
#——————————————————————————————————————————————————————————————————————————#
for (i in 1:4) {
    segments(x0 = propSampled[i], y0 = lowHDI[i], x1 = propSampled[i], y1 = highHDI[i
        ])
}
```



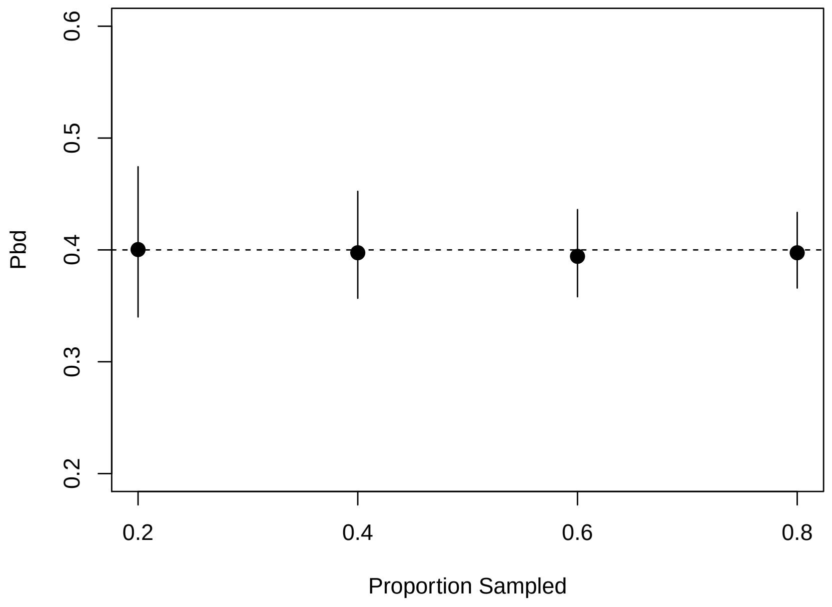
As with the estimates of N, we can see that this approach produces unbiased estimates of P_s, with increasing precision as the proportion of sampled individuals increases. Note that the uncertainty around estimates of P_s are much lower than those for N, particularly when only a small proportion of the population is sampled.

Lastly, we can do the same thing to evaluate the performance of estimation of the number of breeders (P_{bd}), which are columns 4–6 in the returned data.

```
Pbd = c(mean(simSampled_Estimates[1:100, 4]), mean(simSampled_Estimates[101:200, 4]),
    mean(simSampled_Estimates[201:300, 4]), mean(simSampled_Estimates[301:400, 4]))

lowHDI = c(mean(simSampled_Estimates[1:100, 5]), mean(simSampled_Estimates[101:200,
    5]), mean(simSampled_Estimates[201:300, 5]), mean(simSampled_Estimates[301:400,
    5]))

highHDI = c(mean(simSampled_Estimates[1:100, 6]), mean(simSampled_Estimates[101:200,
    6]), mean(simSampled_Estimates[201:300, 6]), mean(simSampled_Estimates[301:400,
    6]))
```

Then, we can plot these results.

```
#------------------------------#
# Plot the data, customizing:
# 1. The plot character & size (pch = 16, cex = 1.5)
# 2. The upper and lower limits for the y-axis (ylim = c(0.2, 0.6))
# 3. The x-axis. Do this by not plotting here, and customizing it in the second line
#------------------------------#
plot(Pbd ~ propSampled, pch = 16, cex = 1.5, ylim = c(0.2, 0.6), xaxt = "n", xlab = "
    Proportion Sampled")
axis(1, at = propSampled, labels = propSampled)

#-------------------------------------------------------#
# Add a dashed line for the true proportion of breeders #
#-------------------------------------------------------#
  abline(h = 0.4, lty = "dashed")

#------------------------------------------------------------------------#
# Add line segments to each point indicating the low and high HDI values #
#------------------------------------------------------------------------#
for (i in 1:4) {
    segments(x0 = propSampled[i], y0 = lowHDI[i], x1 = propSampled[i], y1 = highHDI[i
        ])
}
```



As with the other parameters, we can see that our approach returns unbiased estimates of P_{bd}, regardless of the proportion of individuals sampled. More over, the precision around this estimate does not change drastically as the proportion of sampled individuals changes.

Combined, these analyses show that this approach returns accurate and precise estimates of abundance, across a range of sampling intensities, and should therefore be useful for estimating abundance based on pedigrees.

3 Analysis of Right Whale Data

The data set used for the right whale analyses are in the "`rwData.csv`" file in the **data** folder. This file contains 10 columns:

1. `Year_Set`: the year for which the data are for (representing five actual years)
2. `t_Ns`: the total number of sampled individuals for that year set
3. `t_Nin`: the total number of inferred individuals for that year set
4. `t_Bs`: the total number of breeders sampled for that year set
5. `f_Ns`: the total number of sampled *females* for that year set
6. `f_Nin`: the total number of inferred *females* for that year set
7. `f_Bs`: the total number of breeding *females* sampled for that year set
8. `m_Ns`: the total number of sampled *males* for that year set
9. `m_Nin`: the total number of inferred *males* for that year set
10. `m_Bs`: the total number of breeding *males* sampled for that year set

3.1 All Individuals

We can first run the analyses for all individuals, to obtain total abundance estimates for N. For this analysis, we want to use a hierarchical model, where the estimates for each year also inform the hyperprior from which the distribution of values for each year are also derived. This is a more efficient use of the data, and results in more stable estimates. We can use a hierarchical model by using the function `pedigreeAbundance_h` (the "h" stands for hierarchical).

First, read in the data.

```
rwData = read.table("../data/rwData.csv", header = TRUE, sep = ",")
```

Then, parse out the data as needed for the `pedigreeAbundance_h`. Specifically, it needs three vectors, all of the same length: (1) one vector of the number of sampled individuals for each time period (N_s), (2) one vector of the number of inferred individuals for each time period (N_{in}), and (3) one vector of the number of sampled breeders for each time period (B_s).

```
Ns = rwData$t_NS
Nin = rwData$t_Nin
Bs = rwData$t_Bs
```

Then run the analyses

```
totalN = pedigreeAbundance_h(Ns = Ns, Nin = Nin, Bs = Bs)
```

3.2 Females

We can then run the analyses for just females.

First, parse out the relevant data.

```
f_Ns = rwData$f_NS
f_Nin = rwData$f_Nin
f_Bs = rwData$f_Bs
```

Then run the analyses, stepping through each year. Remember that the data for females are in columns 5–7.

```
females = pedigreeAbundance_h(Ns = f_Ns, Nin = f_Nin, Bs = f_Bs)
```

3.3 Males

Lastly, we can then run the analyses for just males.

First, parse out the relevant data.

```
m_Ns = rwData$m_NS
m_Nin = rwData$m_Nin
m_Bs = rwData$m_Bs
```

Then run the analyses, stepping through each year. Remember that the data for females are in columns 5–7.

```
males = pedigreeAbundance_h(Ns = m_Ns, Nin = m_Nin, Bs = m_Bs)
```

3.4 Visualizing Results

A summary of the results from when we ran these analyses are provided in the `rwSummary.csv` file located within the **results** folder. These also include the mark-recapture abundance estimates based on the photo-identification data reported in Pace *et al.* (2017) for comparison.

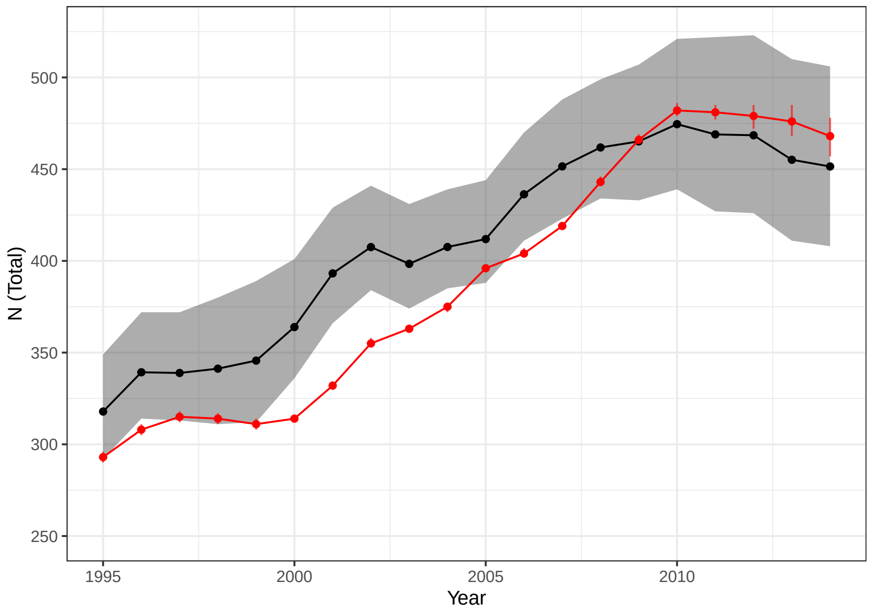
First, we can read these data into R. We will also load the `ggplot2` and `ggpubr` packages.

```
library(ggplot2)
library(ggpubr)

rwsummary = read.table("../results/rwSummary.csv", header = TRUE, sep = ",")
```

First, let's plot the total abundance estimates from our analyses, and compare those to the estimates from Pace *et al.* (2017). For these,

```
ggplot(rwsummary) +
    theme_bw() +
    geom_ribbon(aes(x = Year, ymin = t_Nlow, ymax = t_Nhigh), alpha = 0.4) +
    geom_point(aes(x = Year, y = t_N)) +
    geom_line(aes(x = Year, y = t_N)) +
    geom_point(aes(x = Year, y = t_pace), color = "red") +
    geom_line(aes(x = Year, y = t_pace), color = "red") +
    geom_linerange(aes(x = Year, y = t_pace, ymin = t_paceLow, ymax = t_paceHigh),
        color = "red", alpha = 0.6) +
    ylim(250, 525) +
    ylab("N (Total)")
```

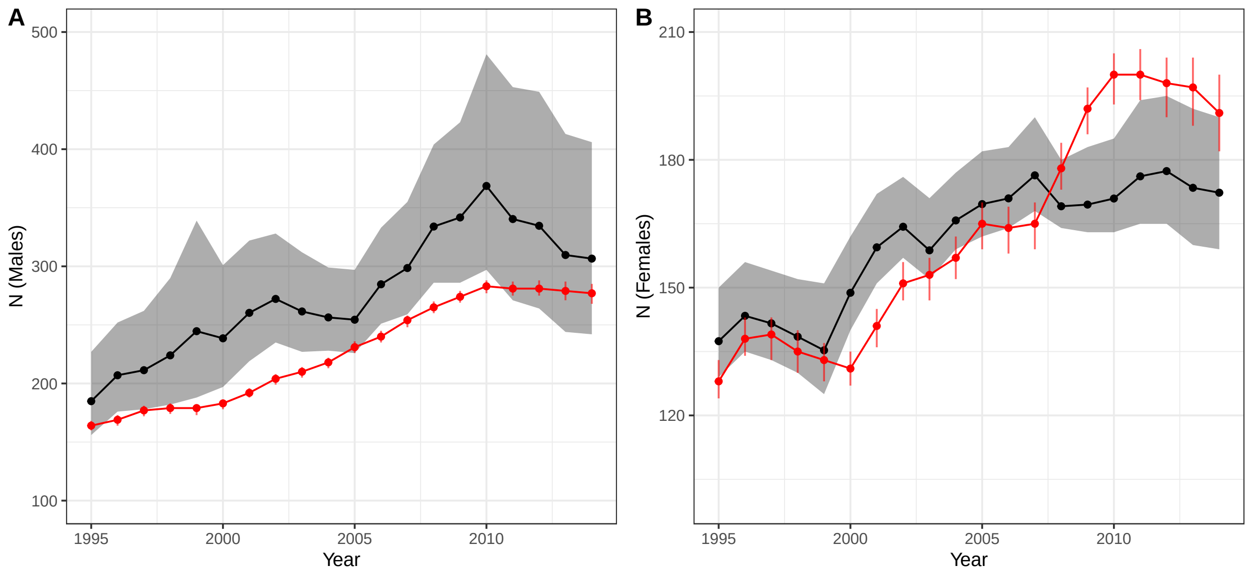
The resulting figure is below.



We can do the same thing, but divide it into data on females and males.

```
females = ggplot(rwsummary) +
    theme_bw() +
    geom_ribbon(aes(x = Year, ymin = f_Nlow, ymax = f_Nhigh), alpha = 0.4) +
    geom_point(aes(x = Year, y = f_N)) +
    geom_line(aes(x = Year, y = f_N)) +
    geom_point(aes(x = Year, y = f_pace), color = "red") +
    geom_line(aes(x = Year, y = f_pace), color = "red") +
    geom_linerange(aes(x = Year, y = f_pace, ymin = f_paceLow, ymax = f_paceHigh),
        color = "red", alpha = 0.6) +
    ylim(100, 210) +
    ylab("N (Females)")

males = ggplot(rwsummary) +
    theme_bw() +
    geom_ribbon(aes(x = Year, ymin = m_Nlow, ymax = m_Nhigh), alpha = 0.4) +
    geom_point(aes(x = Year, y = m_N)) +
    geom_line(aes(x = Year, y = m_N)) +
    geom_point(aes(x = Year, y = m_pace), color = "red") +
    geom_line(aes(x = Year, y = m_pace), color = "red") +
    geom_linerange(aes(x = Year, y = m_pace, ymin = m_paceLow, ymax = m_paceHigh),
        color = "red", alpha = 0.6) +
    ylim(100, 500) +
    ylab("N (Males)")

ggarrange(males, females, labels = c("A", "B"), ncol = 2, nrow = 1)
```

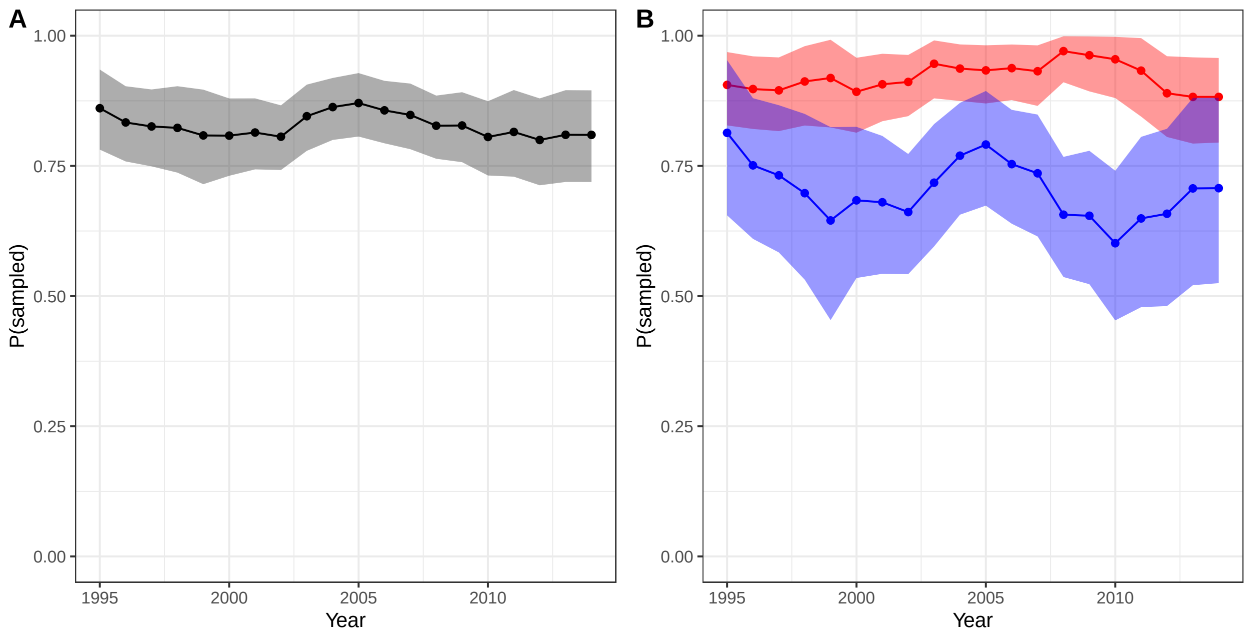
The resulting figure is below.



Next, we can plot the probability of being sample for all individuals, and divided by sex.

```
total = ggplot(rwsummary) +
    theme_bw() +
    geom_ribbon(aes(x = Year, ymin = t_PsLow, ymax = t_PsHigh), alpha = 0.4) +
    geom_point(aes(x = Year, y = t_Ps)) +
    geom_line(aes(x = Year, y = t_Ps)) +
    ylim(0, 1) +
    ylab("P(sampled)")

#---- By Sex ----#
sex = ggplot(rwsummary) +
    theme_bw() +
    geom_ribbon(aes(x = Year, ymin = f_PsLow, ymax = f_PsHigh), fill = "red", alpha =
        0.4) +
    geom_point(aes(x = Year, y = f_Ps), colour = "red") +
    geom_line(aes(x = Year, y = f_Ps), colour = "red") +
    geom_ribbon(aes(x = Year, ymin = m_PsLow, ymax = m_PsHigh), fill = "blue", alpha =
         0.4) +
    geom_point(aes(x = Year, y = m_Ps), colour = "blue") +
    geom_line(aes(x = Year, y = m_Ps), colour = "blue") +
    ylim(0, 1) +
    ylab("P(sampled)")

ggarrange(total, sex, labels = c("A", "B"), ncol = 2, nrow = 1)
```

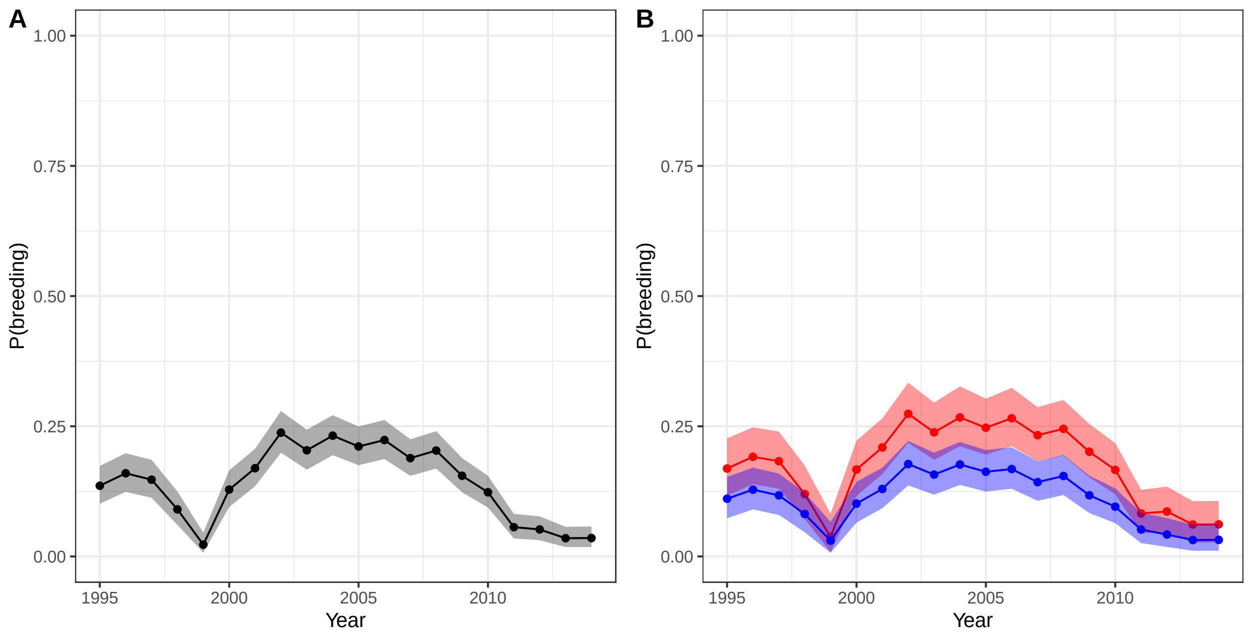
The resulting figure is below.



Next, we can plot the probability of breeding for all individuals, and divided by sex.

```
       total = ggplot(rwsummary) +
           theme_bw() +
           geom_ribbon(aes(x = Year, ymin = t_PbdLow, ymax = t_PbdHigh), alpha = 0.4) +
           geom_point(aes(x = Year, y = t_Pbd)) +
           geom_line(aes(x = Year, y = t_Pbd)) +
           ylim(0, 1) +
           ylab("P(breeding)")

       #--- By Sex ---#
       sex = ggplot(rwsummary) +
           theme_bw() +
           geom_ribbon(aes(x = Year, ymin = f_PbdLow, ymax = f_PbdHigh), fill = "red", alpha
               = 0.4) +
           geom_point(aes(x = Year, y = f_Pbd), colour = "red") +
           geom_line(aes(x = Year, y = f_Pbd), colour = "red") +
           geom_ribbon(aes(x = Year, ymin = m_PbdLow, ymax = m_PbdHigh), fill = "blue", alpha
                = 0.4) +
           geom_point(aes(x = Year, y = m_Pbd), colour = "blue") +
           geom_line(aes(x = Year, y = m_Pbd), colour = "blue") +
           ylim(0, 1) +
           ylab("P(breeding)")

       ggarrange(total, sex, labels = c("A", "B"), ncol = 2, nrow = 1)
```

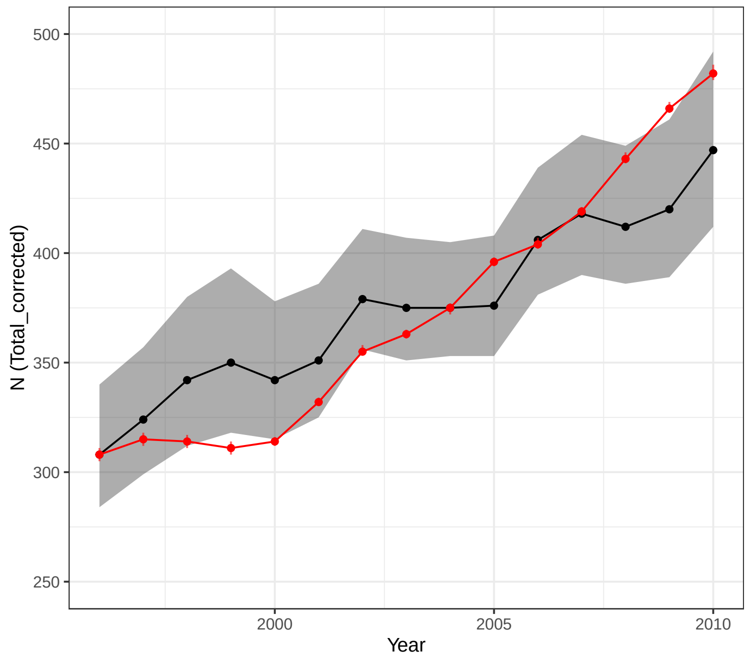
The resulting figure is below.



Lastly, our abundance estimates for each year are actually based on three years of data (e.g., our estimate for 1995 actually includes all individuals alive in 1994, 1995, and 1996). This will artificially inflate our estimates. To account for this, we need to do two things (rationale explained in the manuscript): (1) remove the number of individuals born in the year subsequent to the year of interest, and (2) remove the number of individuals that died prior to the year of interest. We can conduct these calculations, and plot our corrected estimates, using the commands below.

```
growthRate = 0.028
years = 1996:2010
nYears = length(years)
N_corr_Mean =  rep(NA, times = nYears)
N_corr_Low = rep(NA, times = nYears)
N_corr_High = rep(NA, times = nYears)
for (i in 1:nYears) {
    N_corr_Mean[i] = round(rwsummary$t_N[i+1] - rwsummary$calves[i+2] - (rwsummary$
        calves[i+1] - (rwsummary$t_N[i] * growthRate)))
    N_corr_Low[i] = round(rwsummary$t_Nlow[i+1] - rwsummary$calves[i+2] - (rwsummary$
        calves[i+1] - (rwsummary$t_Nhigh[i] * growthRate)))
    N_corr_High[i] = round(rwsummary$t_Nhigh[i+1] - rwsummary$calves[i+2] - (rwsummary
        $calves[i+1] - (rwsummary$t_Nlow[i] * growthRate)))
}

N_corrected = data.frame(years, N_corr_Mean, N_corr_Low, N_corr_High, rwsummary$t_pace
    [2:16], rwsummary$t_paceLow[2:16], rwsummary$t_paceHigh[2:16])

colnames(N_corrected) = c("years", "N_corr", "NLow_corr", "NHigh_corr", "t_pace", "t_
    paceLow", "t_paceHigh")

ggplot(N_corrected) +
    theme_bw() +
    geom_ribbon(aes(x = years, ymin = NLow_corr, ymax = NHigh_corr), alpha = 0.4) +
    geom_point(aes(x = years, y = N_corr)) +
    geom_line(aes(x = years, y = N_corr)) +
    geom_point(aes(x = years, y = t_pace), color = "red") +
    geom_line(aes(x = years, y = t_pace), color = "red") +
    geom_linerange(aes(x = years, y = t_pace, ymin = t_paceLow, ymax = t_paceHigh),
        color = "red", alpha = 0.6) +
    ylim(250, 500) +
    ylab("N (Total_corrected)") +
    xlab("Year")
```

The resulting figure is below.


N (Total_corrected)
Year