Introduction to R for distance sampling

Workshop, 26 August 2018

Centre for Research into Ecological and Environmental Modelling

1.5 More complicated analyses

This practical is based on the Montrave songbird case study in Buckland *et al.* (2015), Chapter 5 with computer code under 'Montrave songbird case study. Both point and line transect surveys were conducted and here we use the data from the line transect data, although the issues (and solutions) will be similar for both.

These data are provided in a 'flat file' format (i.e. it contains all the necessary columns to estimate a detection function, density and abundance) rather than having separate files for detections, transects, regions (as in Practical 2). While both formats are equally valid, the 'flat file' approach has a particular idiosyncrasy which we exploit here to introduce more functions and data manipulation.

Several species of birds were identified but not all species were detected on all transects. If a simple data selection is performed to select records for a particular species, then not all of the transects will be included in the resulting data (because that species may not have been seen). This doesn't matter if we are only interested in fitting detection functions, but will matter if we wish to estimate density and abundance because the effort will be too low since some of the transects are missing. To correct for this, some data frame manipulation is required. There is generally more than one way to do something in R - for an alternative way see the computer code 'Montrave song bird case study' associated with Buckland *et al.* (2015).

Objectives of the practical

1. Data frame selection and manipulation
2. Extracting estimates from `dht` object
3. Customising detection function plots

Importing the data

The data is in a 'flat file' format and contains the following columns:

- Region.Label - name of study
- Area - size of study region (km^2)

- repeats - number of visits to transect
- Sample.Label - line transect identifier
- Effort - length of transect (km)
- distance - perpendicular distance (m)
- species - species of bird (c=chaffinch, g=great tit, r=robin and w =wren)
- visit - on which visit bird was detected.

Use the following command to import the data and then use the `head` command to ensure it has been imported correctly.

```
# Read in data file
birds <- read.csv("montrave-line.csv", header=T)
```

To start with, let's find out a bit about the data.

How many transects are there? The names of all the transect can be listed using:

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

We can easily see that there are 19 transects in total, however, we can check by combining two commands:

```
length(unique(birds$Sample.Label))
```

For now, save the transect labels to a new object as we will use them later on:

```
# Save the transect labels
tran.lab <- unique(birds$Sample.Label)
```

The `table` command is a quick way to determine how many detections there are of each species:

```
# Number of detections by species
table(birds$species)
```

Each of the line transects was visited twice which is not taken into account at present. However, it is straightforward to do so:

```
# Account for transects walked twice
birds$Effort <- birds$Effort * birds$repeats
```

Manipulating the robin data

For the purposes of this practical, we are interested in estimating the density of robins and so we select only these records:

```
# Select robins
robins <- birds[birds$species=="r", ]
```

Now let's see how many transects on which robins were detected:

```
length(unique(robins$Sample.Label))
```

If we were to use the `robins` data as it is at present to estimate density, then density would be underestimated because the search effort associated with three transects is missing. Adding these missing transects to the `robins` data, requires several steps:

1. identify the missing transects,
2. select the information for the missing transects,
3. get the missing information in the correct format,
4. add the missing information to the `robins` data.

The following commands identifies the missing commands. After each command, type the name of the object which has been created to see what each command has done.

```
# List of transect labels on which robins were seen
robin.lab <- unique(robins$Sample.Label)

# Transects missing from robin data
miss.lab <- tran.lab[!is.element(el=tran.lab, set=robin.lab)]
```

To understand what this latter command has done, it can be broken down into several elements:

- elements of `tran.lab` are selected using `[]`
- the `is.element` function (without the `!` symbol) selects the elements in `tran.lab`, which are also in the `set` argument (i.e. `robin.lab`)
- the `!` is used to select the elements in `tran.lab` that are NOT in `robin.lab`.

Now we know which transects are missing, we can select these records from the `birds` data frame:

```
# Select transects from data
miss.data <- birds[is.element(birds$Sample.Label, miss.lab), ]
```

However, the information about the transects are repeated in this new data frame because we have just selected all records for these transects. A quick check of the number of rows will confirm this:

```
length(miss.data$Sample.Label)
```

To get rid of rows where `Sample.Label` is duplicated use the command:

```r
# Get rid of duplicated information
miss.data <- miss.data[!duplicated(miss.data$Sample.Label), ]
```

This command has selected the records from `miss.data` for which the transect label is not duplicated.

We only want to keep the information about search effort and so data in the `distance`, `species` and `visit` columns are set to missing:

```r
# Set distances and species to missing - note use of "" for characters
miss.data$distance <- rep(NA, 3)
miss.data$species <- rep("NA", 3)
miss.data$visit <- rep(NA, 3)
```

Check what `miss.data` looks like.

The final thing to do is to add the missing data (`miss.data`) to the `robins` data frame using the `rbind` function (this combines data frames with the same columns).

```r
# Add missing transect info onto robin data
robins <- rbind(robins, miss.data)
```

Let's see the result of all this manipulation:

```r
# See what that has done
tail(robins)
```

If we wanted to be very tidy, then the data frame could be sorted so that the transect labels were in order:

```r
# Being very tidy - order by transect
robins <- robins[order(robins$Sample.Label), ]
```

Analysis

Before we fit any models, have a quick look at the histogram of distances:

```r
hist(robins$distance, breaks=20)
```

In line with Buckland *et al.* (2015), three different detection functions are fitted:

```r
# Half normal with hermite poly adjustment
robin.hn.herm <- ds(robins, truncation=95, transect="line", key="hn",
                    adjustment="herm", convert.units=0.1)

# Uniform with cosine adjustment
robin.uni.cos <- ds(robins, truncation=95, transect="line", key="unif",
                    adjustment="cos", convert.units=0.1)
```

```
# Hazard rate with simple polynomial adjustment
robin.haz.simp <- ds(robins, truncation=95, transect="line", key="hr",
                     adjustment="poly", convert.units=0.1)
```

Examining the `dht` object

As we have seen in a previous practical, the fitted model object (e.g. `robin.uni.cos`) is made up of two parts; the detection function in the `ddf` part and the estimates in the `dht` part. In this section, we look at the `dht` part.

To list the elements that are contained in `dht`, use the `names` function:

```
names(robin.uni.cos$dht)
```

Detections were of individual birds and so group size was not included in these data - if it had been included (in a column called `size`), then as well as `individuals` there would have been elements `clusters` and `Expected.S`.

The estimates stored in the `individuals` object can be listed in a similar manner:

```
names(robin.uni.cos$dht$individuals)
```

To collect together the density estimates (and estimates of precision) from all the fitted models, we can use the following command:

```
# Collect together results
model.results <- rbind(robin.uni.cos$dht$individuals$D,
                       robin.haz.simp$dht$individuals$D,
                       robin.hn.herm$dht$individuals$D)
```

Type the name of the new object to see how the estimates compare between the different detection function models.

Goodness of fit

Here we look at goodness of fit test with unequal bin intervals and just consider one of the fitted models. First we specify the required bin intervals.

```
# Specify breaks - note irregular spacing
robin.brks <- c(0, 12.5, 22.5, 32.5, 42.5, 52.5, 62.5, 77.5, 95.0)
```

Perform the tests:

```
ddf.gof(robin.uni.cos$ddf, breaks=robin.brks)
```

Customising the detection function plot

The `plot` function provides a basic plot of the fitted detection function overlaid onto the scaled distribution of distances:

```r
plot(robin.uni.cos)
```

However, the plot can be customised for reporting:

```r
# Plot detection function
plot(robin.uni.cos, showpoints=FALSE, pl.den=10, lwd=2, breaks=robin.brks,
     main="Uniform-cosine", xlab="Distance (m)")
```

The arguments are:

- `showpoints` - logical indicating whether observed distances are shown
- `lwd` - line width (1=default)
- `pl.den` - density of shading of histogram (0=no shading)

For other options see `help(plot.ds)` (Note `plot` is a generic function which selects a relevant type of plot based the the object).

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)