Distance Sampling

Ellen Bledsoe

2025-04-21

Distance Sampling Lab

Set-Up

We need to load the package we will need to complete this lab, which is `unmarked`.

```
library(unmarked) # Load unmarked
```

We also need to load our data. I've compiled it for you already, and it should be in your list of files, called `Distance_Data_Spring2025.csv`.

```
beads <- read.csv("../data_raw/Distance_Data_Spring2025.csv")
```

Before we begin any analyses, let's make sure our data looks alright.

```
str(beads)    # View the structure of the dataframe
```

```
## 'data.frame':    371 obs. of  4 variables:
##  $ Group      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ BeadColor  : chr  "Green" "Green" "Green" "Green" ...
##  $ Distance_in: num  2.5 7 77 37 10 ...
##  $ Transect_m : int  10 10 10 10 10 10 10 10 10 10 ...
```

```
head(beads)   # First few observations
```

```
##   Group BeadColor Distance_in Transect_m
## 1     1     Green        2.50         10
## 2     1     Green        7.00         10
## 3     1     Green       77.00         10
## 4     1     Green       37.00         10
## 5     1     Green       10.00         10
## 6     1     Green       35.25         10
```

Prepping the Data

First, we need to convert all of our data from inches into meters because those are the measurements `unmarked` is expecting. You won't need to do this in your assignment, thankfully, because those values are already in meters.

```
beads$Distance_m <- beads$Distance_in / 39.37
```

Since we have 2 different colors of beads, we need to create two different data frames. You won't need to do this in your assignment, either, thankfully!

```
clear <- beads[beads$BeadColor == "Clear", ]
green <- beads[beads$BeadColor == "Green", ]
```

Take a look at the environment to see how many rows the `clear` and `green` data frames have. The `green` data frame has more observations than the `clear`. Is that expected? Why or why not?

Green Beads

Transect Data

The first thing we need to do is to get a data frame that includes the length of transect that each group walked so we have a measure of survey effort.

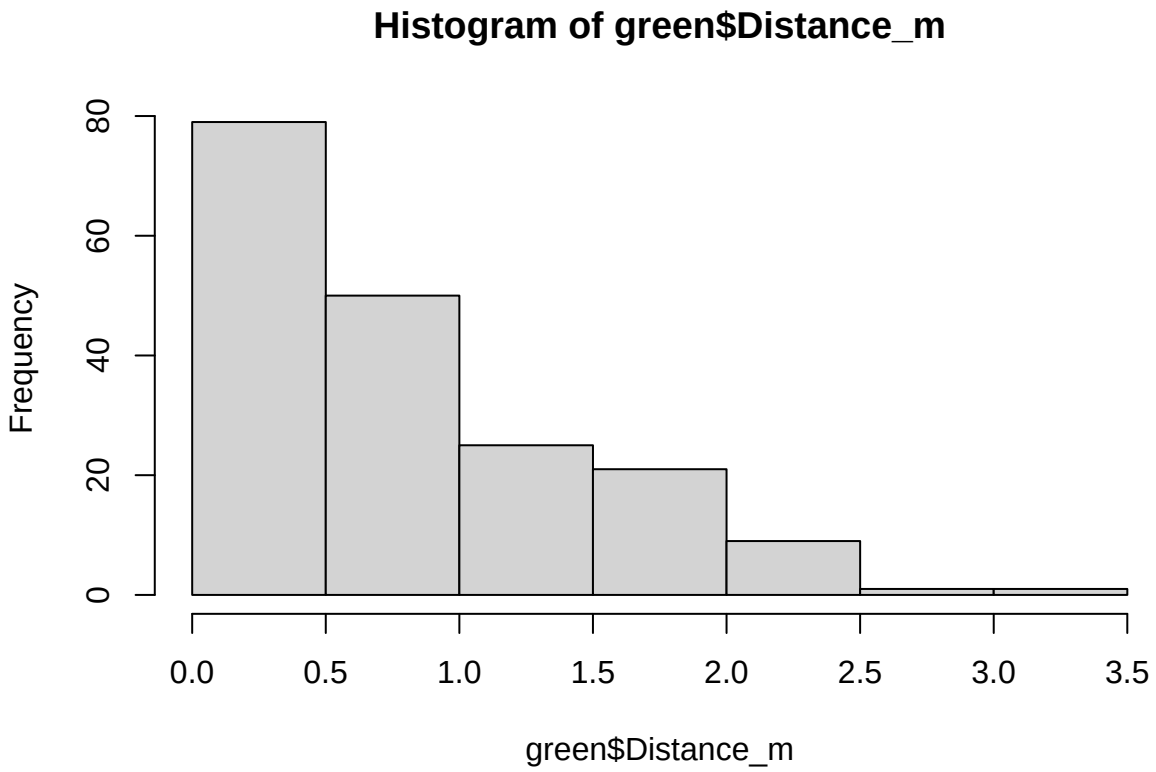
We will use the `unique()` function, which returns only one of each value in a column. We want the unique values from two of our columns: the group and the transect length.

```
# create a data frame that has a column for each group id and a column for the length that each group s
transect_length <- unique(green[c(1,4)])
transect_length
```

```
##     Group Transect_m
## 1       1         10
## 55      2         10
## 103     3         10
## 116     4         10
## 155     5         10
## 197     6         10
## 246     7         10
## 278     8         10
## 298     9         10
## 325    10         10
```

Let's check out our distribution of detection distances. We can use the `hist()` function to do this.

```
hist(green$Distance_m)
```



Based on the histogram, I'll suggest we truncate at 2.5 m. Let's set this value so we can refer to it later.

```
# Set truncation distance to eliminate extreme observations
trunc <- 2.5
```

We can set "cut points" for distance bins; in our case, we want every half meter.

This line of code creates a vector that will have values every half meter (`by = 0.5`), starting at 0 and running through the value we set for truncation (`trunc`).

```
distance_bins <- seq(0, trunc, by = 0.5)
```

`unmarked` Data Prep

As we've seen before, the `unmarked` package likes to have data set up in a very specific way and has specific functions for this.

To get our detection functions and density estimates, we will use the `formatDistData` to get the data into the correct format.

```
# create an unmarked data frame
green_data <- formatDistData(distData = green,          # data frame with the data points
                         distCol = "Distance_m",        # column that holds the distance values
                         transectNameCol = "Group",     # column that holds the transect groups
                         dist.breaks = distance_bins)   # vector of distance groupings
```

```
## Warning in formatDistData(distData = green, distCol = "Distance_m",
## transectNameCol = "Group", : The transects were converted to a factor
```

```
green_data
```

```
##    [0,0.5] (0.5,1] (1,1.5] (1.5,2] (2,2.5]
## 1        7       5       4       5       0
## 2       11       4       3       0       1
## 3        5       2       4       1       1
## 4        7       4       3       0       0
## 5        6       5       3       5       3
## 6        7      11       0       0       0
## 7        7       3       0       4       2
## 8        9       6       1       3       1
## 9        5       0       3       3       1
## 10      15      10       4       0       0
```

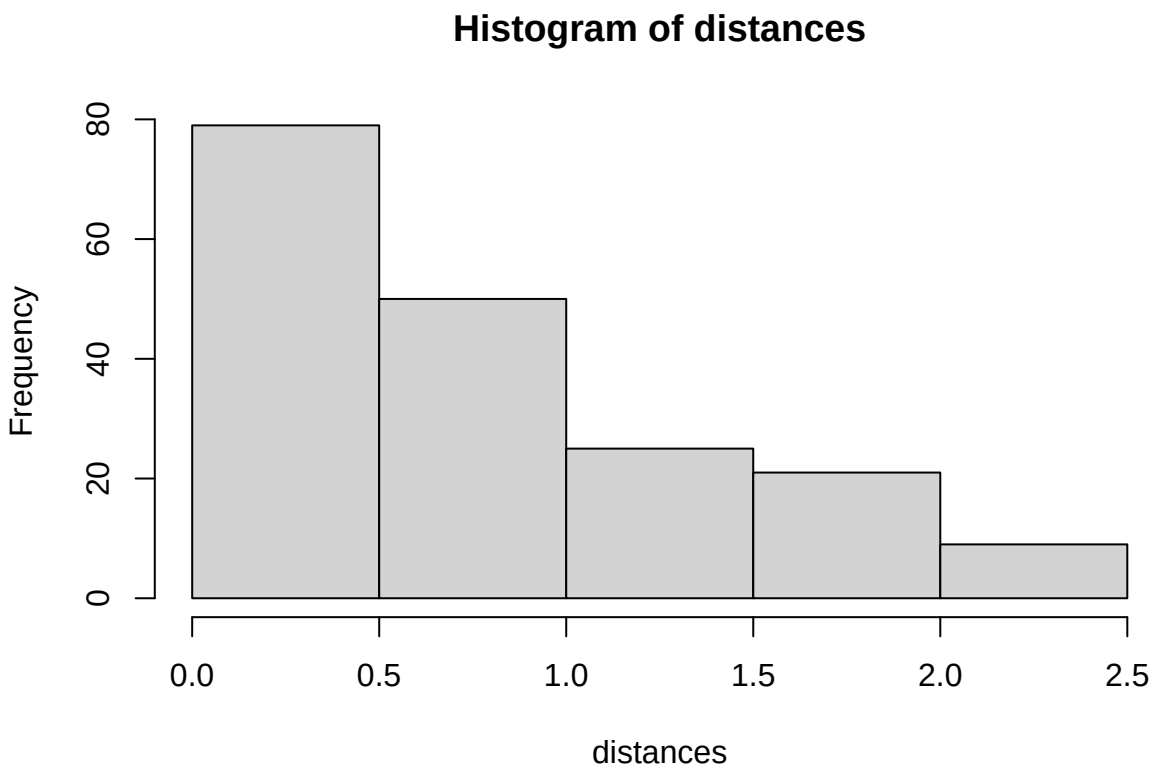
Next, we need to assemble data into the format required by `unmarked`, called an "unmarked frame". We've seen something similar before, but this one is `unmarkedFrameDS`, with the DS meaning "distance sampling."

```
UMF_green <- unmarkedFrameDS(y = green_data,                    # prepared data
                        survey = "line",                        # type of survey conducted (line or poin
                        tlength = transect_length$Transect_m,  # column with each transect length
                        dist.breaks = distance_bins,            # vector of how distances are binned
                        unitsIn = "m")                          # prepared data
UMF_green
```

```
## Data frame representation of unmarkedFrame object.
##    y.1 y.2 y.3 y.4 y.5
## 1    7   5   4   5   0
## 2   11   4   3   0   1
## 3    5   2   4   1   1
## 4    7   4   3   0   0
## 5    6   5   3   5   3
## 6    7  11   0   0   0
## 7    7   3   0   4   2
## 8    9   6   1   3   1
## 9    5   0   3   3   1
## 10  15  10   4   0   0
```

Let's check the distribution of detection distances to be used for analysis. These should *not* include the values that we truncated.

```
hist(UMF_green)
```



Models

Now that our data is in the correct format for `unmarked`, we can use the `distsamp` function from `unmarked` to create our 4 types of models: half normal, hazard rate, uniform, and negative exponential.

Our options for the `unitsOut` argument is either hectares (ha) or kilomenter (km). We will use hectares. This means that the density we calculate will be the density of beads per hectare.

```
HN   <- distsamp(~1 ~1, UMF_green, keyfun = "halfnorm", output = "density", unitsOut = "ha")
HR   <- distsamp(~1 ~1, UMF_green, keyfun = "hazard", output = "density", unitsOut = "ha")
```

```
## Warning: Hessian is singular. Try providing starting values or using fewer
## covariates.
```

```
Unif <- distsamp(~1 ~1, UMF_green, keyfun = "uniform", output = "density", unitsOut = "ha")
Exp  <- distsamp(~1 ~1, UMF_green, keyfun = "exp", output = "density", unitsOut = "ha")
```

Model Selection

Which model should we choose? We will use AIC again to help us figure it out. We want the model with the *lowest* AIC value.

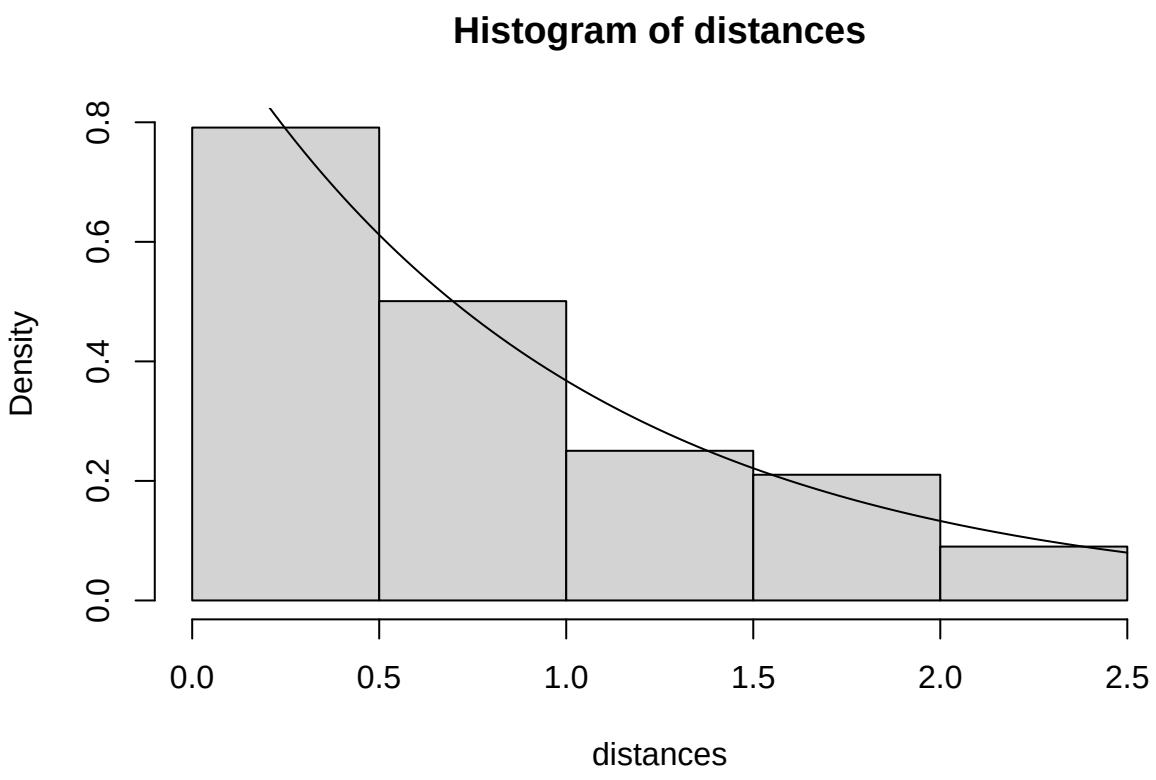
```
models <- fitList('Half Normal' = HN,
                  'Hazard Rate' = HR,
                  'Uniform'     = Unif,
                  'Exponential' = Exp)
modSel(models)
```

```
## Hessian is singular.
```

```
##              nPars    AIC delta   AICwt cumltvWt
## Exponential      2 214.13  0.00 9.0e-01     0.90
## Half Normal      2 218.59  4.46 9.7e-02     1.00
## Uniform          1 295.69 81.56 1.8e-18     1.00
## Hazard Rate      3 297.69 83.56 6.5e-19     1.00
```

Let's plot what the half-normal model looks like with our data.

```
hist(Exp)
```



Density Estimate

Now that we've chosen our top model (half normal, in this case), we can get our density estimate!

There is a handy function that we can use to pull the density estimate out of the model: `backTransform()`

```
density <- backTransform(Exp, type = "state") # Density estimate (no./ha)
density
```

```
## Backtransformed linear combination(s) of Density estimate(s)
##
##  Estimate   SE LinComb (Intercept)
##     10159 1211    9.23           1
##
## Transformation: exp
```

We can also calculate the confidence intervals for our parameter estimate

```
exp(confint(Exp, type = "state"))  # CI for density
```

```
##              0.025    0.975
## lam(Int) 8042.138 12831.96
```

Convert to Our Sample Area

You won't need to do this in your assignment—we will just leave all of our density estimates in the numbers per hectare. For our bead data, however, we definitely didn't sample a whole hectare (100m x 100m).

To see how close our estimate was to the actual number of beads we put outside (#), we need to convert the density in hectares to the density of our survey area.

Our first step is to calculate our effective strip width.

```
# density estimate
density_est <- density@estimate
density_est
```

```
## [1] 10158.56
```

```
# standard deviation
rate <- backTransform(Exp, type = "det")@estimate
rate
```

```
## [1] 0.9828822
```

```
# calculate the effective strip width
esw <- integrate(gxexp, lower = 0, upper = 2.5, rate = rate)$value
esw
```

```
## [1] 0.9056397
```

We can then calculate the area surveyed

```
# Our survey area = 10 groups x (2*esw*L)
area <- 10 * (2 * esw * 10)
area
```

```
## [1] 181.1279
```

```
# One hectare = 100 m x 100 m
ha <- 100 * 100 # 10,000 m^2
ha
```

```
## [1] 10000
```

```r
# Proportion of hectare we surveyed
surveyed <- area/ha
surveyed
```

```
## [1] 0.01811279
```

Finally, if we multiple our density estimate by the amount of area we surveyed, we will get the density of beads for the entirety of our sample area (same as abundance!)

```r
# D-hat * surveyed
density_est * surveyed
```

```
## [1] 184
```

I put out 150 green beads, so this is off by a bit! That's ok, though. We did *repeated* surveys in the same area and did not take that into account in our model. If we were going to accurately analyze our data, we would definitely need to do that.