# CAPWIRE TUTORIAL

Matthew W. Pennell & Craig R. Miller
University of Idaho
https://github.com/mwpennell/capwire

## Contents

## 1 Introduction

This tutorial demonstrates the use and functionality of the R package `capwire` Pennell *et al.* (submitted). It is designed for the estimation of population size from non-invasive sampling of wild populations. It is based on the models presented by Miller *et al.* (2005). For details on the models and the implementation, see Miller *et al.* (2005) and Pennell *et al.* (submitted).

## 2 Estimating Population Size From Non-Invasive Genetic Sampling: An Example

### 2.1 Reading In the Data

For a demonstration, we use a published data set from Banks *et al.* (2003) from a population of northern hairy-nosed wombats (*Lasiorhinus krefftii*). Individuals were identified with microsatellites.

```
> library(capwire)
```

Load in the data.

```
> data(wombat)
```

Alternatively, user data can be loaded into R using `read.table`:

```
> # data <- read.table(file="my_data_set.txt")
```

Taking a look at the structure of the wombat data

```
> wombat
```

```
  capture.class No.Ind
1             1     25
2             2     11
3             3     10
```

```
4               4    10
5               5     9
6               6    11
7               7     5
```

notice that the first column contains the capture classes (i.e. individuals in capture class $i$ were captured $i$ times). The second column is the number of individuals that are in each capture class. The models used here assume that individuals can be identified with certainty. The data must be entered as a two-column data frame as in this example. The column names do not need to be specified; we do so here for the purposes of clarity.

If one enters the capture counts as a vector

```
> some.fake.data <- c(1,1,1,2,2,3)
```

this can be converted to the capture class table format with the function `buildClassTable`.

```
> fake.data.table <- buildClassTable(some.fake.data)
> fake.data.table

  capture.class No.Ind
1             1      3
2             2      2
3             3      1
```

## 2.2 Fitting the Equal Capture Model (ECM) to Estimate Population Size

Under the Equal Capture Model (ECM) of Miller *et al.* (2005), all individuals are assumed to have an equal probability of being captured. Under the assumption that a population can be modeled as an urn, then the sampling of an individual's DNA can be equated to drawing a ball at random from the urn. The sampled individual is then replaced and another draw is made. We can write down the likelihood $\mathcal{L}$ of obtaining the vector of capture counts $\vec{c}$ for the $T$ individuals sampled given a total poplulation size $N$:

$$\mathcal{L}(N) = \left(\frac{N!}{T!(N-T)!}\right)\left(\frac{S!}{c_1!c_2!\dots c_T!}\right)\prod_{i=1}^{T}(1/N)^{c_i} \tag{1}$$

where $S$ is the total number of samples collected ($S = \sum_{i=1}^{T} c_i$). The natural logarithm of $\mathcal{L}$ is then proportional to:

$$\ln \mathcal{L}(N) \propto \sum_{x=1}^{N}\ln(x) - \sum_{x=1}^{N-T}\ln(x) + \ln(1/N)\sum_{i=1}^{T}c_i \tag{2}$$

We can estimate the population size under this model using the function `fitEcm`:

```
> res.ecm <- fitEcm(data=wombat, max.pop=200)

> res.ecm


$model

[1] "Equal.capture"


$likelihood

[1] -1153.84


$ml.pop.size

[1] 84


$cap.ind

[1] 3.246914


$sampled.ind

[1] 81
```

```
$sample.size
[1] 263


$max.pop
[1] 200
```

The argument `max.pop` should be much larger than we reasonably expect the population to be. `max.pop` needs to be specified in order to generate an upper bound for the purposes of optimization. For our examples, we know that the populations are likely to be very small (i.e. $<$ `100` individuals) so we used a `max.pop` that was roughly twice as large as we expected the population to be. Note that the Maximum Likelihood Estimate (MLE) of the population size is infinite if the data is uniformative (e.g. under the scenario where all individuals are captured only once). We can estimate confidence intervals for the population size under this model using a parametric bootstrap approach in which we use the MLE for population size and the sample size $S$ from our observed data to simulate data sets under the ECM Model. The function `bootstrapCapwire` takes three arguments: an object produced from fitting a model (here the object is `res.ecm`, the desired confidence interval (the default is to estimate the 95% confidence intervals) and a number of parametric bootstraps to perform. The argument `CI` takes a vector of quantiles to estimate. Multiple conidence intervals can thus be estimated simultaneously and the confidence intervals need not be symmetric. We recommend conducting at least 1000 parametric bootstraps (the default value).

```
> boot.ecm <- bootstrapCapwire(x=res.ecm, bootstraps=1000, CI=c(0.025, 0.975))
> boot.ecm

$ml.pop.size
[1] 84


$conf.int
```

```
 2.5% 97.5%
   81    88
```

## 2.3 Fitting the Two-Innate Rates Model (TIRM) to Estimate Population Size

For many data sets the ECM model will not be an accurate model as there is likely individual-based heterogeneity in the probability of capture. The simplest approach to dealing with this rate heterogeneity is to assume that a population contains a mixture of individuals with two distinct capture probabilities. Miller *et al.* (2005) call this approach the Two-Innate Rates Model (TIRM). They denote individuals of class B to be those which are difficult to capture and individuals of class A to be those which are relatively easy to capture. Let the relative probability of capturing an individual of class B be 1 and the relative probability of capturing an individual of class A be $\alpha$ (where $\alpha > 1$). If we assume that we can assign all $T$ captured individuals with certainty to either capture class A or B ($T_A$ and $T_B$, respectively), then the joint (log) likelihood of obtaining the vectors of capture counts ($\vec{c}_A$ and $\vec{c}_B$) is given by the equation:

$$\begin{aligned}\ln \mathcal{L}(N_A, N_B) \propto \sum_{x=1}^{N_A} \ln(x) - \sum_{x=1}^{N_A-T_A} \ln(x) + \ln\left[\frac{\alpha}{\alpha N_A + N_B}\right] \sum_{i_A=1}^{T_A} c_{iA} \\ + \sum_{x=1}^{N_B} \ln(x) - \sum_{x=1}^{N_B-T_B} \ln(x) + \ln\left[\frac{1}{\alpha N_A + N_B}\right] \sum_{i_B=1}^{T_B} c_{iB}\end{aligned} \tag{3}$$

We can use this model to obtain the MLE for the population size ($N = N_A + N_B$) using the function `fitTirm`.

```
> res.tirm <- fitTirm(data=wombat, max.pop=200)
> res.tirm

$model
[1] "Two.innate.rates"
```

```
$likelihood
[1] -1104.967


$ml.pop.size
[1] 98


$ml.na
[1] 35


$ml.nb
[1] 63


$alpha
[1] 4.002968


$cap.ind
[1] 3.246914


$sampled.ind
[1] 81


$sample.size
[1] 263


$max.pop
[1] 200
```

For details on how the MLE of $N$ is estimated with the Two-Innate Rates Model, see Miller *et al.* (2005).

## 2.4 Model Selection

To select between the ECM and TIRM models, we use a likelihood ratio test (LRT). To evaluate the significance of the Likelihood Ratio (LR), we generate a null distribution for the LR by simulating data under the ECM using the ML estimate of population size obtained from fitting the ECM. For each simulated data set we fit both the ECM and the TIRM to the data and then calculate the LR. Significance can then be assesed by comparing the observed LR to the distribution of LR obtained via simulation.

```
> lik.ratio <- lrtCapwire(ecm=res.ecm, tirm=res.tirm, bootstraps=100)
> lik.ratio

$LR
[1] 48.87286


$p.value
[1] 0
```

Here we can reject the null model that there is no within population heterogeneity in the probability of capture.

We can then perform parametric bootstraps to obtain 95% confidence intervals for our estimate of population size under TIRM:

```
> boot.tirm <- bootstrapCapwire(x=res.tirm, bootstraps=1000, CI=c(0.025, 0.975))
> boot.tirm

$ml.pop.size
[1] 98
```

```
$conf.int
 2.5% 97.5%
   86   110
```

## 2.5 Partitioning the Data with PART

Some data sets contain count data which is overdispersed where some individuals are captured far too frequently for the assumptions of either the ECM or TIRM to be valid. Stansbury *et al.* (in prep.) developed an algorithm (which they refer to as PART) for dealing with these data sets. The PART method removes individuals detected a large number of times from the data. These individuals provide little information about the population size and, because they are inconsistent with the modeling assumption of just two capture rates, they cause estimation problems in `capwire`.

The PART algorithm first fits the Two-Innate Rates Model (TIRM) to the entire data set to obtain $\hat{N_A}$, $\hat{N_B}$ and $\hat{\alpha}$. Assuming the distribution of capture counts can be approximated as either a two ($H_0$) or three ($H_A$) class multinomial distribution, the algorithm considers all possible ways to partition the count data under both the null and alternative distributions. The partitioning schemes which maximizes the respective likelihoods is retained. A test-statistic $\Lambda_{obs}$, the ratio of multinomial likelihoods $\ln(\mathcal{L}_3) - \ln(\mathcal{L}_2)$ are obtained for the observed data. $\hat{N_A}$, $\hat{N_B}$ and $\hat{\alpha}$ are used to simulate $n$ data sets under the TIRM. $\Lambda$ is then calculated for each simulated data set as described above. $\Lambda_{obs}$ is then compared to the simulated distribution of $\Lambda$. If $\Lambda_{obs}$ falls in the tail of the distribution of $\Lambda$ from the simulated data (e.g. $p \leq 0.05$), the null model can be rejected.

To demonstrate the partitioning method, we use a data set from Stansbury *et al.* (in prep.) from a population of gray wolves (*Canis lupis*) located in western Idaho, U.S.A. Individuals were identified with microsatellites. Note that this data set is not yet published. Taking a look at the data, it is clear that the capture counts are overdispersed compared to what we expect under either the ECM or the TIRM.

```
> wolf.data

   capture.class No.Ind
1              1     17
2              2      1
3              3      1
4              4      4
5              5      6
6              6      2
7              7      1
8              8      2
9              9      1
10            12      4
11            13      2
12            14      1
13            15      1
14            16      1
15            17      1
16            19      2
17            20      1
18            21      2
```

To partition the data set with the PART algorithm, we use the function `partitionCountData`.

```
> wolf.data.part <- partitionCountData(data=wolf.data, n.boots.part=100, max.pop=200)
> wolf.data.part

$low.2.thirds
  capture.class No.Ind
```

```
1             1     17
2             2      1
3             3      1
4             4      4
5             5      6
6             6      2
7             7      1
8             8      2
9             9      1


$up.1.third
  capture.class No.Ind
1            12      4
2            13      2
3            14      1
4            15      1
5            16      1
6            17      1
7            19      2
8            20      1
9            21      2


$p.2.3
[1] 0
```

The argument `n.boots.part` specifies the number of parametric bootstraps to perform in order to obtain the null distribution of partitions.

We then can fit the TIRM to the count data in the lower partition and then add individuals from the upper partition to the MLE of population size *post hoc*. This can be done using the original data with the function `fitTirmPartition`, which is a wrapper for the functions `partitionCountData` and `fitTirm`.

```
> res.part <- fitTirmPartition(data=wolf.data, max.pop=200, n.boots.part=100)
> res.part
```

```
$model
[1] "Two.innate.rates.partitioned"


$likelihood
[1] -361.4131


$ml.pop.size
[1] 67


$ml.na
[1] 16


$ml.nb
[1] 36


$alpha
[1] 7.414773


$cap.ind
[1] 3.2


$sampled.ind
```

```
[1] 50


$sample.size
[1] 112


$max.pop
[1] 200


$excluded
  capture.class No.Ind
1            12      4
2            13      2
3            14      1
4            15      1
5            16      1
6            17      1
7            19      2
8            20      1
9            21      2


$p.value.partition
[1] 0
```

In addition to the output from fitting the TIRM, `fitTirmPartition` outputs a data frame containing individuals that were excluded from the initial estimation procedure and a p-value denoting whether partitioning the data significantly improves the fit of the model. Here partitioning the data significantly improves model fit ($p \simeq 0$). Note that because the PART algorithm assumes that the count data can be partitioned into at least two classes, it is inappropriate to peform a Likelihood Ratio Test comparing the ECM to the TIRM on the partitioned data set. Again, we can then perform a parametric bootstrap to obtain confidence intervals around our population size estimate.

```
> boot.part <- bootstrapCapwire(x=res.part, bootstraps=1000)
> boot.part

$ml.pop.size
[1] 67


$conf.int
 2.5% 97.5%
   54    73
```

## 3 Simulating Data

`capwire` also contains a number of functions for simulating data. The purpose of this is to allow for rigorous evaluation of the robustness of models contained within the package. Data can be simulated under either the ECM (using `simEcm`) or the TIRM (using `simTirm`).

```
> simdata.ecm <- simEcm(n=30, s=100)
> head(simdata.ecm)

  capture.class No.Ind
1             2      9
2             3      8
3             4      3
4             5      2
5             6      1
6             7      2
```

```
> simdata.tirm <- simTirm(na=15, nb=15, s=100, alpha=4)
> head(simdata.tirm)

  capture.class No.Ind
1             1      7
2             2      3
3             3      4
4             4      2
5             5      2
6             6      2
```

`n`, `na` and `nb` refer to the total number of individuals in the population, number of individuals in class A and number of individuals in class B, respectively. `s` denotes the total number of samples collected and in `simTirm`, $\alpha$ is the ratio of capturabilities between class A and class B individuals. Data sets produced by both `simEcm` and `simTirm` can be input directly into any of the model-fitting functions (`fitEcm`, `fitTirm`, or `fitTirmPartition`) available in `capwire`.

```
> res.ecm.sim <- fitEcm(data=simdata.ecm, max.pop=200)
> res.ecm.sim

$model
[1] "Equal.capture"


$likelihood
[1] -329.5837


$ml.pop.size
[1] 27


$cap.ind
```

```
[1] 3.703704


$sampled.ind
[1] 27


$sample.size
[1] 100


$max.pop
[1] 200
```

The methods currently implemented in `capwire` make assumptions about the distribution of individual capture probabilities within the population. The ECM assumes that all individuals are equally likely to be captured and the TIRM assumes that the population contains a mixture of two classes of capturabilities. A more realistic scenario is that the individual capture probabilities come from some underlying distribution. Data sets can be simulated under such a model with the function `simCapture`. `simCapture` has four arguments: `n` the number of individuals in the population; `s` the number of samples obtained; `dist.func` which is a function describing the distribution from which the capture probabilities are drawn from; and `return.cap.probs`, a logical argument specifying whether individual capture probabilities should also be returned (default=`FALSE`). `dist.func` can take the form of any function which takes the number of individuals $n$ and returns a vector of capture probabilities of length $n$. Functions for drawing from several common probability distributions are implemented in `capwire`. However, `simCapture` is designed to be flexible so that any distribution (discrete or continuous) can be specified by the user. To give an example, say one wants to assume that capture probabilites come from an exponential distribution. First specify the distribution with the function `drawCapRatesExp`:

```
> my.dist <- drawCapRatesExp(r=0.5)
```

Then simulate a data set consisting of 100 samples obtained from a population containing 30 individuals:

```
> exp.data <- simCapture(n=30, s=100, dist.func=my.dist, return.cap.probs=FALSE)
> head(exp.data)
```

```
  capture.class No.Ind
1             1      7
2             2      3
3             3      2
4             4      2
5             5      3
6             7      2
```

Similarly, capture probabilities can be drawn from a gamma (`drawCapRatesGamma`), uniform (`drawCapRatesUnif`), beta (`drawCapRatesBeta`) or geometric (`drawCapRatesGeom`) distribution.

If we want to simulate capture probabilities from a non-standard distribution, we can create a new function specifying the desired distribution which takes $n$, the number of individuals, as an argument and returns a vector of length $n$ capture probabilities. For example, say we want to use a truncated normal distribution where all capture probabilities are $> 0$, we can make such a function as follows:

```
> ## Create a function which takes argumens mean, sd, and trunc.point
> drawCapRates.truncnorm <- function(mean, sd, trunc.point){
+
+           ## Draw a large number of samples from a normal distribution
+           ## with specified mean and sd
+           normal.dist <- rnorm(100000, mean, sd)
+
+           ## Truncate the distribution at the point specified by trunc.point
+           trunc.dist <- normal.dist[which(normal.dist >= trunc.point)]
+
+           ## Create a function which draws n samples from trunc.dist
+           function(n){
+                   sample(trunc.dist, size=n, replace=TRUE)
+           }
+
+   }
```

We then specify the particular distribution we want to draw from

```
> my.dist <- drawCapRates.truncnorm(mean=2, sd=1, trunc.point=0)
```

Take a look at the function we have created:

```
> my.dist

function(n){
                  sample(trunc.dist, size=n, replace=TRUE)
          }
<environment: 0x12450b430>
```

And then simulate data from this distribution using `simCapture`. Here we specify that we want the individual capture probabilities returned.

```
> my.data <- simCapture(n=20, s=100, dist.func=my.dist, return.cap.probs=TRUE)
```

This now returns two elements: first, a data frame (`ind.data`) consisting of all individuals in the population, the number of times each individual was captured and the individual's capture probability. The second element (`class.data`) contains the capture class information and can be directly entered into other functions in `capwire`.

```
> fitEcm(my.data$class.data, max.pop=200)
```

```
$model
[1] "Equal.capture"


$likelihood
[1] -294.4439


$ml.pop.size
[1] 19


$cap.ind
[1] 5.263158


$sampled.ind
[1] 19


$sample.size
[1] 100


$max.pop
[1] 200
```

## 4 Notes

This package is available on the CRAN repository
http://cran.r-project.org/web/packages/capwire/.

## 5 Acknowledgments

We would like to thank Carisa Stansbury and Lisette Waits for helping to develop the methods implemented here and Paul Joyce for his statistical insights. M.W.P. would like to thank Luke Harmon for encouraging him to pursue projects far outside the scope of his dissertation.

## References

Banks, S. C., Hoyle, S. D., Horsup, A., Sunnucks, P. & Taylor, A. C. (2003). Demographic monitoring of an entire species (the northern hairy-nosed wombat, Lasiorhinus krefftii) by genetic analysis of non-invasively collected material. *Animal Conservation*, **6**, 101–107.

Miller, C. R., Joyce, P. & Waits, L. P. (2005). A new method for estimating the size of small populations from genetic mark-recapture data. *Molecular Ecology*, **14**, 1991–2005.

Pennell, M. W., Stansbury, C. R., Waits, L. P. & Miller, C. R. (submitted). capwire: A R Package for Estimating Population Census Size from Non-Invasive Genetic Sampling. *Molecular Ecology Resources*.

Stansbury, C. R., Ausband, D. E., Zager, P., Mack, C. M., Miller, C. R., Pennell, M. W. & Waits, L. P. (in prep.). Non-invasive genetic sampling of rendezvous sites and population estimation of grey wolves in Idaho, USA.