Introduction to R for distance sampling

2. Using real data

In this practical, we use some real data to fit different detection function models and estimate density and abundance. The data were collected during a line transect survey of duck nests in Monte Vista National Wildlife Refuge, Colorado, USA. Transects were of length 128.75km and a distance out to 2.4m was searched. Distances of detected nests have been provided in a 'csv' text file in a basic format required by 'Distance'. The columns in the the file are:

- Study.Area - name of the study region (Monte Vista NWR)
- Region.Label - identifier of regions (in this case there is only one region and it is set to 'Default')
- Sample.Label - line transect identifier
- distance - perpendicular distances (m).

The distances allow different key functions/adjustments to be fitted in the detection function model and, by including the transect lengths and area of the region, density and abundance can be estimated.

Objectives of the practical

1. Import a text file
2. Understand the structure of a data frame
3. Fit different key functions/adjustments in the detection function model
4. Explore the model object i.e. `ddf` and `dht`
5. Create a data frame
6. Estimate density and abundance using `ds`.

Importing the data

To let R know where to look for files (and also where to save the R workspace and '.Rmd' files) we can set the 'working directory'; from the menu along the top of the `RStudio` window click on 'Session > Set Working Directory > Choose Directory' and select your chosen directory, for example 'C:/workshop'.

Load the data into `R` with the following command:

```
nests <- read.csv(file="ducknests.csv", header=TRUE)
```

This command is made up of several components:

- `read.csv` is an in-built `R` function which reads in a data file of type 'csv',
- the function has two arguments specified; `file` specifies the name of the data file and `header=TRUE` specifies that the first row of the data file contains the names of the data columns. If the columns had not been named in the data file, then `header=FALSE` should be specified and the columns would have been called `X1`, `X2`, etc. automatically. They can be subsequently renamed.
- the `<-` symbol has assigned the data set to an object called `nests`. Note that there is now an object called `nests` listed on the 'Environment' tab.

To check that the data file has been read into R correctly, use the `head` and `tail` 'functions' to look at the top and bottom rows of the data, respectively. To look at the first few rows of `nests` type the following command.

```
head(nests)
```

```
##        Study.Area Region.Label Sample.Label distance
## 1 Monte Vista NWR      Default            1     0.06
## 2 Monte Vista NWR      Default            1     0.07
## 3 Monte Vista NWR      Default            1     0.04
## 4 Monte Vista NWR      Default            1     0.01
## 5 Monte Vista NWR      Default            1     0.37
## 6 Monte Vista NWR      Default            1     0.36
```

The `head` function as used above displays the first 6 records of the named object. By default, `head` and `tail` display 6 rows of data but this can be changed by specifying a value for the function argument which controls this action. To display the last 2 records in the data, type the command:

```
tail(nests, n=2)
```

```
##          Study.Area Region.Label Sample.Label distance
## 533 Monte Vista NWR      Default           20     2.38
## 534 Monte Vista NWR      Default           20     2.13
```

In this function, `n` is the argument which controls the number of rows to display.

The object `nests` is a dataframe object made up of rows and columns. Use the function `dim` to find out the dimensions of the data set (i.e. the total number of rows and columns):

```
dim(nests)
```

How many rows and columns are there in `nests`?

```
## Number of rows 534  Number of columns  4
```

Another way to look at a data frame is to move to the 'Environment tab' and click on the rectangle (with the grid); this opens a new tab showing the data.

Summarising the perpendicular distances

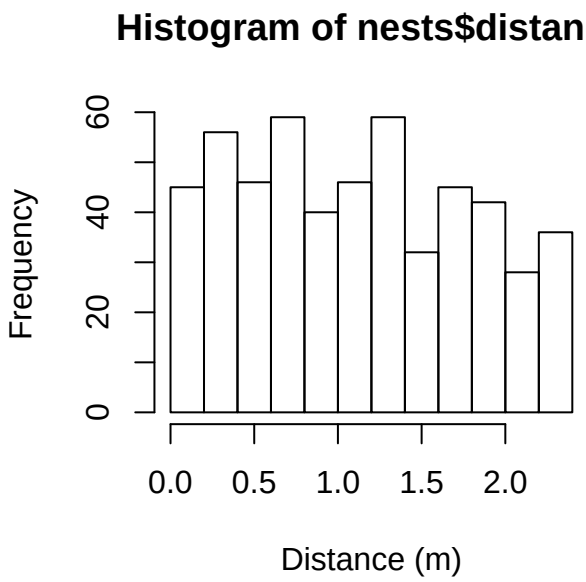
To access an individual column within a data frame we use the `$` symbol, for example to summarise the distances, then the following command is used:

```
summary(nests$distance)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.010   0.540   1.080   1.117   1.670   2.400
```

Similarly to plot the histogram of distances, the command is:

```
hist(nests$distance, xlab="Distance (m)")
```



Fitting different models

To use the `ds` function, first ensure that the `Distance` package (Miller 2017) has been loaded - see practical 1 for help on this.

The function `ds` requires a data frame to have a column called `distance` - since we have this in our `nests` data, we can simply specify the name of the data frame as follows:

```
nest.model1 <- ds(nests, key="hn", adjustment=NULL)
```

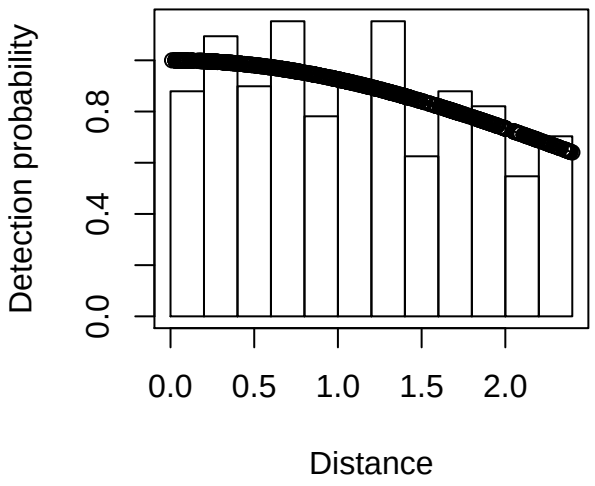
In this command, a half-normal key function is selected with no adjustment terms. Summarise the model:

```
summary(nest.model1)
```

```
##
## Summary for distance analysis
## Number of observations :  534
## Distance range         :  0  -  2.4
##
## Model : Half-normal key function
## AIC   : 928.1338
##
## Detection function parameters
## Scale coefficient(s):
##             estimate        se
## (Intercept) 0.9328967 0.1703933
##
##                        Estimate          SE         CV
## Average p              0.8693482  0.03902053 0.04488481
## N in covered region 614.2533225 29.19683067 0.04753223
```

Plot the detection function with the histogram having 12 bins:

```
plot(nest.model1, nc=12)
```



To fit different detection function shapes, we can change the `key` and `adjustment` arguments. For example to fit a half-normal key function with cosine adjustment terms, then use the command:

```
nest.model2 <- ds(nests, key="hn", adjustment="cos")
```

By default, AIC selection will be used to fit adjustment terms of up to order 5. Have any adjustment terms been selected?

```
## No adjustment terms have been selected.
```

```
##
## Summary for distance analysis
## Number of observations :  534
## Distance range         :  0  -  2.4
##
## Model : Half-normal key function
## AIC   : 928.1338
##
## Detection function parameters
## Scale coefficient(s):
##             estimate        se
## (Intercept) 0.9328967 0.1703933
##
##                         Estimate          SE         CV
## Average p              0.8693482  0.03902053 0.04488481
## N in covered region 614.2533225 29.19683067 0.04753223
```

To fit a hazard rate key function with hermite polynomial adjustment terms, then use the command:

```
nest.model3 <- ds(nests, key="hr", adjustment="herm")
```

```
##
## Summary for distance analysis
## Number of observations :  534
## Distance range         :  0  -  2.4
##
```

```
## Model : Hazard-rate key function
## AIC   : 929.7934
##
## Detection function parameters
## Scale coefficient(s):
##              estimate        se
## (Intercept) 0.9190194 0.2081042
##
## Shape coefficient(s):
##              estimate        se
## (Intercept) 0.2899026 0.6393387
##
##                         Estimate          SE         CV
## Average p              0.8890698  0.04958136 0.05576768
## N in covered region 600.6277685 34.59620441 0.05760007
```

Use the `help` command to find out what other key functions and adjustment terms are available.

The `ds` object

The objects created with `ds` (e.g. `nest.model1`) are made up of several parts. We can list them using the `names` function as below:

```
names(nest.model1)
```

```
## [1] "ddf" "dht"
```

The detection function information is in the `ddf` part and the density and abundance estimates would be stored in the `dht` part. To access each part, then the `$` can be used (as with columns in a data frame). For example to see what information is stored in the `ddf` part, we can use the `names` function again:

```
names(nest.model1$ddf)
```

```
##  [1] "call"         "data"         "model"        "meta.data"
##  [5] "control"      "method"       "ds"           "par"
##  [9] "lnl"          "hessian"      "dsmodel"      "criterion"
## [13] "fitted"       "Nhat"         "name.message"
```

The `dht` part is essentially empty because we haven't included the necessary information to calculate density. We will do this later.

```
nest.model1$dht
```

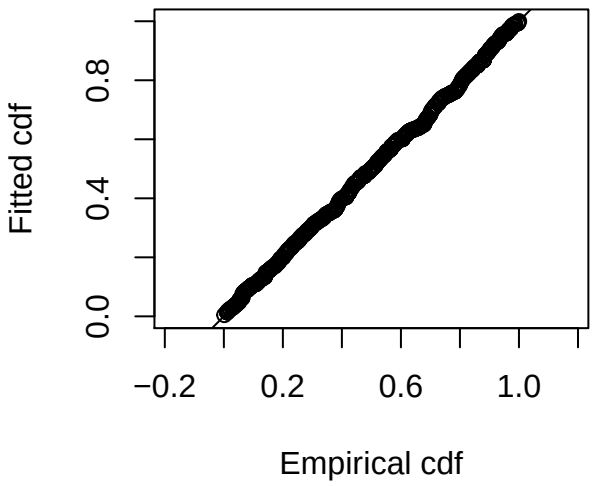
```
## NULL
```

Goodness of fit

The usual tools for checking goodness of fit are available, for example:

```
ddf.gof(nest.model1$ddf, nc=12)
```

Noting that the `ddf` part of the object is passed to the `ddf.gof` function. In this command, the number of bins has been set to 12 for the chi-squared goodness of fit test.




```
##
## Goodness of fit results for ddf object
##
## Chi-square tests
##             [0,0.2]  (0.2,0.4] (0.4,0.6] (0.6,0.8]  (0.8,1]    (1,1.2]
## Observed  43.000000 57.0000000  47.00000 58.000000 38.00000 49.0000000
## Expected  51.135009 50.8195759  50.19453 49.271358 48.06681 46.6024569
## Chisquare  1.294189  0.7516324   0.20331  1.546318  2.10833  0.1233457
##           (1.2,1.4] (1.4,1.6]  (1.6,1.8]    (1.8,2]    (2,2.2]  (2.2,2.4]
## Observed  59.000000 32.000000 45.0000000 36.0000000 33.0000000 37.0000000
## Expected  44.903997 43.000538 40.9237553 38.7070224 36.3845281 33.9904119
## Chisquare  4.424935  2.814194  0.4060178  0.1893189  0.3148325  0.2664758
##              Total
## Observed  534.0000
## Expected  534.0000
## Chisquare  14.4429
##
## P = 0.15373 with 10 degrees of freedom
##
## Distance sampling Kolmogorov-Smirnov test
## Test statistic =  0.027023  P =  0.83034
##
## Distance sampling Cramer-von Mises test (unweighted)
## Test statistic =  0.035363  P =  0.95542
```

Estimating density and abundance

So far, we have concentrated on the detection function but with more information, such transect lengths and the area of the region, we can estimate density and abundance. This information can be included in the text file with the distances but this would involve repeating some information and so `Distance` also uses a series of linked tables. Even though it is a bit more work, we will use the latter approach and illustrate a few more

R commands.

The tables are linked with columns that have reserved names, for example, the detected distances are in a column called `distance`. The other columns (e.g. `Region.Label`, `Sample.Label`) are also reserved names and can be used to link to other tables.

`Sample.Label` contains the transect identifier, the values can be listed using the `unique` function:

```
unique(nests$Sample.Label)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

We can create a data frame containing the name of the transect (sample) and search effort and also the name of the region. In this example, there was only one region and the effort was the same for all transects (128.75km) and so this information gets repeated.

```
nests.sample <- data.frame(Region.Label=rep("Default", times=20), Sample.Label=1:20,
                           Effort=rep(128.75, times=20))
```

Let's see what has this done:

```
head(nests.sample)
```

```
##   Region.Label Sample.Label Effort
## 1      Default            1 128.75
## 2      Default            2 128.75
## 3      Default            3 128.75
## 4      Default            4 128.75
## 5      Default            5 128.75
## 6      Default            6 128.75
```

There is only one region and so the region table is trivial - for the purposes of this practical, we assume that the area of the region was 50km^2.

```
nests.region <- data.frame(Region.Label="Default", Area=50)
```

```
##   Region.Label Area
## 1      Default   50
```

The last table to create is for the observations; some of the required information is in `nests` and so we can select the necessary columns as follows:

```
nests.obs <- nests[ ,2:3]
```

Finally, we need to create a unique identifier for each detected nest:

```
nests.obs$object <- 1:534
```

Check what this has done.

```
head(nests.obs)
```

```
##   Region.Label Sample.Label object
## 1      Default            1      1
## 2      Default            1      2
## 3      Default            1      3
## 4      Default            1      4
## 5      Default            1      5
## 6      Default            1      6
```

Now we have all the necessary information to estimate density and abundance. These tables are included in the `ds` function:

```
nest.model4 <- ds(nests, key="hn", adjustment=NULL, region.table=nests.region,
```

```
sample.table=nests.sample, obs.table=nests.obs, convert.units=0.001)
```

The `convert.units` argument ensures that the correct units are specified - in this example, distances are in metres, lengths in km and the area in km^2.

Having run the command, the estimates are stored in the `dht` part of the object:

```
nest.model4$dht
```

```
## Fitting half-normal key function

## Key only model: not constraining for monotonicity.

## AIC= 928.134

##
## Summary statistics:
##    Region Area CoveredArea Effort   n  k        ER       se.ER      cv.ER
## 1 Default   50       12.36   2575 534 20 0.2073786 0.007970756 0.03843576
##
## Abundance:
##   Label Estimate       se         cv      lcl      ucl       df
## 1 Total 2484.844 146.8363 0.05909276 2210.165 2793.659 99.55689
##
## Density:
##   Label Estimate       se         cv     lcl      ucl       df
## 1 Total 49.69687 2.936725 0.05909276 44.2033 55.87318 99.55689
```

References

Buckland ST, Rexstad EA, Marques TA and Oedekoven CS (2015) Distance Sampling: Methods and Applications. Springer 277 pp. ISBN: 978-3-319-19218-5 (Print) 978-3-319-19219-2 (Online)

Miller DL (2017). Distance: Distance Sampling Detection Function and Abundance Estimation. R package version 0.9.7. https://CRAN.R-project.org/package=Distance

R Core Team (2017) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/