REM analysis using camtools

Marcus Rowcliffe

11 March 2020

Background

This vignette steps through data processing and analysis for generating random encounter model density estimates and associated parameters (trap rates, activity level, detection zone dimensions). Functions and example data are available at https://github.com/MarcusRowcliffe/camtools.

Preparing data

Four dataframes are needed for the full REM analysis:

1. Metadata extracted from tagged images (`exifdat` below). The function `read.exif` from package `CTtracking` can be used, but before using it for the first time, you need to download the exiftool executable to your computer (https://www.sno.phy.queensu.ca/~phil/exiftool). For windows (the function isn't currently geared up for Mac OS), rename the executable file exiftool.exe and place in a new directory named C:/Exiftool. You can then run `read.exif` with a path to the image folder as the sole argument. This may take some time to complete. To avoid having to repeat this in future sessions, save the data as a csv file for future use.

```
path <- "./Data"
```

```
source(CTtracking)
exifdat <- read.exif(file.path(path, "CameraImages"))
write.csv(exifdat, file.path(path, "exifdata.csv"), row.names = FALSE)
```

2. Animal position data generated using tracking function `predict.pos` from package `CTtracking` (`posdat` below) - see tools and vignette at https://github.com/MarcusRowcliffe/CTtracking. .

3. Animal speed data generated using tracking function `seq.summary` from package `CTtracking` (`seqdat` below).

4. A deployment table, indicating the site and start and end date/times of each camera deployment (`depdat` below). For ths table, create a csv file with (at least) three columns named exactly "station"",”start"" and "stop". **station** should contain station identifiers that will match with those used during tagging. **start** and **stop** should contain text date/time values in a consistent format, preferably *yyyy:mm:dd HH:MM:SS*. This is the usual format for image metadata, and later processing functions expect this by default, although you can specify a different format if you need to. A portion of an example deployment dataframe is shown below. Note that all columns should be read as text format, not converted to factors (`stringsAsFactors=FALSE`). Note that station 10 has three camera deployments at different times, so takes three rows in this example. Station identiers here and in the exif data tags should ideally include an alphabetic charater.

```
exifdat <- read.csv(file.path(path, "exifdata.csv"), stringsAsFactors = FALSE)
posdat <- read.csv(file.path(path, "posdat.csv"), stringsAsFactors = FALSE)
seqdat <- read.csv(file.path(path, "seqdat.csv"), stringsAsFactors = FALSE)
depdat <- read.csv(file.path(path, "depdat.csv"), stringsAsFactors = FALSE)
depdat[8:14,]
```

```
##    station               start                stop
## 8     RP08 2017:10:02 20:26:08 2017:10:06 09:58:12
## 9     RP09 2017:10:23 20:26:02 2017:10:30 06:36:11
## 10    RP10 2017:09:28 02:00:55 2017:10:01 10:41:36
## 11    RP10 2017:10:12 09:37:34 2017:10:13 05:37:36
## 12    RP10 2017:10:21 20:19:59 2017:10:26 08:26:56
## 13    RP11 2017:10:12 00:46:50 2017:10:14 10:35:38
## 14    RP12 2017:09:28 12:17:18 2017:10:12 12:04:03
```

Generating and checking trap rate data

You will need functions from package `camtools`, available at github.com/MarcusRowcliffe/camtools.

```
source("camtools.R")
```

First, extract the tag and time/date information from the image metadata using function `extract.tags`. This returns a dataframe with a column for each tag field, by default with source file, creation time/date and original tag string columns added, plus an additional column, **time**, which extracts time of day from the time/date values and expresses it in radians. Subsequent processing requires specific column names for particular data elements, specifically **date** for record time/date, and **station** for station identifier. These columns may therefore need to be renamed.

```
tagdat <- extract.tags(exifdat)
tagdat <- plyr::rename(tagdat, c(CreateDate="date", placeID="station"))
head(tagdat)
```

```
##                                          SourceFile                date
## 1 D:/Survey_xxx/DeploymentImages/RP01/IMG_0001.JPG 2017:10:02 19:06:43
## 2 D:/Survey_xxx/DeploymentImages/RP01/IMG_0002.JPG 2017:10:02 19:06:44
## 3 D:/Survey_xxx/DeploymentImages/RP01/IMG_0003.JPG 2017:10:02 19:06:45
## 4 D:/Survey_xxx/DeploymentImages/RP01/IMG_0004.JPG 2017:10:02 19:06:45
## 5 D:/Survey_xxx/DeploymentImages/RP01/IMG_0005.JPG 2017:10:02 19:06:46
## 6 D:/Survey_xxx/DeploymentImages/RP01/IMG_0006.JPG 2017:10:02 19:06:47
##                                Keywords     time Calibration contact
## 1 contact1, placeID: RP01, species1: Fox 5.003495          NA       1
## 2           placeID: RP01, species1: Fox 5.003568          NA      NA
## 3           placeID: RP01, species1: Fox 5.003641          NA      NA
## 4           placeID: RP01, species1: Fox 5.003641          NA      NA
## 5           placeID: RP01, species1: Fox 5.003714          NA      NA
## 6           placeID: RP01, species1: Fox 5.003786          NA      NA
##   good.photo station species
## 1         NA    RP01     Fox
## 2         NA    RP01     Fox
## 3         NA    RP01     Fox
## 4         NA    RP01     Fox
## 5         NA    RP01     Fox
## 6         NA    RP01     Fox
```

Next, take a subset that includes just the first contact records (these are the ones that we need to tally to generate trap rates), and check that all records sit within the deployment times given in the deployment table. First subsetting contacts:

```
contactdat <- subset(tagdat, contact==1)
```

Then a visual check using `plot.deployments` (Fig. 1). Obviously problematic data will show up in this plot as red points that do not sit over a deployment period for their site. All looks OK in this case.

```
plot.deployments(contactdat, depdat)
```

You can also split the dataframe into records that do or do not make sense using `check.dates`. This produces a list of two dataframes: `good.data` and `bad.data`, respectively holding the records that do and do not sit within their deployments. In this case there were no problematic records.

```
chk <- check.dates(contactdat, depdat)
chk$bad.data
```

```
## [1] SourceFile  date        Keywords    time        Calibration contact
## [7] good.photo  station     species
## <0 rows> (or 0-length row.names)
```

Finally, create a dataframe of trap rate data using `event.count`. This requires as input your `contactdat` dataframe and your `depdat` dataframe, and produces a new dataframe with a row per station, and data columns for station identifier, effort in days, and record counts for each species in the database:

```
trdat <- event.count(contactdat, depdat)
head(trdat)
```

```
##   station effort.days Fox Hedgehog
## 1    RP01   10.878507   3        0
## 2    RP02    5.379664   2        0
## 3    RP03   10.601586   3        2
## 4    RP04   10.670775   4        0
## 5    RP05    6.273380   6        0
## 6    RP06   13.831262   4        6
```

REM analysis

The analysis has four steps:

1. activity level estimation
2. speed estimation
3. detection zone estimation
4. density estimation

If you're working with a multi-species dataset, first create an indicator with which you can select relevant records from the various dataframes, for example:

```
sp <- "Fox"
```

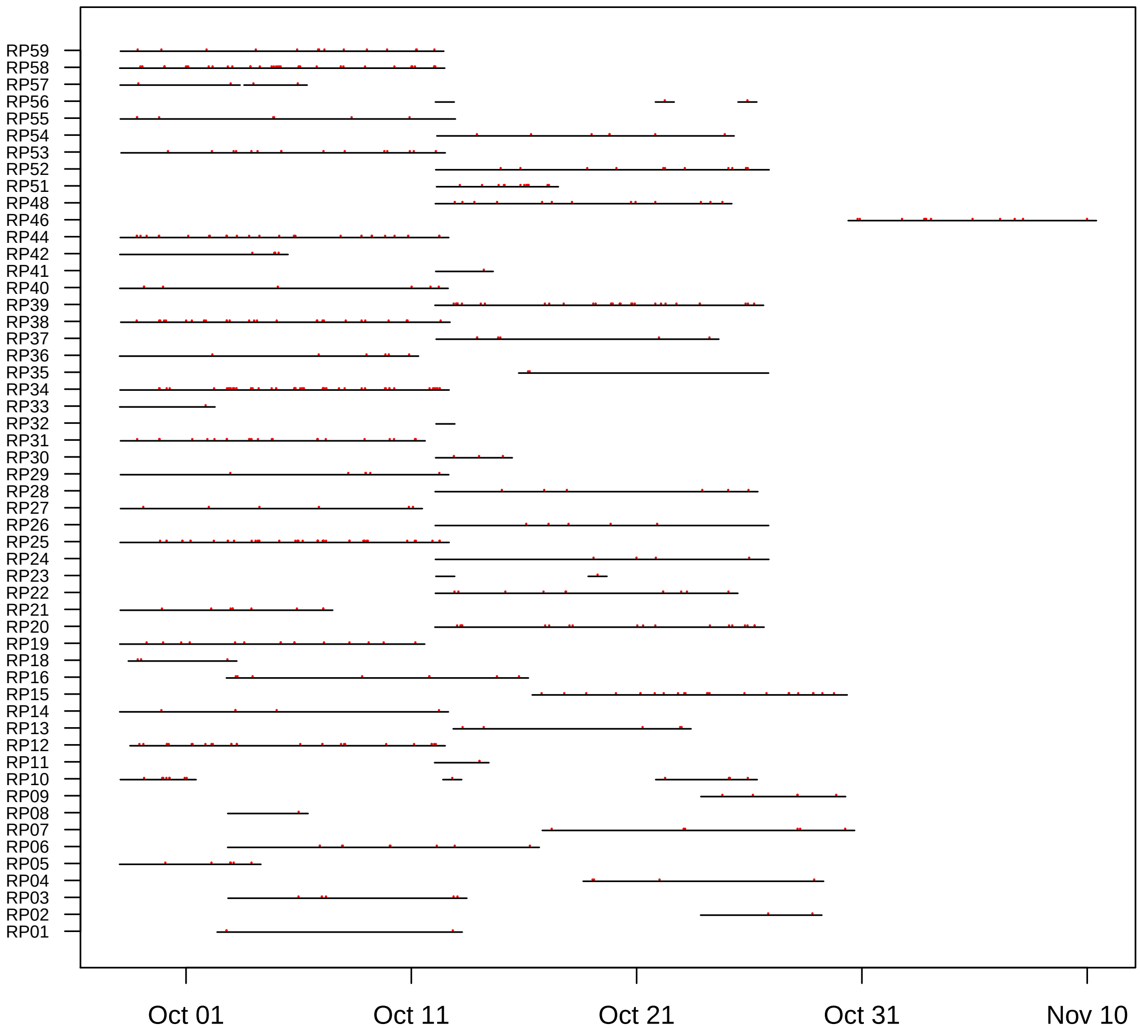

Figure 1: A plot of camera deployment and animal record times for each station. Black lines indicate deployments while red points indicate animal records.

Activity level estimation

You will need function `fitact` from R package `activity` (available at cran.r-project.org), applying this to radian time of day data for contact records of your species of interest. The function fits a circular kernel model to the data, and provides an estimate of activity level (see Rowcliffe et al. 2014 MEE 5: 1170-1179). By default, the model is fitted without bootstrapping and no standard errors are provided for the activity level estimate. To obtain a standard error, set the `sample` argument to either "data" or "model". This can be slow, so in the example below, the number of boostrap replicates is reduced using the `reps` argument. If you need to consider only records from part of the diel cycle, provide a two-element vector to the `bounds` argument, specifying the beginning and end of the the period you wish to consider. The result is an object of class `actmod`, which can be plotted to examine model fit (Fig. 2), and has slot `act` containing the activity level estimate.

```
library(activity)
actmod <- fitact(subset(contactdat, species==sp)$time,
                 bounds=c(18,8)*pi/12,
                 sample="data",
                 reps=100)
actmod@act
```

```
##        act         se   lcl.2.5%  ucl.97.5%
## 0.69220952 0.04764451 0.60301541 0.78528710
```

```
plot(actmod, centre="night", dline=list(col="grey"))
```

Speed estimation

You will need function `hmean` from package `sbd` (available on github.com/MarcusRowcliffe/sbd) in order to calculate the harmonic mean of speed observations and it's standard error. This approach is required because faster speeds are more likely to be observed than slower speeds, and the harmonic mean appropriately down-weights the influence of faster speeds on the average (see Rowcliffe et al. RSEC 2016 2: 84-94).

```
source("sbd.r")
```

Before calculating the mean, it is desirable to inspect the distribution of observations. Typically there may be a few extremely slow speed observations (less than about 0.001 m s^{-1}) that might reflect errors in sequence definition, or sequences in which the animal essentially didn't move, and probably shouldn't therefore be included in the data. There may also be unrealistically fast speeds because of errors in the calculation process. Inspecting the distribution will help to show the extent of these problems (Fig. 3).

```
speeds <- subset(seqdat, species==sp )$speed
hist(log10(speeds), main="", xlab="Speed (log10[m/s])")
```

Re-inspecting the image sequences of extreme observations may help to identify errors so that they can be either corrected or excluded. Here, to simplify the demonstration, extreme observations are simply excluded before estimating average speed. The result is a named vector of estimated harmonic mean speed and it's standard error.

```
speeds <- subset(seqdat, species==sp & speed>0.001 & speed<10)$speed
(spdest <- hmean(speeds))
```

```
##       mean         se
## 0.06834510 0.01673901
```

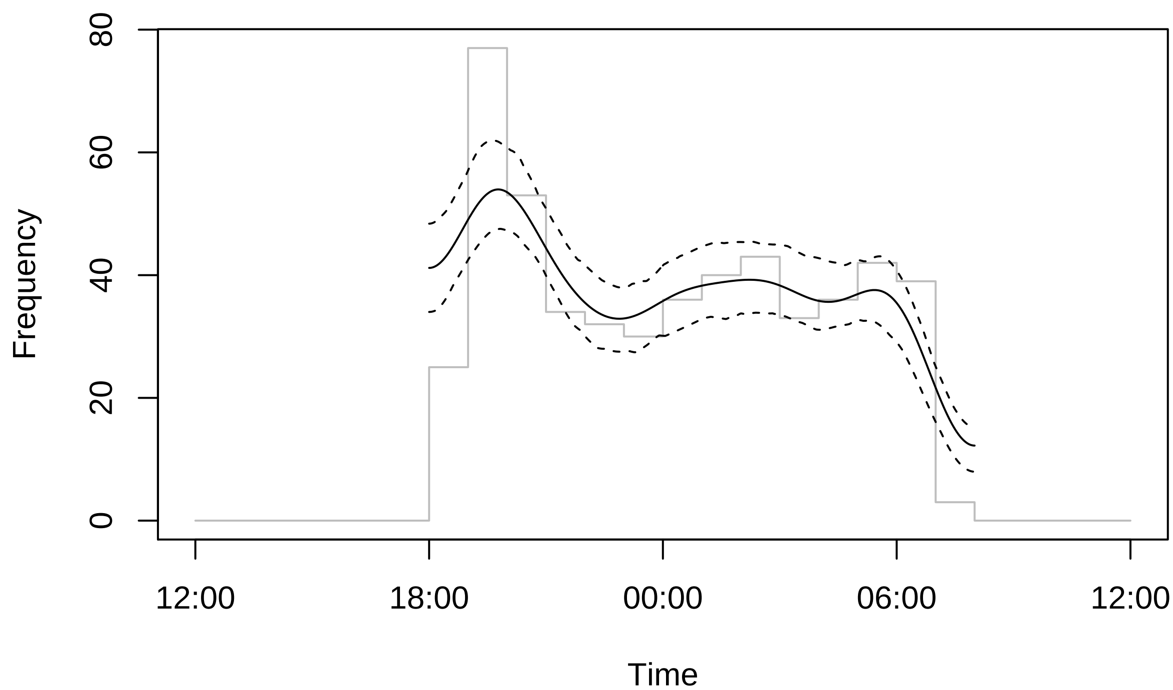

Figure 2: An activity pattern from camera trap data, showing the data distribution and fitted circular kernel model, truncated to consider only nocturnal images

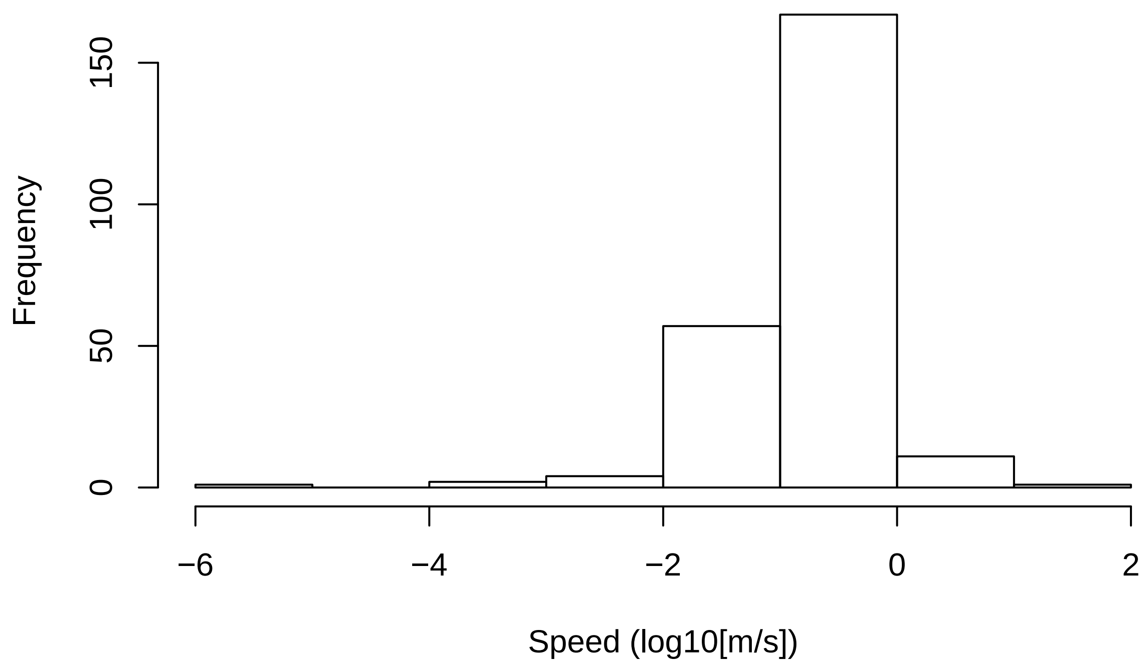

Figure 3: Example distribution of speed observations, showing a small number of extreme observations, particularly at the lower end.

Detection zone estimation

You will need function `fitdf` from package `distanceDF` (available on github.com/MarcusRowcliffe/distanceDF) in order to fit detection functions to radial and angular distances at first contact, and so estimate effective detection zone dimensions (see Rowcliffe et al. MEE 2011 2: 464-476).

```
source("distancedf.r")
```

The first argunment to `fitdf` is a formula indicating the column containing distance data on the left, and covariate columns on the right (or 1 if there are none). The second argument is used to name a dataframe containing the required data. Within the `fitdf` function, models are fitted using the the `ds` function from the package `Distance`, and additional arguments to `fitdf` are passed to `ds`. Full option details can be found in the `Distance` documentation. For radial distance, a point transect is required. Typically a hazard rate model is appropriate (`key=hr`), and flexibility adjustment terms are un-necessary (order=0) (Rowcliffe et al. 2011), so for simplicity here we fit a single model with these characteristics. The result is a named list with elements `ddf` (a detection function model object as described in `Distance` documentation), and `edd` (a list holding estimates of the effective detection distance and its precision). Model fit can be inspected by plotting the ddf component of the result (Fig. 4). If extreme values are detected, as in standard distance sampling they can be removed by using the `truncation` argument to define the maximum distance to include in the analysis.

```
dzdat <- subset(posdat, frame_count==1 & species==sp)
radmod <- fitdf(radius~1, dzdat, transect="point", key="hr", order=0, truncation=10)
radmod$edd
```

```
##    estimate        se
## p2 5.777056 0.2165647
```

```
plot(radmod$ddf, pdf=TRUE)
```

Angle estimation proceeds in more or less the same way, but with a line-type transect and half-normal detection function (both default options). Angle observations left of the centre of the camera's field of view are registered as negative values in the example data, which can't be modelled. It is therefore necessary to convert angles to absolute values before analysis.

```
dzdat$angle <- abs(dzdat$angle)
angmod <- fitdf(angle~1, dzdat, order=0)
angmod$edd
```

```
##    estimate         se
## 1 0.2698825 0.01539376
```

```
plot(angmod$ddf)
```

Density estimation

You will need function `bootTRD` from package `camtools` (see above) in order to estimate density (see Rowcliffe et al. J App Ecol 2008 45: 1228-1236). This function takes arguments for the number of records per station, the amount of effort (camera time per station), and additional parameters needed for the estimation, which were estimated in the steps above. Parameters and their standard errors must be provided as named lists,

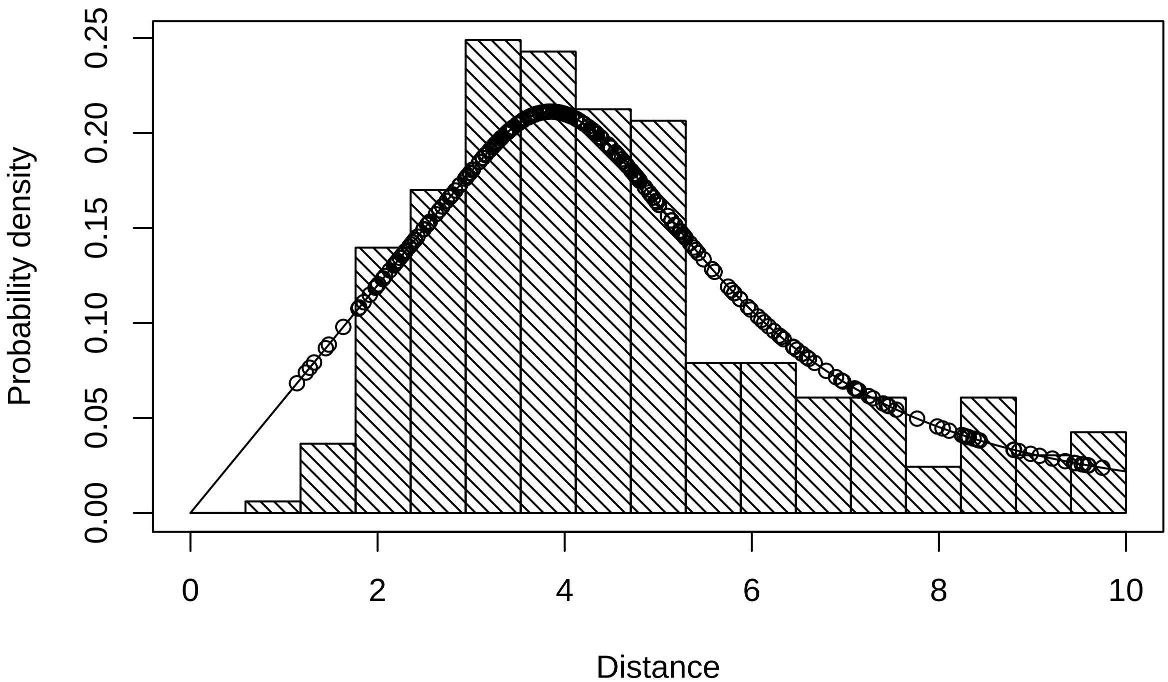

Figure 4: Example distribution of radial distances and fitted detection function.

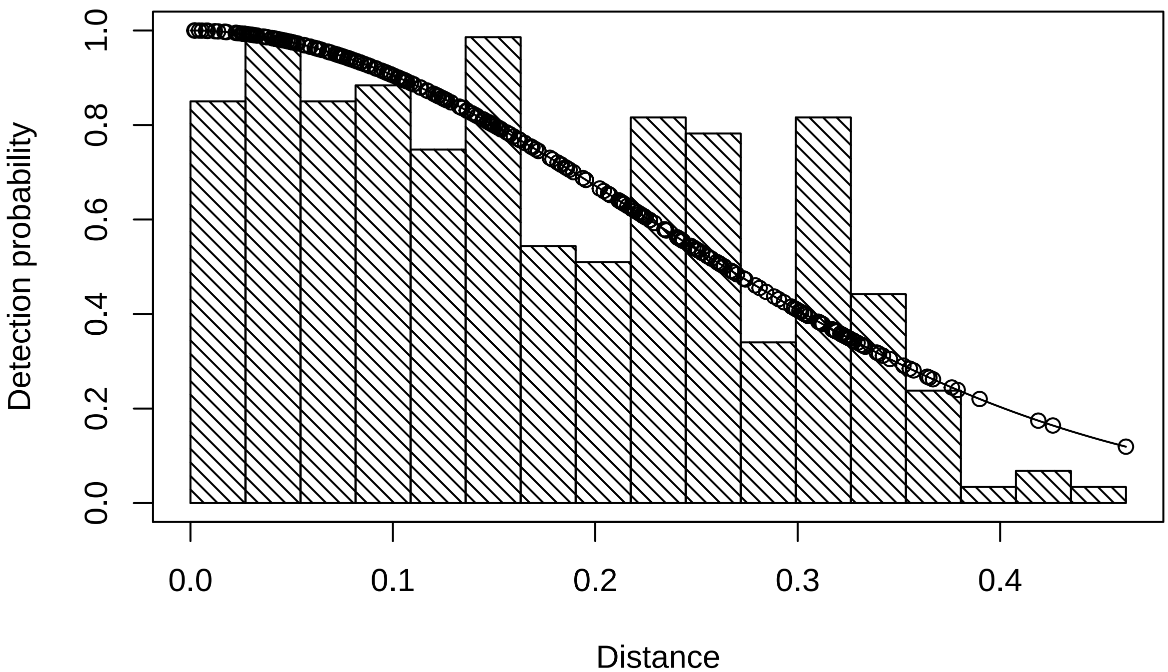

Figure 5: Example distribution of angular distances and fitted detection function.

using names v (speed while active), p (activity level), r (detection zone radius) and *theta* (detection zone angle). It is also crucial to ensure that units are harmonised across parameters. In this case, camera time is measured in days, but the speed time unit is seconds, while the activity level estimate was truncated to only 14 hours of the day. To convert speed to distance covered per day we therfore multiply by 14*60^2. Distance units for both speed and radius are both m, so could be left un-transformed, but we would like our density estimate expressed per km^2, so divide both values by 1000. Finally, the angle detectection function was one-sided, with observations from both sides of the field of view centre being analysed as if they came from one side. We therefore need to mutiply angle by 2.

```
param <- list(v = spdest["mean"] * 14*60^2 / 1000,
              p = actmod@act["act"],
              r = radmod$edd$estimate / 1000,
              theta = angmod$edd$estimate * 2)
paramse <- list(v = spdest["se"] * 14*60^2 / 1000,
                p = actmod@act["se"],
                r = radmod$edd$se / 1000,
                theta = angmod$edd$se * 2)
bootTRD(trdat[, sp], trdat$effort.days, param, paramse)
```

```
##       Density       SE
## [1,] 82.53251 24.41895
```