

CVCS Escapement Modeling

Arthur Barros

2024-09-25

Introduction

This document provides an overview of the modeling methods used to estimate the annual escapement, or the number of adults returning to spawn, of **Central Valley Chinook Salmon (CVCS)**. We will review the basic concepts of mark-recapture methods, delve into more complex models such as the Cormack-Jolly-Seber model, and demonstrate how to write scripts that calculate escapement estimates. The document relies heavily on R scripting and assumes that readers have a basic understanding of the language, as well as the ability to write and implement R scripts and functions. Additionally, it assumes familiarity with statistical concepts such as linear regression and probability.

This effort aims to translate and make clear the current **CVCS escapement model** utilized by the California Department of Fish and Wildlife (CDFW) and other groups within the **escapeMR** package. This model, R package, and application were developed by contractor Trent McDonald, Ph.D., in 2020 and have proven invaluable to state scientists in producing annual escapement estimates. However, despite its utility, it has become somewhat of a “black-box model,” where users input data and receive outputs without a clear understanding of the underlying mechanisms.

The documentation is intended to complement a CVCS escapement modeling workshop being held at the West Sacramento CDFW offices on Tuesday, August 20th, 2024. It will also likely be a “living” document, with continues updates and edits being made while modeling methods are tested and refined. Throughout this document, various chunks of R code will be presented in the following format:

```
print("hello reader")
```

```
[1] "hello reader"
```

These chunks are designed for users to copy and paste, or rewrite entirely, into their own R scripts to replicate the methods. The functions discussed in this document can also be found in the “CVCS_functions.R” script included in the workshop repository. Additionally, the code used in this document relies on the following R packages: `tidyverse`, `ggplot2`, `escapeMR`, `mra`, and `Rcpp`.

Additional Resources

For general help and assistance with understanding the CVCS escapement modeling methods outlined here feel free to contact me (the author) at Arthur.Barros@wildlife.ca.gov.

For a good introduction to writing functions in R I suggest this brief review of the use and writing of functions: <https://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/functions.html>. There are also plenty of resources on YouTube, such as this short video: <https://www.youtube.com/watch?v=CNPUDajM2X0>.

Another important coding method to understand are “for loops”, I suggest the following link as a good starting place for better understanding how to write these: <https://www.geeksforgeeks.org/for-loop-in-r/>.

An Introduction to Mark-Recapture Models

Mark and recapture is a common method for ecologists to estimate the size of a population of animals without needing to count each individual. Typically, an initial ‘marking’ period is conducted in which a sample of the population is captured and marked in some way so that those individuals can be recognized in the future. The marked individuals are then returned to the greater population and allowed time to mix back in. Later, another sampling period is conducted, during which a sample of the population is collected again, and the number of marked individuals in this sample is counted. This ratio can be used to estimate the size of the total population because the proportion of marked individuals in the second sample is assumed to be proportional to the number of total marked individuals in the population.

Regarding the CVCS escapement, mark-recapture surveys are conducted throughout the Central Valley in what is known as a “carcass survey.” Periodically, field crews survey Chinook salmon spawning grounds, looking for adult carcasses. These carcasses are marked with disk tags and returned to the river. The crews usually return within a week to see if the carcasses are still in the system and mark new carcasses that have arrived.

The Lincoln-Petersen estimate

The classic mark-recapture estimator is known as the **Lincoln-Petersen** [9, 6] and utilizes the equation:

$$\hat{N} = \frac{MC}{R} \quad (1)$$

- M is the number of individuals marked and released during the initial marking period.
- C is the number of individuals (both marked and unmarked) captured in the recapture period.
- R is the number of marked individuals recaptured in the later recapture period.
- \hat{N} is the estimate of population abundance.

In the diagram below (Figure 1), 7 fish from the population are captured and marked (K) during an initial capture and marking period. Then during a follow up recapture period, 7 fish are again captured (n), and of those 3 have markings (k). This provides us with an estimate of a total population of roughly 16 fish.

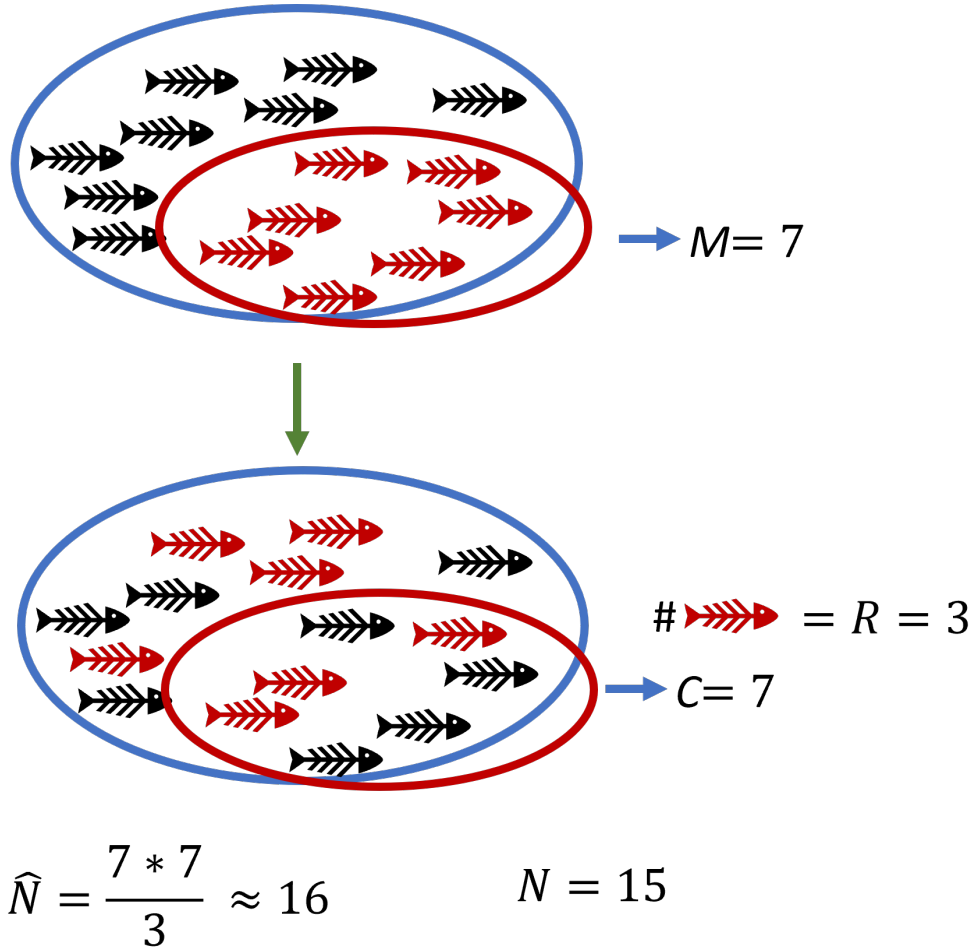


Figure 1: A conceptual diagram of the classic Lincoln-Petersen mark-recapture method.

There are several important assumptions of the Lincoln-Petersen method:

1. The population is closed so that N is constant during the study period. No deaths, no births, no immigration or emigration.
2. The probability of capture is equal across all individuals.
3. Fish do not lose their markings between the two sampling periods.
4. All marked fish recaptured during the second sampling period are accurately enumerated and recorded.

The Lincoln-Petersen method was expanded upon upon by Zoe Schnabel [10] with the **Schnabel estimator**, which allows for multiple periods of capture and recapture instead of just two.

$$\hat{N} = \frac{\sum_t (C_t M_t)}{\sum_t R_t} \quad (2)$$

This allows for marking to occur during multiple sampling periods t . One primary advantage of this method is multiple sampling periods allow for the detection of violation of the above assumptions. Regression of the proportion of marked animals to previously marked animals ($\frac{R_t}{C_t} \sim M_{t-1}$) will be straight if assumptions are met, and curved if the assumptions are violated.

If we examine our the annual carcass surveys performed across the Central Valley for returning Chinook salmon, we can easily identify ways in which the system violates the assumptions of the Lincoln-Petersen method:

- **Populations are open:** throughout the survey period adult Chinook arrive in the survey area, spawn, and die. At the same time spawned out carcasses that may or may not have been marked and counted are removed from the system either by scavengers, decay, or being flushed downstream.
- **Probability of capture is not homogeneous:** probability of "capture" for individual Chinook carcasses is really the probability it is spotted by survey field crews. This can be dependent on a number of factors including carcass size, carcass sex, water flow rates, and turbidity.

The above violations of the Lincoln-Petersen estimates could cause large biases in abundance estimates, and suggests that using this closed population model is ill advised.

The Cormack-Jolly-Seber models

To deal with open populations and heterogeneous capture probabilities we can utilize a Cormack-Jolly-Seber (CJS) model [3, 5, 11], which can deal with population changes during the survey period, as well as differences in capture probability among individuals. The CJS method does not directly provide estimates of abundance, instead it utilizes multiple sampling events during the survey period and allows us to calculate two important parameters:

- ϕ_{ij} : the probability carcass i "survives" (remains in the system) from period j to $j + 1$.
- p_{ij} : the probability carcass i is captured during period j .

We will generate estimates for p_{ij} and ϕ_{ij} using the CJS Maximum Likelihood Estimation methods later, but for now know that the probability of capture can be used to estimate the population size at period j using the "**Horvitz-Thompson estimator**" [4, 8]:

$$\hat{N}_j = \sum_{i=1}^n \frac{h_{ij}}{\hat{p}_{ij}} \quad (3)$$

Where h_{ij} is either a 0 or 1 capture indicator for carcass i at period j , and n is the number of all carcasses observed over all survey periods. Basically it's saying the abundance of carcasses at time j is equal to the sum of all the carcasses captured during that period divided by their capture probability.

Typically the total population estimate \hat{N} is calculated by taking the average value of \hat{N}_j , however there are problems with this method when many individuals are entering and leaving the population during the survey. This can be improved by using a *superpopulation* modification [2] to estimate total escapement as:

$$\hat{N}_{escapement} = \hat{N}_2 \frac{\ln(\tilde{\phi}_1)}{\tilde{\phi}_1 - 1} + B_2^* + B_3^* + \dots + B_{S-2}^* \quad (4)$$

Where S is the total number of all sampling occasions and B_j^* , the adjusted number of “births” (new adults entering the system and dying) between j and $j + 1$, is:

$$B_j^* = \tilde{B}_j \frac{\ln(\tilde{\phi}_j)}{\tilde{\phi}_j - 1} \quad (5)$$

\tilde{B}_j is the total “births” for period j :

$$\tilde{B}_j = \hat{N}_{j+1} - \tilde{\phi}_j [\hat{N}_j - (n_j - R_j)] \quad (6)$$

and R_j is the number of carcasses released with disctags during period j , n_j is the number of carcasses captured during period j , and \hat{N}_j is the estimate of abundance during period j derived from the Horvitz-Thompson estimator in Equation 3.

The CVCS Escapement Model

The Data

For the CVCS carcass surveys, carcasses are captured over k occasions, usually weekly sampling events, marked with disctags, and deposited back into the system for future recapture. The CJS model used for CVCS carcass surveys incorporates three different data inputs to estimate ϕ_{ij} and p_{ij} : capture histories, “chops” records, and covariate data.

Capture histories are the records of capture and recapture for a given carcass and usually look similar to:

$$\begin{bmatrix} DiscTag & S1 & S2 & S3 & S4 & \dots & k \\ 1601 & 1 & 0 & 1 & 1 & \dots & \\ 1602 & 1 & 0 & 0 & 1 & \dots & \\ 1603 & 0 & 0 & 1 & 1 & \dots & \\ 1604 & 1 & 1 & 1 & 2 & \dots & \end{bmatrix}$$

Here each record is for a tagged carcass and the subsequent sampling periods j . A value of 0 indicates the carcass was not captured, a value of 1 indicates the carcass was captured, and a value of 2 indicates the carcass was captured and “chopped” (beheaded) and removed from the system.

Chops is the record of the number of carcasses removed from the system upon first capture. This is done when the carcass is deteriorated beyond identification or measurement, and wont last long enough for further recaptures. The data can look like:

$$\begin{bmatrix} S1 & S2 & S3 & S4 & S5 & \dots & k \\ 0 & 4 & 27 & 18 & 20 & \dots & \end{bmatrix}$$

With just one row of total chopped counts for each sampling period.

Covariate data is the record of relevant covariates for each captured carcass or each survey period. For CVCS surveys, the covariates currently used are the sex and the length of the carcass. In the future we hope to incorporate environmental variables such as temperature and flow. A covariate table usually looks like:

$$\begin{bmatrix} DiscTag & Sex & Length \\ 1601 & F & 200 \\ 1602 & F & 223 \\ 1603 & M & 195 \\ 1604 & M & 180 \end{bmatrix}$$

With the covariates recorded for each carcass marked with a disctag, and used to help determine different values of p and ϕ for each carcass.

Probabilities of capture and survival

While Equation 4 used to estimate abundance using carcass survey data may seem relatively straight forward, they are heavily dependent on estimates of p_{ij} and ϕ_{ij} , which are much more complex to get.

With these probabilities the “capture history” of a carcass i can be represented for each sampling period j as:

$$j = 1 \xrightarrow{\phi_{1i}} j = 2 \xrightarrow{\phi_{2i}} j = 3 \xrightarrow{\phi_{3i}} j = 4 \xrightarrow{\phi_{4i}} j = 5 \dots$$

$$p_{2i} \quad p_{3i} \quad p_{4i} \quad p_{5i}$$

Let's say we have the following ,very simple, set of capture histories for four carcass, across four different sampling periods:

$$\begin{bmatrix} DiscTag & S1 & S2 & S3 & S4 \\ 1601 & 1 & 0 & 1 & 1 \\ 1602 & 1 & 0 & 0 & 1 \\ 1603 & 0 & 0 & 1 & 1 \\ 1604 & 1 & 1 & 1 & 2 \end{bmatrix}$$

If we look at the first row of data, the capture history for the carcass with disc-tag number 1601, we see it was captured and marked during period 1, not captured in period 2, and then captured again in periods 3 and 4. We can assign a probability of observing this capture history as:

$$P(1011|\text{release at period 1}) = \phi_1(1 - p_2)\phi_2p_3\phi_3p_4$$

In the above, $(1 - p_2)$ is the probability that the carcass is not captured at period $j = 2$. Now we can do the same with the rest of the example capture histories that we saw:

$$P_1(1011|\text{release at period 1}) = \phi_1(1 - p_2)\phi_2p_3\phi_3p_4$$

$$P_2(1001|\text{release at period 1}) = \phi_1(1 - p_2)\phi_2(1 - p_3)\phi_3p_4$$

$$P_3(0011|\text{release at period 3}) = \phi_3p_4$$

$$P_4(1112|\text{release at period 1}) = \phi_1p_2\phi_2p_3\phi_3p_4$$

In the above note the 2 at the end of the last capture history representing a chopped event. Chopped carcasses will be dealt with later. Also note that for P_3 the likelihood conditions on the initial observation, and excludes estimates of ϕ or p before that period.

We can use those equations to estimate the total likelihood of observing all the capture histories we saw as:

$$L = P_1 * P_2 * P_3 * P_4 \quad (7)$$

Or as the product of all n of the capture history probabilities [1]:

$$L = \prod_{i=1}^n P_i \quad (8)$$

To make the computing of this probability easier we can log-transform our likelihoods so that:

$$\ln(L) = \sum_{i=1}^n \ln(P_i) \quad (9)$$

This is known as the “log-likelihood” of the probability. Using this simple example above, we have a way to estimate the probability of observing a set of given capture histories if we have the capture (p) and survival (ϕ) probabilities associated with each capture history.

Of course, we don’t know these probabilities, and they will likely vary across different covariates for each carcass and each period. Possible covariates could include size of carcass, carcass sex, or environmental variables such as temperature, flow, or turbidity. For CVCS work, we currently utilize covariates of sex and length to try and estimate capture and survival probabilities. For our example we will consider a CJS model in which capture probability is a function of length, and survival probability is a function of sex.

$$P(\text{capture}|\text{length}) = \hat{p}_{ij} \quad (10)$$

$$P(\text{survival}|\text{sex}) = \hat{\phi}_{ij} \quad (11)$$

These functions can best be represented with logistic regression models:

$$\hat{\phi}_{ij} = \frac{e^{\Upsilon_0 + \Upsilon_1 x_{ij}}}{1 + e^{\Upsilon_0 + \Upsilon_1 x_{ij}}} \quad (12)$$

$$\hat{p}_{ij} = \frac{e^{\beta_0 + \beta_1 y_{ij}}}{1 + e^{\beta_0 + \beta_1 y_{ij}}} \quad (13)$$

Where x_{ij} is the sex of a given carcass, and y_{ij} it's length. To ease in estimating the coefficients Υ and β we can turn the above into linear models:

$$\ln\left(\frac{\hat{\phi}_{ij}}{1 - \hat{\phi}_{ij}}\right) = \Upsilon_0 + \Upsilon_1 x_{ij} \quad (14)$$

$$\ln\left(\frac{\hat{p}_{ij}}{1 - \hat{p}_{ij}}\right) = \beta_0 + \beta_1 y_{ij} \quad (15)$$

with $\ln(\frac{\hat{\phi}_{ij}}{1 - \hat{\phi}_{ij}})$ and $\ln(\frac{\hat{p}_{ij}}{1 - \hat{p}_{ij}})$ as our logit functions, used as “link functions” to model the probabilities as linear functions of covariates [1].

You can see above that if we can estimate the coefficients of those two equations, for a given sex or length (x_{ij}, y_{ij}) we can estimate the corresponding probability of survival or capture. To estimate the coefficients of Υ_0 , Υ_1 , β_0 , and β_1 , we can use a method know as **maximum log-likelihood estimation (MLE)**. To do this we first need to be able to estimate the log-likelihood of our model on a given set of coefficients and capture histories, basically: what is the log-likelihood of the capture histories we observed given a set of coefficients?

If we refer back to Equation 9, we can remember that the log-likelihood of all the observed capture histories is equal to the sum of the log-likelihood of each individual capture history P_i . The log-likelihood of the capture history for a given carcass i can be given as the sum of its capture and survival probabilities from its first to last encounter, plus χ_i : the probability it is not captured after its last release [7]. This individual capture history log-likelihood can be represented as:

$$\ln(P_i) = \sum_{j=first}^{last} [h_{ij} * \ln(\hat{p}_{ij}) + (1 - h_{ij}) * \ln(1 - \hat{p}_{ij}) + \ln(\hat{\phi}_{j-1,i})] + \chi_i \quad (16)$$

Where h_{ij} is 1 if the carcass was observed during j , and 0 otherwise. χ_i can be represented as:

$$\chi_i = (1 - \hat{\phi}_{last,i}) + \prod_{j=last}^n \hat{\phi}_{ij} * (1 - \hat{p}_{j+1,i}) * (1 - \hat{\phi}_{(n-1),i}) \quad (17)$$

The first part of Equation 17 ($1 - \hat{\phi}_{last,i}$) represents the probability the given carcass is removed from the system immediately after its last observation. The second part of Equation 17 is the

product of all probabilities that carcass remained in the system but wasn't observed during all subsequent observation periods.

Now we have Equation 16 that provides us with the log-likelihood of observing a given capture history with specific survival and capture probabilities. We have also shown that values of ϕ_{ij} and p_{ij} can be linearly regressed across the coefficients Υ_0 , Υ_1 , β_0 , and β_1 (Equation 14 and Equation 15). So now we can plug in values for the coefficients to give us capture and survival probabilities for a given carcass i during period j , which we can put into Equation 16 to give us the log-likelihood of observing the capture history of that carcass using those coefficients. This process can be repeated for each carcass and summed to give us the total log-likelihood of all capture histories for all carcasses in Equation 9. We can then repeat this whole process with new coefficient values until we find the coefficients of Equation 14 and Equation 15 that provide us with the **maximum log-likelihood**. For a good explanation of how maximum likelihood works, I highly recommend this YouTube video: <https://www.youtube.com/watch?app=desktop&v=XepXtl9YKwc>, which does a good job summarizing the basic concept.

Of course, repeating the above steps to find the highest log-likelihood for even just one carcass history would take a long time to do manually, and we need to find the coefficients that will provide the highest likelihood across all of our capture histories and covariates. Fortunately, we can use an optimization algorithm to automate this process, and find the coefficients that produce the maximum likelihood estimation for us. I won't expand on how MLE optimization algorithms work specifically, but we will be using optimization functions that are available in base R, which will be discussed later.

Once an optimization algorithm provides us with the coefficients that give us the maximum likelihood of observing our capture history data with the given covariates, we can use it to calculate the capture probability for each carcass (\hat{p}_{ij}) given it's length and sex. Those probabilities can then be utilized with the Horvitz-Thompson estimator (Equation 3) to estimate total escapement for each period, and in turn providing an overall escapement estimate.

Coding the CVCS Escapement Model

Reviewing the escapeMR package

The escapeMR package (<https://gitlab.com/tmcd/escapemr>), written by Trent McDonald, Ph.D., was designed specifically for CDFW CVCS escapement surveys as a user-friendly Rshiny interface. The Rshiny application itself can either be accessed online here: <https://tmcd.shinyapps.io/escapemr/>, or by loading the escapeMR package into R and using the `escapeMR()` call to load the shiny app.

```
install.packages("escapeMR")
library(escapeMR)
escapeMR()
```

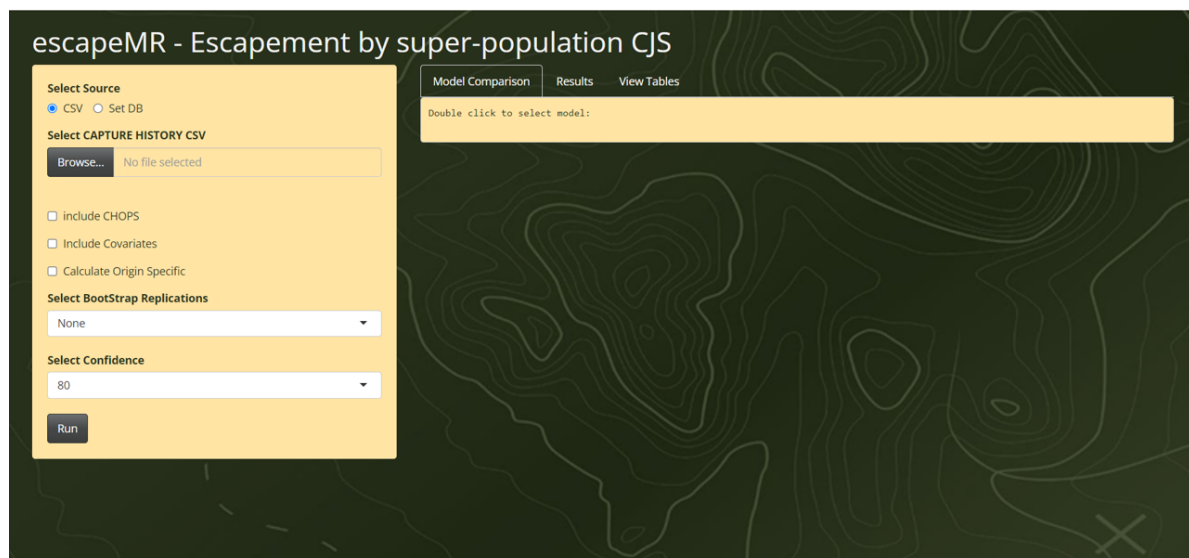


Figure 2: The user interface of the escapeMR shiny application.

The shiny application allows users to upload capture histories, chops data, and covariate data as .csv files. Then users can specify how the capture and survival probabilities are modeled using either sex or length covariates, and set bootstrap replications and target confidence intervals before running the model. After processing, the model outputs an estimate of total escapement with upper and lower bounds, and a histogram and summary of bootstrapping results.

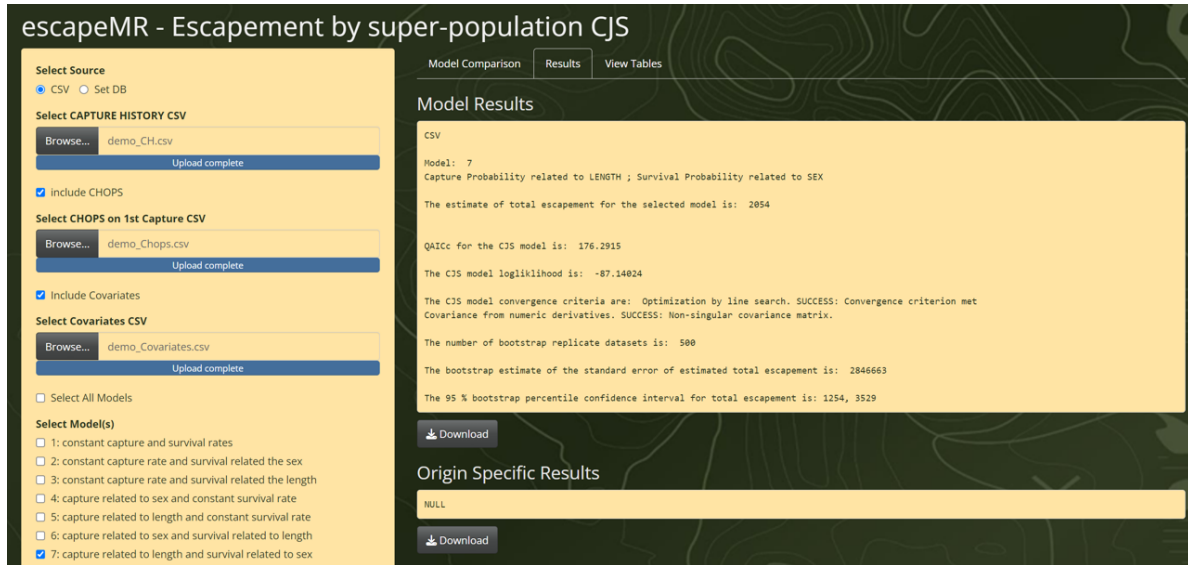


Figure 3: Reported outputs of escapeMR shiny application.

Before attempting to code the equations and methods described above for our CVCS escape-ment model, it is helpful to review, at least at an overview level, how the **escapeMR** model and code works. This helped me develop the code and functions I wrote, and also identify potential restrictions and possible avenues for improvement.

The **escapeMR** package is actually a “front-end” package that loads and formats user data before passing it to a “back-end” package **mra**, which is the “Mark Recapture Analysis” package (<https://cran.r-project.org/web/packages/mra/mra.pdf>). The ‘mra’ package was also written by Trent McDonald, prior to his work with CDFW on the **escapeMR** package, and is a more generalized mark-recapture package without the user friendly interface.

Let’s first examine the **escapeMR** package by examining it’s gitlab repository: <https://gitlab.com/tmcd/escapemr/-/tree/master/R>. In this repository are several different R scripts that the package utilizes to load, format, and pass data to the **mra** package. I will make note of the ones important to our efforts here:

- **escapeMR.R** - loads and runs the shiny application for user interface with the package.
- **askForData.R** - prompts user to load catch history, chops, and covariate data, and then formats it for use in the **mra** package.
- **CJSscript.R** - a script based version of the escapeMR shiny application that can be run in R without needing to use shiny. Takes user inputs and runs the “esc_model.R” script based on those inputs (model selection, bootstrap iterations, and confidence intervals).

- **esc_model.R** - the workhorse script of the **escapeMR** package, this passes the formatted data to the **mra** package using the `F.cjs.estim()` call. The script then receives the estimates of \hat{N}_j , \hat{p} , and $\hat{\phi}$, and uses them to estimate final escapement.

The **mra** package repository can be accessed here: <https://github.com/tmcd82070/MRA/tree/master>, note its been sometime since it was updated. There are a lot of folders and scripts in this repository, so again I will highlight the one's I referenced in my efforts, and how they are applicable to our work.

- **F.cjs.estim.R** - receives the input data from the **escapeMR** package, and set's initial parameters to use with MLE to make our estimates of \hat{N}_j , \hat{p} , and $\hat{\phi}$. This script then passes the initial parameters and relevant data to a `.Fortran("cjsmod")` call. This then utilizes the FORTRAN coding language and the “Mrawin.f90” script in the package to do all the leg-work of MLE.
- **Mrawin.f90** - this is FORTRAN script of various subroutines used in finding the MLE of our model. the “cjs” subroutine takes the formatted data and passes it to other subroutines which replicate many of the equations we’ve discussed so far.
 - Of important note is the subroutine **VA09AD**, which is the MLE algorithm that the **mra** package uses to find the most likely survival and capture probability coefficients. I have not been able to find out much information about this algorithm, other than that is a minimization algorithm (so the **mra** package just multiplies it by -1), and that it originates from the Harwell Subroutine Library. I was also unable to replicate this method in R, and am instead relying on the R `optim()` call for maximization. In a later section I will discuss the benefits and drawbacks of this.

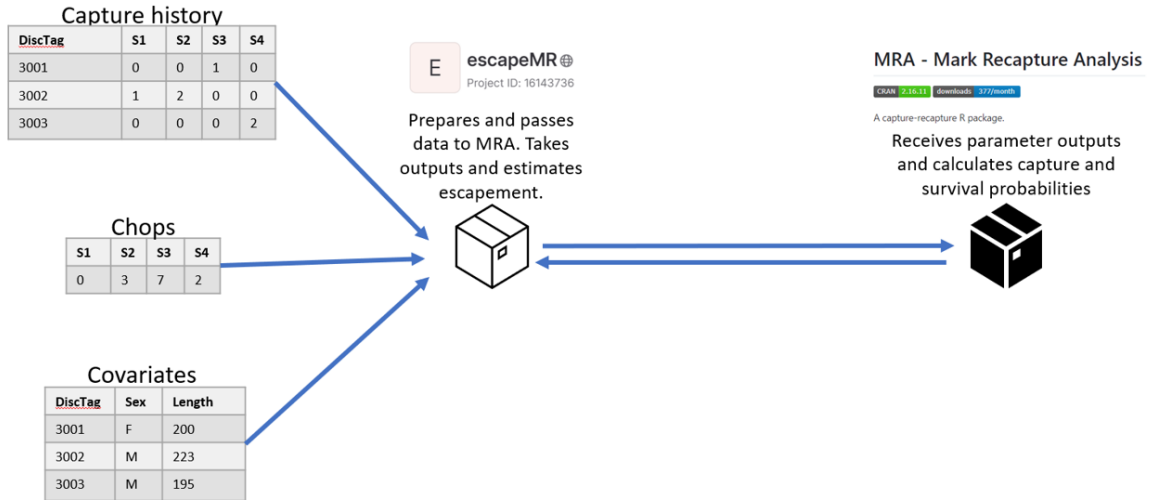


Figure 4: Conceptual model overview of how the **escapeMR** and **mra** package operate.

Reviewing some example data

To start our coding section, we'll begin by reviewing some data I've prepared as a demonstration. We have three different .csv files in the "data" folder of the R project, "demo_CH.csv", "demo_Covariates.csv", and "demo_Chops.csv". we can read these into our environment with:

```
library(tidyverse)
ch<-read.csv('data/demo_CH.csv')
covars<-read.csv('data/demo_Covariates.csv')
chops<-read.csv('data/demo_Chops.csv')
```

Let's review our different data sets, first with the capture history data we loaded in as 'ch'. For this demo we are utilizing a smaller data set with just 153 capture histories to keep the processing time lower. The capture history should have the expected format, with the first column showing the individual disc-tag for each carcass. The following columns contain the observation values for each survey period, with either a 0, 1, or 2 representing no-observation, observation, or observation and removal, respectively.

```
head(ch)
```

	DiscTag	X1	X2	X3	X4	X5
1	111	0	1	2	0	0
2	114	0	1	0	0	2
3	12	0	1	0	0	0
4	123	0	1	0	0	0
5	127	0	1	0	0	0
6	129	0	1	0	0	0

Next we can review our covariate data "covars", containing the sex and fork-length for each carcass that has a corresponding disc-tag and capture history.

```
head(covars)
```

	DiscTag	sex	length
1	111	M	755
2	114	M	737
3	12	F	718
4	123	F	763
5	127	M	754
6	129	F	728

Finally we'll review the "chops" data, which is just one record, with the number of carcasses for each survey period that were removed from the system on first observation. This carcasses won't have capture histories represented in the "ch" data set.

```
head(chops)
```

```
SurveyMetaID X1 X2 X3 X4 X5
1             1 10 42 95 61 13
```

Coding the superpopulation model

When coding our equations here, we'll start at the end first. We'll first write the Horvitz-Thompson estimator (Equation 3), that we'll then use in our superpopulation model (Equation 4) to get our final estimate of escapement \hat{N} . The `Horvitz_Thompson()` function I've created below does this in the following:

- Loads `p_hat`, a matrix of capture probabilities \hat{p}_{ij} , and `ch`, our capture history dataframe.

i Note

At this point we haven't calculated values for \hat{p}_{ij} yet, that will be done further down the line.

- Finds `nan` and `ns` ('number of animals' and 'number of surveys') of the capture history.
- Makes an empty matrix `n_mat` with dimensions `nan * ns` that it fills with a for-loop with either a 0, if carcass i wasn't observed in that period, or with $\frac{1}{\hat{p}_{ij}}$.
- Calculates \hat{N} for each period j by summing all the records of `n_mat` during period j .

```
Horvitz_Thompson<-function(p_hat,ch){
  nan=nrow(ch)
  ns=ncol(ch)
  N_hat<-list()
  n_mat<-matrix(NA,nan,ns)
  for(j in 1:ns){
    for(i in 1:nan){
      n_mat[[i,j]]<-if(ch[[i,j]]>=1){
        1/p_hat[[i,j]]
      } else {0}
    }
    N_hat[[j]]=sum(n_mat[,j])
  }
}
```



```

}
return(N_hat)
}

```

If we next refer back to Equation 4, we see we'll also need estimates of B_j^* or the number of new carcasses entering the system between j and $j + 1$, which is given by Equation 5 and Equation 6. We'll do all of that in one go here:

```

B_star<-function(ch,s_hat,p_hat){
  R=n=list()
  nan=nrow(ch)
  ns=ncol(ch)
  for(j in 1:ns){
    d<-ch #select just the capture matrix data
    R[j]<-as.numeric(length(which(d[j]==1)))
    n[j]<-as.numeric(length(which(d[j]==1))+length(which(d[j]==2)))
  }

  N_hat<-Horvitz_Thompson(p_hat,ch)
  #next B1, or total number of births for each period
  B1<-list()
  for(j in 2:(ns-2)){
    B1[j]<-N_hat[[j+1]]-mean(s_hat[,j])*(N_hat[[j]]-(n[[j]]-R[[j]]))
  }

  #next Bstar, number of births adjusted for those entering the system between
  #j and j+1, but not surviving to j+1
  Bstar<-NULL
  for(j in 2:(ns-2)){
    Bstar[j]<-as.numeric(B1[[j]]*(log(mean(s_hat[,j]))/(mean(s_hat[,j])-1)))
  }
  Bstar<-Bstar[-(1)]
  return(Bstar)
}

```

The above function `B_star()` runs through a few steps:

- First it uses a for-loop to calculate R_j , the number of carcasses released with tags during j ; and n_j , the number of carcasses captured during j .
- Then it uses the `Horvitz_Thompson()` function we wrote to estimate \hat{N}_j .

- It then uses `s_hat` (a matrix of $\hat{\phi}_{ij}$ that like \hat{p}_{ij} we haven't yet written a function to produce), and n_j and R_j to estimate $B1$ (\tilde{B}_j) the total number of births for period j .
- Finally it uses $B1$ and `s_hat` to calculate B_j^* , the number of births between j and $j + 1$.

To wrap everything up we can write a final function `total_escapement()`, which uses what we've created so far to estimate $\hat{N}_{escapement}$ (Equation 4):

```
total_escapement<-function(ch,s_hat,p_hat){
  N_hat=Horvitz_Thompson(p_hat,ch)
  Bstar=B_star(ch,s_hat,p_hat)
  escapement<-N_hat[[2]]*(log(mean(s_hat[,1]))/(mean(s_hat[,1])-1)) +
    sum(Bstar,na.rm=T)
  return(escapement)
}
```

The `total_escapement()` function uses the `Horvitz_Thompson()` and `B_star()` functions we just created and returns our estimate of $\hat{N}_{escapement}$. Note that the above requires we have both `s_hat` and `p_hat`, which will be matrices with dimensions `nan` and `ns` for values of $\hat{\phi}$ and \hat{p} for each carcass i and period j . We don't yet have a way to estimate those, so that will be the next big step.

Getting estimates of $\hat{\phi}$ and \hat{p}

So far we've written functions that will allow us to take values for $\hat{\phi}$ and \hat{p} and produce our final reported value of $\hat{N}_{escapement}$. This brings us to the complicated part of estimating $\hat{\phi}_{ij}$ and \hat{p}_{ij} for each carcass and survey period.

If we recall Equation 14 and Equation 15, we had ways of estimating $\hat{\phi}_{ij}$ and \hat{p}_{ij} for each carcass based on linear regression for given covariates, in our example sex (x_{ij}) and length (y_{ij}). So if we can figure out values for the coefficients Υ_0 , Υ_1 , β_0 , and β_1 , we have a linear equation that solves for $\hat{\phi}_{ij}$ and \hat{p}_{ij} .

Now let's write a function that replicates Equation 14 and Equation 15, which will utilize coefficients for Υ and β to estimate values for $\hat{\phi}$ and \hat{p} .

```

pro_capsur<-function(i,j,ch, beta,cap_X,surv_X){
  p=length(beta)

  #purpose: evaluate probability of capture and survival for each animal i
  cap_beta<-beta[1:(p/2)]
  surv_beta<-beta[((p/2)+1):p]

  zp<-exp(cap_beta[1]*1+cap_beta[2]*cap_X[i,j])
  zs<-exp(surv_beta[1]*1+surv_beta[2]*surv_X[i,j])

  p.hat<-zp/(1+zp)
  s.hat<-zs/(1+zs)

  est_list<-list('p.hat'=p.hat,'s.hat'=s.hat)
  return(est_list)
}

```

The plan of the above function is to use it on each capture history record to estimate the capture and survival probabilities for each carcass during each period. The `pro_capsur()` function incorporates several variables: `i` and `j` of course represent the given carcass and survey period; `beta` is a list of our four coefficients (Υ_0 , Υ_1 , β_0 , and β_1 , see Equation 14 and Equation 15); and `cap_X` and `surv_X` are each matrices of the capture and survival covariates from the `covars` dataframe.

The function begins by producing `nan` and `ns` from the `ch` dataframe, and assigning the capture coefficients and survival coefficients as `cap_beta` and `surv_beta` respectively. We also save a value of `p` which is just the number of coefficients we are using, in this case four. It then calculates `zp` and `zs` for a given carcass and survey i, j . Here `zs` is $\ln(\frac{\hat{\phi}_{ij}}{1-\hat{\phi}_{ij}})$ and `zp` is $\ln(\frac{\hat{p}_{ij}}{1-\hat{p}_{ij}})$, our logistic link functions. Finally the function uses those values of `zs` and `zp` to estimate $\hat{\phi}$ and \hat{p} , which it then adds to a list called `est_list` and labels them as `s.hat` and `p.hat` respectively.

Now that we have a function that can give us values of $\hat{\phi}_{ij}$ and \hat{p}_{ij} , we can utilize it to give us the log-likelihood for each individual capture history (P_i) we have observed using Equation 16.

If we refer to Equation 16, you'll remember one important aspect to calculating the log-likelihood of a given capture history is that it sums all the probabilities of capture and survival from the first to the last detection event for a given carcass. This means we'll need a way to find the first and last observation for a given carcass in our `ch` dataframe. To help in this process we can write a function that does this work for us and can be used later:

```

location<-function(nan,ns,ch){
  #purpose: compute first and last capture for each animal
  first <- rep(0, nan)
  last <- rep(0, nan)
  for(i in 1:nan){
    findch=TRUE
    for(j in 1:ns){
      if(ch[i,j]>=1){
        if(findch==T && (j<ns)){
          first[i]=(j+1)
          findch=FALSE
        }
        last[i]=j
      }
    }
  }
  est_list<-list('first'=first,'last'=last)
  return(est_list)
}

```

The `location()` function utilizes `ch` our capture history data frame, and runs a for-loop through each record and survey, looking for the first and last periods j that carcass i was observed during. It then returns two lists, `est_list$first` and `est_list$last` which list the first and last encounters respectively. While reviewing the above function, you may wonder why the record for `first[i]` is equal to $j+1$, where j is the first time it was encountered. This is because our estimate of likelihood for the given capture history is conditioned on the initial observation of the carcass, so when estimating capture probability we want `first[i]` to be the occasion after the first encounter, which is the period with the first estimable capture probability.

Our next function is the longest and most complicated, and aims to replicate Equation 9. The end output is the estimated total log-likelihood of observing the provided capture histories with a given `beta`, our four coefficients (Υ_0 , Υ_1 , β_0 , and β_1). I've put the whole function in the following code chunk, but have included commented parts which I break down below:

```

CJS_loglik<-function(beta,cap_X,surv_X,ch){

  #Part 1: Initialize variables
  xlnlik <- 0
  nan=nrow(ch)
  ns=ncol(ch)

```

```

#Part 2: get locations
first<-location(nan,ns,ch)$first
last<-location(nan,ns,ch)$last

#Part 3: Calculating total log-likelihood
for(i in 1:nan){

  #set initial values
  sum1=0
  sum2=0
  vp_ij<-NA
  vs_ij<-NA

  if(first[i]==0){
    init_cap=ns+1
    init_surv=ns+1
  } else if(first[i]>0){
    init_cap=first[i]
    init_surv=first[i]-1
  }

  #Part 3.1: Compute all probabilities of capturing carcass i,
  #from first occasion to end
  if (init_cap<=ns){
    for (j in init_cap:ns) {
      vp_ij[j] <- pro_capsur(i,j,ch,beta,cap_X,surv_X)$p.hat
    }
  }

  #Part 3.2:Compute all probabilities of carcass i surviving from j to j+1,
  #from first occasion to end
  if(init_surv<ns){
    for(j in init_surv:(ns-1)){
      vs_ij[j] <- pro_capsur(i,j,ch,beta,cap_X,surv_X)$s.hat
    }
  }

  #Part 3.3: compute log-likelihood contribution for animal i
  if((first[i]>0 && first[i]<=last[i])){
    for(j in first[i]:last[i]){#for each of the observations for the animal,
      #from first contact to last
      hij <- ifelse(ch[i, j] >= 1, 1, 0)
      #for each observation, calculate a running sum of

```

```

    #ln(L)=ln(p)+(ln(1-p))+ln(survival)
    sum1=sum1+hij*log(vp_ij[j])+
      (1-hij)*log(1-vp_ij[j])+
      log(vs_ij[j-1])
  }
}

#Part 3.4: Find second part of likelihood, Chi
#Chi = probability of animal i not being seen again after last i
#If animal died on capture before release, prob of not seeing again is 1
if(ch[i,last[i]]>=2){
  sum2=0
}else if(last[i]>0 && last[i]<ns){
  sum2=1-vs_ij[last[i]] #chance it died at last[i]
  for(ii in (last[i]+1):ns){#for each obs after last[i]
    prod<-1
    for(jj in last[i):(ii-1)){
      prod<-prod*vs_ij[jj]*(1-vp_ij[jj+1])#product of all chances carcass
      #survived but wasn't seen for each observation point between
      #last and ns-1
    }
    if(ii<ns){
      prod<-prod*(1-vs_ij[ii])
    }
    sum2<-sum2+prod
  }
  sum2<-log(sum2)
}

#Part 3.5: estimate total likelihood of carcass capture history
xlnlik<-xlnlik+sum1+sum2
}
return(xlnlik)
}

```

- Part 1: initializes some variables used throughout the rest of the function, including the initial value of `xlnlik` (which will be our final output) to 0.
- Part 2: the `CJS_loglik()` function creates a `first` and `last` list for our capture histories using the `location()` function we just wrote.
- Part 3: initializes a for-loop over each capture history record in `ch`, and sets initial values for the loop. Also sets values for `init_cap` and `init_surv`, which will tell the next steps

which period j to begin with when estimating values for $\mathbf{vp_ij}$ ($\hat{\phi}_{ij}$) and $\mathbf{vs_ij}$ (\hat{p}_{ij}) for each carcass.

i Note

For the above if statement, `first[i]==0` is used because when the first encounter ends up being the last period of observation, the `location()` function we wrote will give it a first value of 0.

- Part 3.1: computes values of \hat{p} for carcass i from the first observation to the last on utilizing the `pro_capsur()` function we wrote earlier, with the given coefficients `beta`. It assigns those values to the vectors `vp_ij`.
- Part 3.2: computes values of $\hat{\phi}$ for carcass i from the first observation to the last on utilizing the `pro_capsur()` function we wrote earlier, with the given coefficients `beta`. It assigns those values to the vectors `vs_ij`.
- Part 3.3: uses the estimates of \hat{p} and $\hat{\phi}$ to compute the log-likelihood contribution for carcass i . This is the first part of the equation calculating the log-likelihood of everything up to χ . Saves output as value `sum1`.
- Part 3.4: estimates the log-likelihood of χ as `sum2`, the chance a carcass remains in the system but is not recaptured after it's final observation.
- Part 3.5: Adds `sum1` and `sum2` together into `xlnlik`, the running sum of total likelihood. Once the for-loop is complete, the function returns the final value of `xlnlik`, the total sum of all the log-likelihood of observing all the observed capture histories given the provided covariates and coefficients in `beta` (Υ_0 , Υ_1 , β_0 , and β_1).

Now let's review the functions we've made so far:

- The `total_escaping()` function, which utilizes our capture history (`ch`); `p_hat` and `s_hat`, matrices of \hat{p}_{ij} and $\hat{\phi}_{ij}$; as well as the `Horivtz_Thompson()` and `B_star()` functions to produce a final estimate of \hat{N} for the study period.
- The `pro_capsur()` function was made to estimate the individual values of \hat{p}_{ij} and $\hat{\phi}_{ij}$ which populate the `p_hat` and `s_hat` matrices using the covariates provided and the coefficients (Υ_0 , Υ_1 , β_0 , and β_1) stored in the `beta` list..
- The `location()` function which finds the first and last encounter for each carcass. This function is used in the `CJS_loglik()` function.
- The `CJS_loglik()` function that utilizes the `beta`, `ch`, `cap_X` and `surv_X` data to estimate the total log-likelihood of the given capture histories and covariates being observed, given the provided coefficients.

Recall that we can use Equation 16 to find the coefficients in Equation 14 and Equation 15 that produce the highest likelihood of occurring, or the maximum log-likelihood estimation. We can leverage the `optim()` call in R, a general-purpose optimization function, to pass different values of `beta` into the `CJS_loglike()` function to find the set that maximizes the log-likelihood.

Before we can use the `optim()` function though, we need to define a wrapper function that takes the `beta` argument and passes the other required values (`ch`, `cap_X`, and `surv_X`) to the `CJS_loglik()` function. Then `optim()` can call the wrapper function with different values of `beta` to find the coefficients with the maximum log-likelihood. This wrapper function can be written like so:

```
CJS_loglik_wrapper <- function(beta, cap_X, surv_X, ch) {  
  -CJS_loglik(beta, cap_X, surv_X, ch) # Return the negative log-likelihood  
}
```

i Note

In the above `CJS_loglike_wrapper()` function, it returns the negative of the log-likelihood output, this is because by default the `optim()` function minimizes the negative of the log-likelihood.

Bringing it all together

Now we've written all the functions to replicate the equations covered, and allow us to find the most likely coefficients for Equation 14 and Equation 15 that will produce the observed capture histories and covariates. We can now take the functions we created, and apply them to the example data set we have to give us an estimate of total escapement.

Data preparation

Before we begin, we need to prepare the data we have for use in our functions. Because we'll want to be testing our equations and functions using various example data sets, here I've written a function called `CJS_data_prep()` that does data formatting for us. Note that currently this function only incorporates sex and length data as covariates for survival and capture probabilities. In the future, if we aim to incorporate other covariates, data will need to be prepared differently.

```
CJS_data_prep<-function(ch,chops,covars){  
  #set.seed(123)  
  #Part 1: prepare ch and covars data  
  ch=ch[-1] #remove disctag vector from capture histories
```



```

covars$sex<-ifelse(covars$sex=='F',1,0) #change sex to numeric value

#Part 2: prep chops data
chops<-chops[-1]
clean_chops<-matrix(ncol=ncol(chops))
for(i in 1:ncol(chops)){
  d<-chops[i]
  r<-rep(0,ncol(chops))
  r[i]=2
  r<-matrix(rep((r),d),ncol=ncol(chops),byrow=T)
  clean_chops<-clean_chops%>%rbind(r)
}
clean_chops<-clean_chops[-1,]
colnames(clean_chops)<-colnames(ch)
#add in chops
ch<-ch%>%rbind(clean_chops)

#Part 3: generate sex data for chopped data
index = 1:dim(ch)[1]
samp<- sample(index, replace = T)

covariates_new<-as.matrix(covars[samp,])

sex_vector<-covariates_new[, 'sex']
no.miss = sex_vector[!is.na(sex_vector)]
prop.female<-mean(no.miss)
prop.male<-1-prop.female
sex_vector<-ifelse(is.na(sex_vector),
                   sample(c(1,0),1,prob=c(prop.female,prop.male)),
                   sex_vector)
sex_matrix<-matrix(sex_vector,nrow=nrow(ch),ncol=ncol(ch))

#Part 4: generate length data for chopped data
lengths_vector<-covariates_new[, 'length']
females<-covariates_new[covariates_new[,2]==1,]
males<-covariates_new[covariates_new[,2]==0,]
avg.female.length<-round(mean(as.numeric(females[,3]),na.rm=T),1)
avg.male.length<-round(mean(as.numeric(males[,3]),na.rm=T),1)
for(i in 1:length(lengths_vector)){
  if(is.na(lengths_vector[i])){
    lengths_vector[i] = ifelse(sex_matrix[i,1] == 1,
                              avg.female.length,

```

```

                                avg.male.length)
  }
}
lengths_matrix<-matrix(as.numeric(lengths_vector),
                        nrow=nrow(ch),ncol=ncol(ch))

return(list('ch'=ch,'lengths_matrix'=lengths_matrix,
            'sex_matrix'=sex_matrix))
}

```

Again, the above is a longer function, so I'll break it down here:

- Part 1: we begin by removing the first column of the capture history matrix `ch`, which contains the disctag data that can't be used in our escapement functions. We also change the values of sex from "F" and "M" to 1 and 0, respectively.
- Part 2: in this function we deal with the chops data `chops` by appending it to the original capture histories `ch`. We do this by creating a new matrix of capture histories for the chopped data and appends them to the original `ch` data.
- Part 3: now that we've generated capture histories for all the chopped data, we also need to generate covariate data to utilize. First we start with the sex data, we do this by finding the proportion of male and female carcass in the original covariate data, and then assigning sex to the new capture histories based on that proportion.
- Part 4: Finally we can generate the lengths covariate data for the added chops capture histories. This is done by taking the average male and female lengths, and assigning it to each of the added capture histories based on sex.

i Note

Assigning the average length data to carcasses is based on the methods of the `escapeMR` package. I think this could be improved upon by sampling from the distribution of known lengths for each chopped capture history.

Running our code

Now let's load our demo data, and then run the new `CJS_data_prep()` function we just built. Once we run the data prep function, we can assign the outputs from the output "prepped_data" list. For this demonstration we are modeling capture and survival based on length and sex respectively (Equation 10, Equation 11), so we will save our `prepped_data$lengths_matrix` as `cap_X_prepped` and our `prepped_data$sex_matrix` as `surv_X_prepped`.

```

ch<-read.csv('data/demo_CH.csv')
covars<-read.csv('data/demo_Covariates.csv')
chops<-read.csv('data/demo_Chops.csv')

prepped_data<-CJS_data_prep(ch,chops,covars)

ch_prepped<-prepped_data$ch
cap_X_prepped<-prepped_data$lengths_matrix
surv_X_prepped<-prepped_data$sex_matrix

```

We've prepared our data, now we can run it and our `CJS_loglik_wrapper()` function through the R `optim()` function. First we set our initial values for `beta`, then we can run the optimizer to produce the coefficients with the maximum log-likelihood. The R `optim()` call uses the following parameters:

- `par`: the initial coefficient parameters we'll be optimizing, in our case `initial_beta`.
- `fn`: the function we will be optimizing, `CJS_loglike_wrapper()` that we created earlier.
- `method`: "BFGS", or the specific algorithm the `optim()` function will use for optimization. See the help page for the function by calling `?optim` for more information on potential algorithms it can utilize.
- `cap_X`: the capture covariate lengths matrix we just prepared to be passed to the function.
- `surv_X`: the survival covariate sex matrix we just prepared to be passed to the function.
- `ch`: the prepared capture history matrix to be passed to the function.
- `control`: allows for various function control parameters, we will set the maximum number of iterations for here to 1000.

```

initial_beta=numeric(4)

starttime<-Sys.time()
optim_results<-optim(par=initial_beta,
                     fn=CJS_loglik_wrapper,
                     method="BFGS",
                     ch=as.matrix(ch_prepped),
                     cap_X=as.matrix(cap_X_prepped),
                     surv_X=as.matrix(surv_X_prepped))
endtime<-Sys.time()
optim_speed_R<-endtime-starttime

```

```
optim_beta<-optim_results$par
```

Once the `optim()` call is finished we can see that the coefficients that produce the maximum log-likelihood are $\Upsilon_0=3.5987491$, $\Upsilon_1=-0.008089$, $\beta_0= 2.4728445$, $\beta_1=-1.6810092$.

Next we can plug those parameters into our `pro_capsur()` function to estimate the capture and survival probabilities for each carcass i at each survey period j . We're likely going to have to do this process many times, so again we'll write another function to do the work for us, this time calling it `fill_prob_matrices()`. This function will use the `pro_capsur()` function and apply it to all of our prepared data with the optimized parameters we've estimated and give us two output matrices `p_hat` and `s_hat`.

```
fill_prob_matrices<-function(ch,beta,cap_X,surv_X){
  nan=nrow(ch)
  ns=ncol(ch)
  p_hat<-s_hat<-matrix( 0, nan, ns ) #create empty matrices
  for(i in 1:nan){
    for(j in 1:ns){ #fill each cell with pro_capsur() function
      p_hat[i,j]<-pro_capsur(i,j,ch,beta,
                             cap_X,
                             surv_X)$p.hat
      s_hat[i,j]<-pro_capsur(i,j,ch,beta,
                             cap_X,
                             surv_X)$s.hat
    }
  }
  p_hat[,1]<-NA

  est_list<-list('p_hat'=p_hat,'s_hat'=s_hat)
  return(est_list)
}
```

Now we have the capture histories (`ch_prepped`) and a function to create our matrices of `p_hat` and `s_hat` using the optimized parameters and our `pro_capsur()` function. We could just run our new `fill_prob_matrices()` function and plug in the outputs to our `total_escapement()` function, but to really stream line things we can update `total_escapement()` to do it automatically:

```
total_escapement<-function(ch,beta,cap_X,surv_X){

  p_hat<-fill_prob_matrices(ch,beta,
```

```

        cap_X,
        surv_X)$p_hat
s_hat<-fill_prob_matrices(ch,beta,
        cap_X ,
        surv_X)$s_hat

N_hat=Horvitz_Thompson(p_hat,ch)

Bstar=B_star(ch,s_hat,p_hat)

escapement<-N_hat[[2]]*(log(mean(s_hat[,1]))/(mean(s_hat[,1])-1)) +
sum(Bstar,na.rm=T)

return(ceiling(escapement))
}

```

Now we can plug the `ch_prepped`, `optim_beta` results, and the `cap_X_prepped` and `surv_X_prepped` all into our `total_escapement()` function to produce our final estimate of escapement.

```

est_escapement<-total_escapement(ch_prepped,optim_beta,
                                cap_X_prepped,surv_X_prepped)
print(est_escapement)

```

```
[1] 1846
```

i Note

When we utilize the `CJS_data_prep()` function in our script, it randomly samples sex and length data for our chopped capture histories. Because of this randomness, final estimates for escapement and the capture and survival coefficients can vary when your run this script. This could be fixed by using the `set.seed()` call, but would remove some of the variability.

If we go back to our optimization code using the `optim()` function, note that the code written stores the speed of the function as `optim_speed1`. If we check that value, we can see the function was completed in 2.1 seconds. The next step in our efforts involves a process known as “bootstrapping” so that we can calculate confidence intervals for the escapement estimate we produce. This process involves repeating the optimization process for hundreds of iterations, meaning that the above process could take quite a while to run. If we increase the complexity of the optimization problem with larger capture histories and covariate data sets, which many

Central Valley river systems often observe, it could take hours to bootstrap our model. We need a way to speed up our model.

Incorporating C++ to speed things up

My solution to trying to speed up our model was to translate all of our functions into C++, using Visual Studio . I'm not experienced in C++ but was able to utilize ChatGPT 3.5, and a lot of trial and error, to translate the `pro_capsur()`, `location()`, and `CJS_loglik()` functions into the language. Within the “Rcpp sources/CJS_functions.cpp” script I also wrote the `cpp_optim()` function, which utilizes the base R `optim()` call in the C++ script. I won't get into the process of translating the packages into C++ in this text, as it was long and error prone, but if you read the “CJS_functions.cpp” script one important note is that C++ and R indexes matrices differently. With R, the first cell in a matrix is indexed at `matrix[1,1]`, starting at 1, while in C++, the first cell in a matrix is indexed at `matrix[0,0]`, starting at 0.

What allows us to utilize the C++ code in our R script is the `Rcpp` package and the `sourceCpp()` function that comes with it (this takes a few seconds to read everything in, but only needs to be done once). In the C++ script, functions marked with `Rcpp::export` are pulled into our R environment and made available.

```
library(Rcpp)
sourceCpp('Rcpp sources/CJS_functions.cpp')

starttime<-Sys.time()
{gc()
  optim_results<-cpp_optim(beta=initial_beta,
                           ch=as.matrix(ch_prepped),
                           cap_X=as.matrix(cap_X_prepped),
                           surv_X=as.matrix(surv_X_prepped))
}
endtime<-Sys.time()
optim_speed_cpp<-endtime-starttime

est_escapement<-total_escapement(ch_prepped,optim_results$par,
                                 cap_X_prepped,surv_X_prepped)
```

In the above code, we first load the `Rcpp` package, then use the `sourceCpp()` call to pull the functions from our “CJS_functions.cpp” script. Then we run the `cpp_optim()` optimization call similarly to how we used the base `optim()` call earlier, except here the C++ version of the call is built to utilize the `CJS_wrapper()` function automatically, so it requires less inputs. The `cpp_optim()` call is run within brackets and preceded by the `gc()` function which is used

to perform “garbage collection”, which is the process of reclaiming memory space for future allocation. This is done to fix an issue I ran into when I originally testing this function, in which utilizing the `cpp_optim()` call multiple times in a row would result in memory losses that would often cause R studio to crash, and would make iterative bootstrapping impossible.

After running the optimization we see that it ran at 0.44 seconds, quite a bit faster than our original method at 2.1 seconds, and produced our same escapement estimate of 1846. This speed increase will help make sure that when we do many hundreds of bootstrap iterations for larger data sets, that the model takes only hours to run, instead of days.

i Note

Even though we sped up the optimization process of our function by utilizing C++, we still can’t match the speed of the FORTRAN algorithm used by `escapeMR`. In short, while FORTRAN is a very old coding language, it tends to be faster then either C++ or R. This speed difference will lessen however when we begin to process larger data sets.

Bootstrapping for confidence intervals

With the methods above used to estimate total escapement, it’s difficult to produce an estimate of precision or confidence intervals. Fortunately we can once again take advantage of the speed of modern computing to perform what is known as “**bootstrapping**”, an iterative resampling technique we can use to estimate the distribution of the escapement estimate by repeatedly sampling, with replacement, the original dataset. Figure 5 shows a conceptual model of how sampling the original capture histories and covariates with replacement creates a new dataset where some of the original records are either over represented or not included.

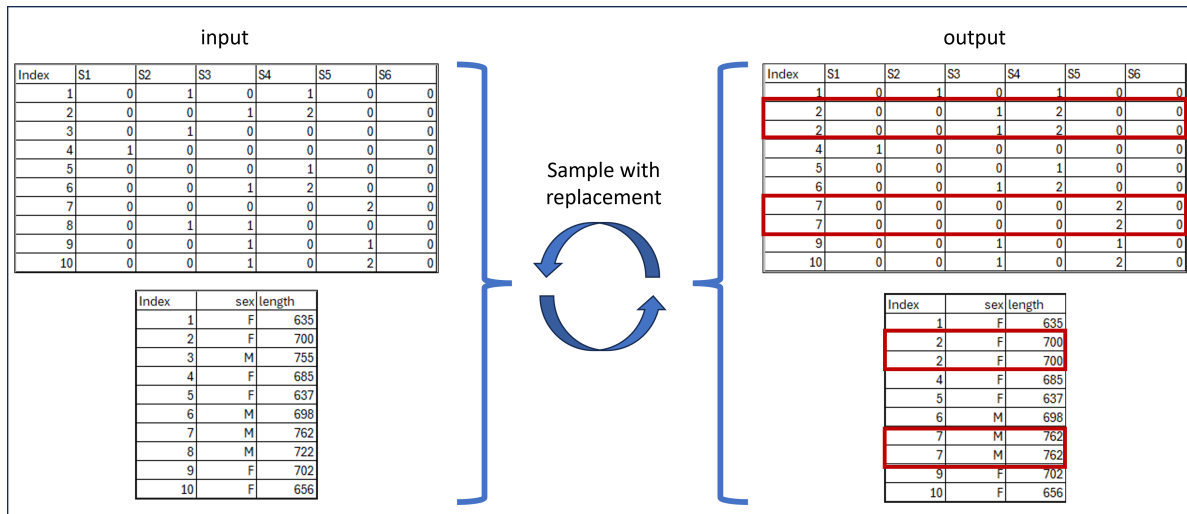


Figure 5: Conceptual model of how bootstrapping iterations creates variation in observed capture histories and covariates. Records 2 and 7 are repeated in the output dataset, and records 3 and 8 are excluded.

Below is another new function `CJS_bootstrap()` which is written to run bootstrapping for a number of user defined iterations.

```
CJS_bootstrap<-function(iterations,ch,cap_X,surv_X){
  #Part 1: set initial values
  results<-data.frame()
  initial_beta=numeric(4)

  #Part 2: run the for loop for each iteration
  for(r in 1:iterations){
    iter_start<-Sys.time()

    #Part 2.1: index and sample the capture histories
    index=1:dim(ch_prepped)[1]
    samp<- sample(index, replace = T)

    #Part 2.2: create new capture histories and
    #cap_X and surv_X matrices based on sampled indices
    ch_iteration = ch_prepped[samp,]
    cap_X_iteration=cap_X_prepped[samp,]
    surv_X_iteration=surv_X_prepped[samp,]

    #Part 2.3: use cpp_optim to find the optimizal beta
```



```

#parameters for the given iteration data
{gc()
  optim_iter<-cpp_optim(beta=initial_beta,
                        ch=as.matrix(ch_iteration),
                        cap_X=as.matrix(cap_X_iteration),
                        surv_X=as.matrix(surv_X_iteration))
}
iter_beta<-optim_iter$par
iter_lik<-optim_iter$value

#Part 2.4: estimate total escapement for iteration
est_esc_iter<-total_escapement(ch_iteration,
                               iter_beta,
                               cap_X_iteration,
                               surv_X_iteration)

iter_end<-Sys.time()
iter_time<-iter_end-iter_start

#Part 2.5: store relevant iteration information in results
d<-data.frame("iteration"=r,
              "log-likelihood"=iter_lik,
              "escapement"=est_esc_iter,
              "time"=iter_time)
results<-results%>%rbind(d)
print(r) #print iteration number for progress tracking
}
return(results)
}

```

The above `CJS_bootstrap()` function takes our `ch`, `cap_X`, and `surv_X` inputs, the target number of `iterations` (how many times to sample and run the model), and returns a dataset with the log-likelihood, escapement estimate, and duration for each iteration. Again we have another long and complicated function, so let's break it down:

- Part 1: begins by setting an empty data frame for our results and the `initial_beta` of `c(0,0,0,0)`.
- Part 2: begins the for loop for the number of iterations set by the user.
- Part 2.1: creates an index for each row of the original capture history data, and then uses the `sample()` call from base R to sample that list of indices with replacement. Sampling

with replacement ensures that some of the original records will not be represented in the new iteration data, and some of the records will be over represented.

- Part 2.2: uses that `samp` list of sampled indices to create new capture histories and covariate matrices `cap_X` and `surv_X`.
- Part 2.3: runs the `cpp_optim()` function using the new iteration data sets, stores parameter and loglikelihood estimates in `iter_beta` and `iter_lik` values.
- Part 2.4: estimates escapement using the `total_escapement()` call and stores as `est_esc_iter`.
- Part 2.5: stores the iteration outputs in the `results` data frame before continuing loop again.

Let's use the `CJS_bootstrap()` function with 500 iterations to generate confidence intervals for our escapement estimate. Depending on the size of the dataset being used this can take anywhere from several minutes to several hours, so feel free to use less iterations when testing the functions.

```
bootstrap_results<-CJS_bootstrap(iterations=500,ch_prepped,  
                                cap_X_prepped,surv_X_prepped)
```

Running 500 iterations took my computer about 7.5 minutes to complete, with each iteration averaging 0.89 seconds to complete.

Now that we have a large dataset of many iterative randomly samples capture histories and corresponding covariates, we can calculate confidence intervals,

```
#confidence intervals  
conf.level = 95  
alpha = 1 - conf.level/100  
lower = alpha/2  
upper = 1 - alpha/2  
mid=.5  
  
ci<-bootstrap_results  
ci<-ci%>%  
  summarise(lower_ci=ceiling(quantile(escapement,probs = c(lower),na.rm=T)),  
            mid_ci=ceiling(quantile(escapement,probs = c(mid),na.rm=T)),  
            upper_ci=ceiling(quantile(escapement,probs = c(upper),na.rm=T)))
```

and then plot our results using the `ggplot` package.

```

library(ggplot2)
ggplot(bootstrap_results,aes(x=escapement))+
  geom_histogram(color = "#000000", fill = "#0099F8")+
  geom_segment(data=ci,aes(x=lower_ci,xend=lower_ci,y=0,yend=Inf),
              linewidth=1,linetype='dashed')+
  geom_segment(data=ci,aes(x=upper_ci,xend=upper_ci,y=0,yend=Inf),
              linewidth=1,linetype='dashed')+
  geom_segment(data=ci,aes(x=est_escapement,
                          xend=est_escapement,y=0,yend=Inf),
              linewidth=1,linetype='dashed',color='red')+
  geom_text(data=ci,aes(x=lower_ci-500,y=200,label=paste('Lower:',lower_ci)))+
  geom_text(data=ci,aes(x=upper_ci+500,y=200,label=paste('Upper:',upper_ci)))+
  geom_text(data=ci,aes(x=est_escapement,y=200,
                      label=paste('Est Escapement:',est_escapement)))+
  scale_x_continuous(breaks = seq(0,10000,500)) +
  xlim(0,5000)+
  theme_classic()

```

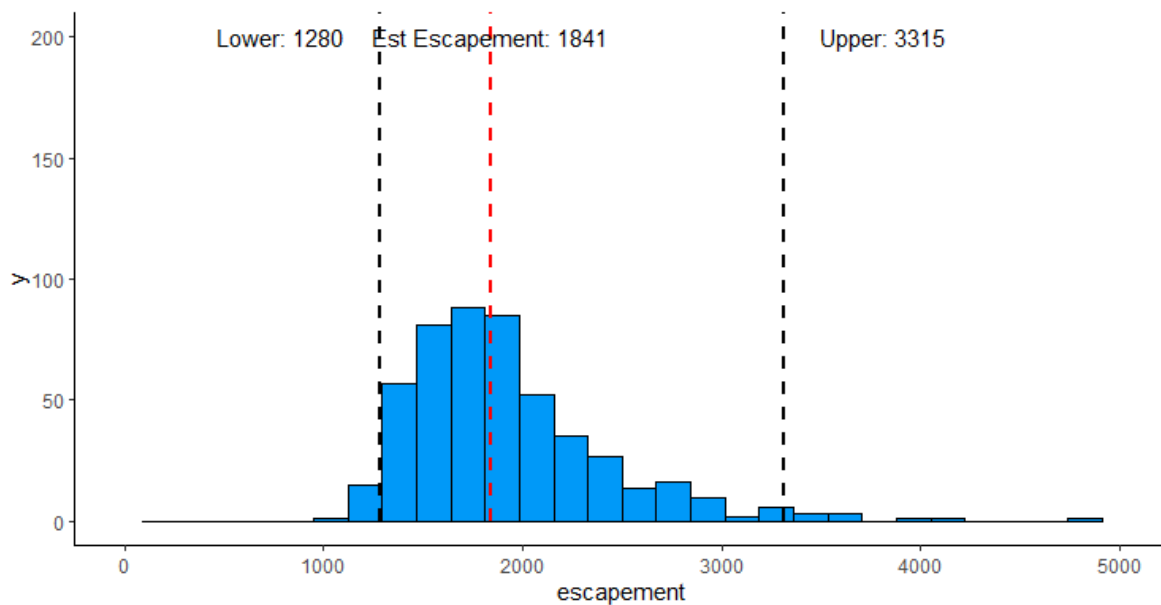


Figure 6: Plot of bootstrapped escapement estimates, with upper and lower confidence intervals.

In the above plot (Figure 6) we can see that with 500 bootstrap iterations, our estimate of $\hat{N}_{escapement}$ is 1841, and our 95% confidence interval is between 1280 and 3315, so a pretty large rang of values for our estimate.

Comparing our functions to escapeMR

The only practical difference in escapement estimate production between the methods we've created so far and the `escapeMR` package is how the coefficient parameters are estimated through optimization. As discussed previously, the `escapeMR` package utilizes the `VA09AD` FORTRAN algorithm for optimization, while our code uses the R `optim()` "BFGS" algorithm. We can easily compare our results above to the the what the `escapeMR` package generates by using calling the `escapeMR()` function in R studio, and using the shiny application. The application produces an estimate of $\hat{N}_{escapement}$ is 2054, and the confidence intervals generated are between 1251 and 3203.

```
library(escapeMR)
escapeMR()
```

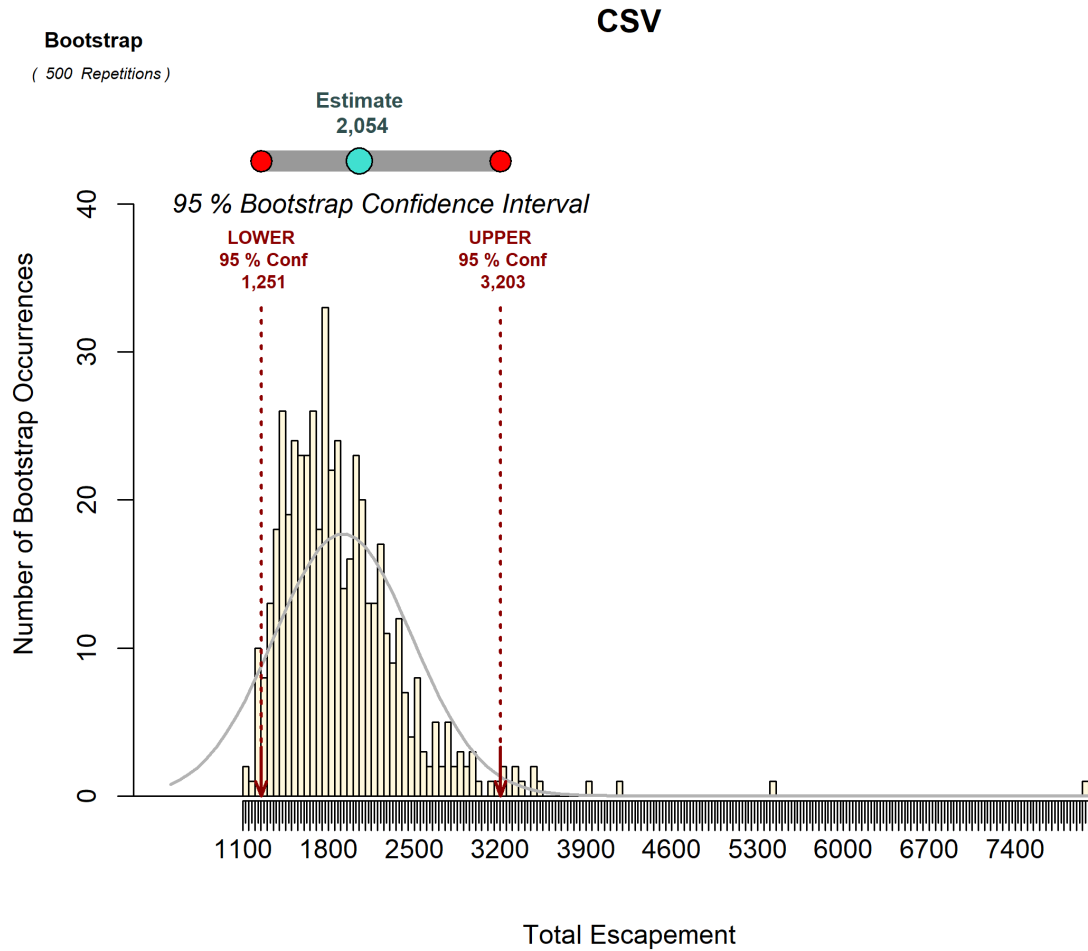


Figure 7: Histogram of escapeMR bootstrap results.

Overall the **escapeMR** package did give us slightly tighter confidence intervals with this dataset, and while we can't really time the shiny app, it took between two and three minutes to complete 500 bootstrap iterations on my computer.

Let's compare a larger data set, using 2021 American River carcass survey data. After preparation, there are a total of 5244 individual capture histories and corresponding covariates. That's a lot more than the demo data we used above. In this following chunk we will run all the steps at once, beginning by loading the data, preparing it with `CJS_data_prep()`, optimizing the parameters with `cpp_optim()`, estimating escapement with `total_escapement()`, and then running a series of bootstraps with our `CJS_bootstrap()` function.

```

ch<-read.csv('data/additional_data/AR_2021_CH.csv')
covars<-read.csv('data/additional_data/AR_2021_Covariates.csv')
chops<-read.csv('data/additional_data/AR_2021_Chops.csv')

prepped_data<-CJS_data_prep(ch,chops,covars)

ch_prepped<-prepped_data$ch
cap_X_prepped<-prepped_data$lengths_matrix
surv_X_prepped<-prepped_data$sex_matrix

starttime<-Sys.time()
{gc()
  optim_results<-cpp_optim(beta=initial_beta,
                           ch=as.matrix(ch_prepped),
                           cap_X=as.matrix(cap_X_prepped),
                           surv_X=as.matrix(surv_X_prepped))
}
endtime<-Sys.time()
optim_speed<-endtime-starttime

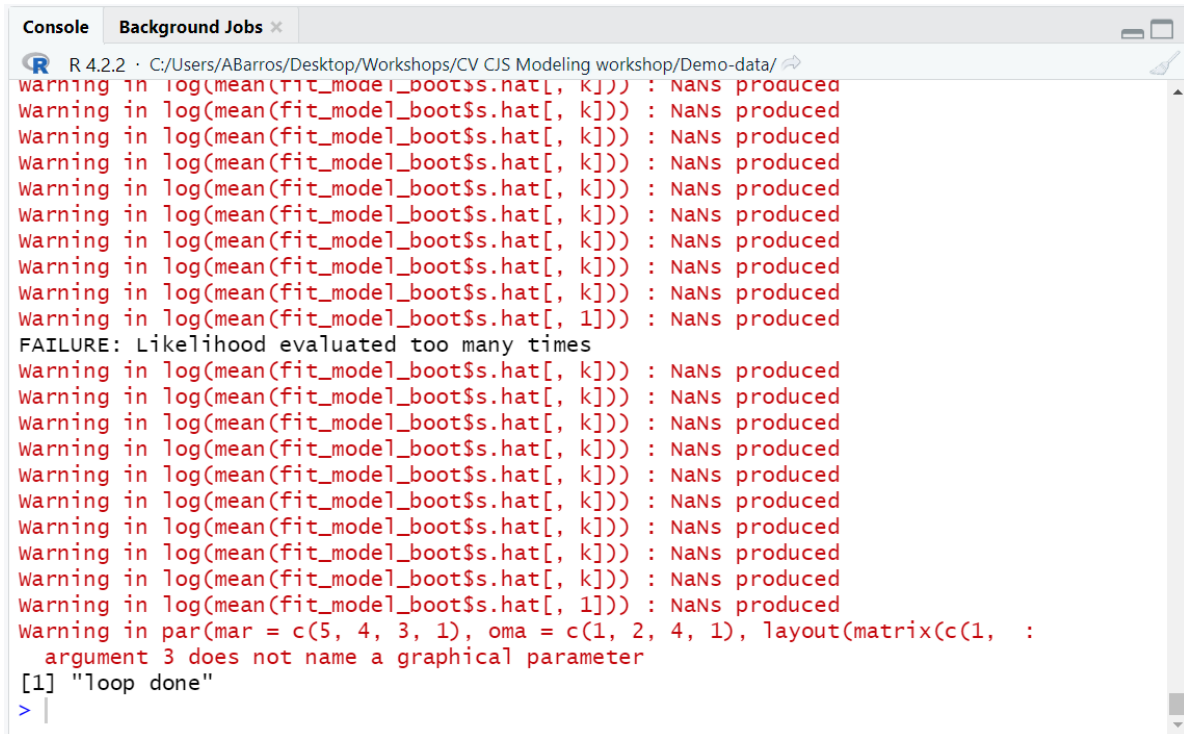
est_escapement<-total_escapement(ch_prepped,optim_results$par,
                                 cap_X_prepped,surv_X_prepped)

starttime<-Sys.time()
bootstrap_results<-CJS_bootstrap(iterations=500,
                                 ch_prepped,
                                 cap_X_prepped,
                                 surv_X_prepped)

endtime<-Sys.time()
bootstrap_speed<-endtime-starttime #2.16 hours for 500 iterations
saveRDS(bootstrap_results,'outputs/AR_bootstrap_results.rds')

```

When testing this myself, I ran 500 bootstrap iterations, which took roughly 2.16 hours. The initial estimate of $\hat{N}_{escapement}$ was 10,976 carcasses, and our 95% confidence intervals were between 10,246 and 12,352. I ran the same dataset through the escapeMR shiny application, and had some issues. First off, it produces an escapement estimate of “NaN”. If we run the application in R instead of online, we’ll actually see a lot of warning messages appear (Figure 8).



```
R 4.2.2 · C:/Users/ABarros/Desktop/Workshops/CV CJS Modeling workshop/Demo-data/
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, 1])) : NaNs produced
FAILURE: Likelihood evaluated too many times
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, k])) : NaNs produced
Warning in log(mean(fit_model_boot$s.hat[, 1])) : NaNs produced
Warning in par(mar = c(5, 4, 3, 1), oma = c(1, 2, 4, 1), layout(matrix(c(1, :
argument 3 does not name a graphical parameter
[1] "loop done"
> |
```

Figure 8: Example of escapeMR optimization failing to converge on successive bootstrapping iterations.

These warnings suggest that the optimization algorithm used by **escapeMR** failed to converge on a maximum likelihood during those iterations. Enough of the iterations did succeed to produce some confidence intervals (Figure 9). I wrote some code to pull the log-likelihood from each bootstrap iteration, as well as measure the time it takes to process. For the 2021 American River data set, the escapeMR bootstrapping took 3.4 hours to run 500 bootstrap iterations, and failed to converge for 163 of those iterations.

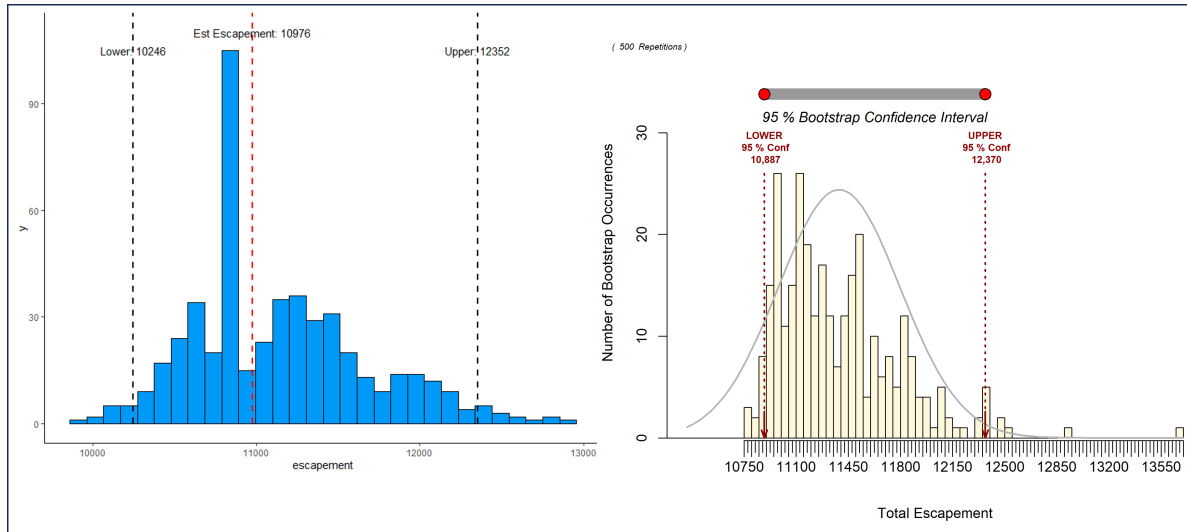


Figure 9: Histogram of American River bootstrapped escapement results produced by our functions (left) and the escapeMR package (right).

So far we've been able to compare the escapeMR and methods using two very different data sets, and see how the speed and precision of these both methods compare. However, we can't really judge how these two methods compare in regards to their accuracy to the true escapement, as of course we don't know what those numbers are for our example data.

Simulating a population

To try and gauge accuracy of our escapement functions and the escapeMR package, I've taken some time to write an R script that simulates capture histories and covariate data by taking inputs of a target population size `N1`, the number of simulated surveys `ns`, and the covariate coefficients `coef`. To run these simulations I've used the coefficients generated when we estimated escapement for our demo data.

```
source('scripts/archive/ch_simulator.R')

sim<-ch_sim(N1=2000,ns=15,coef=optim_beta)

ch_prepped<-sim$ch
cap_X_prepped<-sim$lengths
surv_X_prepped<-sim$sex

write.csv(ch_prepped,'data/additional_data/sim_ch.csv')
covars<-data.frame("length"=cap_X_prepped[,1],"sex"=surv_X_prepped[,1])
```



```

write.csv(covars,'data/additional_data/sim_covars.csv')

starttime<-Sys.time()
{gc()
  sim_results<-cpp_optim(beta=initial_beta,
                        ch=as.matrix(ch_prepped),
                        cap_X=as.matrix(cap_X_prepped),
                        surv_X=as.matrix(surv_X_prepped))
}
endtime<-Sys.time()
endtime-starttime

est_escapement<-total_escapement(ch_prepped,optim_results$par,
                                cap_X_prepped,surv_X_prepped)

bootstrap_results<-CJS_bootstrap(iterations=500,ch_prepped,
                                cap_X_prepped,surv_X_prepped)
saveRDS(bootstrap_results,'outputs/sim_N2000.rds')

```

The script above pulls the `ch_sim()` function from the “ch_simulator.R” script, and creates simulated capture histories and covariate data sets. I tested two different population sizes for comparison, $N=2000$, and $N=4000$. It then saves those simulated data sets and then runs them through our bootstrapping function. I also ran both simulated populations through the escapeMR optimization method to compare results.

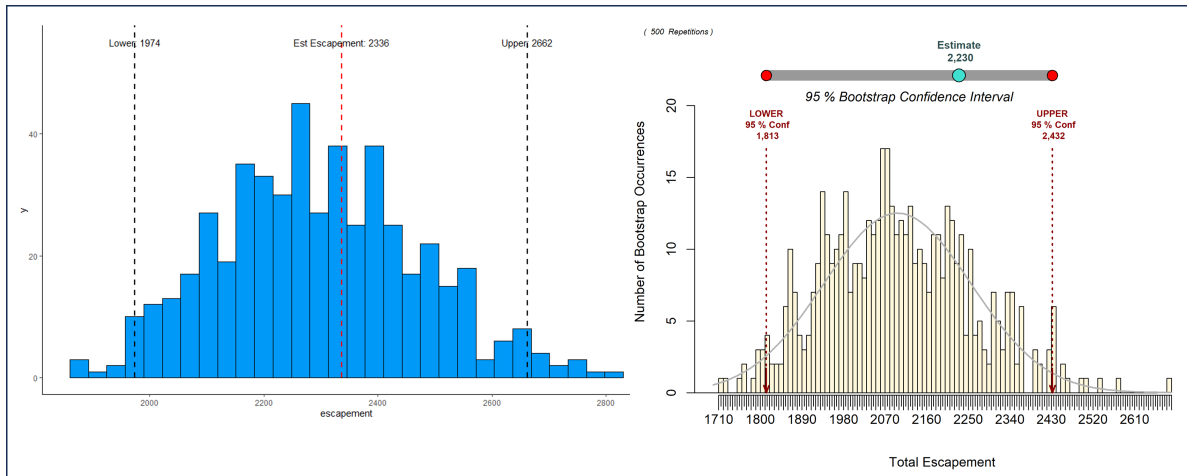


Figure 10: Histogram of simulated population bootstrapped escapement results produced by our functions (left) and the escapeMR package (right). Simulated population size $N=2000$. Number of surveys $ns=15$.

For a population size of 2000, the R `optim` bootstrapping took 36 minutes to run 500 iterations, while the `escapeMR` package took only 10 minutes to run. The results in Figure 10 are pretty similar for both methods, with \hat{N} about 300 carcasses above the true population size, but the confidence intervals straddling $N=2000$.

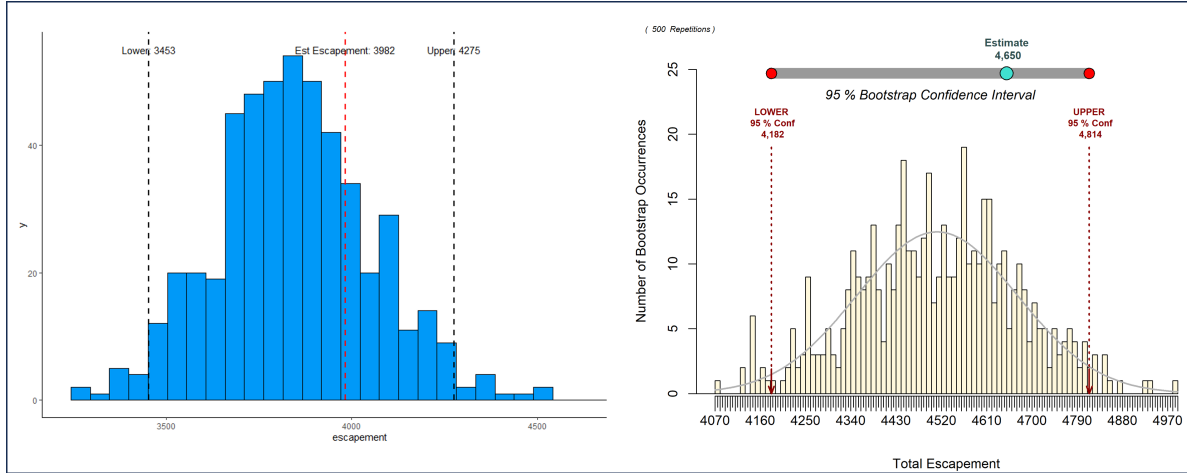


Figure 11: Histogram of simulated population bootstrapped escapement results produced by our functions (left) and the `escapeMR` package (right). Simulated population size $N=4000$. Number of surveys $ns=15$.

For a population size of 4000, the bootstrapping took 82 minutes to run 500 iterations, while the `escapeMR` package took about 2 hours to run. I find it interesting that at these larger population sizes, the `escapeMR` package tends to take a lot longer than our functions. Also, at this larger population size, in Figure 11 shows that the `escapeMR` estimate of \hat{N} was much farther from the true population size than the estimate produced using `R optim()`, and the confidence intervals fell above $N = 4000$. Here our bootstrapping functions produced an estimate very close to the true value with confidence intervals on either side of $N=4000$.

i Note

It's important to not read too much into the results of these simulated populations, as the accuracy of one method over the other could just be attributed to differences in how the optimization process works on a simulated population, as opposed to a real world population. In order to 'ground-truth' these accuracy tests, it could be helpful to use auxiliary data sets to produce alternate estimates of escapement, such as redd surveys, DIDSON sonar recordings, and video surveys.

Comparison summaries

Reviewing the outputs of our methods comparisons we can see the following results:

- For smaller capture history data sets like our ‘demo_data’, the escapeMR method is faster and both methods produced a similar range of confidence intervals and estimates of \hat{N} (Figure 6, Figure 7).
- With a much larger data set, like the 2021 American River data, the escapeMR method was actually much slower than our R methods, and ended up failing to converge during optimization for 163 of the 500 bootstrapping iterations (Figure 8).
- Using data from simulated populations, both methods produced estimates and confidence intervals close to the true population when N was 2000. However when N was 4000 the escapeMR method resulted in confidence intervals that overestimated escapement (Figure 10, Figure 11).

Next steps

I hope this document has provided a foundation for understanding how CVCS escapement estimation is performed and will aid participants in the August modeling workshop in discussing ways to improve this process. Below, I've summarize my thoughts on potential next steps.

Adopting R optim based methods

While the escapeMR package generally works, and works well for smaller data sets, we've seen that the VA09AD optimization algorithm it depends on can fail to optimize for covariate coefficients with larger, more complex data sets. This failure to converge is concerning for the precision and accuracy of this method.

The Rshiny application and escapeMR package are generally user friendly, and provide an easy interface for users who are not experienced with coding to produce escapement estimates. However, this Rshiny interface could easily be replicated and applied over an R `optim()` based method, to generate estimates. These packages are also no longer supported, and cannot be edited or updated by state scientists, as the owner and creator Trent McDonald has retired. Moving to the outlined R based method allows us ownership and editing control over an "in-house" based model, and will allow us to make changes and updates to the system when we decide they may be useful.

Incorporating additional covariates.

Transitioning to an in-house R based method and package will allow us to improve our modeling methods by incorporating covariates beyond sex and length in our escapement estimate calculation. Environmental covariates such as turbidity, temperature, and flow could have significant influence on capture and survival probabilities, and currently the Rshiny application cannot incorporate those in estimates.

While developing an RShiny application that could incorporate and format all potential covariates for use in the model might be impractical, we could create guidance for users on how to format their covariate data for modeling. This would allow an RShiny application to incorporate data formatted and prepared by the user.

It could also be possible to update our modelling methods to incorporate multiple covariates in one probability function. For example, we may be able to model the probability of capture based on carcass length as well as water flow. I haven't delved into the details of advancing the model to that stage yet, but it is theoretically possible. Of course, adding more complexity to the model would increase computing time, which is always important to consider.

References

- [1] Steven C. Amstrup, Trent L. McDonald, and Bryan F.J. Manly. *Handbook of Capture-Recapture Analysis*. Princeton Univ. Press, 2005. URL: <http://www.jstor.org/stable/j.ctt7sdj6>.
- [2] Jennifer M. Bergman, Ryan M. Nielson, and Alice Low. “CENTRAL VALLEY CHINOOK SALMON IN-RIVER ESCAPEMENT MONITORING PLAN”. In: *Fisheries Branch Administrative Report Number: 2012-1* (2013), pp. 12–26.
- [3] R.M. Cormack. “Estimates of Survival from the Sighting of Marked Animals”. In: *Biometrika* 51.3 (1964), pp. 429–438.
- [4] D.G. Horvitz and D.J. Thompson. “A Generalization of Sampling Without Replacement From a Finite Universe”. In: *Journal of the American Statistical Association* 44.260 (1952), pp. 663–685. ISSN: 00283932. URL: <http://www.ncbi.nlm.nih.gov/pubmed/16243365>.
- [5] G.M. Jolly. “Explicit Estimates from Capture-Recapture Data with Both Death and Immigration- Stochastic Model”. In: *Biometrika* 52.1 (1965), pp. 225–247.
- [6] Frederick C. Lincoln. *Calculating Waterfowl Abundance on the Basis of Banding Returns*. Tech. rep. US Department of Agriculture, 1930. arXiv: [arXiv:1011.1669v3](https://arxiv.org/abs/1011.1669v3).
- [7] Trent L. McDonald and Steven C. Amstrup. “Estimation of Population Size Using Open Capture – Recapture Models”. In: *American Statistical Association and the International Biometric Society* 6.2 (2001), pp. 206–220.
- [8] J.D. Nichols. *Modern Open-Population Capture-Recapture Models*. Ed. by Steven C. Amstrup, Trent L. McDonald, and Bryan F.J. Manly. Princeton Univ. Press, 2008, pp. 88–123.
- [9] C. G. J. Petersen. “The yearly immigration of young plaice in the Limfjord from the German sea”. In: *Report of the Danish Biological Station* 6 (1896), pp. 5–84.
- [10] Zoe E. Schnabel. “The Estimation of the Total Fish Population of a Lake”. In: *American Mathematics Monthly* 45 (1938), pp. 348–352.
- [11] G A F Seber. “A Note on the Multiple-Recapture Census Published”. In: *Biometrika* 52.1 (1965), pp. 249–259.