

Trip Estimation: a demonstration of the R package

Michael D. Sumner

April 30, 2007

1 Introduction

The `tripEstimation` package provides specialist estimation algorithms and supporting code for light level geo-location. Here we provide a fully worked example, from a raw text table to mapped outputs.

1.1 Raw data preparation

First, we extract the data from the tag files as CSV (see sample), and import the entire set :

```
‘ ‘C699_02.csv’:  
  "Date", "Time", "Depth", "Light Level", "Temperature"  
[1] 17/01/1999,12:00:16,8,189,22.1  
[2] 17/01/1999,12:00:46,8,172,  
[3] 17/01/1999,12:01:16,6,177,  
...  
  
label <- "c699_02"  
file <- paste(label, ".csv", sep = "")  
d <- read.csv(file, header = FALSE)
```

Now provide appropriate names to the data columns, convert the date-time strings to `POSIXct` type and save the required dataset to an R binary file. This process will be different depending on your dataset and tag type. It is important to check that the date-time data are correctly interpreted (including time zone), and that strange or missing values are handled appropriately.

```
names(d) <- c("datetime", "depth", "speed", "temp", "light")  
d$gmt <- as.POSIXct(strptime(d$datetime, "%Y-%m-%d %H:%M:%S"), "GMT")  
## create a new data frame with the data we need  
d <- data.frame(id = 1:nrow(d), gmt = d$gmt, depth = d$depth,  
  temp = d$temp, light = d$light)  
save(d, file = "rawData.Rdata")
```

1.2 Data preparation

Before estimation can proceed, we must provide some simple housekeeping. Load the required package, the previously saved raw data and check the light data. (The file paths simply reflect my directory setup.

```
library(tripEstimation)
fl <- "rawdata\\rawData.Rdata"
load(fl)
```

```
n <- nrow(d)
```

We define a maximum spatial range for this animal's trip, and a start and end location (optional). Also, we determine interactively the record numbers that delimit some light data from a known location.

```
lon.min <- 125
lon.max <- 165
lat.min <- -70
lat.max <- -48
```

```
start <- c( 158.9341, -54.5016)
end <- start
```

```
dstart <- 20000
dend <- 20000
```

Using the light data from a known location, the `mkCalibration` function allows us to choose some twilight segments to use for light calibration. `gam` from the `mgcv` package is used to generate the function from the individual light segments. The calibration is shown in Fig. 1.

```
library(mgcv)
```

```
d.calib <- d[c(1:dstart, dend:n), ]
```

```
calib <- mkCalibration(d.calib, known = start, elim = c(-10, 8))
```

Finally we run through the entire dataset, plotting the light data and interactively choosing twilight segments for use in the estimation. This step could be automated, but it doesn't take that long and provides a visual overview of the entire data. The `pick` function provides the running plot from which twilight segments are interactively chosen from `nsee` records. `picksegs` then uses those paired record numbers to generate segment numbers. The session is then saved with the calibration function, the bounding box values and the start and end positions.

```
twindex <- pick(d$id, d$light, nsee = 80000)
segs <- picksegs(twindex, nrow(d))
```

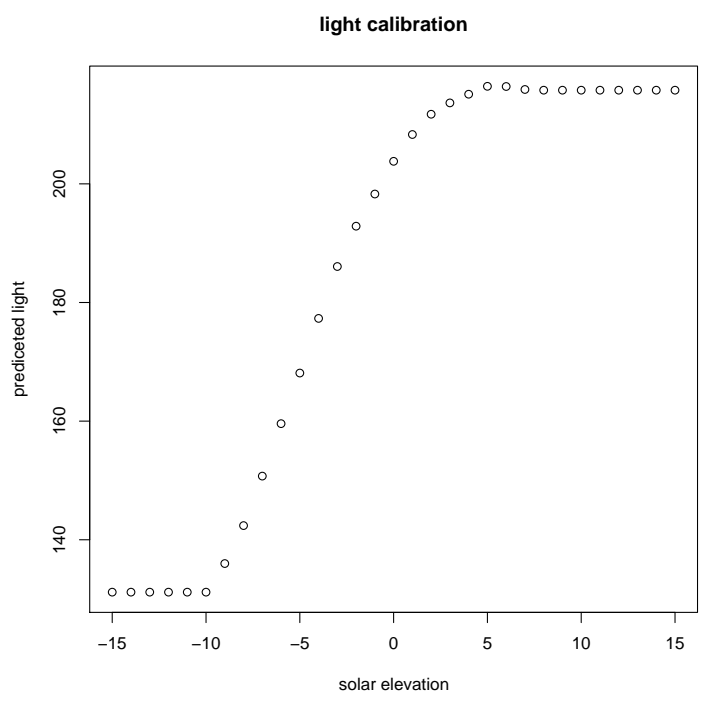


Figure 1: Calibration function: solar elevation vs. light

```
d$segment <- segs

save(calib, lon.min, lon.max, lat.min, lat.max, start, file =
"readyRun.Rdata")
```

1.3 Generate temperature and depth masks (or likelihood)

(For now we leave out this step to simplify this document).

2 Final preparation

Now we require some further packages for spatial data handling and saving the estimation results. We load the data prepared earlier, subset the based on surface depths and segment periods, and save the time values ready for use when summarizing results.

```
library(tripEstimation)
library(satin)
library(rgdal)
library(chain)

fl <- "readyRun.Rdata"
load(fl)

kp <- is.finite(d$segment) & d$depth < 20

d$segment <- unclass(d$segment)
d1 <- d[kp, ]
rm(d)

## save the times of each segment
tm <- tapply(d1$gmt, d1$segment, mean)
save(tm, file = "tm_TimesOfSegments.Rdata")
```

We now specify whether the start and/or end location are fixed and prepare the model. Proposal objects are created for handling the parameter sampling and tuning, for now we choose sampling variances that are set for every location estimate.

```
fixed.release <- TRUE
fixed.recapture <- TRUE
m <- nlevels(factor(d1$segment)) + fixed.release + fixed.recapture

proposal.x <- norm.proposal(m, 3, sqrt(c(0.05, 0.05, 0.05)))
proposal.z <- norm.proposal(m-1, 2, sqrt(c(0.1, 0.1)))
```

2.1 Land mask

We can specify a land mask to disallow any estimates on land. This can be augmented by temperature or depth as required. When estimation begins we may update the bounding box in this step in order to conserve memory. Again, the file path illustrates my directory structure. The `lookup` function handles the reference to the topographic mask.

```
topo <- readData("C:/spatdata/topodata/Etopo2.tif",
                xlim = c(lon.min, lon.max),
                ylim = c(lat.min, lat.max),
                mkEtopo())

topo <- as.image.SpatialGridDataFrame(topo)
topo$z <- topo$z < -1

## set up mask function
lookup <- mkLookup(topo, FALSE)
```

2.2 Initialization

Now we specify an appropriate behavioural model to constrain the estimation, and provide all further information for the estimation model. Here we use lognormal speed, and a fairly wide Ekstrom range.

```
speed.mu <- 2
speed.sigma <- 1

## create the model
d.model <- solar.model(d1$segment, d1$gmt, d1$light,
                      proposal.x = proposal.x$proposal,
                      proposal.z = proposal.z$proposal,
                      fix.release = fixed.release,
                      fix.recapture = fixed.recapture,
                      mask.x = lookup,
                      mask.z = lookup,
                      calibration=calib,
                      behav = "speed",
                      behav.dist = "log",
                      light.sigma=7,
                      k.sigma=10,
                      behav.mean = speed.mu,
                      behav.sd = speed.sigma,
                      ekstrom = c(-8, 5, 128))
```

In order to run the estimation, we must find some starting points that satisfy the land (and other masks). Here we evaluate the light likelihood across a coarse grid and find the maximum for each segment. Then an initialization version of the metropolis sampler is used to find valid starting locations.

```

xx <- seq(lon.min, lon.max, by = .3)
yy <- seq(lat.min, lat.max, by = .3)
plg <- position.logp(d.model, xx, yy, xrest = -10,
  start = start, end = end)
X <- plg$X

ch <- metropolis0(d.model, iters=10, thin=1, start.x = X,
  start.z = (X[-m,1:2]+X[-1,1:2])/2)

```

That initialization run may need to be repeated until valid points are found.

3 Running the estimation

Finally, we can run the estimation and cache the results to disk. Once the proposals are tuned the chain may be saved, variations on this may be required depending on the data.

```

library(chain)
ch <- metropolis(d.model, iters=500, thin=10,
  start.x=ch$last.x, start.z = ch$last.z)

ch <- metropolis(d.model, iters=1000, thin=10,
  start.x=ch$last.x, start.z = ch$last.z)
proposal.x$tune(ch$x, scale = 0.3)
proposal.z$tune(ch$z, scale = 0.3)

xfile <- "X.bin"
zfile <- "Z.bin"
for (i in 1:5) {
  ch <- metropolis(d.model, iters=1000, thin=10,
    start.x=ch$last.x, start.z = ch$last.z)

  proposal.x$tune(ch$x, scale = 0.3)
  proposal.z$tune(ch$z, scale = 0.3)
  chain.write(ch$x, xfile, append = i != 0)
  chain.write(ch$z, zfile, append = i != 0)
}

```

4 Summarizing the chain

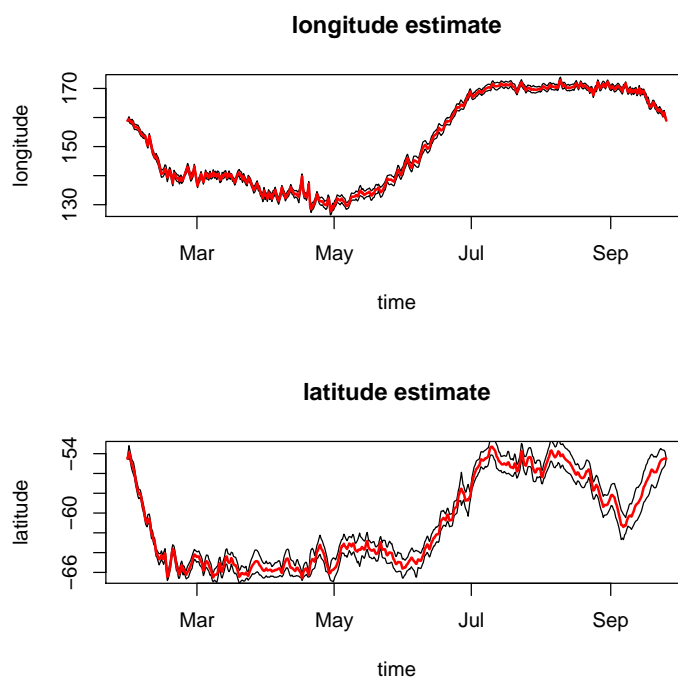


Figure 2: Longitude and latitude: mean estimates with full path range

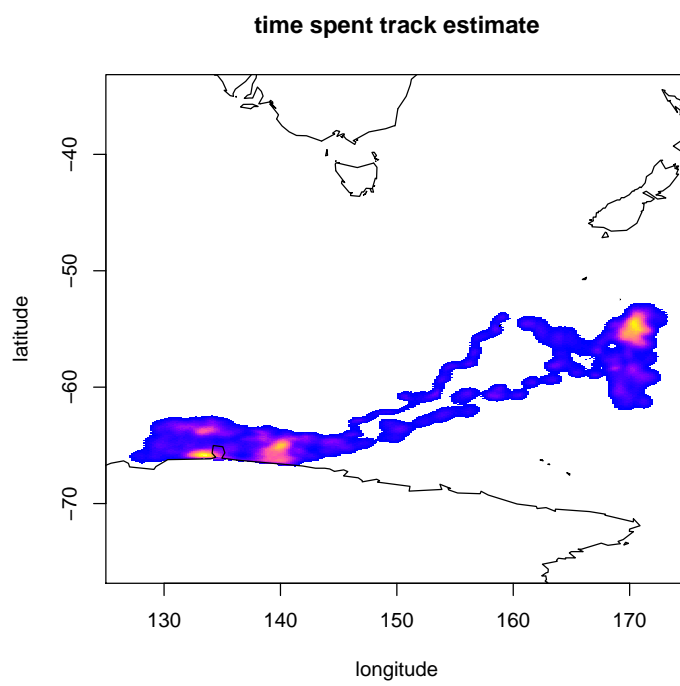


Figure 3: Time spent map from full path