Introduction to Resampling in R

DQuinn

Friday, November 06, 2015

Ecological studies often suffer from limited resource availability, including time, money, and people power. Oversampling not only taxes these resources needlessly, but may also negatively influence the system being studied; for example, through lethal collections of organisms or unnecessary handling of non-target species. Projects relying on data collection over multiple field seasons may benefit from resampling algorithms that can identify efficient collection protocols if oversampling is known or thought to have occurred.

We'll be using a mock data set of tagged and non-tagged eels captured in a small lake to estimate population size, then build a resampling algorithm to explore the sensitivity of using varying numbers of traps and sampling events.

* 50 American eels were captured from a small lake, marked with an external tag, and released back into the lake
* 15 traps were checked 20 times over a season
* number of tagged and non-tagged eels in each trap was recorded

**Load Required Packages**

library(dplyr)  
library(FSA)  
library(ggplot2)  
library(digest)  
library(lubridate)

You can find the dataset here: <https://github.com/DanielleQuinn/RLessons/blob/master/SensitivityAnalysis/recapture_data.csv>

df.data<-read.csv("recapture\_data.csv")

First, take a look at the data frame df.data

head(df.data)

X trap check present tag  
1 1 1 1 1 1  
2 2 1 1 1 0  
3 3 1 1 1 0  
4 4 1 1 1 0  
5 5 1 1 1 0  
6 6 1 1 1 0

str(df.data)

'data.frame': 150036 obs. of 5 variables:  
 $ X : int 1 2 3 4 5 6 7 8 9 10 ...  
 $ trap : int 1 1 1 1 1 1 1 1 1 1 ...  
 $ check : int 1 1 1 1 1 1 1 1 1 2 ...  
 $ present: int 1 1 1 1 1 1 1 1 1 1 ...  
 $ tag : int 1 0 0 0 0 0 0 0 0 1 ...

**Baseline Population Size Estimate**

We'll be using the mrClosed() function from the FSA package to estimate the population size using the Schnabel method. Without getting into too much detail, the Schnbabel method is used when we have multiple sampling occassions and assumes a closed population. The function requires specific input, including:

* n: the number of captured animals
* m: the number of recaptured marked animals
* M: the number of extant marked animals prior to the sample

*Step 1:* Build cmr data frame

df.cmr<-data.frame(df.data%>%  
 group\_by(check)%>%  
 summarise(date=unique(check),n=sum(present), m=sum(tag), M=50)%>%  
 select(n,m,M))

*Step 2:* Estimate population size

pop.est<-summary(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel"))) #From FSA package  
pop.low<-stats::confint(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel")))[1] # Confidence intervals #  
pop.high<-stats::confint(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel")))[2] # Confidence intervals #  
baseline<-data.frame(pop.est, pop.low, pop.high)  
baseline

N pop.low pop.high  
1 152 151 154

We see that using all of the data avilable (50 traps each checked 20 times), our population estimate is 152, with confidence intervals of 151 and 154.

Because the lake is very small, we're fairly certain that oversampling may have occurred, and our goal is to design a sampling protocol for the next year that maximizes the efficiency of our sampling effort. We want to know the minimum number of traps and the minimum number of checks that we can do that will result in a realistic estimate of population size.

**Sample 10 Random Traps**

all\_traps<-unique(df.data$trap) # A vector of all available traps  
use\_traps<-sample(all\_traps, 10) # Choose 10 random traps

Subset the data to include only those traps, then create df.cmr and use mrClosed() to estimate population size.

my\_subset<-data.frame(df.data%>%  
 filter(trap %in% use\_traps))  
  
df.cmr<-data.frame(my\_subset%>%  
 group\_by(check)%>%  
 summarise(date=unique(check),n=sum(present), m=sum(tag), M=50)%>%  
 select(n,m,M))  
  
pop.est<-summary(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel"))) #From FSA package  
pop.low<-stats::confint(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel")))[1] # Confidence intervals #  
pop.high<-stats::confint(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel")))[2] # Confidence intervals #  
results<-data.frame(pop.est, pop.low, pop.high)  
results

N pop.low pop.high  
1 163 159 167

**Sample X Random Traps**

Repeat this analysis using varying numbers of traps and see how our results vary from the baseline population estimates. We are going to resample from 10 to 50 random traps and apply to analysis for each subset.

*Step 1:* Build a function that creates the cmr data frame and produces the population estimate and confidence intervals

my\_cmr<-function(my\_subset)  
{  
 # Build cmr data frame  
 df.cmr<-data.frame(my\_subset%>%  
 group\_by(check)%>%  
 summarise(date=unique(check),n=sum(present), m=sum(tag), M=50)%>%  
 select(n,m,M))  
   
 # Estimate Population Size  
 pop.est<<-summary(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel"))) #From FSA package  
 pop.low<<-stats::confint(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel")))[1] # Confidence intervals #  
 pop.high<<-stats::confint(with(df.cmr,mrClosed(n=n,m=m,M=M,method="Schnabel")))[2] # Confidence intervals #  
}

*Step 2:* Set up a data frame to handle the output of the analyses We want a data frame that has information about how many traps were used, and the results of our analysis. In this case, we're going to have 41 sets of output.

results<-data.frame(traps=c(10:50), pop.est=NA, pop.low=NA, pop.high=NA)  
results

traps pop.est pop.low pop.high  
1 10 NA NA NA  
2 11 NA NA NA  
3 12 NA NA NA  
4 13 NA NA NA  
5 14 NA NA NA  
6 15 NA NA NA  
7 16 NA NA NA  
8 17 NA NA NA  
9 18 NA NA NA  
10 19 NA NA NA  
11 20 NA NA NA  
12 21 NA NA NA  
13 22 NA NA NA  
14 23 NA NA NA  
15 24 NA NA NA  
16 25 NA NA NA  
17 26 NA NA NA  
18 27 NA NA NA  
19 28 NA NA NA  
20 29 NA NA NA  
21 30 NA NA NA  
22 31 NA NA NA  
23 32 NA NA NA  
24 33 NA NA NA  
25 34 NA NA NA  
26 35 NA NA NA  
27 36 NA NA NA  
28 37 NA NA NA  
29 38 NA NA NA  
30 39 NA NA NA  
31 40 NA NA NA  
32 41 NA NA NA  
33 42 NA NA NA  
34 43 NA NA NA  
35 44 NA NA NA  
36 45 NA NA NA  
37 46 NA NA NA  
38 47 NA NA NA  
39 48 NA NA NA  
40 49 NA NA NA  
41 50 NA NA NA

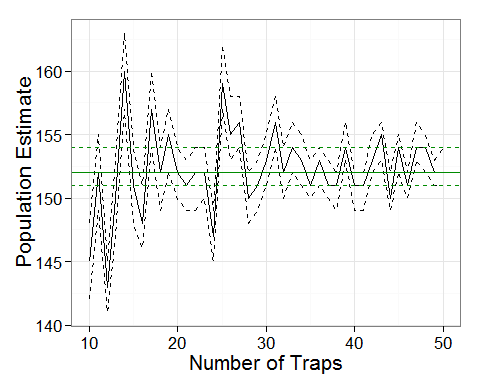
*Step 3:* Resample the data and fill in the results data frame

for(i in 1:nrow(results)) # For each row in our results data frame  
{  
 use\_traps<-sample(all\_traps, results$traps[i]) # Sample x number of random traps  
 my\_subset<-data.frame(df.data%>%filter(trap %in% use\_traps)) # Subset the data to only include those traps  
 my\_cmr(my\_subset) # Apply our my\_cmr function to the subset  
 results$pop.est[i]<-pop.est # Fill in our results  
 results$pop.low[i]<-pop.low  
 results$pop.high[i]<-pop.high  
}  
results

traps pop.est pop.low pop.high  
1 10 145 142 148  
2 11 152 149 155  
3 12 143 141 145  
4 13 150 147 153  
5 14 160 157 163  
6 15 151 148 154  
7 16 148 146 151  
8 17 157 154 160  
9 18 152 149 154  
10 19 155 152 157  
11 20 152 150 154  
12 21 151 149 153  
13 22 152 149 154  
14 23 152 150 154  
15 24 147 145 149  
16 25 159 157 162  
17 26 155 153 158  
18 27 156 154 158  
19 28 150 148 152  
20 29 151 149 153  
21 30 153 151 155  
22 31 156 154 158  
23 32 152 150 154  
24 33 154 152 156  
25 34 153 151 155  
26 35 151 150 153  
27 36 153 151 154  
28 37 151 150 153  
29 38 151 149 152  
30 39 154 153 156  
31 40 151 149 152  
32 41 151 149 152  
33 42 153 152 155  
34 43 155 153 156  
35 44 150 149 152  
36 45 154 152 155  
37 46 151 150 152  
38 47 154 153 156  
39 48 154 152 155  
40 49 152 151 153  
41 50 152 151 154

*Step 4:* Visualize results

ggplot(results)+  
 geom\_line(aes(x=traps, y=pop.est))+  
 geom\_line(aes(x=traps, y=pop.low), linetype='dashed')+  
 geom\_line(aes(x=traps, y=pop.high), linetype='dashed')+  
 geom\_hline(yint=baseline$N, col='green4')+  
 geom\_hline(yint=baseline$pop.low, col='green4',linetype='dashed')+  
 geom\_hline(yint=baseline$pop.high, col='green4',linetype='dashed')+  
 theme\_bw(16)+xlab("Number of Traps")+ylab("Population Estimate")



Because random traps are selected each time, your results will look slightly different. However, you should see the same amount of variation in population estimates. Why is there so much variation? It's because we're only randomly selecting each number of traps a single time. To really get an idea of the overall sensitivity of the population estimate to the number of traps used, we'll repeat this resampling algorithm many times.

**Bootstrapping**

Repeat this algorithm 25 times, but instead of varying the number of traps to be 1 to 50, we'll limit those options to be 10, 15, 20, ...50.

*Step 1:* Build a function - already done!

*Step 2:* Set up a data frame to handle the output of the analyses In this case, we're going to have 9 (different numbers of traps to select) x 25 (repeats) outputs:

results<-data.frame(traps=rep(seq(from=10, to=50, by=5),25), pop.est=NA, pop.low=NA, pop.high=NA)  
head(results)

traps pop.est pop.low pop.high  
1 10 NA NA NA  
2 15 NA NA NA  
3 20 NA NA NA  
4 25 NA NA NA  
5 30 NA NA NA  
6 35 NA NA NA

*Step 3:* Resample and fill in the results data frame In addition, we'll set up a means of keeping track of how long the resampling takes, and how long each iteration takes to complete.

starttime=Sys.time() # What time does the resampling begin?  
for(i in 1:nrow(results))  
{  
 use\_traps<-sample(all\_traps, results$traps[i])  
 my\_subset<-data.frame(df.data%>%filter(trap %in% use\_traps))  
 my\_cmr(my\_subset)  
 results$pop.est[i]<-pop.est  
 results$pop.low[i]<-pop.low  
 results$pop.high[i]<-pop.high  
}  
totaltime<-difftime(Sys.time(),starttime, unit="secs") #How long did it take?  
totaltime

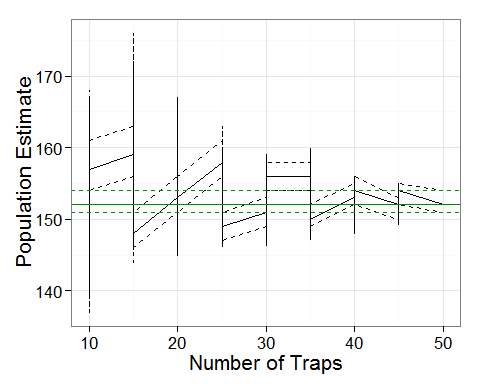
Time difference of 5.752329 secs

paste(round(as.numeric(totaltime)/nrow(results),2),"seconds per iteration") # How long did each iteration take?

[1] "0.03 seconds per iteration"

*Step 4:* Visualize results

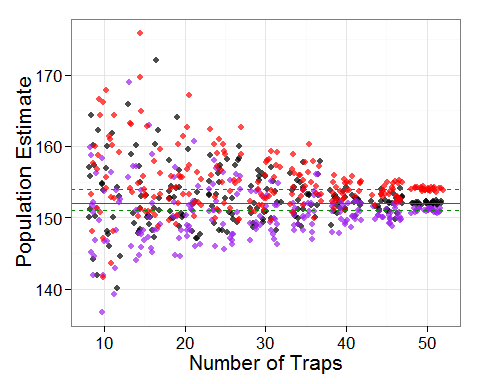
ggplot(results)+  
 geom\_line(aes(x=traps, y=pop.est))+  
 geom\_line(aes(x=traps, y=pop.low), linetype='dashed')+  
 geom\_line(aes(x=traps, y=pop.high), linetype='dashed')+  
 geom\_hline(yint=baseline$N, col='green4')+  
 geom\_hline(yint=baseline$pop.low, col='green4',linetype='dashed')+  
 geom\_hline(yint=baseline$pop.high, col='green4',linetype='dashed')+  
 theme\_bw(16)+xlab("Number of Traps")+ylab("Population Estimate")



Why do we have jagged lines? It's because each value of traps has 25 points associate with it. There are different ways to deal with this.

*Option 1:* Don't use a line to represent the resampled data.

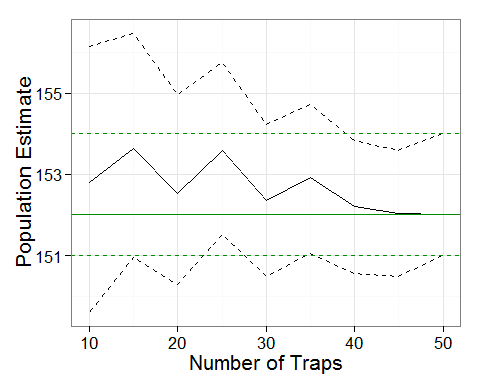
ggplot(results)+  
 geom\_jitter(aes(x=traps, y=pop.est), alpha=0.7)+  
 geom\_jitter(aes(x=traps, y=pop.low), col='purple', alpha=0.7)+  
 geom\_jitter(aes(x=traps, y=pop.high), col='red', alpha=0.7)+  
 geom\_hline(yint=baseline$N, col='green4')+  
 geom\_hline(yint=baseline$pop.low, col='green4',linetype='dashed')+  
 geom\_hline(yint=baseline$pop.high, col='green4',linetype='dashed')+  
 theme\_bw(16)+xlab("Number of Traps")+ylab("Population Estimate")



You can start to see that when few traps are used, the population estimates are much more variable than if many traps are used.

*Option 2:* Summarize the data.

results.sum<-data.frame(results%>%  
 group\_by(traps)%>%  
 summarise(mean.pop=mean(pop.est), mean.low=mean(pop.low), mean.high=mean(pop.high)))  
  
ggplot(results.sum)+  
 geom\_line(aes(x=traps, y=mean.pop))+  
 geom\_line(aes(x=traps, y=mean.low), linetype='dashed')+  
 geom\_line(aes(x=traps, y=mean.high), linetype='dashed')+  
 geom\_hline(yint=baseline$N, col='green4')+  
 geom\_hline(yint=baseline$pop.low, col='green4',linetype='dashed')+  
 geom\_hline(yint=baseline$pop.high, col='green4',linetype='dashed')+  
 theme\_bw(16)+xlab("Number of Traps")+ylab("Population Estimate")



**More Bootstrapping**

Recall that each trap was checked 20 times. Imagine all of the time and effort that would be saved if we could sample traps fewer times and still be confident in our population estimates! In addition to exploring the sensitivity of the number of traps, let's add another level to this analysis and include a varying number of checks. We'll say that we want to see what would happen if 10 to 20 checks were done.

all\_checks<-unique(df.data$check)

Because we've been keeping track of how long each iteration takes (approx. 0.03 sec, depending on the processing power of your computer), we know that we can easily do more iterations within a reasonable time frame. We'll do 40 repeats this time.

*Step 1:* Build a function - already done!

*Step 2:* Set up a data frame to handle the output of the analyses In this case, we're going to have 9 (different numbers of traps to select) x 11 (different numbers of checks to select) x 40 (repeats) outputs. Using expand.grid() will be useful here; it creates a dataframe of every combination of each argument you give it.

results<-expand.grid(traps=seq(from=10, to=50, by=5), checks=c(10:20), repeats=c(1:40))  
nrow(results) # How many iterations?

[1] 3960

nrow(results)\*0.03/60 # It should take roughly 2 minutes

[1] 1.98

# We can get rid of the repeats column  
results<-results[,-3]  
  
# And need to add the pop.est, pop.low, and pop.high columns  
results$pop.est<-NA  
results$pop.low<-NA  
results$pop.high<-NA

*Step 3:* Resample and fill in the results data frame

starttime=Sys.time()  
for(i in 1:nrow(results))  
{  
 # print(paste(i/nrow(results)\*100, "% Complete")) # This is useful for keeping track of how many iterations have been completed  
 use\_traps<-sample(all\_traps, results$traps[i])  
 use\_checks<-sample(all\_checks, results$checks[i])  
 my\_subset<-data.frame(df.data%>%filter(trap %in% use\_traps)%>%filter(check %in% use\_checks))  
 my\_cmr(my\_subset)  
 results$pop.est[i]<-pop.est  
 results$pop.low[i]<-pop.low  
 results$pop.high[i]<-pop.high  
}  
totaltime<-difftime(Sys.time(),starttime, unit="secs")  
totaltime

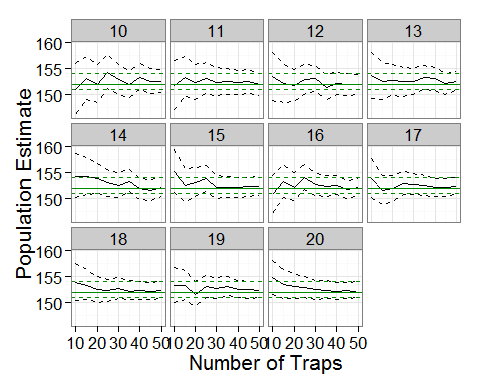
Time difference of 109.2773 secs

paste(round(as.numeric(totaltime)/nrow(results),2),"seconds per iteration")

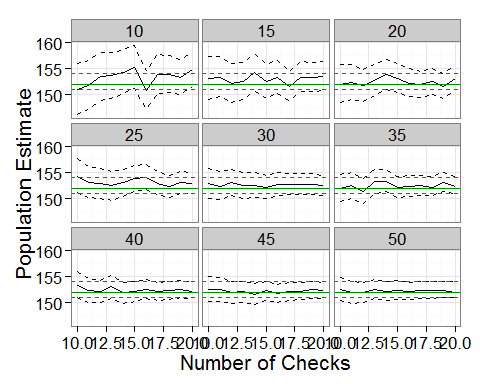
[1] "0.03 seconds per iteration"

*Step 4:* Visualize results

results.sum<-data.frame(results%>%  
 group\_by(traps, checks)%>%  
 summarise(mean.pop=mean(pop.est), mean.low=mean(pop.low), mean.high=mean(pop.high)))  
  
ggplot(results.sum)+  
 geom\_line(aes(x=traps, y=mean.pop))+  
 geom\_line(aes(x=traps, y=mean.low), linetype='dashed')+  
 geom\_line(aes(x=traps, y=mean.high), linetype='dashed')+  
 geom\_hline(yint=baseline$N, col='green4')+  
 geom\_hline(yint=baseline$pop.low, col='green4',linetype='dashed')+  
 geom\_hline(yint=baseline$pop.high, col='green4',linetype='dashed')+  
 theme\_bw(16)+xlab("Number of Traps")+ylab("Population Estimate")+  
 facet\_wrap(~checks)



ggplot(results.sum)+  
 geom\_line(aes(x=checks, y=mean.pop))+  
 geom\_line(aes(x=checks, y=mean.low), linetype='dashed')+  
 geom\_line(aes(x=checks, y=mean.high), linetype='dashed')+  
 geom\_hline(yint=baseline$N, col='green4')+  
 geom\_hline(yint=baseline$pop.low, col='green4',linetype='dashed')+  
 geom\_hline(yint=baseline$pop.high, col='green4',linetype='dashed')+  
 theme\_bw(16)+xlab("Number of Checks")+ylab("Population Estimate")+  
 facet\_wrap(~traps)



You can use these plots to start identifying the number of traps and the number of checks that result in population estimates that you consider to be acceptable, and from there you can design more efficient sampling protocols for next year!