BIOE 440R

HW4: Analysis of line-transect data for magpies/crows using distance sampling methods with the R package *unmarked*.

1. Enter your data into two files using Excel or similar software that will let you enter the data easily and then export it as a comma-delimited text file (.csv) with no hidden formatting.

   One file (here, I will call it **dists.csv)** should have one row for each occasion that you detected an individual or flock. At a minimum, it must have columns that include:

   - the transect ID (trans in the example with puku)
   - flock size (herd.size in the example with puku)
   - perpendicular distance (dist in the example with puku).

   To calculate perpendicular distances:

   (a) Calculate the sighting angle: this is the difference between the transect bearing and the bearing to the animal. This should be a number between 0 and 90.
   (b) If the sighting angle is between 90 and 179, recalculate it as 180 minus sighting angle.
   (c) If the sighting angle is between 180 and 270, recalculate it as sighting angle minus 180
   (d) If the sighting angle is between 271 and 360, recalculate it as 360 minus sighting angle
   (e) (b) Perpendicular distance = sin(sighting angle) X distance to animal.

   The second file (I will call it **covs.csv**) should have one row for each transect. It must have columns that include:

   - The transect ID, using the same variable name and same transect codes as in dists.csv (trans, in the example with puku)
   - The length of the transect in meters (length in the example with puku)
   - one column for each of the variables you measured to test for effects on density or detection.

2. Save dists.csv and covs.csv as comma-delimited text files. Use the 'Save as type' option to do this in Excel

3. Create an R script to do the analysis. All of the remaining steps of the analysis are accomplished by adding lines to your script.

4. Clean up the memory so that there are no complications due to objects already in memory. Do this with `rm(list = ls())`

5. Read the files into R using using the import dataset button in R Studio, or with `read.csv()` or `read.table(sep = ',')`, after you've set the working directory to the location with the data files. Something like:

   ```
   dists <- read.csv(dists.csv)
   ```

   And examine the data to be sure everything is correct

   ```
   head(dists)
   tail(dists)
   ```

6. Load the unmarked package with `library(unmarked)`. You'll have to install it first, if you have not yet done so.

7. Be sure that the transect id variable is a factor, with code like:

   ```
   dists$trans <- as.factor(dists$trans)
   ```

8. Do not truncate the data (as I did in the example with puku). Use all of your data.

9. Format your distance data with code like:

   ```
   yDat <- formatDistData(dists, distCol="dist", transectNameCol="trans",
   dist.breaks=c(0,100, 200,300))
   ```

   The first argument of the `formatDistData()` function is the object you used in step 4 to save the dists.csv file.

   The second argument (`distCol`) identifies the column name with the perpendicular distances.

   The third argument (`transectNameCol`) identifies the column name with the transect ids.

   The last argument (`dist.breaks`) sets up the bins of perpendicular distances that will be used to fit the detection function. You'll have to look at your data to identify a set of breaks that make sense for the set of perpendicular distances in your data set. 3-5 bins that go out to the maximum distance is reasonable. You can use the histogram (see step 14) to refine this decision, by going through this step more than once.

10. Examine the formatted distance data

```
ydat (or whatever name you assigned it to)
```

Will display the entire formatted dataset on the console.

11. Read in the covariates file and inspect it.

```
covs <- read.csv(covs.csv, header=TRUE)
head(covs)
tail(covs)
```

12. Make the unmarked dataframe.

```
umf <- unmarkedFrameDS(y=as.matrix(yDat), siteCovs=covs, survey="line",
dist.breaks=c(0, 100, 200,300), tlength=covs$length,  unitsIn="m")
```

use F1 help if necessary to understand the arguments of the `unmarkedFrameDS()` function

13. Examine the histogram of detections as a function of distance

```
hist(umf, xlab="distance (m)", main="", cex.lab=0.8, cex.axis=0.8)
```

Go back to step 9 if the distribution does not look reasonable with the breaks you've selected.

14. Fit the model or set of models that test your hypothesis/hypotheses, using the `distsamp()` function.

```
m.haz.1.type <- distsamp(~1 ~type, umf, keyfun="hazard",
output="density", unitsOut="kmsq")
```

the first argument is the model for detection
the second argument is the model for density
the third argument is the object name for the unmarked data frame you created in step 12
the third argument (`keyfun`) is the function used for detection as a function of distance
the fourth (`output`) specifies that you want the output to be densities
the last (`unitsOut`) specifies that the reported density will be in units of flocks per km^2

15. Use the coefficients, their standard errors and the P-values to make inferences about your hypothesis.

16. You do **not** have to do any goodness of fit testing. You do **not** have to convert the density of flocks to the density of individuals. You also do **not** have to use the `predict()` function or make a graph to show the results, but that would be worth 5 points of extra credit.

17. Turn in a document with the following

A) A clear statement of your hypothesis
B) Printout of your data files, or at least the first and last rows
C) The R script you used to analyze your data
D) The output from the R script
E) A summary paragraph with your inferences and the supporting statistics you used to make the inferences.

A FAST AND EFFECTIVE WAY TO DO B,C & D is available in R Studio. Once you have completed the analysis (so you have your R script written and run), you can click on the File menu and then on the ***Compile Notebook*** option. This will create a nicely formatted file with the code and the output. If you examine your data in the code with head() and tail(), it will be included in the output.