Mark-Recapture Distance Sampling

Workshop, 28, 31 October 2022

Centre for Research into Ecological and Environmental Modelling

Hidden Markov Line Transect Model Exercise

1 The `hmltm` package

This document takes you through an example using the `hmltm` package to fit a hidden Markov line transect model to line transect survey data. The package implements the method of Borchers *et al.* (2013), with some extensions, and is available on GitHub here: https://github.com/david-borchers/hmltm. It was not written as a user-friendly general package and has very little in the way of error traps and data checks. Examples of its use can be found in Rekdal *et al.* (2015) and Heide-Jorgensen *et al.* (2017)

2 The Data

The data used in this exercise were gathered on a shipboard survey of beaked whales in the Alboran Sea in 2008, from a vessel moving at about 3.5 m/s. Beaked whales have long dives and spend a lot of time underwater and unobservable. Focal follows of whales gave estimates of mean times available (at surface) and unavailable (diving) in a single dive cycle, of 122 seconds (about 2 minutes) and 1,580 seconds (about 26 minutes), respecively, with standard errors of these means of 9.62 seconds and 134.9 seconds, respectively. With these mean times available and unavailable, animals are available only about 7% of the time!

A total of 81 groups were detected on the survey. The sightings data contain forward distance (y) as well as perpendicular distance (x), and also group size ($size$), and the height of the observer who made the detection (ht). We are going to focus on x and y only here. The survey was not a MRDS survey and so there is no duplicate data.You can load and look at the data like this:

```r
library(hmltm)  # load the library
data("beaked.ship") # load the data (contained in the library)
head(beaked.ship)
```

3 Conventional Distance Sampling Analysis

Let's start by fitting a conventional disance sampling model to the data. To do that we first have to rename some columns in the data frame `beaked.ship` to comply with the conventions of the `Distance` package. We will put the data frame with the new column names in something called `bkdsdat`, as per these R commands

```
bkdsdat = beaked.ship
names(bkdsdat)[1:4] = c("Region.Label", "Area", "Sample.Label", "Effort")
names(bkdsdat)[which(names(bkdsdat)=="x")] = "distance"

head(bkdsdat)
```

To fit a CDS model, we need the `Distance` package and we need to specify the units of the distances, transect lengths and areas, which are as follows:

```
library(Distance)
conversion.factor <- convert_units("meter", "kilometer", "Square kilometer")
```

We will use a hazard-rate model as this turns out to be adequate for these data:

```
bk.hr <- ds(data=bkdsdat, key="hr", adjustment=NULL,convert_units=conversion.factor)
```

Check that it seems adequate before proceeding:

```
cutpoints = seq(0,max(na.omit(bkdsdat$distance)),length=21)
par(mfrow=c(1,2))
plot(bk.hr, breaks=cutpoints, main="Hazard-rate model, beaked shipboard survey")
gof_ds(bk.hr)
```

We know that $g(0)$ is less than 1 (because of how long the whales dive), so the CDS estimates will be negatively biased, but let's look at the total abundance estimate anyway:

```
cds.N = c(est=bk.hr$dht$individuals$N$Estimate[3],
          lcl=bk.hr$dht$individuals$N$lcl[3],
          ucl=bk.hr$dht$individuals$N$ucl[3])
t(round(as.data.frame(cds.N)))
```

4 CDS with Correction Factors

Let's now calculate various correction factors and use them to "correct' ' for $g(0) < 1$. We will do this using the simple (instantaneous) correction factor (which is just the proportion of time available), McLaren's factor, and Laake's factor. The `hmltm` package has functions that allow you to do this, given the mean times available and unavailable, and (optionally) estimates of the CVs of these times. McLaren's and Laake's factors require us to specify a forward distance at which animals are first detectable if they are available (we will call this `ymax`).

Let's plot the data in x- and y- dimensions to figure out what a reasonable `ymax` would be

```
xcuts = seq(0,max(na.omit(beaked.ship$x)),length=21)
ycuts = seq(0,max(na.omit(beaked.ship$y)),length=21)
layout(matrix(1:4,nrow=2))
hist(beaked.ship$x,breaks=xcuts,xlab="Perpendicular Distance",main="")
plot(beaked.ship$x, beaked.ship$y, pch="+",xlab="Perpendicular Distance",
     ylab="Forward Distance")
plot(0,0,xaxt="n",yaxt="n",bty="n",xlab="",ylab="",type="n")
hist(beaked.ship$y,breaks=xcuts,xlab="Forward Distance",main="")
```

From this, a `ymax` equal to 4,500 seems reasonable, so let's go with that. We will also calculate the "time in view'' – the time that that it takes a stationary animal to go from a distance `ymax` ahead of the observer, to abeam of the observer:

```r
ymax = 4500 # distance beyond which cannot detect animal
# Calculate time in view, in minutes
spd = 3.5 # ship speed in m/s
tiv = ymax/spd / 60 # minutes in view
```

The `hmltm` package has a funtion `make.hmm.pars.from.Et` that constructs the HMM parameter object needed for inference from the mean times unavailable, mean time available, and their standard errors, so let's make this object. (By default these mean times are assumed to be independent, although the function also allows you to specify a covariance.)

```r
library(hmltm)  # load the library

Eu=1580 # mean time UNavailable -per dive cycle
Ea=122 # mean time available per dive cycle
seEu=134.9 # std error of mean time Unavailable
seEa=9.62 # std error of mean time available

# Make availability parameters object
availpars = make.hmm.pars.from.Et(Eu=Eu, Ea=Ea, seEu=seEu, seEa=seEa)
```

The time an animal is within detectable distance is 21.43 minutes, while the mean time UNavailable is 26.33 minutes and the mean time available is 2.03 minutes.

To get a feel where and how long animals will be available for detection using these numbers, here is a little function that takes an object like `availpars`, and a given time-in-view (in minutes), simulates an availability sequence from it, and does a crude plot. The blue is sea, the black is when the animal is available, and the "obs window'' indicates the region within which available animals are at risk of detection when available.

```r
plot.sim.avail = function(availpars,mins.sim,mins.in.view,seed=NULL) {
  Pi = availpars$Pi[,,1]
  delta = as.vector(availpars$delta)
  pm = list(prob=c(0.00001,0.99999))
  nsim = mins.sim*60
  if(!is.null(seed)) set.seed(seed)
  pn=list(size=rep(1,nsim))
  dthmmobj = dthmm(NULL,Pi,delta,distn="binom",pm,pn,discrete=TRUE)
  avail = simulate(dthmmobj,nsim)$x
  mins = (1:length(avail))/60
  ylim=c(0.95,1.05)

  plot(mins,avail,type="n",xlab="Minutes",col="lightgray",
       ylim=ylim,yaxt="n",ylab="")
  sealevel = 0.999
  xs = range(-0.05*mins,1.05*mins)
  ys = c(0,sealevel)
  polygon(c(xs[1],xs[1],xs[2],xs[2]),c(ys[1],ys[2],ys[2],ys[1]),density=25,
          col="lightblue")
  abline(sealevel,0,col="lightblue",lwd=2)
  up = which(avail==1)
 # lines(mins,avail,col="gray")
  points(mins[up],avail[up],pch=19,cex=1)
  polygon(c(0,0,mins.in.view,mins.in.view),c(1,ylim[2],ylim[2],1))
  legend(0,ylim[2],legend="obs window",bty="n",cex=0.5)
}

# now use the function to plot a scenario:
minutes.to.simulate = 100
minutes.in.view = 21
plot.sim.avail(availpars,minutes.to.simulate,minutes.in.view, seed=1)
```

If you remove the `seed` argument and run the command again, you'll get a different realisation of the availability process. It is worth doing this and simulating a few processes to get a feel for where and how long animals will be available:

```
# try again without the seed (run this a few times)
plot.sim.avail(availpars,minutes.to.simulate,minutes.in.view)
```

Clearly $g(0)$ is going to be a lot less than 1, because animals spend most of their time being unavailable for detection - but how much less? Let's calculate the naive estimate and ask McLaren and Laake. Package `hmltm` provides the functions `simple.a`, `mclaren.a`, and `laake.a` to do the calculations, using objects like `availpars`, which we created above, as input:

```
simple.cf = simple.a(availpars)
laake.cf = laake.a(availpars,spd=spd, ymax=ymax)
mclaren.cf = mclaren.a(availpars,spd=spd, ymax=ymax)
# put them in a data frame for convenience
cf = data.frame(simple=simple.cf$mean,
                mclaren=mclaren.cf$mean,
                laake=laake.cf$mean)
round(cf,4)
```

Let's use these to "correct'' the CDS abundance estimates:

```
# Point estimates of total abundance and 95% CI with each correction method
simple.N = cds.N/cf[,"simple"]
mclaren.N = cds.N/cf[,"mclaren"]
laake.N = cds.N/cf[,"laake"]
Nests = t(data.frame(simple=round(simple.N),
                     mclaren=round(mclaren.N),
                     laake=round(laake.N)))
Nests
```

There are big differences between the methods! Which one to believe?

Let's take another look at the forward distance data and think about what this implies for the validity of each of the above correction methods:

```
hist(beaked.ship$y,breaks=xcuts,xlab="Forward Distance",main="")
```

In the light of this plot and what you know about the survey, say whether each of the corrected estimates is biased or not, and in which direction each is likely biased if it is biased.

5 The Hidden Markov Line Transect Abundance Estimator

Now let's use the `hmltm` package to estimate abundance.

5.1 Setting things up

To use the hidden Markov line transect (HMLT) method, we need first to specify some key survey parameters:

```
# set survey and fitting parameters
spd=3.5 # observer speed
Wl=0 # left-truncation for perpendicular distance (none here)
W=4000 # right-truncatino for perpendicular distance
W.est=W # set perpendicular truncation distance for Horvitz-Thompson-like
#         abundance estimator
ymax=4500 # maximum forward distance beyond which nothing could be detected

# specify survey parameters and put them in an object called `survey.pars`,
# using the hmltm funciton `make.survey.pars
survey.pars=make.survey.pars(spd=3.5,Wl=0,W=4000,ymax=4500)

# set some fitting parameters
# hessian=FALSE says don't estimate Hessian matrix, and
# nx=64 says use 64 cutpoints when integrating over the perp distance dimension
control.fit=list(hessian=FALSE,nx=64)

# set some parameters for the optimiser
# trace specifies how much to report as function is optimised
# maxit is maximum number of iterations when optimising
# see help(optim) for more
control.opt=list(trace=0,maxit=1000)
```

5.2 Fitting a model

Now we fit the model with the function `est.hmltm`. We will use one particular 2-D detection hazard form here, but there are others, which we will say a bit more about elsewhere. All the detection hazard forms assume that an available animal that is at ***radial*** distance zero (not perpendicular distance zero) will be detected. Here we don't use any covariates. The function `est.hmltm` requires you to specify paramerter starting values for the detection hazard model, which is a bit of a pain, but the vignette that comes with the `hmltm` package gives a little guidance on this (and suitable starting values are given for the example below).

```
# specify model and starting parameter values and fit model
hfun="h.EP2x.0" # detection hazard model to use
models=list(y=NULL,x=NULL) # specify detection hazard model form (no covariates)
```

```r
# You need to give starting values to the optimiser:
pars=c(0.45, 0.31, 7.7, 62.8) # detection hazard parameter starting values

# Point estimation:
bkEP2x.null=est.hmltm(beaked.ship,pars,hfun,models,survey.pars,availpars,
                      control.fit,control.opt,W.est=W.est)
```

5.3 Checking goodness-of-fit

We can look at diagnostic plots similarly to the way we do for CDS models, but now we need to consider the goodness-of-fit in both the x- and the y-dimensions. (You can look at the help files for the functions used below for some information on what the various arguments mean - or you can just use them as given.)

```r
# Look at goodness of fit
par(mfrow=c(2,2))
# Plot the fit in the perpendicular (x) dimension, and save the results
bkEP2x.null.fx=fxfit.plot(bkEP2x.null,breaks=xcuts,type="prob",allx=FALSE)
# Plot the fit in the forward (y) dimension, and save the results
bkEP2x.null.fy=fyfit.plot(bkEP2x.null,breaks=ycuts,allx=FALSE,nys=250)
# Calculate goodness-of-fit in the perpendicular (x) dimension, do CDF-EDF plot,
# and save results
bkEP2x.null.gof.x=hmmlt.gof.x(bkEP2x.null)
# Calculate goodness-of-fit in the forward (y) dimension, do CDF-EDF plot,
# and save results
bkEP2x.null.gof.y=hmmlt.gof.y(bkEP2x.null,breaks=ycuts)
```

These plots and the associated Kolmogorov-Smirnov (KS) test statistics for the fits in the x- and the y-dimensions. Below are commands that put the KS p-values into a data frame and prints the data frame. Are the fits adequate?

```r
gof_hmltm = data.frame(perp_dist=bkEP2x.null.gof.x$p.ks,
                       forward_dist=bkEP2x.null.gof.y$p.ks)
round(gof_hmltm,3)
```

The point estimates are stored in the element `$point$ests` of the fitted object, and we can look at them thus:

```r
bkEP2x.null$point$ests
```

We need to bootstrap to get interval estimates.

5.4 Obtaining confidence intervals by bootstrap

Confidence intervals can be obtained by bootstrapping. For comparison with the correction factor methods used above, we treat the avaiability model as fixed and known here (`fixed.avail=TRUE`), but we need not do so in general. Similarly to other `hmltm` functions, you can get help on the use of the boostrap function `bs.hmltm` by typing `help(bs.hmltm)`. Below is a command to do 99 bootstraps (which will probably take around 6-12 minutes, depending on how fast your computer is).

```
set.seed(1) # for reproducibility of bootstraps
bests = bs.hmltm(bkEP2x.null,B=99,hmm.pars.bs=availpars,bs.trace=0,report.by=10,
                 fixed.avail=TRUE)
```

The functions `bootsum` and `strat.estable` help you summarise the estimates from bootstraps in a convenient way. Below are commands that do this, then add the abundance estimate and confidence intervals to the object `Nests` that we created earlier to hold the "corrected'' CDS estimates, and prints all:

```
bsum = bootsum(bests,bkEP2x.null)
hmltmNests = bsum[3,c("N","N.lo","N.hi")]
names(hmltmNests) = colnames(Nests)
Nests = rbind(Nests,hmltm=hmltmNests)
round(Nests)
```

Are these results consistent with the biases (if any) that you expected the various CDS correction methods to give? Which estimate(s) do you consider most reliable?

5.5 References:

Borchers, D.L., Zucchini, W., Heide-Jørgensen, M.P., Canadas, A. & Langrock, R. (2013). Using hidden Markov models to deal with availability bias on line transect surveys. *Biometrics* **69**, 703– 713.

Heide-Jørgensen M.P.,Hansen, R.G, Fossette, S., Nielsen, N.H., Borchers, D.L., Stern, H. and Witting, L. (2017). Rebuilding belugas stocks in West Greenland. *Animal Conservation* **20**, 282-293.

Rekdal, S.L., Hansen, R.G., Borchers, D.L., Bachmann, L., Laidre, K.L., Wiig, Ø., Nielsen, N.H., Fossette, S., Tervo, O. & Heide-Jørgensen, M.P. (2015). Trends in bowhead whales in West Greenland: aerial surveys vs. genetic capture-recapture analyses. *Mar. Mamm. Sci.* **31**, 133– 154.