Chapter 4

Home Range Estimation

Contents

Figures

4.1 Kernel Density Estimation (KDE) with reference bandwidth selection (h_{ref})

In KDE, a kernel distribution (i.e. a three-dimensional hill or kernel) is placed on each telemetry location. The height of the hill is determined by the bandwidth of the distribution, and many distributions and methods are available (e.g. fixed versus adaptive, univariate versus bivariate bandwidth). We will focus here on "fixed kernel" but will alter the bandwidth

selection. Datasets for avian and mammalian species can include as many as 10,000 locations and only the reference or default bandwidth (h_{ref}) was able to produce KDE in both Home Range Tools and adehabitat or adehabitatHR (Calenge 2007, 2011). Estimation with (h_{ref}) typically is not reliable for use on multimodal datasets because it results in over-smoothing of home ranges and multimodal distribution of locations is typical for most species (Worton 1995, Seaman et al. 1999).

1. Exercise 4.1 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(adehabitat)
NOTE: adehabitatHR package requires Spatial Points instead of data frame
so code below will not work if adehabitatHR is also loaded
library(sp)
library(rgdal)
library(raster)
```

4. Now open the script "HrefScript.R" and run code directly from the script

```
panther<-read.csv("pantherjitter.csv", header=T)
str(panther)
```

5. Now we can run fixed kernel home range with h_{ref} bandwidth selection

```
#Note below the code uses the original adehabitat package to run home range
loc <- panther[, c("x", "y")]
## Estimation of UD for each animal separately
id <- panther[, "id"]
udbis <- kernelUD(loc, id, h = "href")
ud <- kernelUD(loc, id = id, h = "href", grid = 40, same4all = FALSE,
   hlim = c(0.1, 1.5), kern = c("bivnorm"), extent = 0.5)
image(ud) ## Note that the contours are under the locations
```

6. Calculation of the 95 percent home range

```
ver <- getverticeshr(ud, 95)
plot(ver, add=TRUE)
```

7. We estimated size of home range using h_{ref} now we need to output the size of these home ranges that were estimated

```
#Look at the estimates of home range by contour
cuicui1 <- kernel.area(loc, id)
plot(cuicui1)
cuicui1

#Results for 2 animals by probability contour (i.e., 20-95 percent)
       FP121     FP128
20  224.7252  346.8085
25  304.0400  468.6602
30  396.5739  609.2582
35  508.9366  759.2295
40  634.5183  946.6936
45  786.5383 1162.2773
50  971.6062 1424.7270
```

```
55 1189.7218 1743.4159
60 1440.8854 2165.2101
65 1731.7062 2652.6167
70 2062.1845 3205.6357
75 2445.5394 3824.2671
80 2928.0377 4546.0038
85 3529.5082 5464.5778
90 4316.0465 6748.7067
95 5545.4257 8820.1848
```

8. We can export these estimates with the previous code
```
write.table(cuicui1,"output.csv", row.names=TRUE,
sep=" ", col.names=TRUE, quote=TRUE, na = "NA")
```

9. We can also output the respective shapefiles with code below:

```
#######
# OVERRIDE the default kver2spol function so that we
#can include the projection info
#######
kver2spol <- function(kv,projstr){
    x <- kv
    if (!inherits(x, "kver"))
        stop("x should be of class \"kver\"")
    if (!require(sp))
        stop("sp package needed")
    lipols <- lapply(1:length(x), function(i) {
        y <- x[[i]]
        class(y) <- c("data.frame", "list")
        res <- split(y[, 2:3], y[, 1])
        lipol <- lapply(res, function(z) {
            if (sum(abs(z[1, ] - z[nrow(z), ])) > 1e-16)
                z <- rbind(z, z[1, ])
            Polygon(as.matrix(z))
        })
        pols <- Polygons(lipol, ID = names(x)[i])
        return(pols)
    })
    return(SpatialPolygons(lipols, proj4string=CRS(as.character(projstr))))}

##############
# Function to export specific levels of isopleths of a "kv" object
##############

#Code creates contours only for animal 1 at each level so need to repeat for
#each animal if needed
kv<-list()
class(kv) <- "kver"

kvtmp <- getverticeshr(udbis, lev = 99)
kv$KHR99<- kvtmp[[1]]
kvtmp <- getverticeshr(udbis, lev = 95)
kv$KHR95<- kvtmp[[1]]
```

```
kvtmp <- getverticeshr(udbis, lev = 90)
kv$KHR90<- kvtmp[[1]]
kvtmp <- getverticeshr(udbis, lev = 75)
kv$KHR75<- kvtmp[[1]]
kvtmp <- getverticeshr(udbis, lev = 50)
kv$KHR50<- kvtmp[[1]]
kvtmp <- getverticeshr(udbis, lev = 25)
kv$KHR25<- kvtmp[[1]]

spolTmp <- kver2spol(kv,"+proj=utm +zone=17N +ellps=WGS84")
dfTmp <- data.frame(Isopleth=c("99","95","90","75","50","25"),
row.names=c("KHR99","KHR95","KHR90","KHR75","KHR50","KHR25"))
spdfTmp <- SpatialPolygonsDataFrame(spolTmp, dfTmp, match.ID = TRUE)
writeOGR(spdfTmp,"HREF","FP048HREF", "ESRI Shapefile")
```

10. We can also use package adehabitatHR to determine home range but requires different code

```
panther <- subset(panther, panther$CatID == "143")
panther$CatID <- factor(panther$CatID)
loc <- data.frame("x"=panther$x,"y"=panther$y)
proj4string <- CRS("+proj=utm +zone=17N +ellps=WGS84")
cats <- SpatialPointsDataFrame(loc,panther, proj4string = proj4string)
udbis <- kernelUD(cats[,3], h = "href")
image(udbis)

ver <- getverticeshr(udbis, standardize = FALSE)
ver50 <- getverticeshr(udbis, percent=50)
ver80 <- getverticeshr(udbis, percent=80)
ver90 <- getverticeshr(udbis, percent=90)
ver95 <- getverticeshr(udbis, percent=95)
ver99 <- getverticeshr(udbis, percent=99)
ver
plot(ver99, col="grey",axes=T);plot(ver95, add=T);plot(ver90, add=T);
   plot(ver80, add=T);plot(ver50,  add=T)
points(cats)
```

4.2 KDE with least-squares cross validation bandwidth selection (h_{lscv})

Both the least squares cross-validation (h_{lscv}) and bias crossed validation (h_{bcv}) have been suggested instead of href in attempts to prevent over-smoothing of KDE (Rodgers and Kie 2010). However, (h_{lscv}) and (h_{bcv}) have been minimally evaluated on GPS datasets because previous literature only evaluated datasets collected on VHF sampling protocols or simulated data that included at most 1,000 locations. Least-squares cross validation, suggested as the most reliable bandwidth for KDE was considered better than plug-in bandwidth selection ($h_{plug\text{-}in}$; for description see section 3.3) at identifying distributions with tight clumps but risk of failure increases with h_{lscv} when a distribution has a "very tight cluster of points" (Gitzen et al. 2006, Pellerin et al. 2008, Walter et al. 2011).

1. Exercise 4.2 - Download and extract zip folder into your preferred location

2. Set working directory to the extracted folder in R under File - Change dir...

3. First we need to load the packages needed for the exercise

```
library(adehabitatHR)
library(sp)
```

4. Now open the script "HlscvScript.R" and run code directly from the script

```
panther <-read.csv("pantherjitter.csv", header=T)
str(panther)

loc <- data.frame("x"=panther$X,"y"=panther$Y)
proj4string <- CRS("+proj=utm +zone=17N +ellps=WGS84")
pantherspdf <- SpatialPointsDataFrame(loc,panther, proj4string = proj4string)
plot(pantherspdf, col=pantherspdf$CatID)
```

5. Now we can run fixed kernel home range with h_{lscv} bandwidth selection

```
## Example of estimation using LSCV
udbis2 <- kernelUD(pantherspdf[,4], h = "LSCV", hlim = c(10,50),extent=1)
image(udbis2)

#Now change h to href for comparison to LSCV later

## Compare the estimation with ad hoc and LSCV method
## for the smoothing parameter
cuicui2 <- kernel.area(udbis2)
cuicui2
```

6. After running some very important appears for both animals that reads:

```
Warning messages:
1: In .kernelUDs(SpatialPoints(x, proj4string = CRS(as.character(pfs1))),  :
  The algorithm did not converge
within the specified range of hlim: try to increase it
2: In .kernelUDs(SpatialPoints(x, proj4string = CRS(as.character(pfs1))),  :
  The algorithm did not converge
within the specified range of hlim: try to increase it
```

7. Note that regardless of change hlim or extent, LSCV will not converge for these animals so let's try a trick here. I believe LSCV is a poor estimator with GPS locations being too numerous and very close together compared to traditional VHF datasets which LSCV were originally evaluated. So lets jitter locations 50 meters from their original location and try again.

```
panther$jitterX <- jitter(panther$x, factor=50)
panther$jitterY <- jitter(panther$y, factor=50)
locjitter <- data.frame("x"=panther$jitterX,"y"=panther$jitterY)
proj4string <- CRS("+proj=utm +zone=17N +ellps=WGS84")
jitterspdf <- SpatialPointsDataFrame(locjitter,panther, proj4string = proj4string)
plot(jitterspdf, col=pantherspdf$id)
points(pantherspdf, col="blue")
udbis3 <- kernelUD(jitterspdf[,4], h = "LSCV")#, hlim = c(1, 5),extent=1)
image(udbis3)
```

```
#Now rerun with jitter factor = 100 then 500 instead of 50 and see what happens?

cuicui3 <- kernel.area(udbis3) ## LSCV
cuicui3

#Now rerun with jitter factor = 500 instead of 100 and see what happens?

#Combine all estimates for comparison
iso <- cbind(cuicui2,cuicui3)
colnames(iso) <- c("FP121_lscv","FP143_lscv","FP121_Jitter","FP143_Jitter")
iso
```

4.3 KDE with plug-in bandwidth selection ($h_{plug\text{-}in}$)

Using $h_{plug\text{-}in}$ in ks, we were able to calculate KDEs for the sample GPS datasets on 3 avian species and 2 mammalian species where first generation methods (h_{lscv}) failed or generated a considerably over-smoothed KDE (h_{ref}). While home range polygons generated with $h_{plug\text{-}in}$ appeared fragmented, they may be appropriate when studying a species in highly fragmented landscapes such as urban areas. Based on our results and previous research, conclusions presented in Loader (1999) should be re-evaluated for analyses of large GPS dataset because sample size and clumping of locations has consistently failed using h_{lscv}, while estimates using $h_{plug\text{-}in}$ converged for large multimodal datasets and resulted in reasonable estimates (Girard et al. 2002, Amstrup et al. 2004, Millspaugh et al. 2006).

1. Exercise 4.3 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(ks)
   library(rgdal)
   library(maptools)
   library(gpclib)
   library(PBSmapping)
   ```

4. Now open the script "PelicanPlug.R" and run code directly from the script
5. We will use an abbreviated dataset to save processing time
   and the code will also output shapefiles of home ranges

   ```
   # get input file
   indata <- read.csv("White10pelicans.csv")
   innames <- unique(indata$ID)
   innames <- innames[1:5]
   outnames <- innames
   ```

6. We then want to set up a table to output estimates of size of home ranges

   ```
   # set up output table
   output <- as.data.frame(matrix(0,nrow=length(innames),ncol=9))
   colnames(output) <- c("ID","noFixes","h11","h12","h21","h22",
                         "iso50areaKm","iso95areaKm","iso99areaKm")
   ```

   NOTE: the h followed by a number are outputs from the $h_{plug\text{-}in}$ for the bandwidth matrix estimated for each animal. They are in there for reference

but don't really need them.

7. We also need to tell R which contours to output

```
# set up levels for home range
levels <- c(50,95,99)
```

8. Set up a directory to place the resulting shapefiles

```
dirout <- "output"
# begin loop to calculate home ranges
for (i in 1:length(innames)){
 data <- indata[which(indata$ID==innames[i]),]
 if(dim(data)[1] != 0){
 # export the point data into a shp file
 data.xy = data[c("Longitude", "Latitude")]
 coordinates (data.xy) <- ~Longitude+Latitude
 sppt <- SpatialPointsDataFrame(coordinates(data.xy),data)
 proj4string(sppt) <- CRS("+proj=longlat +ellps=WGS84")
 writePointsShape(sppt,fn=paste(dir,outnames[i],sep="/"),
 factor2char=TRUE)
 # start populating output table by column heading "pelicanID"
 output$ID[i] <- data$ID[1]
 output$noFixes[i] <- dim(data)[1]
 locs <- cbind(data$Longitude,data$Latitude)
 try(HpiOut <- Hpi(locs,pilot="samse",binned=TRUE))
 if(is.null(HpiOut)=="FALSE"){
  output$h11[i] <- HpiOut[1,1]
  output$h12[i] <- HpiOut[1,2]
  output$h21[i] <- HpiOut[2,1]
  output$h22[i] <- HpiOut[2,2]
 fhatOut <- kde(x=locs,H=HpiOut)
  }
 if(is.null(fhatOut)=="FALSE"){
 for (j in 1:length(levels)){
  fhat.contlev <- contourLevels(fhatOut, cont=c(levels[j]))
  fhat.contlines <- contourLines(x=fhatOut$eval.points[[1]],
  y=fhatOut$eval.points[[2]], z=fhatOut$estimate, level=fhat.contlev)
 # convert contour lines into spatial objects to export as
   polygon shp file
  sldf <- ContourLines2SLDF(fhat.contlines)
  proj4string(sldf) <- CRS("+proj=longlat +ellps=WGS84")
  ps <- SpatialLines2PolySet(sldf)
  attr(ps,"projection") <- "LL"
  sp <- PolySet2SpatialPolygons(ps)
  dataframe <- as.data.frame(matrix(as.character(1,nrow=1,ncol=1)))
  spdf <- SpatialPolygonsDataFrame(sp,dataframe,match.ID=TRUE)
 # get area and export shp files
  if (j == 1){
  pls <- slot(spdf, "polygons")[[1]]
  gpclibPermit()
  xx <- checkPolygonsHoles(pls)
  a <- sapply(slot(xx, "Polygons"), slot, "area")
  h <- sapply(slot(xx, "Polygons"), slot, "hole")
```

```
  output$iso50areaKm[i] <- sum(ifelse(h, -a, a))/1000000
  writeOGR(spdf,dirout,paste(outnames[i],"KUD50",sep=""),
  "ESRI Shapefile")}
  if (j == 2){
  pls <- slot(spdf, "polygons")[[1]]
  gpclibPermit()
  xx <- checkPolygonsHoles(pls)
  a <- sapply(slot(xx, "Polygons"), slot, "area")
  h <- sapply(slot(xx, "Polygons"), slot, "hole")
  output$iso95areaKm[i] <- sum(ifelse(h, -a, a))/1000000
  writeOGR(spdf,dirout,paste(outnames[i],"KUD95",sep=""),"ESRI Shapefile")}
  if (j == 3){
  pls <- slot(spdf, "polygons")[[1]]
  gpclibPermit()
  xx <- checkPolygonsHoles(pls)
  a <- sapply(slot(xx, "Polygons"), slot, "area")
  h <- sapply(slot(xx, "Polygons"), slot, "hole")
  output$iso99areaKm[i] <- sum(ifelse(h, -a, a))/1000000
  writeOGR(spdf,dirout,paste(outnames[i],"KUD99",sep=""),
  "ESRI Shapefile")}
  }}}
  rm(data,data.xy,sppt,locs,HpiOut,fhatOut,fhat.contlev,
     fhat.contlines,sldf,ps,sp,dataframe,spdf)
}
# write output
write.csv(output,paste(dir,"\\output.csv",sep=""))
```

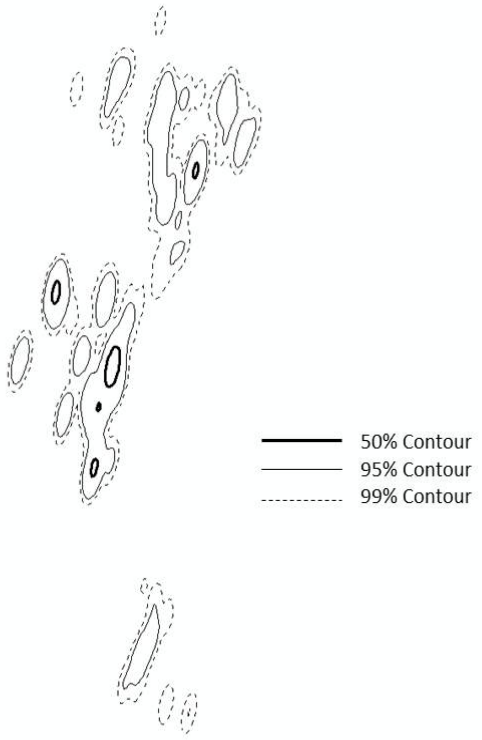

Figure 4.1: Example of KDE with $h_{plug\text{-}in}$ smoothing parameter to estimate size of home range for an American White Pelican.

9. There is also an alternative way to create KDE $h_{plug\text{-}in}$ without using the looping environment in the code above. This also uses the *ks* package in a more straight forward manner (Duong 2007, Duong and Hazelton 2003).

10. Exercise 4.3a - Download and extract zip folder into your preferred location

11. Set working directory to the extracted folder in R under File - Change dir...

12. First we need to load the packages needed for the exercise

```
library(ks)
library(rgdal)
library(maptools)
library(gpclib)
library(PBSmapping)
library(adehabitat)
library(adehabitatHR)
library(raster)
```

13. Now open the script "PantherKSplug.R" and run code directly from the script

```
#Load dataset
panther<-read.csv("pantherjitter.csv",header=T)
str(panther)
panther$CatID <- as.factor(panther$CatID)

cat143 <- subset(panther, panther$CatID == "143")

##Get only the coordinates
loc <- data.frame("x"=cat143$X, "y"=cat143$Y)

##Define the projection of the coordinates
proj4string <- CRS("+proj=utm +zone=17N +ellps=WGS84")

##Make SpatialPointsDataFrame using the XY, attributes, and projection
spdf <- SpatialPointsDataFrame(loc, cat143, proj4string = proj4string)

#Calculate the bandwidth matrix to use later in creating the KDE
Hpi1 <- Hpi(x = loc)
Hpi1

##write out the bandwidth matrix to a file as you might want to refer to it later
#write.table(Hpi1, paste("hpivalue_", "143", ".txt", sep=""), row.names=FALSE,
#sep="\t")

##Create spatial points from just the xyâĂŹs
loc.pts <- SpatialPoints(loc, proj4string=proj4string)
```

14. For home range calculations, some packages require evaluation points (ks) while others require grid as spatial pixels (adehabitatHR). Understanding these simple concepts of what formats of data are required by each package will save a tremendous amount of time as we move forward!

```
##Set the expansion value for the grid and get the bbox from the
##SpatialPointsDataFrame
```

```
expandValue <- 2500 #This value is the amount to add on each side of the bbox
#Change to 5000 if error occurs at 99% ud
boundingVals <- spdf@bbox

##Get the change in x and y and adjust using expansion value
deltaLong <- as.integer(((boundingVals[1,2]) - (boundingVals[1,1]))
      + (2*expandValue))
deltaLat <- as.integer(((boundingVals[2,2]) - (boundingVals[2,1]))
      + (2*expandValue))

##100 meter grid for data in this exercise
gridRes <- 100
gridSizeX <- deltaLong / gridRes
gridSizeY <- deltaLat / gridRes
##Offset the bounding coordinates to account for the additional area
boundingVals[2,1] <- boundingVals[2,1] - expandValue
boundingVals[2,2] <- boundingVals[2,2] + expandValue
boundingVals[1,1] <- boundingVals[1,1] - expandValue
boundingVals[1,2] <- boundingVals[1,2] + expandValue

##Grid Topology object is basis for sampling grid (offset, cellsize, dim)
gridTopo <- GridTopology((boundingVals[,1]),
      c(gridRes,gridRes),c(gridSizeX,gridSizeY))

##Using the Grid Topology and projection create a SpatialGrid class
sampGrid <- SpatialGrid(gridTopo, proj4string = proj4string)

##Cast over to Spatial Pixels
sampSP <- as(sampGrid, "SpatialPixels")

##convert the SpatialGrid class to a raster
sampRaster <- raster(sampGrid)

##set all the raster values to 1 such as to make a data mask
sampRaster[] <- 1

##Get the center points of the mask raster with values set to 1
evalPoints <- xyFromCell(sampRaster, 1:ncell(sampRaster))

##Here we can see how grid has a buffer around the locations and trajectory.
##This will ensure that we project our home range estimates into a slightly
##larger extent than the original points extent (bbox) alone.
plot(sampRaster)
#lines(tele.range.ltraj.lines, cex=0.5, lwd=0.1, col="grey")
points(loc, pch=1, cex=0.5)

##Create the KDE using the evaluation points
hpikde <- kde(x=loc, H=Hpi1, eval.points=evalPoints)

##Create a template raster based upon the mask and then assign the values from
##the kde to the template
```

```
hpikde.raster <- raster(sampRaster)

hpikde.raster <- setValues(hpikde.raster,hpikde$estimate)

##Lets take this raster and put it back into an adehabitat object
##This is convenient to use other adehabitat capabilities such as overlap
## indices or percent volume contours

##Cast over to SPxDF
hpikde.px <- as(hpikde.raster,"SpatialPixelsDataFrame")

##create new estUD using the SPxDF
hpikde.ud <- new("estUD", hpikde.px)

##Assign values to a couple slots of the estUD
hpikde.ud@vol = FALSE
hpikde.ud@h$meth = "Plug-in Bandwidth"

##Convert the UD values to volume using getvolumeUD from adehabitatHR
##and cast over to a raster
hpikde.ud.vol <- getvolumeUD(hpikde.ud, standardize=TRUE)
hpikde.ud.vol.raster <- raster(hpikde.ud.vol)

##Here we generate volume contours using the UD
hpikde.50vol <- getverticeshr(hpikde.ud, percent = 50,ida = NULL, unin = "m",
     unout = "ha", standardize=TRUE)
hpikde.80vol <- getverticeshr(hpikde.ud, percent = 80,ida = NULL, unin = "m",
     unout = "ha", standardize=TRUE)
hpikde.90vol <- getverticeshr(hpikde.ud, percent = 90,ida = NULL, unin = "m",
     unout = "ha", standardize=TRUE)
hpikde.95vol <- getverticeshr(hpikde.ud, percent = 95,ida = NULL, unin = "m",
     unout = "ha", standardize=TRUE)
hpikde.99vol <- getverticeshr(hpikde.ud, percent = 99,ida = NULL, unin = "m",
     unout = "ha", standardize=TRUE)

##Let's put the HR, volume, volume contours, and points on a plot
##and write out the shapefiles for contours for use in ArcMap
plot.new()
breaks <- c(0, 50, 80, 90, 95, 99)
plot(hpikde.ud.vol.raster, col=heat.colors(3), breaks=breaks,interpolate=TRUE,
     main="Kernel Density Estimation, Plug-in Bandwidth",xlab="Coords X",
     ylab="Coords Y",      legend.shrink=0.80,legend.args=list(text="UD by
     Volume (%)",side=4, font=2, line=2.5, cex=0.8))
plot(hpikde.99vol, add=T)#col="grey",axes=T)
plot(hpikde.95vol, add=TRUE)
plot(hpikde.90vol, add=TRUE)
plot(hpikde.80vol, add=TRUE)
plot(hpikde.50vol, add=TRUE)
points(loc, pch=1, cex=0.5)

writeOGR(hpikde.50vol,dsn="output",  layer="cat143plug50",driver="ESRI Shapefile",
```

```
   overwrite=TRUE)
writeOGR(hpikde.80vol,dsn="output", layer="cat143plug80",driver="ESRI Shapefile",
   overwrite=TRUE)
writeOGR(hpikde.90vol,dsn="output", layer="cat143plug90",driver="ESRI Shapefile",
   overwrite=TRUE)
writeOGR(hpikde.95vol,dsn="output", layer="cat143plug95",driver="ESRI Shapefile",
   overwrite=TRUE)
writeOGR(hpikde.99vol,dsn="output", layer="cat143plug99",driver="ESRI Shapefile",
   overwrite=TRUE)
```

4.4 Brownian Bridge Movement Models (BBMM)

The BBMM requires (1) sequential location data, (2) estimated error associated with location data, and (3) grid-cell size assigned for the output utilization distribution. The BBMM is based on two assumptions: (1) location errors correspond to a bivariate normal distribution and (2) movement between successive locations is random conditional on the starting and ending location (Horne et al. 2007). Normally distributed errors are common for GPS data and 1 h between locations likely ensured that movement between successive locations was random (Horne et al. 2007). The assumption of conditional random movement between paired locations, however, becomes less realistic as the time interval increases (Horne et al. 2007).

1. Exercise 4.4 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   require(survival)
   library(maptools)
   require(sp)
   require(gpclib)
   require(foreign)
   require(lattice)
   require(BBMM)
   ```

4. Now open the script "BBMMscript.R" and run code directly from the script

   ```
   panther<-read.csv("pantherjitter.csv",header=T)
   str(panther)
   panther$CatID <- as.factor(panther$CatID)#make CatID a factor
   ```

5. First, we need to get Date and time into proper format for R because Time in DateTimeET2 is single digit for some hours

   ```
   panther$NewTime <- str_pad(panther$TIMEET2,4, pad= "0")
   panther$NewDate <- paste(panther$DateET2,panther$NewTime)
   #Used to sort data in code below for all deer
   panther$DT <- as.POSIXct(strptime(panther$NewDate, format='%Y %m %d %H%M'))
   #Sort Data
   panther <- panther[order(panther$CatID, panther$DT),]
   ```

6. We need to define time lag between locations, GPS collar error, and cell size

   ```
   timediff <- diff(panther$DT)*60
   ```

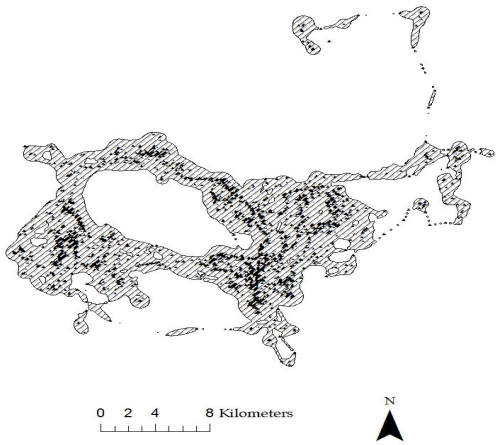

Figure 4.2: Example of 95% BBMM home range for a Florida Panther.

```
# remove first entry without any difference
panther <- panther[-1,]
panther$timelag <-as.numeric(abs(timediff))

#Subset for only one panther
cat143<-subset(panther, panther$CatID == "143")
cat143 <- cat143[-1,] #Remove first record with wrong timelag
cat143$CatID <- factor(cat143$CatID)
```

7. Use brownian.bridge function in package BBMM to run home range

```
BBMM = brownian.bridge(x=cat143$X, y=cat143$Y, time.lag=cat143$timelag,
   location.error=34, cell.size=100)
bbmm.summary(BBMM)

#Plot results for all contours
contours = bbmm.contour(BBMM, levels=c(seq(50, 90, by=10), 95, 99),
    locations=cat143, plot=TRUE)
# Print result
print(contours)
```

NOTE:

(a) Time lag refers to the elapsed time between consecutive GPS locations that was presented in section 2.3

(b) GPS collar error can be from error reported by the manufacturer of the GPS collar or from error test conducted at the study site

(c) Cell size refers to grid size we want to estimate the BBMM

8. We need to create output ascii files or shapefiles for graphical representation of size of BBMM (Fig. 4.2). We can also compare BBMM for the Florida panther to KDE using $h_{plug\text{-}in}$ (Fig. 4.3) and h_{ref} (Fig. 4.4). We start by creating a data.frame indicating cells within the contour desired and export as Ascii Grid

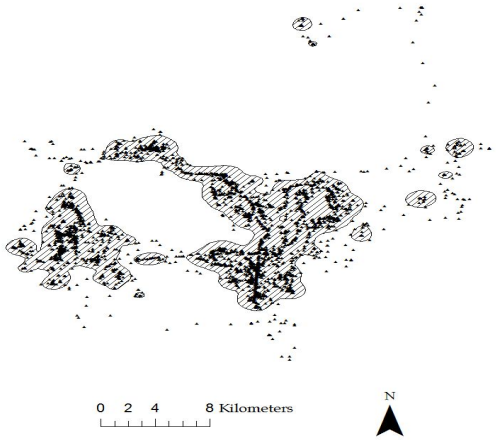

Figure 4.3: Example of 95% KDE home range with $h_{plug\text{-}in}$ for a Florida Panther.

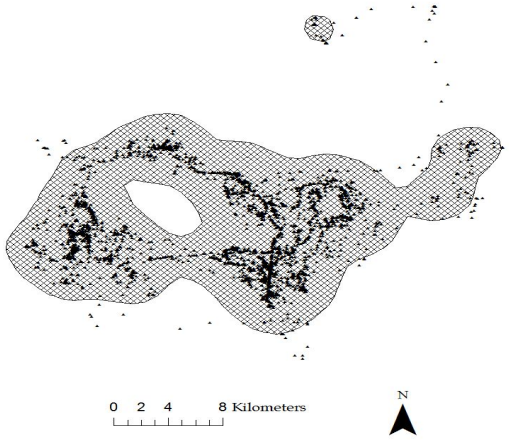

Figure 4.4: Example of 95% KDE home range with h_{ref} for a Florida Panther.

```
bbmm.contour = data.frame(x = BBMM$x, y = BBMM$y, probability = BBMM$probability)

# Pick a contour for export as Ascii
bbmm.50 = data.frame(x = BBMM$x, y = BBMM$y, probability =
 BBMM$probability)
bbmm.50 = bbmm.50[bbmm.50$probability <= contours$Z[1],]
# Output ascii file for cells within specified contour.
m = SpatialPixelsDataFrame(points = bbmm.50[c("x", "y")], data=bbmm.50)
m = as(m, "SpatialGridDataFrame")
writeAsciiGrid(m, "50ContourInOut.asc", attr=ncol(bbmm.50))

# Print result for 80 percent BBMM
print(contours)
# Pick a contour for export as Ascii
bbmm.80 = data.frame(x = BBMM$x, y = BBMM$y, probability =
 BBMM$probability)
bbmm.80 = bbmm.80[bbmm.80$probability <= contours$Z[4],]
# Output ascii file for cells within specified contour.
m = SpatialPixelsDataFrame(points = bbmm.80[c("x", "y")], data=bbmm.80)
m = as(m, "SpatialGridDataFrame")
writeAsciiGrid(m, "80ContourInOut.asc", attr=ncol(bbmm.80))

# Print result for 95 percent BBMM
print(contours)
# Pick a contour for export as Ascii
bbmm.95 = data.frame(x = BBMM$x, y = BBMM$y, probability =
 BBMM$probability)
bbmm.95 = bbmm.95[bbmm.95$probability <= contours$Z[4],]
# Output ascii file for cells within specified contour.
m = SpatialPixelsDataFrame(points = bbmm.95[c("x", "y")], data=bbmm.95)
m = as(m, "SpatialGridDataFrame")
writeAsciiGrid(m, "95ContourInOut.asc", attr=ncol(bbmm.95))

# Print result for 99 percent BBMM
print(contours)
# Pick a contour for export as Ascii
bbmm.99 = data.frame(x = BBMM$x, y = BBMM$y, probability =
 BBMM$probability)
bbmm.99 = bbmm.99[bbmm.99$probability <= contours$Z[7],]
# Output ascii file for cells within specified contour.
m = SpatialPixelsDataFrame(points = bbmm.99[c("x", "y")], data=bbmm.99)
m = as(m, "SpatialGridDataFrame")
writeAsciiGrid(m, "99ContourInOut.asc", attr=ncol(bbmm.99))
```

9. Now we can create shapefiles of contours from ascii files in ArcMap

   (a) Convert ASCII files to Rasters using Conversion Toolbox

   ```
   Toolbox>Conversion Tools>To Raster>ASCII to Raster
        Input ASCII raster file: 50ContourInOut.asc
        Output raster: AsciiToRast
        Output data type (optional): INTEGER
   ```

   (b) Convert No Data values to value of 1 and those that are not to value of 0 by:

```
Toolbox>Spatial Analyst Tools>Math>Logical>Is Null
     Input raster: tv53_99contr
     Output raster: IsNull_bv53_3
```

(c) Convert raster probability surface to a shapefile by opening shapefile table Highlight all raster cells with a value=1 then open appropriate Toolbox as follows:

```
Toolbox>Conversion Tools>From Raster>Raster to Polygon
     Input raster:  IsNull_bv53_3
     Field (Optional): Value
     Output Polygon Features: RasterT_IsNull_2.shp
Uncheck the "Simplify polygons (optional)" box for proper results. Select OK.
Tool will convert all cells with value=1 to a shapefile with multiple polygons.
```

(d) Calculate area of new shapefile using appropriate tool (i.e., Xtools) Open table to view area of polygon and summarize to get total size of home range (Fig. 4.5)

```
Right click on column heading "Hectares"
Select Statistics and Sum will be the total hectares in the home range
```

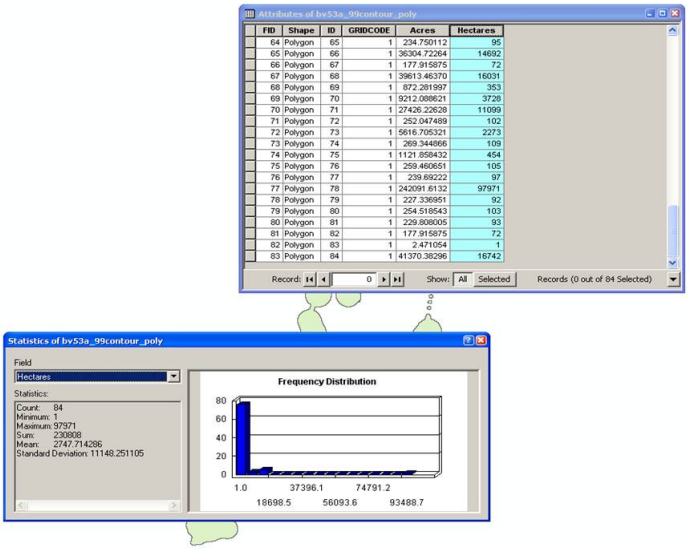

Figure 4.5: This figure shows how to summarize size of home range in ArcMap.

10. Or another way to create shapefiles of contours right in R! There is also an additional way here to create ascii files that will eliminate the need for Step 9 above.

```
# Create data.frame indicating cells within the contour desired and export
# as Ascii Grid
bbmm.contour = data.frame(x = BBMM$x, y = BBMM$y, probability = BBMM$probability)
#contours = bbmm.contour(BBMM2, levels=c(seq(50, 90, by=10), 95, 99),
   locations=cat143, plot=TRUE)
#bbmm.contour = data.frame(x = BBMM2$x, y = BBMM2$y, probability = BBMM2$probability)
#contours = bbmm.contour(BBMM3, levels=c(seq(50, 90, by=10), 95, 99),
   locations=loc2, plot=TRUE)
#bbmm.contour = data.frame(x = BBMM3$x, y = BBMM3$y, probability = BBMM3$probability)

str(contours) #Look at contour or isopleth levels 1 to 7 (50%-99%)
#$List of 2
```

```
 #$ Contour: chr [1:7] "50%" "60%" "70%" "80%" ...
 #$ Z       : num [1:7] 7.35e-05 5.66e-05 4.22e-05 2.81e-05 1.44e-05 .

# Pick a contour for export as Ascii (1 = 50%, 2 = 60%, etc.)
bbmm.50 = bbmm.contour[bbmm.contour$probability >= contours$Z[1],]
bbmm.50$in.out <- 1

bbmm.50 <-bbmm.50[,-3]
# Output ascii file for cells within specified contour.
m50 = SpatialPixelsDataFrame(points = bbmm.50[c("x", "y")], data=bbmm.50)
m50.g = as(m50, "SpatialGridDataFrame")
writeAsciiGrid(m50.g, "50ContourInOut.asc", attr=ncol(bbmm.50))
```

11. Code to start converting the above SpatialPixelsDataFrame to SpatialPolygonsDataFrame and export as ESRI Shapefile

```
shp.50 <- as(m50, "SpatialPolygonsDataFrame")
map.ps50 <- SpatialPolygons2PolySet(shp.50)
diss.map.50 <- joinPolys(map.ps50, operation = 'UNION')

#Set Projection information
diss.map.50 <- as.PolySet(diss.map.50, projection = 'UTM', zone = '17')

#Re-assign the PolySet to Spatial Polygons and Polygon ID (PID) to 1
diss.map.p50 <- PolySet2SpatialPolygons(diss.map.50, close_polys = TRUE)
data50 <- data.frame(PID = 1)
diss.map.p50 <- SpatialPolygonsDataFrame(diss.map.p50, data = data50)
```

12. Use package rgdal to write shapefile to ArcMap

```
writeOGR(diss.map.p, dsn = ".", layer="contour50", driver = "ESRI Shapefile")
```

13. Also use package rgdal to read the shapefile back into R

```
map.50 <- readOGR(dsn=".", layer="contour50")
plot(map.50)
```

14. Repeat steps 8-10 for each contour shapefile desired

15. Another short exercise to subset large dataset by the appropriate time lags in your data but only include locations collected within the 7 hour schedule (i.e., < 421 minutes)

```
loc <- subset(cat143, cat143$timelag != "NA" & cat143$timelag < 421)
BBMM2 = brownian.bridge(x=loc$X, y=loc$Y, time.lag=loc$timelag, location.error=34,
   cell.size=100)
bbmm.summary(BBMM2)
contours1 = bbmm.contour(BBMM2, levels=c(seq(50, 90, by=10), 95, 99),
    locations=loc, plot=TRUE)
```

16. Or we could exclude the extreme 1% of locations based on a time cutoff that we will determine as in Walter et al. 2011

```
freq <- as.data.frame(table(round(cat143$timelag)))
#Result is Var1 = the time difference, and Freq = its frequency in the data
freq$percent <- freq$Freq/dim(cat143)[1]*100
freq$sum[1] <- freq$percent[1]
```

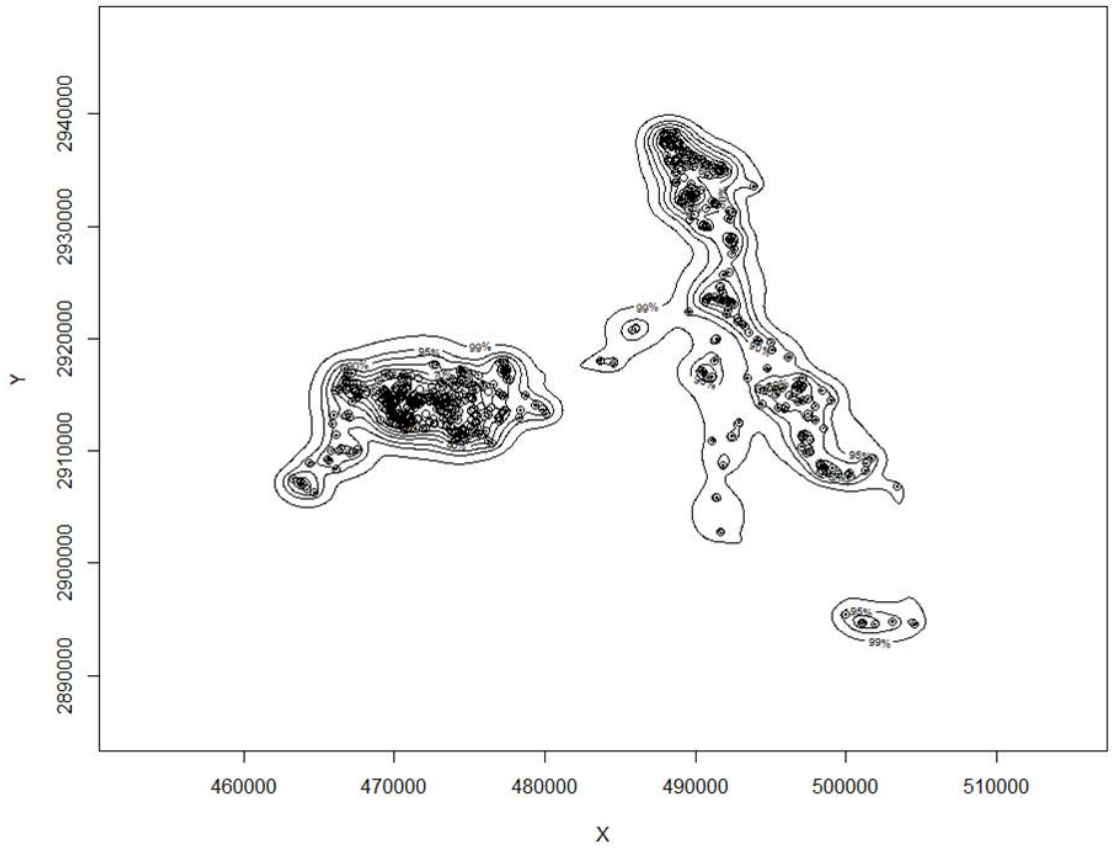

Figure 4.6: Home range of one panther using BBMM showing all contours.

```
#Loop below runs through data to determine "cutoff" time
for (j in 2:dim(freq)[1]){
freq$sum[j] <- freq$percent[j]+freq$sum[j-1]
}

indicator <- which(freq$sum>99)
cutoff <- as.numeric(as.character(freq$Var1[min(indicator)]))
cutoff
# [1] 2939 #2939 minutes
```

17. Need to subset data and only calculate BBMM with timediff < 2940 minutes

```
loc2 <- subset(loc, loc$timelag < 2940)
str(loc2)
BBMM3 = brownian.bridge(x=loc2$X, y=loc2$Y, time.lag=loc2$timelag, location.error=34,
   cell.size=100)
bbmm.summary(BBMM3)
```

18. Plot out resulting home ranges to see differences when altering time difference criteria. Be sure to change bbmm.contours and rerun Steps 10-11 each time before running each section of code below

```
raw.df <- data.frame("x"=cat143$X,"y"=cat143$Y)
```

```
##Define the projection of the coordinates
proj4string <- CRS("+proj=utm +zone=17N +ellps=WGS84")
##Make SpatialPointsDataFrame using the XY, attributes, and projection
spdf <- SpatialPointsDataFrame(raw.df, cat143, proj4string = proj4string)
plot(map.99, col="grey",axes=T)
plot(map.95, add=TRUE)
plot(map.90, add=TRUE)
plot(map.80, add=TRUE)
plot(map.50, add=TRUE)
points(spdf)

windows()

loc.df <- data.frame("x"=loc$X,"y"=loc$Y)
##Define the projection of the coordinates
proj4string <- CRS("+proj=utm +zone=17N +ellps=WGS84")
##Make SpatialPointsDataFrame using the XY, attributes, and projection
spdf2 <- SpatialPointsDataFrame(loc.df, loc, proj4string = proj4string)
plot(map.99, col="grey",axes=T)
plot(map.95, add=TRUE)
plot(map.90, add=TRUE)
plot(map.80, add=TRUE)
plot(map.50, add=TRUE)
points(spdf2)
```

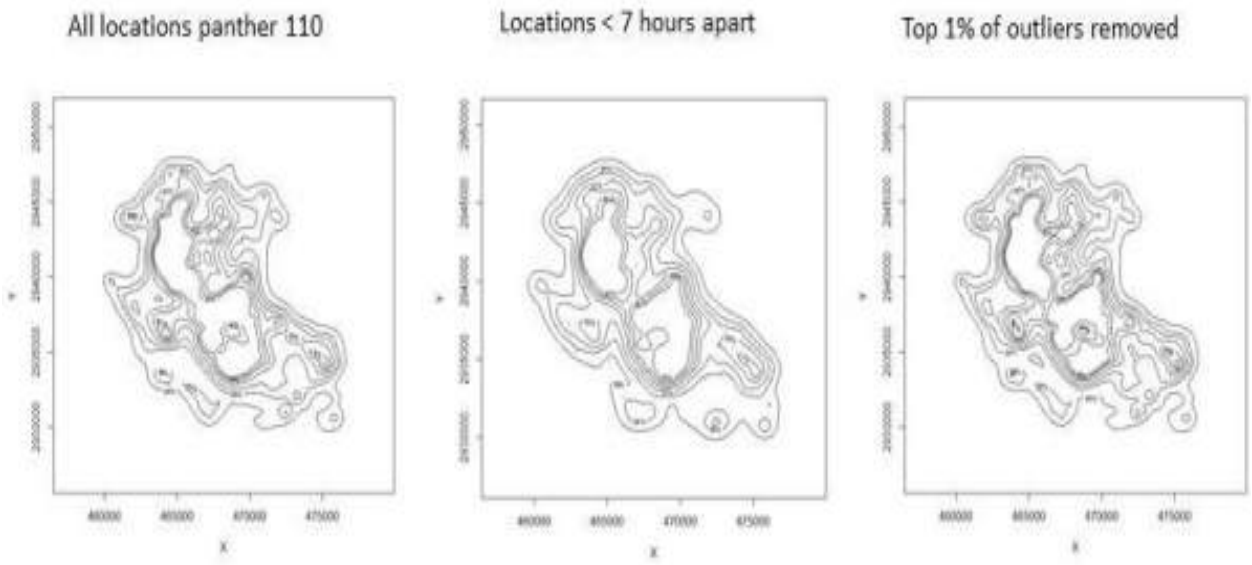

Figure 4.7: Home range using BBMM for panther 110 with various time lags incorporated.

4.5 Movement-based Kernel Density Estimation (MKDE)

If we want to take both BBMM and KDE to a higher level we can incorporate movement-based estimators of home range. Movement-based Kernel Density Estimation (MKDE) incorporates movements trajectories and habitat components of the landscape your animal occupies (Benhamou 2011, Benhamou and Cornelis 2010). This method requires a

habitat layer and the adehabitatHR package requires that no duplicate entries exist for a given date so makes estimates of home range with GPS data problematic. Furthermore, after tirelessly trying this method for days using panther data with time intervals, we changed to vulture data that had dates but then had to remove duplicates. If you have worked out all of these issues, you can skip ahead to MKDE estimates with your data starting at Step 6.

1. Exercise 4.5 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(adehabitatHR)
library(adehabitatLT)
library(sp)
library(rgdal)
library(raster)
library(chron)
library(rgeos)
```

4. Now open the script "MKDEscript.R" and run code directly from the script

   NOTE: Two issues at this step held us back with the method for weeks so we will stress them here:

   Extent of the raster layer selected - Although Extent was the lesser problem, it still needs to be address for several reasons. If the extent is too large or raster cell size too small then processing time increases. Although we would not really want to spend the time to clip raster habitat layers for each animal, you may need to have separate rasters for different areas of your study site to cut down on processing time. More importantly, animals need to be within the same grid for analysis using MKDE/BRB home range estimates. This will become more apparent later but preparing habitat or landscape layers for all animals using the same habitat extent will save time in the end.

   Projection of the raster layer and locations - Even we missed this one with all our experiences and constant issues with data layers in different projections. We assumed that defining the projection with R would take care of this issue but we could not have been more wrong. So before we move forward, we want to demonstrate our thought processes here and how we solved this problem.

   Using the raster habitat layer that was created using Albers Equal Area Conic similar to exercises from Chapter 1, we can already see the difference in the raster layer by it's orientation and the numbers on the x and y axis are on a different scales compared to the locations in Figure 4.8 (Fig. 4.9).

5. So the raster layer that is included in this exercise is in UTM Zone 17. Now, import the raster layer to have layers in the same projection (Fig. 4.10).

```
habitat = as(readGDAL("beauzoom100.asc"), "SpatialPixelsDataFrame")
#beauzoom100.asc has GDAL driver AAIGrid
#and has 276 rows and 355 columns
str(habitat)
image(habitat)
```

6. Now we need to import our locations and get them in the form we need for this exercise before we can move forward

```
#Creates a Spatial Points Data Frame for 2 animals by ID
```

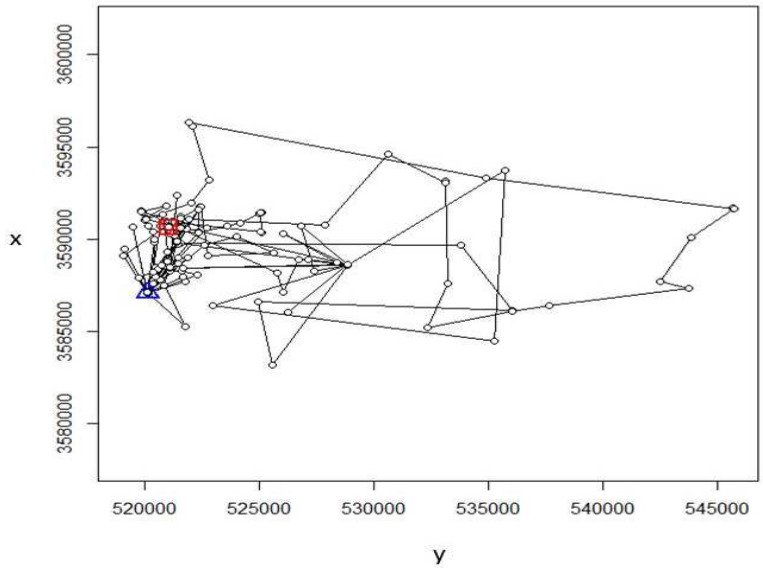

Figure 4.8: Locations of one vulture in UTM 17N.

```
twobirds <-read.csv("Twobirds.csv", header=T)
twobirds$id <-as.factor(twobirds$id)

#Needs to be done to get proper digits of date into R then POSIXct
xtime <- paste(twobirds$OldDate,twobirds$Hour)
twobirds$PosTime <- xtime

#Calculates time difference to use as dt
twobirds$date_time <- chron(as.character(twobirds$OrigDate),twobirds$Hour,
   format=c(dates="m/d/y", times="h:m:s"))
timediff <- diff(twobirds$date_time)*24*60
twobirds <-twobirds[-1,]
twobirds$timediff <-as.numeric(abs(timediff))

#Creates class Spatial Points for all locations
data.xy = twobirds[c("x","y")]
xysp <- SpatialPoints(data.xy)

#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)

#Creates a spatial data frame of ID
idsp<-data.frame(onebird[2])
#Creates a spatial data frame of Date
datesp<-data.frame(onebird[1])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp,datesp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
proj4string(merge) <- CRS("+proj=utm +zone=17N +ellps=WGS84")

#Cast the Dates as POSIXct dates
```

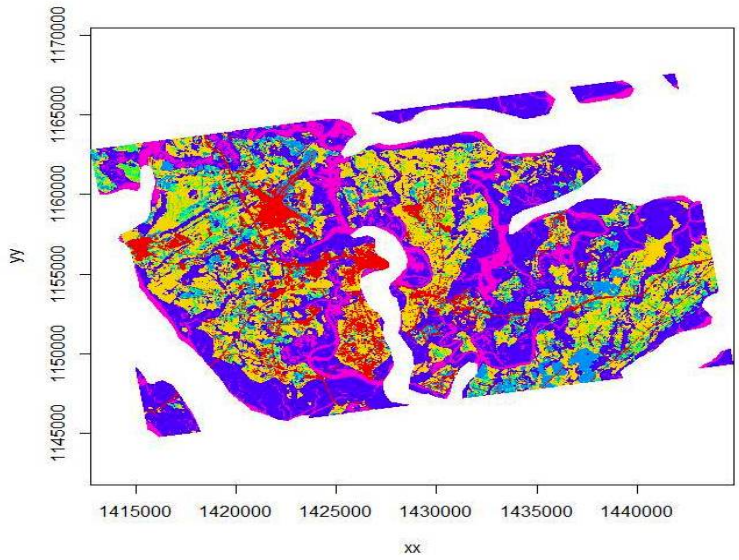

Figure 4.9: Imported habitat layer in Albers Equal Area Conic Projection.

```
merge$DT <-as.POSIXct(strptime(merge$Date, format='%Y%m%d'))
```

7. With the same projections for our 2 data layers, we can move forward. First we need to create ltraj as in Chapter 3 and use some additional code to overlay the trajectory onto the Spatial Pixels Data Frame using the command "spixdf" as in the code below that results in Fig. 4.11. Basically, if this works then we are on the right path to moving forward with MKDE.

```
#Create an ltraj trajectory object.
ltraj <- as.ltraj(coordinates(merge), merge$DT, id = merge$id,
     burst = merge$id, typeII = TRUE)
plot(ltraj)

#Code to overlap trajectory onto raster layer
plot(ltraj, spixdf=habitat)
```

8. Now identify habitats that can and can not be used by vultures (i.e., water is not used; Fig. 4.12)

```
#Be sure to do this step after plotting ltraj onto spixdf or won't work!
#This step just builds a "fake" habitat map with habitat=1
fullgrid(habitat) <- TRUE
hab <- habitat
hab[[1]] <- as.numeric(!is.na(hab[[1]]))
image(hab)#See note below

#NOTE: When viewing this image, there may appear to be a periphery of habitat
around the extent of the raster that you exported as an ascii from ArcMap.  Even
deleting the NoData values or categories in ArcMap does not remove this extra
data from around the periphy of your habitat layer.  To remove this category of
data, clip the raster within a boundary box in R or in ArcMap using the following:

 Data Management Tools-->Raster-->Raster Dataset-->Copy Raster
Select the optional field in this tool called Ignore Background Value and a related
```

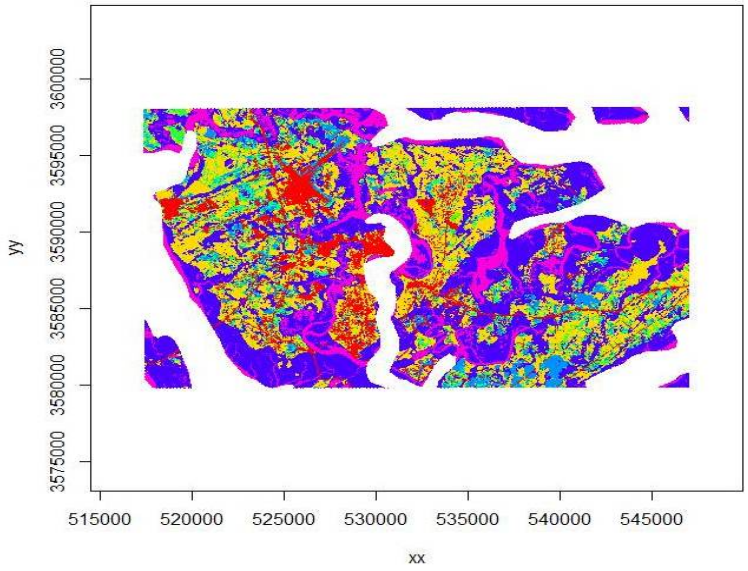

Figure 4.10: Imported habitat layer projected to UTM Zone 17N similar to vulture locations.

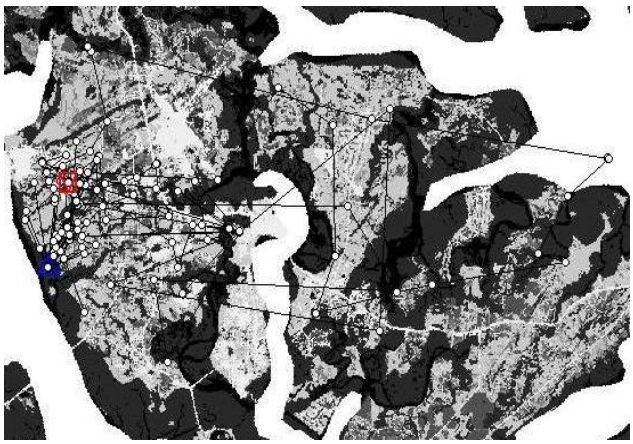

Figure 4.11: Overlay of ltraj for one bird on spixdf=habitat in UTM Zone 17N.

```
NoData Value field (see help window for more info on how to use these fields).


#The step below is needed to convert SpatialGrid to SpatialPixels for use in "ud"
#estimation (i.e., "habitat" in "grid=habitat" must be of class SpatialPixels)
fullgrid(habitat) <- FALSE
class(habitat)#shows it is now class SpatialPixels
```

9. Now we can begin to create Movement-based KDEs using biased random bridges (BRBs)

```
#Assign parameter values for BRB
tmax <- 1*(24*60*60) + 1 ## set the maximum time between locations to be just more
   # than 1 day
lmin <- 50 ## locations less than 50 meters apart are considered inactive.
hmin <- 100 ## arbitrarily set to be same as hab grid cell resolution

#Diffusion component for each habitat type using plug-in method
```

Figure 4.12: Red identifies ocean and tributaries not used by vultures.

```
vv<- BRB.D(ltraj, Tmax = tmax, Lmin = lmin,  habitat = hab)
vv

 vv
$‘49‘
         n         D
global 234 146.46958
0        2  24.88662
1      229 140.07425

$‘50‘
         n        D
global 272 148.5204
0        0      NaN
1      270 142.3398

attr(,"class")
[1] "DBRB"

ud <- BRB(ltraj, D = vv, Tmax = tmax, Lmin = lmin, hmin=hmin, grid = hab, b=TRUE,
   extent=0.1, tau = 300)
ud
image(ud)

#Address names in ud by assigning them to be the same as the ids in ltraj
#Must be done before using "getverticeshr" function
names(ud) <- id(ltraj)
```

10. Create contours using *getverticeshr* to display or export as shapefiles (Fig. 4.15).

```
ver1_95 <- getverticeshr(ud, percent=95, standardize = TRUE, whi = id(ltraj[1]))
plot(ver1_95)
windows()
```

```
ver2_95 <- getverticeshr(ud, percent=95, standardize = TRUE, whi = id(ltraj[2]))
plot(ver2_95)
```

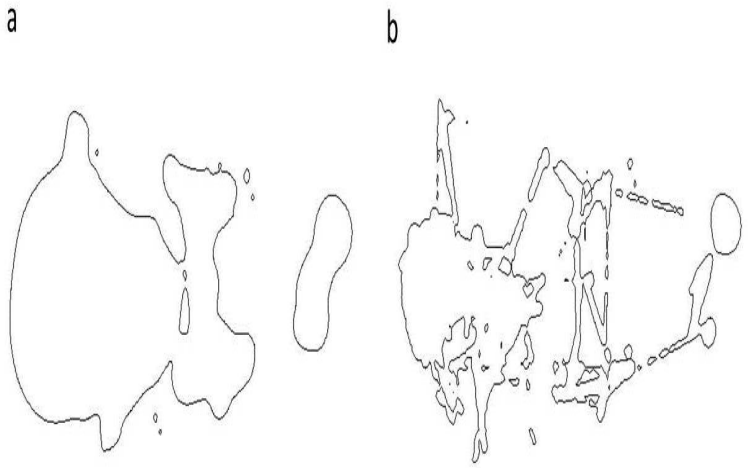

Figure 4.13: Contours of home range for 2 black vultures estimated using the Movement-based kernel density method (MKDE)

11. Now let's create a new UD using an actual habitat layer that has more than "used/unused" such as the 7 habitat categories from original dataset

```
#Start by importing the habitat layer again and run the following
habitat2 = as(readGDAL("beauzoom100.asc"), "SpatialPixelsDataFrame")
plot(ltraj, spixdf=habitat2)

#CODE TO CONDUCT BRB
#Assigne parameter values for BRB
# Parameters for the Biased Random Bridge Kernel approach
tmax <- 1*(24*60*60) + 1 ## set the maximum time between locations to be just
    more than 1 day
lmin <- 50 ## locations less than 50 meters apart are considered inactive.
hmin <- 100 ## arbitrarily set to be same as hab grid cell resolution

#Diffusion component for each habitat type using plug-in method
vv2 <-  BRB.D(ltraj, Tmax = tmax, Lmin = lmin,  habitat = habitat2)
vv2

 vv2
$‘49‘
         n           D
global 234 146.4695789
1        4   1.5909335
```

```
2        9   1.5728743
3        0         NaN
4        4   1.2998600
5        1   0.4621171
6        2   0.4812110
7        0         NaN

$`50`
         n           D
global 272 148.5203800
1        1   1.2267057
2        3  21.6201300
3        0         NaN
4        1   0.7170513
5        0         NaN
6        5   9.5334889
7        0         NaN

attr(,"class")
[1] "DBRB"
```

```
ud2 <- BRB(ltraj, D = vv2, Tmax = tmax, Lmin = lmin, hmin=hmin, habitat = habitat2,
   b=TRUE, extent=0.1, tau = 300, same4all=FALSE)
image(ud2)

names(ud2) <- id(ltraj)

ver1a <- getverticeshr(ud2, percent=95, standardize = TRUE, whi = id(ltraj[1]))
plot(ver1a)
windows()
ver2a <- getverticeshr(ud2, percent=95, standardize = TRUE, whi = id(ltraj[2]))
plot(ver2a)
```

12. Now let's create a new UD without incorporating an actual habitat layer which reverts to simply KDE with BRB to see if there is difference in the resulting home range shapes are for each estimate.

```
#Diffusion component for each habitat type using plug-in method
vv3<- BRB.D(ltraj, Tmax = tmax, Lmin = lmin,  habitat = NULL)
vv3

ud3 <- BRB(ltraj, D = vv3, Tmax = tmax, Lmin = lmin, hmin=hmin, habitat = NULL,
   b=TRUE, extent=0.1, tau = 300, same4all=FALSE)
image(ud3)

names(ud3) <- id(ltraj)

windows()
ver1b <- getverticeshr(ud3, percent=95, standardize = TRUE, whi = id(ltraj[1]))
plot(ver1b)
windows()
ver2b <- getverticeshr(ud3, percent=95, standardize = TRUE, whi = id(ltraj[2]))
```

```
plot(ver2b)
```

4.6 Dynamic Brownian Bridge Movement Model (dBBMM)

With the wide-spread use of GPS technology to track animals in near real time, estimators of home range and movement have developed concurrently. Unlike the traditional point-based estimators (i.e., MCP, KDE with $h_{ref}/h_{plug\text{-}in}$) that only incorporate density of locations into home range estimation, newer estimators incorporate more data provided by GPS technology. While BBMM incorporates a temporal component and GPS error into estimates, dynamic Brownian Bridge Movement Models (dBBMM) incorporate temporal and behavioral characteristics of movement paths into estimation of home range (Kranstauber et al. 2012). However, estimating a movement path over the entire trajectory of data should be separated into behavorial movement patterns (i.e., resting, feeding) prior to estimating the variance of the Brownian motion ($\sigma^2{}_m$). Overestimating the $\sigma^2{}_m$ will cause an imprecision in estimation of the utilization distribution that dBBMM seeks to address (Kranstauber et al. 2012).

1. Exercise 4.6 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(adehabitatLT)
library(move)
library(circular)
library(sp)
```

4. Now open the script "MuleydBBMM.R" and run code directly from the script

```
muleys <-read.csv("muleysexample.csv")
str(muleys)

#TIME DIFF ONLY NECESSARY AS A MEANS TO EXCLUDE POOR DATA LATER
muleys$Date <- as.numeric(muleys$GPSFixTime)
timediff <- diff(muleys$Date)*24*60
muleys <-muleys[-1,]
muleys$timediff <-as.numeric(abs(timediff))
```

5. make dates as.POSIXct so we can sort data later

```
muleys$DT <-as.POSIXct(strptime(muleys$GPSFixTime, format='%Y.%m.%d %H:%M:%OS'))
muleys$DT
```

6. Sort data here to address error in code - must be done or move object will not be created! I am not sure why but if data is sorted properly in excel you may be able to ignore this step.

```
muleys <- muleys[order(muleys$id,muleys$DT),]

#EXCLUDE OUTLIERS AND POOR DATA FIXES
newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.80)
muleys <- newmuleys
newmuleys <-subset(muleys, muleys$Long < -107)
muleys <- newmuleys
```

7. Create a move object for all deer using the *Move* package

```
loc <- move(x=muleys$X, y=muleys$Y, time=as.POSIXct(muleys$GPSFixTime,
     format="%Y.%m.%d %H:%M:%S"), proj=CRS("+proj=utm"),data=muleys,
     animal=muleys$id)
```

8. Now create a dBBMM object

```
d8_dbbmm <- brownian.bridge.dyn(object=ld8, location.error=22, window.size=19,
    margin=7, dimSize=100,time.step=180)
```

9. Alternatively, create a *Move* object and run dBBMM for each deer individually for increased flexibility in plotting home range contours. Need to subset using muley$id column if proceeding with the code below this point.

```
dataD8 <- subset(muleys, muleys$id == "D8")
dataD8$id <- factor(dataD8$id)
d8 <- move(x=dataD8$X, y=dataD8$Y, time=as.POSIXct(dataD8$GPSFixTime,
   format="%Y.%m.%d %H:%M:%S"),    proj=CRS("+proj=utm +zone=12 +datum=NAD83"),
   data=dataD8, animal=dataD8$id)
d8_dbbmm <- brownian.bridge.dyn(object=d8, location.error=22, window.size=19,
   margin=7, dimSize=100,time.step=180)
plot(d8_dbbmm)
contour(d8_dbbmm, levels=c(.5,.9,.95,.99), add=TRUE)
show(d8_dbbmm)
```

10. Plot the movement of the animal

```
par(mfcol=1:2)
plot(loc2, type="o", col=3, lwd=2, pch=20, xlab="location_east",
      ylab="location_north")
```

11. Code in this exercise can be used to create ascii or shapefiles of resulting home range simiilar to code used in BBMM

4.7 Characteristic Hull Polygons (CHP)

Now we are going to get into another class of home range estimators that use polygons created by Delaunay triangulation of a set of relocations and then removing a subset of the resulting triangles. These polygons can have concave edges, be composed of disjoint regions, and contain empty portions of unused space within hull interiors. This estimator has been described in the adehabitatHR package and evaluated on black-footed albatross (*Phoebastria nigripes*; Downs and Horner 2009). Polygon-based estimators may be a useful method for a variety of species but research has been limited.

1. Exercise 4.7 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(adehabitatHR)
library(maptools)
```

4. Now open the script "CHPscript.R" and run code directly from the script

5. Now read in the locations and create a Spatial Points Data Frame for the 2 animals by ID using the code that follows:

```
twocats <-read.csv("pantherjitter.csv",
header=T)

data.xy = twocats[c("X","Y")]
```

6. Creates a class of Spatial Points for all locations with projection defined

```
xysp <- SpatialPoints(data.xy)
proj4string(xysp) <- CRS("+proj=utm +zone=17N +ellps=WGS84")
```

7. Creates a Spatial Data Frame from Spatial points

```
sppt<-data.frame(xysp)
```

8. Creates a spatial data frame of ID

```
idsp<-data.frame(twocats[1])
```

9. Merges ID and Date into the same spatial data frame

```
coordinates(idsp)<-sppt

head(as.data.frame(idsp))
#Results from the above code

    ID      X       Y
1   121 494155.6 2904240
2   121 498182.3 2905598
3   121 498476.2 2905114
4   121 499210.5 2905661
5   121 499467.3 2905533
6   121 502960.9 2904391
```

10. Code for estimation of home range using CharHull from adehabitatHR

```
res <- CharHull(idsp[,1])
class("res")
```

11. Code to display the home range in R (Fig. 4.14).

```
plot(res)
```

12. Code to compute the home range size for 20−100 percent

```
MCHu2hrsize(res)

          121         143
20    10.70262    150.2998
30    28.73158    342.9798
40    79.15079    688.7553
50   172.95317   1418.7588
60   343.12024   2319.1578
70   700.94072   3574.5525
80  1333.94110   5910.8221
```

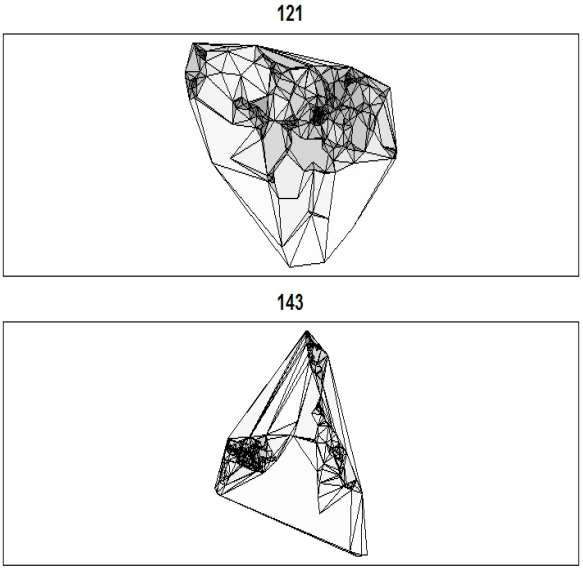

Figure 4.14: Example of CHP home range for 2 Florida panther.

```
90  2431.27046  12912.0320
100 5538.22402 103361.1336

OR
```

13. Computes the home range size for 95 percent

```
MCHu2hrsize(res, percent=95)

OR use

getverticeshr(res, percent=95)

Object of class "SpatialPolygonsDataFrame" (package sp):

Number of SpatialPolygons:  2

Variables measured:
   id      area
1 121  2413.518
2 143 12845.298
```

Keep in mind that CHP estimates in Figure 4.16 are for 20−100 percent with 100% filling in the entire area traversed by each animal. If you want to visualize core area (i.e., 50%) or 95% that are commonly reported, the resulting home ranges might be more appropriate visualization of the area an animal traverses.

4.8 Local Convex Hull (LoCoH)

Local convex hull nonparametric kernel method (LoCoH), which generalizes the minimum convex polygon method, produces bounded home ranges and better convergence properties than parametric kernel methods as sample size increases (Getz et al. 2007, Getz and Wilmers

2004). The use of LoCoH also has been investigated for identifying hard boundaries (i.e. rivers, canyons) of home ranges because it is essentially a non-parametric kernel method using minimum convex polygon construction. The use of polygons instead of kernels gives LoCoH the ability to produced hard edges or boundaries that will not overlap into unused spaces common to kernel methods (Getz et al. 2007). Without getting into to much detail, LoCoH has 3 modifications that reference the k-nearest neighbor convex hulls (NNCH) in some form. The 3 terms are fixed k, fixed radius (r-NNCH), and adaptive (a-NNCH) that are comparable to kernel smoothing of href, lscv, and plug-in, respectively.

1. Exercise 4.8a - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(adehabitatLT)
   library(move)
   library(circular)
   library(sp)
   ```

4. Now open the script "LocohGUIscript.R" and run code directly from the script
5. We will begin using LoCoH by first downloading the proper script NNCH.R
   NOTE: LoCoH GUI will not work in earlier versions of R with adehabitat in Windows 10 but worked with Windows Vista so may need to download most recent version of R along with package adehabitatHR
6. Then install the script to use with the adehabitatHR package:

   ```
   source("NNCH.R")
   ```

7. Next we need to install the graphic user interface locoh_gui.R

   ```
   source("locoh_gui.R")
   ```

8. To access the GUI we can simply invoke LoCoH using the command

   ```
   locoh()
   ```

   The LoCoH GUI will pop up in a separate window in R (Fig. 4.18).
   Refer to adehabitatHR manual and Getz et al. (2007) for more details on LoCoH inputs.
9. Then browse for the appropriate shapefile of animal locations (Fig. 4.18)
10. Choose the algorithm (k, r, a) and enter the value of the variable (Fig. 4.18)
11. Select the option that is appropriate for handling duplicate points (Fig. 4.18)
12. Save resulting home range as shapefiles or pdfs (Fig. 4.18)

Alternatively, for the purists that don't like GUIs or need estimates of home range for multiple animals we can calculate LoCoH directly in R using the original adehabitat package (Calenge 2007) which is not recommended and removed from a previous version of this manual. We can use the adehabitatHR package to estimate LoCoH with any of the 3 algorithms (i.e., k, r, adaptive):

1. Exercise 4.8b - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

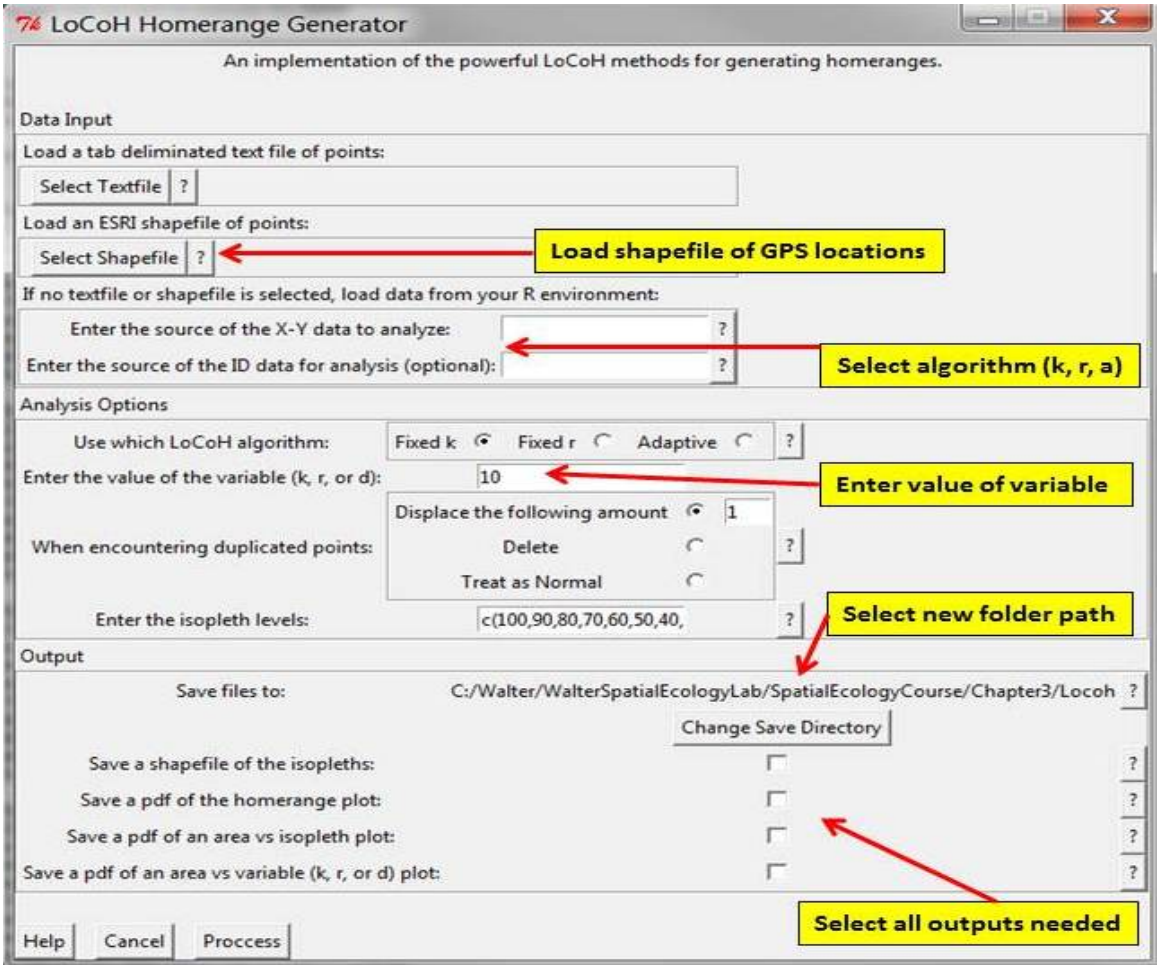

Figure 4.15: The LoCoH GUI.

```
library(adehabitatHR)
library(shapefiles)
library(rgeos)
library(rgdal)
library(maptools)
```

4. Now open the script "CodeHRscript.R" and run code directly from the script

```
panther <- read.csv("pantherjitter2.csv")
str(panther)
panther$CatID <- as.factor(panther$CatID)

#Or explore with one panther with 381 relocations
cat159 <- subset(panther, CatID=="159")
str(cat159)
cat159$CatID <- factor(cat159$CatID)


#Get the relocation data from the source file
data.xy = cat159[c("x","y")]
xysp <- SpatialPoints(data.xy)
#Creates a Spatial Data Frame from
```

```
sppt<-data.frame(xysp)
#Creates a spatial data frame of ID
idsp<-data.frame(cat159[1])
#Adds ID and Date data frame with locations data frame
coordinates(idsp)<-sppt
proj4string(idsp) <- CRS("+proj=utm +zone=17 +ellps=WGS84")
locsdf <-as.data.frame(idsp)
head(locsdf)
## Shows the relocations
plot(data.xy, as.numeric(locsdf[,1]), col="red")

## Examines the changes in home-range size for various values of k
## Be patient! the algorithm can be very long
ar <- LoCoH.k.area(idsp, k=c(16:25))
## 24 points seems to be a good choice (rough asymptote for all animals)
## the k-LoCoH method:
nn <- LoCoH.k(idsp, k=24)
## Graphical display of the results
plot(nn, border=NA)
## the object nn is a list of objects of class
## SpatialPolygonsDataFrame
length(nn)
names(nn)
class(nn[[1]])
## shows the content of the object for the first animal
as.data.frame(nn[[1]])

## The 95% home range is the smallest area for which the
## proportion of relocations included is larger or equal
## to 95% In this case, it is the 339th row of the
## SpatialPolygonsDataFrame.
## The area covered by the home range panther 159
## is equal to 5506.04 ha.
## shows this area:
plot(nn[[1]][339,], lwd=2)

#The 50% home range code is on line 146
plot(nn[[1]][146,], add=TRUE)

#The 99% home range code is on line 133
plot(nn[[1]][363,], lwd=3, add=TRUE)

##We can write shapefiles for specific sizes of home range
ver <-getverticeshr(nn)
ver
plot(ver)
writeOGR(ver,dsn="FixedK",layer="FixedK24", driver = "ESRI Shapefile",
     overwrite=TRUE)
##Or we can write shapefiles for specific sizes of home range but overwrite will
##not work so must edit path so "FixedK" folder is created with code below.
ver50 <-getverticeshr(nn, percent=50)
```

```
writeOGR(ver50,dsn="FixedK",layer="50FixedK24", driver = "ESRI Shapefile",
     overwrite=TRUE)
ver95 <-getverticeshr(nn, percent=95)
writeOGR(ver95,dsn="FixedK",layer="95FixedK24", driver = "ESRI Shapefile",
     overwrite=TRUE)
ver99 <-getverticeshr(nn, percent=99)
writeOGR(ver99,dsn="FixedK",layer="99FixedK24", driver = "ESRI Shapefile",
     overwrite=TRUE)
```