4.3a KDE with plug-in bandwidth selection (hplug-in)

Manual of Applied Spatial Ecology

3/11/2022

Here we will estimate home range using all 3 of the previous estimators on the same animal. We will conclude with estimating home range using the Brownian Bridge Movement Models (BBMM) for comparison to kernel density estimators.

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

2. Set working directory to the extracted folder in R under Session - Set Working Directory...

3. Now open the script "Panther_All4.Rmd" and run code directly from the script

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

```
library(rgdal)
library(adehabitatHR)
library(ks)
library(raster)
library(stringr)
library(BBMM)
library(maptools)
library(PBSmapping)
library(move)
#library(ctmm)
```

5. Now include have a separate section of code to include projection information we will use throughout the exercise. In previous versions, these lines of code were within each block of code

```
utm.crs <- CRS("+proj=utm +zone=17N +ellps=WGS84")
```

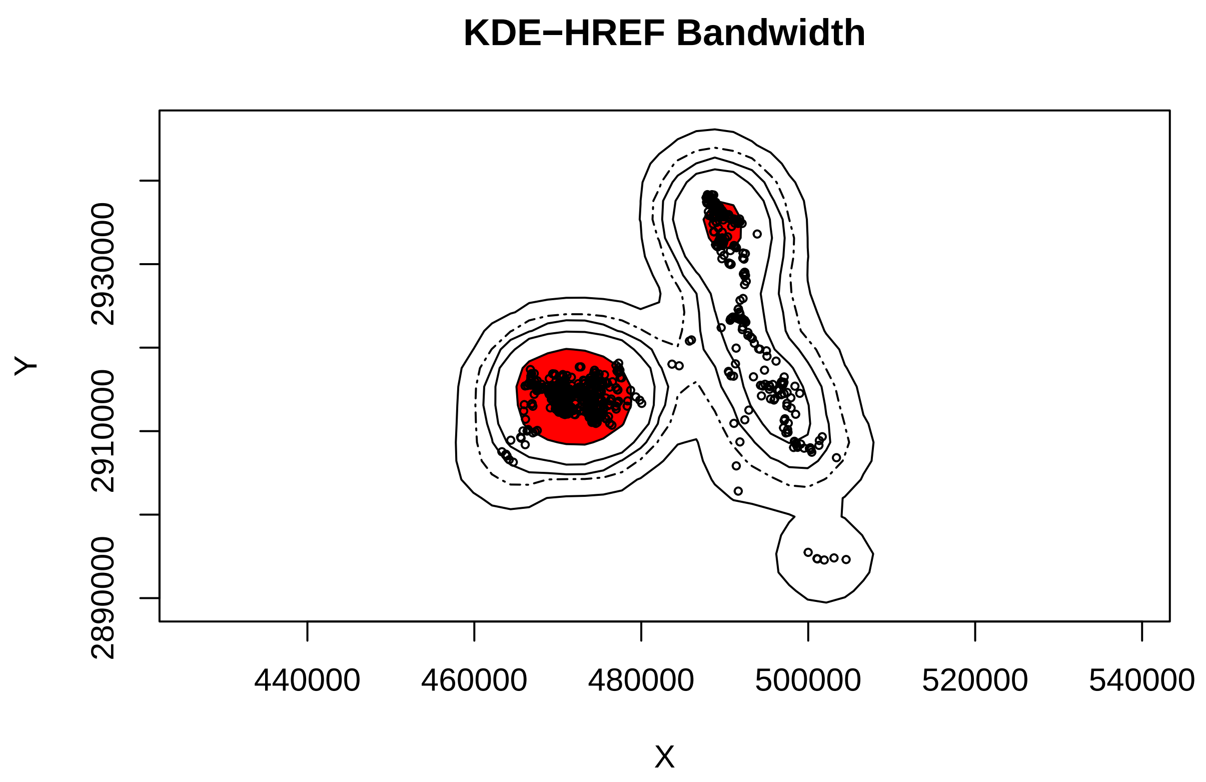
6. We will use an abbreviated dataset to save processing time and the code will also output shapefiles of home ranges

```
panther<-read.csv("pantherjitter.csv",header=T)
panther$CatID <- as.factor(panther$CatID)
cat143 <- subset(panther, panther$CatID == "143")
cat143$CatID <- droplevels(cat143$CatID)
```

7. We will start by running KDE with href similar to exercise 4.1.

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

cats <- SpatialPointsDataFrame(loc,cat143, proj4string = utm.crs)
udbis <- kernelUD(cats[,2], h = "href")

ver <- getverticeshr(udbis, unin = "m", unout = "km2", standardize=TRUE)
ver50 <- getverticeshr(udbis, percent=50,unin = "m", unout = "km2", standardize=TRUE)
ver80 <- getverticeshr(udbis, percent=80,unin = "m", unout = "km2", standardize=TRUE)
```

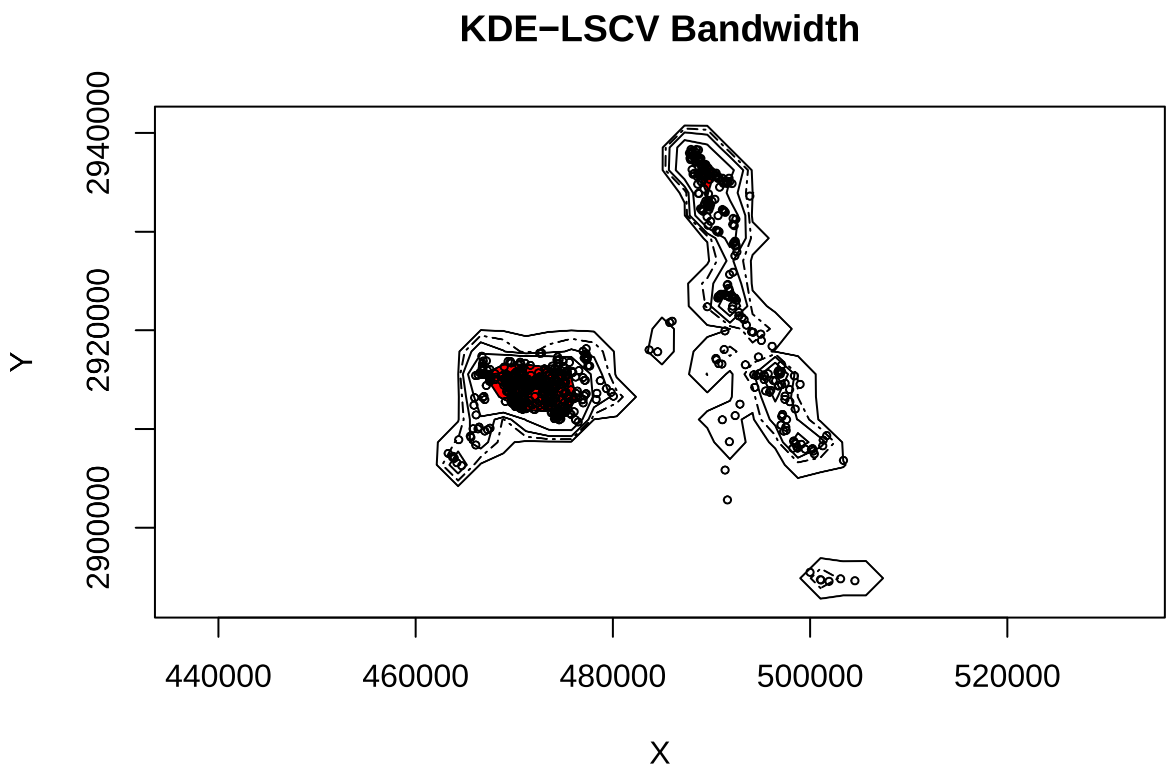
```
ver90 <- getverticeshr(udbis, percent=90,unin = "m", unout = "km2", standardize=TRUE)
ver95 <- getverticeshr(udbis, percent=95,unin = "m", unout = "km2", standardize=TRUE)
ver99 <- getverticeshr(udbis, percent=99,unin = "m", unout = "km2", standardize=TRUE)

plot(ver99,main="KDE-HREF Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(ver95, lty=6, add=TRUE)
plot(ver90, add=TRUE)
plot(ver80, add=TRUE)
plot(ver50, col="red",add=TRUE)
points(loc, pch=1, cex=0.5)
```



8. Next we will run KDE with hlscv similar to exercise 4.2.

```
udbis2 <- kernelUD(cats[,2], h = "LSCV")
#Notice the error
# The algorithm did not converge
# within the specified range of hlim: try to increase it
## Example of estimation using LSCV
cat143$jitterX <- jitter(cat143$X, factor=500)
cat143$jitterY <- jitter(cat143$Y, factor=500)
locjitter <- data.frame("x"=cat143$jitterX,"y"=cat143$jitterY)
jitterspdf <- SpatialPointsDataFrame(locjitter,cat143, proj4string = utm.crs)
udbis3 <- kernelUD(jitterspdf[,2], h = "LSCV")#, hlim = c(100, 500),extent=1)

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

ver2 <- getverticeshr(udbis3, unin = "m", unout = "km2", standardize=TRUE)
```

```r
#Or individually by isopleth
ver2_50 <- getverticeshr(udbis3, percent=50,unin = "m", unout = "km2", standardize=TRUE)
ver2_80 <- getverticeshr(udbis3, percent=80,unin = "m", unout = "km2", standardize=TRUE)
ver2_90 <- getverticeshr(udbis3, percent=90,unin = "m", unout = "km2", standardize=TRUE)
ver2_95 <- getverticeshr(udbis3, percent=95,unin = "m", unout = "km2", standardize=TRUE)
ver2_99 <- getverticeshr(udbis3, percent=99,unin = "m", unout = "km2", standardize=TRUE)

plot(ver2_99,main="KDE-LSCV Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(ver2_95, lty=6, add=TRUE)
plot(ver2_90, add=TRUE)
plot(ver2_80, add=TRUE)
plot(ver2_50, col="red",add=TRUE)
points(loc, pch=1, cex=0.5)
```



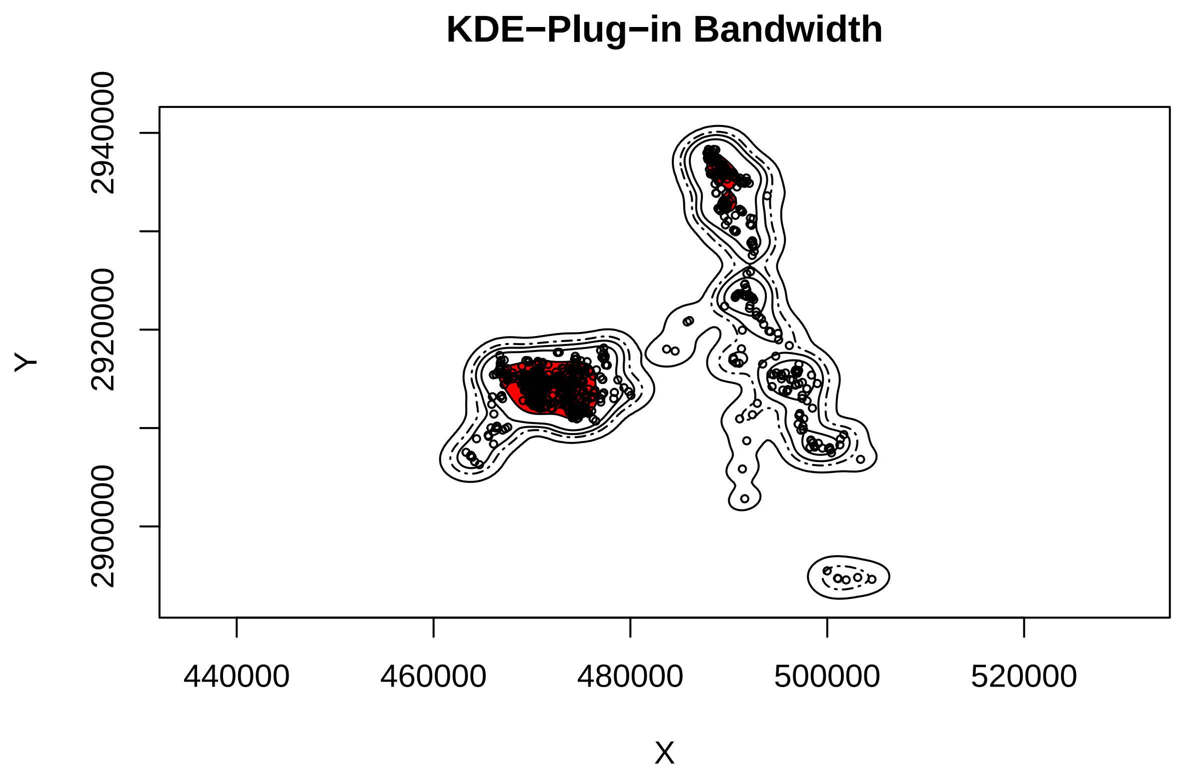
9. Then we will run KDE with hplug-in similar to exercise 4.3.

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

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

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

```
##              [,1]       [,2]
## [1,] 1917158.8  277154.8
## [2,]  277154.8 1030023.5
```

```r
##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 xy?s
loc.pts <- SpatialPoints(loc, proj4string=utm.crs)

##For home range calculations, ##some packages require evaluation points (ks) while others
##require grid as spatial pixels (adehabitatHR).

##Set the expansion value for the grid and get the bbox from the SpatialPointsDataFrame
expandValue <- 5000 #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
deltaX <- as.integer(((boundingVals[1,2]) - (boundingVals[1,1])) + (2*expandValue))
deltaY <- as.integer(((boundingVals[2,2]) - (boundingVals[2,1])) + (2*expandValue))

##100 meter grid for data in this exercise
gridRes <- 100
gridSizeX <- deltaX / gridRes
gridSizeY <- deltaY / 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 = utm.crs)

##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))

##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
```

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

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

##Lets take this raster and put it back into an adehabitatHR object
##This is convenient to use other adehabitatHR 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)#,unin = "m", unout = "km2")
hpikde.80vol <- getverticeshr(hpikde.ud, percent = 80,unin = "m", unout = "km2")
hpikde.90vol <- getverticeshr(hpikde.ud, percent = 90,unin = "m", unout = "km2")
hpikde.95vol <- getverticeshr(hpikde.ud, percent = 95,unin = "m", unout = "km2")
hpikde.99vol <- getverticeshr(hpikde.ud, percent = 99,unin = "m", unout = "km2")

plot(hpikde.99vol,main="KDE-Plug-in Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(hpikde.95vol, lty=6, add=TRUE)
plot(hpikde.90vol, add=TRUE)
plot(hpikde.80vol, add=TRUE)
plot(hpikde.50vol,col="red", add=TRUE)
points(loc, pch=1, cex=0.5)
```



10. We will finish by running BBMM.

```
#To run BBMM we first need to use the original dataset to calculate time between locations
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$DT),]
#TIME DIFF NECESSARY IN BBMM CODE
timediff <- diff(panther$DT)*60
# remove first entry without any difference
panther <- panther[-1,]
panther$timelag <-as.numeric(abs(timediff))

cat143<-subset(panther, panther$CatID == "143")
cat143 <- cat143[-1,] #Remove first record with wrong timelag
cat143$CatID <- factor(cat143$CatID)
BBMM = brownian.bridge(x=cat143$X, y=cat143$Y, time.lag=cat143$timelag, location.error=34,
cell.size=100)

bbmm.summary(BBMM)
```

```
##
## Brownian motion variance :  4352.462
## Size of grid :  381294 cells
## Grid cell size :  100
```

```
contours = bbmm.contour(BBMM, levels=c(seq(50, 90, by=10), 95, 99), locations=cat143, plot=FALSE)

#Create a new data frame for all contours
bbmm.contour = data.frame(x = BBMM$x, y = BBMM$y, probability = BBMM$probability)

# Pick a contour for export as Ascii
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)
```

```
## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 1

## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 2
```

```
#m50.g = as(m50, "SpatialGridDataFrame")
#writeAsciiGrid(m50.g, "50ContourInOut.asc", attr=ncol(bbmm.50))

# Convert 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')
diss.map.50 <- as.PolySet(diss.map.50, projection = 'UTM', zone = '17')
diss.map.p50 <- PolySet2SpatialPolygons(diss.map.50, close_polys = TRUE)
```

```
## Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer_proj
## = prefer_proj): Discarded datum Unknown based on WGS84 ellipsoid in Proj4
## definition
```

```
data50 <- data.frame(PID = 1)
diss.map.p50 <- SpatialPolygonsDataFrame(diss.map.p50, data = data50)
# writeOGR(diss.map.p50, dsn = ".", layer="contour50", driver = "ESRI Shapefile")
# map.50 <- readOGR(dsn=".", layer="contour50")
# plot(map.50)

# Pick a contour for export as Ascii
bbmm.80 = bbmm.contour[bbmm.contour$probability >= contours$Z[4],]
bbmm.80$in.out <- 1

bbmm.80 <-bbmm.80[,-3]
# Output ascii file for cells within specified contour.
m80 = SpatialPixelsDataFrame(points = bbmm.80[c("x", "y")], data=bbmm.80)
```

```
## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 1

## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 2
```

```
m80.g = as(m80, "SpatialGridDataFrame")
#writeAsciiGrid(m80.g, "80ContourInOut.asc", attr=ncol(bbmm.80))
```

```r
# Convert to SpatialPolygonsDataFrame and export as ESRI Shapefile
shp.80 <- as(m80, "SpatialPolygonsDataFrame")
map.ps80 <- SpatialPolygons2PolySet(shp.80)
diss.map.80 <- joinPolys(map.ps80, operation = 'UNION')
diss.map.80 <- as.PolySet(diss.map.80, projection = 'UTM', zone = '17')
diss.map.p80 <- PolySet2SpatialPolygons(diss.map.80, close_polys = TRUE)
data80 <- data.frame(PID = 1)
diss.map.p80 <- SpatialPolygonsDataFrame(diss.map.p80, data = data80)
# writeOGR(diss.map.p80, dsn = ".", layer="contour80", driver = "ESRI Shapefile")
# map.80 <- readOGR(dsn=".", layer="contour80")
# plot(map.80)

# Pick a contour for export as Ascii
bbmm.90 = bbmm.contour[bbmm.contour$probability >= contours$Z[5],]
bbmm.90$in.out <- 1

bbmm.90 <-bbmm.90[,-3]
# Output ascii file for cells within specified contour.
m90 = SpatialPixelsDataFrame(points = bbmm.90[c("x", "y")], data=bbmm.90)
```

```
## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 1

## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 2
```

```r
#m90.g = as(m90, "SpatialGridDataFrame")
#writeAsciiGrid(m90.g, "90ContourInOut.asc", attr=ncol(bbmm.90))

# Convert to SpatialPolygonsDataFrame and export as ESRI Shapefile
shp.90 <- as(m90, "SpatialPolygonsDataFrame")
map.ps90 <- SpatialPolygons2PolySet(shp.90)
diss.map.90 <- joinPolys(map.ps90, operation = 'UNION')
diss.map.90 <- as.PolySet(diss.map.90, projection = 'UTM', zone = '17')
diss.map.p90 <- PolySet2SpatialPolygons(diss.map.90, close_polys = TRUE)
data90 <- data.frame(PID = 1)
diss.map.p90 <- SpatialPolygonsDataFrame(diss.map.p90, data = data90)
# writeOGR(diss.map.p90, dsn = ".", layer="contour90", driver = "ESRI Shapefile")
# map.90 <- readOGR(dsn=".", layer="contour90")
# plot(map.90)

# Pick a contour for export as Ascii
bbmm.95 = bbmm.contour[bbmm.contour$probability >= contours$Z[6],]
bbmm.95$in.out <- 1

bbmm.95 <-bbmm.95[,-3]
# Output ascii file for cells within specified contour.
m95 = SpatialPixelsDataFrame(points = bbmm.95[c("x", "y")], data=bbmm.95)
```

```
## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 1

## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 2
```

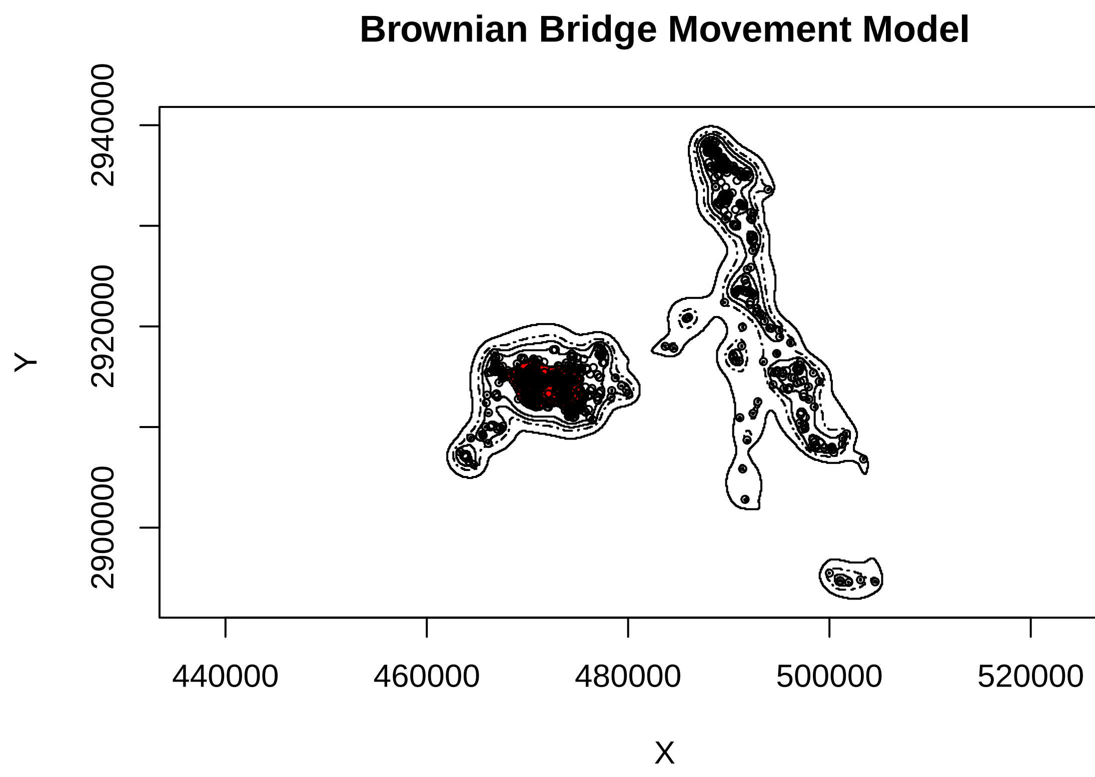
```r
#m95.g = as(m95, "SpatialGridDataFrame")
#writeAsciiGrid(m95.g, "95ContourInOut.asc", attr=ncol(bbmm.95))

# Convert to SpatialPolygonsDataFrame and export as ESRI Shapefile
shp.95 <- as(m95, "SpatialPolygonsDataFrame")
map.ps95 <- SpatialPolygons2PolySet(shp.95)
diss.map.95 <- joinPolys(map.ps95, operation = 'UNION')
diss.map.95 <- as.PolySet(diss.map.95, projection = 'UTM', zone = '17')
diss.map.p95 <- PolySet2SpatialPolygons(diss.map.95, close_polys = TRUE)
data95 <- data.frame(PID = 1)
diss.map.p95 <- SpatialPolygonsDataFrame(diss.map.p95, data = data95)
# writeOGR(diss.map.p95, dsn = ".", layer="contour95", driver = "ESRI Shapefile")
# map.95 <- readOGR(dsn=".", layer="contour95")
# plot(map.95)

# Pick a contour for export as Ascii
bbmm.99 = bbmm.contour[bbmm.contour$probability >= contours$Z[7],]
bbmm.99$in.out <- 1

bbmm.99 <-bbmm.99[,-3]
# Output ascii file for cells within specified contour.
m99 = SpatialPixelsDataFrame(points = bbmm.99[c("x", "y")], data=bbmm.99)
```

```
## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 1

## Warning in points2grid(points, tolerance, round): grid has empty column/rows in
## dimension 2
```

```r
# m99.g = as(m99, "SpatialGridDataFrame")
# writeAsciiGrid(m99.g, "99ContourInOut.asc", attr=ncol(bbmm.99))

# Convert to SpatialPolygonsDataFrame and export as ESRI Shapefile
shp.99 <- as(m99, "SpatialPolygonsDataFrame")
map.ps99 <- SpatialPolygons2PolySet(shp.99)
diss.map.99 <- joinPolys(map.ps99, operation = 'UNION')
diss.map.99 <- as.PolySet(diss.map.99, projection = 'UTM', zone = '17')
diss.map.p99 <- PolySet2SpatialPolygons(diss.map.99, close_polys = TRUE)
data99 <- data.frame(PID = 1)
diss.map.p99 <- SpatialPolygonsDataFrame(diss.map.p99, data = data99)
#writeOGR(diss.map.p99, dsn = ".", layer="contour99", driver = "ESRI Shapefile")
# map.99 <- readOGR(dsn=".", layer="contour99")
# plot(map.99)
```

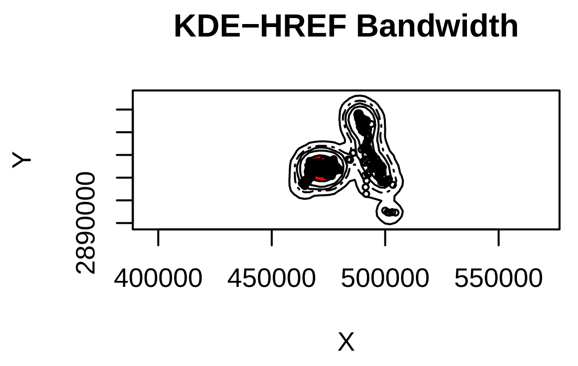
11. Plot BBMM

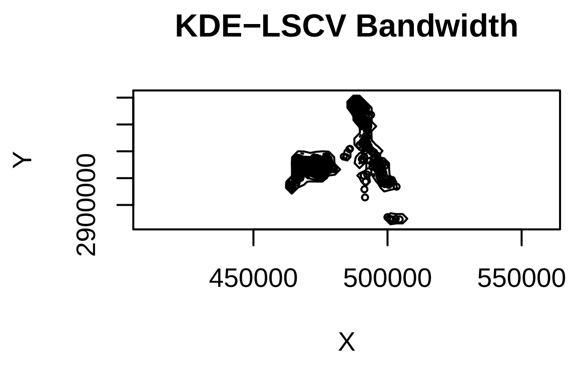
```r
plot(diss.map.p99,main="Brownian Bridge Movement Model",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(diss.map.p95, lty=6, add=TRUE)
plot(diss.map.p90, add=TRUE)
plot(diss.map.p80, add=TRUE)
plot(diss.map.p50,col="red", add=TRUE)
points(loc, pch=1, cex=0.5)
```

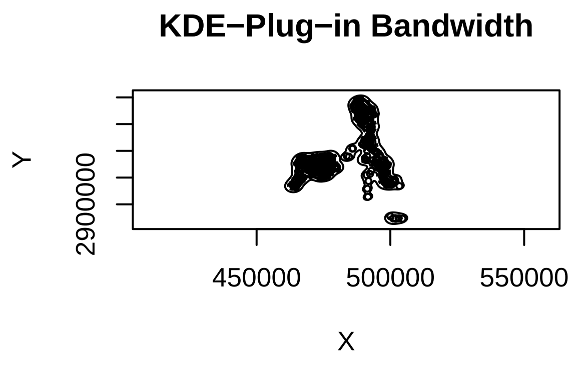


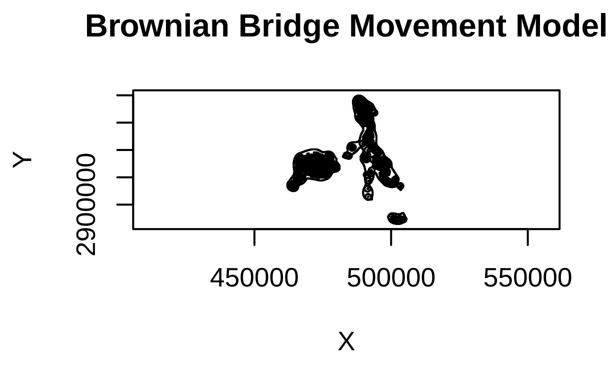
12. We can add 4 estimators to the plot window to compare across estimators

```
par(mfrow=c(2,2))

plot(ver99,main="KDE-HREF Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(ver95, lty=6, add=TRUE)
plot(ver90, add=TRUE)
plot(ver80, add=TRUE)
plot(ver50, col="red",add=TRUE)
points(loc, pch=1, cex=0.5)

plot(ver2_99,main="KDE-LSCV Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(ver2_95, lty=6, add=TRUE)
plot(ver2_90, add=TRUE)
plot(ver2_80, add=TRUE)
plot(ver2_50, col="red",add=TRUE)
points(loc, pch=1, cex=0.5)

plot(hpikde.99vol,main="KDE-Plug-in Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(hpikde.95vol, lty=6, add=TRUE)
plot(hpikde.90vol, add=TRUE)
plot(hpikde.80vol, add=TRUE)
plot(hpikde.50vol,col="red", add=TRUE)
points(loc, pch=1, cex=0.5)

plot(diss.map.p99,main="Brownian Bridge Movement Model",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(diss.map.p95, lty=6, add=TRUE)
plot(diss.map.p90, add=TRUE)
plot(diss.map.p80, add=TRUE)
plot(diss.map.p50,col="red", add=TRUE)
points(loc, pch=1, cex=0.5)
```









12. We will quickly explore autocorrelated kernel density estimator. It was easier to convert our dataframe to a move object prior to creating a telemetry object required by package ctmm.

```
cat.move <- move(x=cat143$X, y=cat143$Y, time=as.POSIXct(cat143$NewDate,
     format='%Y %m %d %H%M'), proj=utm.crs,data=cat143,
     animal=cat143$CatID)
#
# cat143$timestamp <- cat143$DT
# cat143$x <- cat143$X
# cat143$y <- cat143$Y

# cat143$Date.lt <- as.POSIXlt(strptime(cat143$NewDate, format='%Y %m %d %H%M'),tz="EDT")
#cat143 <- cat143[c(22,23,20,24:25)]
cat.telem <- as.telemetry(cat.move,timeformat=timestamp, timezone="UTC",projection=utm.crs, na.rm="row"
GUESS <- ctmm.guess(cat.telem,interactive=FALSE)
GUESS2 <- ctmm(tau=1)
FIT <- ctmm.fit(cat.telem,GUESS)
FIT2 <- ctmm.fit(cat.telem,GUESS2)
UD <- akde(cat.telem,FIT)
UD2 <- akde(cat.telem,FIT2)
plot(cat.telem,UD=UD2)
```

13. We can then plot them along with other estimators to compare

```
par(mfrow=c(2,2))

plot(ver99,main="KDE-HREF Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(ver95, lty=6, add=TRUE)
plot(ver90, add=TRUE)
plot(ver80, add=TRUE)
plot(ver50, col="red",add=TRUE)
points(loc, pch=1, cex=0.5)

plot(hpikde.99vol,main="KDE-Plug-in Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(hpikde.95vol, lty=6, add=TRUE)
plot(hpikde.90vol, add=TRUE)
plot(hpikde.80vol, add=TRUE)
plot(hpikde.50vol,col="red", add=TRUE)
points(loc, pch=1, cex=0.5)

plot(diss.map.p99,main="Brownian Bridge Movement Model",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(diss.map.p95, lty=6, add=TRUE)
plot(diss.map.p90, add=TRUE)
plot(diss.map.p80, add=TRUE)
plot(diss.map.p50,col="red", add=TRUE)
points(loc, pch=1, cex=0.5)

plot(cat.telem,UD=UD,main="autocorrelated KDE Best")
```

14. Or a few other options for autcorrelated KDE

```
par(mfrow=c(2,2))

plot(ver99,main="KDE-HREF Bandwidth",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(ver95, lty=6, add=TRUE)
plot(ver90, add=TRUE)
plot(ver80, add=TRUE)
plot(ver50, col="red",add=TRUE)
points(loc, pch=1, cex=0.5)

plot(diss.map.p99,main="Brownian Bridge Movement Model",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(diss.map.p95, lty=6, add=TRUE)
plot(diss.map.p90, add=TRUE)
plot(diss.map.p80, add=TRUE)
plot(diss.map.p50,col="red", add=TRUE)
points(loc, pch=1, cex=0.5)

plot(cat.telem,UD=UD,main="autocorrelated KDE Best")

plot(cat.telem,UD=UD2,main="autocorrelated KDE tau=1")
```