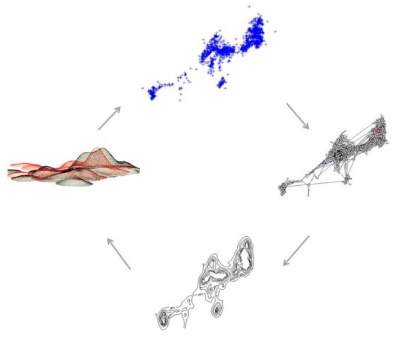


Manual of Applied Spatial Ecology

W. David Walter

2023-12-05

Contents

Preface

The purpose of this manual is to assist researchers on methods for data management and analysis using the R environment or other software after data has been collected in the field. The impetus behind this manual was from many years of frustration in trying to analyze data in R using code and forgetting how it was done upon completion of a study. We wanted to find a way to avoid needing to search computers for folders to find old R code then try to remember what we did to the data to get the code to run properly. Over the years, advancements in data handling and manipulation, GIS capabilities, and methods of estimators for home range, movements, resource selection, and spatial epidemiology have occurred within the R environment. Program R is free and used by researchers world-wide but R also provides a platform to create and display spatial layers without the need for the variety of GUI software, free or otherwise. Furthermore, analyzing spatial data in R enables statistical analysis of data without the errors that may arise from bringing data from spreadsheet or GIS software to statistical programs. We would like to stress that this manual is not the authority on all topics presented herein. Our goal was to create an online manual that could be easily followed by researchers, biologists, or graduate students to analyze their data in R. Although the user would benefit from general introductory knowledge of using R and ArcMap, most of the manual is for mid-level users of R that need guidance beyond the basics of introductory R and GIS coureses. We also provide numerous citations throughout each section should the user choose to learn the theory or more details behind each topic. In addition, this manual provides a handy outline of course materials for an Applied Spatial Ecology course that will surely expand or change as the field evolves. As time permits and errors are brought to our attention, we plan to update and correct problems so be sure to send any corrections or comments our way.

Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

Recommended citation:

Walter, W.D. Manual of Applied Spatial Ecology. Walter Applied Spatial Ecology Lab, Pennsylvania State University, University Park. Access Date.

http://ecosystems.psu.edu/research/labs/walter-lab.

Acknowledgments

Numerous colleagues have provide assistance with R packages they created or with code they have provided in some other form. We would be remiss if we failed to thank these colleagues for their hard work:

Sharon Baruch-Mordo, The Nature Conservancy, Fort Collins, CO 80524 ; sbaruch-mordo@tnc.org Simon Benhamou, Centre d'Ecologie Fonctionnelle et Evolutive, France Clément Calenge, Data Analysis Support Unit, Directorate for Studies and Research, National Office of Hunting and Wild Fauna, Saint Benoist - 78610 Auffargis, France Mevin Hooten, Colorado Cooperative Fish and Wildlife Research Unit, 201 Wagar Bldg, Colorado State University, Fort Collins, CO 8052 ; Mevin.Hooten@colostate.edu Bill Kanapaux, Pennsylvania Cooperative Fish and Wildlife Research Unit, 406 Forest Resources Bldg., Pennyslvania State University, University Park, PA 16802 ; wjk15@psu.edu Bart Kranstauber, Max Planck Institute for Ornithology, Eberhard-Gwinner-Str., 82319 Seewiesen; bart.kranstauber@uni-konstanz.de Ryan Nielsen, West Inc.,415 W. 17th St. Suite 200, Cheyenne, WY 82001 ; rnielson@westinc. com Glen Sargeant, Northern Prairie Wildlife Research Center, Jamestown, North Dakota; glen_sargeant@usgs.gov Peter Singleton, USDA Forest Service, Pacific Northwest Research Station in Wenatchee, WA; singlep@u.washington.edu Marcó Smolla, Max Planck Institute for Ornithology, Dept. Migration and Immuno-ecology, Am Obstberg 1, 78315 Radolfzell, Germany; msmolla@orn.mpg.de Tyler Wagner, US Geological Survey, Pennsylvania Cooperative Fish and Wildlife Research Unit, 402 Forest Resources Bldg., Pennsylvania State University, University Park, PA 16802 ; twagner@psu.edu

Chapter 1

Introduction

The original Manual of Applied Spatial Ecology (hereafter referred to as *Manual*) went online in 2014 with a 173 page manual created in Latex and scripts/datasets available on GitHub a short time thereafter. Since the scripts were .R files, they were then converted into .Rmd files in attempts to create stand-alone pdfs for each chapter. Latex of the entire compilation was too time consuming to maintain so .Rmds for each exercise were an early solution. The resulting chapter pdfs were created with every update and version of each exercise pushed to GitHub to easily update code or package changes. Although the manual pdf was not edited or changed since 2016, the pdfs within each exercise on GitHub were updated each year.

Fast forward to 2023, I was challenged with whether it was even worth maintaining this *Manual* considering so many R packages and code were required to be published with most manuscripts. In addition, with the increase in spatial data available and R code to prepare spatial data (some within my own lab), several chapters in the *Manual* seem outdated with no reason to maintain these chapters (Chapter 2 for example). Combine that with deprecating of packages rgdal and raster for their equivalents, sf and terra, respectively, most exercises in the *Manual* would no longer be functional after December 2023.

The impetus for creating this manual in 2014, however, was a place to store code and revisit functions and code I created or found online in one place to use on new projects. GitHub provides a nice portal to store and organize code that might not be published and for my exercises to be available for graduate students in my course at The Pennsylvania State University - Applied Spatial Ecology in R. Even though I have countless scripts and functions that are not available in the *Manual*, maintaining R code (GitHub) and a pdf manual (bookdown or similar) has never been easier. With this in mind, the link on my lab webpage will stay as an archive as long as the university allows it, but the new manual and all updates will move to my GitHub page walterASEL under a new repository.

Chapter 2

Data Manipulation and Management

2.1 Transformations between coordinate systems

Transformations in ArcMap can be the most troublesome component of spatial analysis that is often overlooked as the reason for errors in data analysis. We will briefly go into the 2 most common problems requiring our assistance from collaborators and potential solutions.

1. What coordinate system were the data collected in?

It seems that every GPS collar, handheld GPS unit, GIS landcover layer, etc. has been created using a different coordinate system and it's not the one you have at your study site. Or perhaps NAD 1927 was used and you decided to be modern and want to use NAD 1983. Regardless, the coordinate systems must match even though ArcMap often overlays them with "on the fly projections". The "on the fly" component of ArcMap is great for visualization but not for spatial ecologists that need data analysis. We often can determine which coordinate system the data were created in using the metadata to define a coordinate system or project the data into a coordinate system for data analysis.

2. A Toolbox in ArcMap may not extract data or clip data properly

As mentioned previously, data collected with a GPS collar or handheld GPS may not be in the same geographic or projected coordinate system as the GIS layers you download or receive from collaborators (i.e., Digital Elevation Data, National Land Cover Data). As you attempt to use a Toolbox function, such as clipping National Land Cover Data within the extent of your GPS locations, an error may result.

We will now explore some transformations of data in R to help understand what Projections and Transformations are all about. The dataset that follows is for a project in Colorado with mule deer equipped with GPS collars that collected locations every 3 hours. The purpose of the study was to determine mule deer use of agricultural crops, sunflowers in this case, in response to years of damage complaints from farmers. We will use this subset of dataset in later exercises as well.

1. Exercise 1.4 - 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 “MDprojections.Rmd” and run code directly from the script

```
#setwd("Exercise_1.4_Projections")
```

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

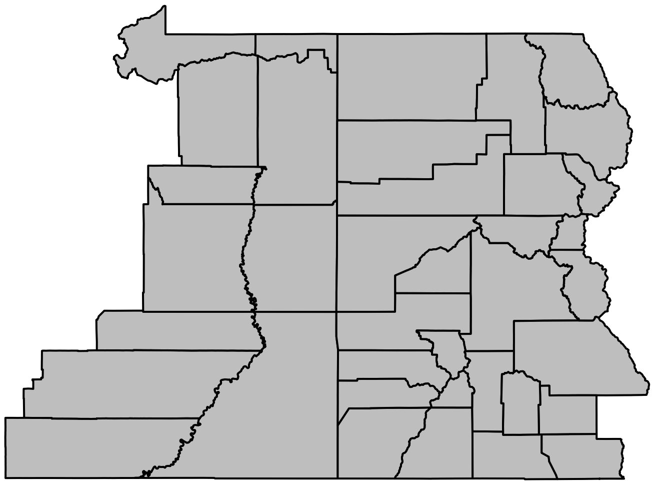
```
library(rgdal)
```

5. Now let’s 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

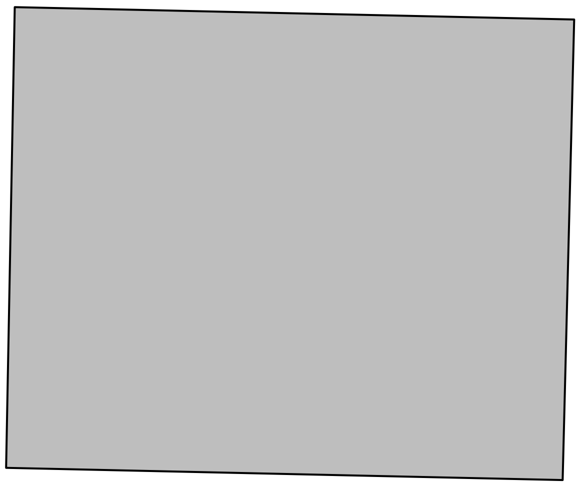
```
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
new.crs <-CRS("+proj=utm +zone=12 +datum=WGS84")
```

6. Read in some shapefiles of the study area

```
study.states<-readOGR(dsn=".",layer="MDcounties",verbose=FALSE)
plot(study.states, col="grey")
```



```
#Let's zoom into the region we have locations instead of county level
study.zoom<-readOGR(dsn=".",layer="MDzoom", verbose=FALSE)
plot(study.zoom, col="grey")
```

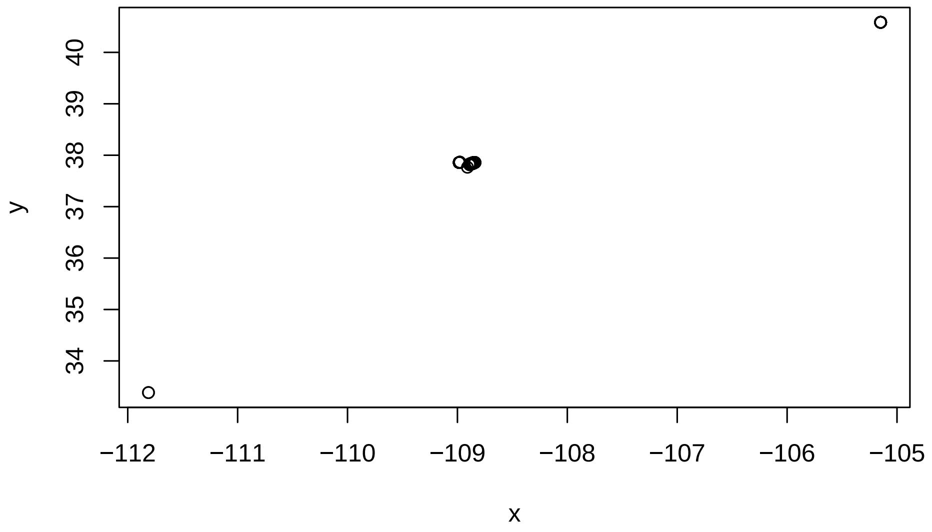


7. Import the csv file that contains all the mule deer locations by ID

```
muleys <-read.csv("muleysexample.csv",header=T)
```

8. Create a spatial data frame of raw mule deer locations with projection defined similar to study site shapefile (i.e., WGS84)

```
coords<-data.frame(x = muleys$Long, y = muleys$Lat)
plot(coords)
```



9. Remove outlier locations

```
newmuleys <-subset(muleys, muleys$Long > -110.50 & muleys$Lat > 37.3
   & muleys$Long < -107)
muleys <- newmuleys
```

10. Create a new spatial data frame of mule deer locations with outliers removed and projection defined similar to study site shapefile (i.e., WGS84)

```
coords<-data.frame(x = muleys$Long, y = muleys$Lat)
plot(coords)
```

11. Create a spatial points data frame of mule deer locations with projection defined similar to study site shapefile (i.e., WGS84)

```
deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys, proj4string = CRS(c
coordinates(deer.spdf)[1:5,]
class(deer.spdf)
proj4string(deer.spdf)
plot(deer.spdf)
points(deer.spdf, col="yellow")
```

12. Now let's project both the mule deer locations and study site shapefile to NAD83 UTM Zone 12 (Fig. 1.4, 1.5)

```
MDzoomUTM12 <-spTransform(study.zoom, CRS=new.crs)
plot(MDzoomUTM12, col="bisque")
class(MDzoomUTM12)
```

```
proj4string(MDzoomUTM12)

#projection for mule deer locations
deerUTM12 <-spTransform(deer.spdf, CRS=new.crs)
points(deerUTM12, col="red")
class(deerUTM12)
proj4string(deerUTM12)

#See new projected coordinates in UTM 12N for the first 5 locations compared to similar line of c
coordinates(deerUTM12)[1:5,]
```

2.2 Import and Format Datasets

1. Exercise 1.5 - 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 "TimeLagScript.Rmd" and run code directly from the script

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

```
library(chron)
library(rgdal)
#library(RAtmosphere)#This package has not been updated for newer versions of R
```

5. Determine the name of your file ("temp" in our case here). We can then enter the path with this name to bring our dataset into R

```
temp <- read.csv("Y2005_UTM_date.csv", header=T)
```

6. It is often necessary to determine the time lag between successive locations within your dataset

```
# Modify time to include seconds
temp$time <- paste(as.character(temp$LMT_TIME),"00",sep=":")

# Convert to chron date
temp$date_time <- chron(as.character(temp$LMT_DATE),temp$time,format=c(dates="m/d/y",times="h:m:s

# Calculate difference in time in minutes
timediff <- diff(temp$date_time)*24*60
summary(timediff)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
##    7.00000   30.00000   30.00000   72.01485   60.00000 7859.00000
```

```
# Remove first entry without any difference
temp <- temp[-1,]

# Assign timediff column to original "temp" dataset for use later
temp$timediff <- as.numeric(timediff)
```

7. We can then either export this file as an excel file for use in other programs or use it in R in subsequent analysis

```
#write.csv(temp, "TimeDiffdata.csv", row.names=TRUE,sep=" ", col.names=TRUE, quote=TRU
```

8. Next we can add code below to include night and day into datasets and to also to account for daylight savings. Package RAtmosphere will eliminate the need for the chunk of code below if working on an earlier version of R (i.e., code not available for R 3.2.1 plus). Thanks to Duane Diefenbach, PA Coop Unit Leader, for compiling all this from online sources.

We may first need to create a SPDF and transform to Lat Long then return to a Data Frame You only need this section of code if you need to acquire Lat Long coordinates for dataset

```
utm.crs <-CRS("+proj=utm +zone=12 +datum=WGS84")
dataSPDF<-data.frame(x = temp$UTMe, y = temp$UTMn)
utm.spdf <- SpatialPointsDataFrame(coords = dataSPDF, data = temp, proj4string = utm.c

ll.crs <- CRS("+proj=longlat +ellps=WGS84")
datall <-spTransform(utm.spdf, CRS=ll.crs)
```

```
## Warning: PROJ support is provided by the sf and terra packages among others
```

```
temp <- as.data.frame(datall)
```

9. Separate times into categories "Day" and "Night" based on sunrise-sunset table by running function below or simply using the RAtmosphere package

10. First run line of code below with d being the day of year, Lat is latitude in decimal degrees, and Long is longitude in decimal degrees (negative == West) available at suncalc suncalc This method is copied from: Teets, D.A. 2003. Predicting sunrise and sunset times. The College Mathematics Journal 34(4):317-321.

At the default location the estimates of sunrise and sunset are within seven minutes of the correct times (http://aa.usno.navy.mil/data/docs/RS_OneYear.php) with a mean of 2.4 minutes error.

NOTE: Function is in package RAtmosphere but does not work in newer version of R along with other issues!

```
suncalc <- function(d,Lat=39.14133,Long=-106.7722){

  ## Function to convert degrees to radians
  rad <- function(x)pi*x/180

  ##Radius of the earth (km)
  R=6378

  ##Radians between the xy-plane and the ecliptic plane
  epsilon=rad(23.45)

  ##Convert observer's latitude to radians
  L=rad(Lat)

  ## Calculate offset of sunrise based on longitude (min)
  ## If Long is negative, then the mod represents degrees West of
  ## a standard time meridian, so timing of sunrise and sunset should
  ## be made later.
  ##NOTE: If working with UTC times use timezone = -4*(abs(Long)%%15)*sign(Long)
  timezone = -6*(abs(Long)%%15)*sign(Long)

  ## The earth's mean distance from the sun (km)
  r = 149598000

  theta = 2*pi/365.25*(d-80)

  z.s = r*sin(theta)*sin(epsilon)
  r.p = sqrt(r^2-z.s^2)

  t0 = 1440/(2*pi)*acos((R-z.s*sin(L))/(r.p*cos(L)))

  ##a kludge adjustment for the radius of the sun
  that = t0+5

  ## Adjust "noon" for the fact that the earth's orbit is not circular:
  n = 720-10*sin(4*pi*(d-80)/365.25)+8*sin(2*pi*d/365.25)

  ## now sunrise and after sunset are:
  sunrise = (n-that+timezone)/60
  sunset = (n+that+timezone)/60
  suntime <- cbind(sunrise,sunset)

  return(suntime)
}
```

11. Read in location data and retain lat, lon, and date

```
temp$Date <- paste((temp$Year),substr(temp$date_time, 2,3),substr(temp$date_time, 5,6)

#calculate calendar day and center of locations
calday <- as.numeric(as.Date(temp$Date)-as.Date("2005-01-01"), units="days")
dat1 <- cbind(temp,calday)
moda <- format(as.Date(temp$Date),"%d-%b")

dat1 <- cbind(dat1, suncalc(dat1$calday, Lat=dat1$LATITUDE, Long=dat1$LONGITUDE),moda)
hrchar <- as.character(substr(dat1$time,1,2))
hr <- as.numeric(as.character(substr(dat1$time,1,2)))
minchar <- as.character(substr(dat1$time,4,5))
min <- as.numeric(minchar)
localhr <- hr+min/60
dat1 <- cbind(dat1,hr,hrchar,minchar,localhr)
Diel <- ifelse(localhr<dat1$sunrise | localhr>dat1$sunset, 'Night', 'Day')
dat1 <- cbind(dat1,Diel)
```

```
dat1[15:50,]
```

2.3 Manipulate Polygon Layer

1. Exercise 1.6 - Download and extract zip folder into your preferred location

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

```
#setwd("Exercise_1.6_ManipulatePolygonLayer")
```

3. Now open the script “SoilScript.Rmd” and run code directly from the script

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

```
library(rgdal)
library(maptools)
library(foreign)
library(here)
```

```
soils<-readOGR(dsn=".",layer="Soil_Properties",verbose=FALSE)
plot(soils)
```

```
names(soils) #get attribute data
```

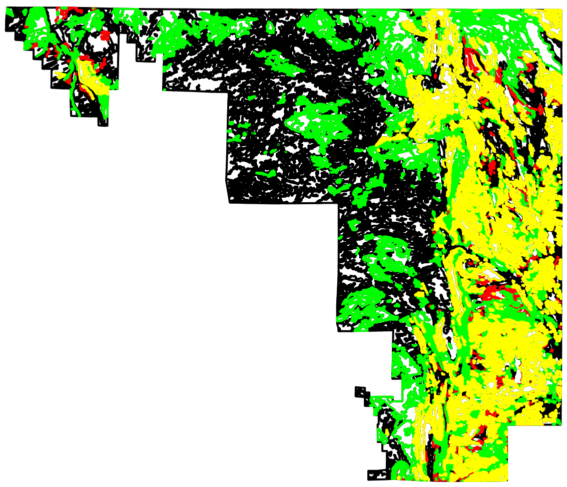
```
##  [1] "Join_Count" "TARGET_FID" "Join_Cou_1" "TARGET_F_1" "AREASYMBOL"
##  [6] "SPATIALVER" "MUSYM"      "MUKEY"      "SdvOutpu_1" "SdvOutpu_2"
## [11] "SdvOutpu_3"
```

```
#Rename original category headings to something more familiar
soils$Clay <- soils$SdvOutpu_1
soils$pH <- soils$SdvOutpu_2
soils$CEC <- soils$SdvOutpu_3
```

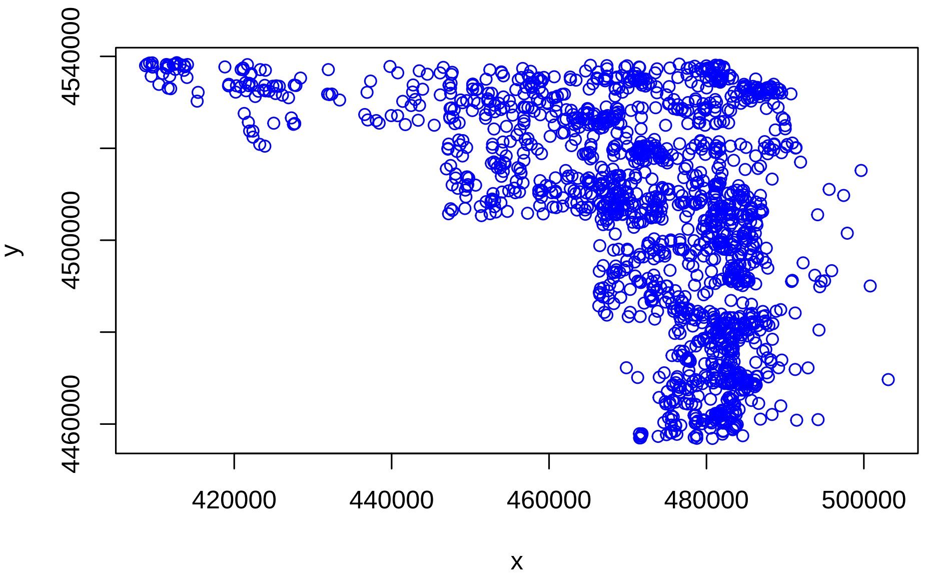
```
#Shapefiles contain several slots which can be called with the "@" symbol or slot(object, "data")
soils@data[1:10,] #a data frame with n observations associated with X covariates,
#soils@polygons #the number of polygons that the shapefile consists of
soils@plotOrder #the order of the polygons
soils@bbox #boundary box
soils@proj4string #projection
#Within the slot
soils@polygons[[1]] #will bring up the first polygon
soils@polygons[[1]]@area #will bring up the area for the first polygon
soils@polygons[[1]]@ID #will retrieve the ID of the first polygon
soils@polygons[[1]]@plotOrder #will retrieve the order of the first polygon
```

5. Select portions of the data that fit some set criteria

```
#Highlights the areas that Percent Clay polygons are over 30%
plot(soils)
high.clay<- soils[soils$Clay>30,]
plot(high.clay, border="red", add=TRUE)

##Highlights the areas that Cation Exchange Capacity is greater than 14
high.CEC<- soils[soils$CEC>14,]
plot(high.CEC, border="green", add=TRUE)

##Highlights the areas that soil pH is greater than 8
high.pH <- soils[soils$pH>8,]
plot(high.pH, border="yellow", add=TRUE)
```


6. Bring in locations of harvested mule deer

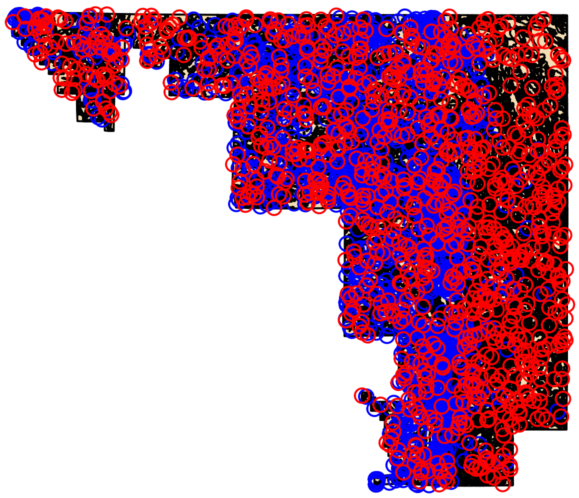
```
#Import mule deer locations from harvested animals tested for CWD
#Note: Locations have been offset or altered so do not reflect actual locations of sam
mule <- read.csv("MDjitterclip.csv", header=T)

coords<-data.frame(x = mule$x, y = mule$y)
crs<-"+proj=utm +zone=13 +datum=WGS84 +no_defs +towgs84=0,0,0"

plot(coords, col="blue")
```



```
#par(new=TRUE)
```

7. Let's generate random points with the extent of the soil layer

```
#Sampling points in a Spatial Object "type=regular" will give a regular grid
samples<-spsample(soils, n=1000, type="random")

plot(soils, col="wheat")
points(coords, col="blue")
points(samples, col="red")
```



8. Creates a SpatialPoints object from the location coordinates

```
samples@bbox <- soils@bbox
samples@proj4string <- soils@proj4string
```

9. Extract and tally Clay soil types for random samples and mule deer locations

```
#Matches points with polygons:
soils.idx<- over(samples,soils)
locs <- SpatialPoints(coords)
locs@proj4string <- soils@proj4string
soils.locs<- over(locs, soils)

#Tally clay soil types for random samples
obs.tbl <- table(soils.idx$Clay[soils.idx$Clay])
obs.tbl
```

```
##
##    0  8.5 11.5 11.7 12.6 12.8 13.2 16.5 19.3 20.1 20.2 20.5 20.7   21 25.8 25.9
##   10  126   39   48   32   49   41  138   14   66   64    6  169    1   58    5
## 26.2 26.7 36.8 37.2 40.6
##   18    5    2   20   32
```

```
#Also tally soil types for each mule deer sampled
obs.tbl2 <- table(soils.locs$Clay[soils.locs$Clay])
obs.tbl2
```

```
##
##    0  8.7  9.7 11.5 11.7 12.1 12.6   13 13.1 16.5 18.5   21   40
##   71   31  670  237   96    1    8   89    2  236   12   30   31
```

10. Converts the counts to proportions

```
obs <- obs.tbl/sum(obs.tbl)
obs
```

```
##
##           0         8.5        11.5        11.7        12.6        12.8
## 0.010604454 0.133616119 0.041357370 0.050901379 0.033934252 0.051961824
##        13.2        16.5        19.3        20.1        20.2        20.5
## 0.043478261 0.146341463 0.014846235 0.069989396 0.067868505 0.006362672
##        20.7          21        25.8        25.9        26.2        26.7
## 0.179215270 0.001060445 0.061505832 0.005302227 0.019088017 0.005302227
##        36.8        37.2        40.6
## 0.002120891 0.021208908 0.033934252
```

```
obs2 <- obs.tbl2/sum(obs.tbl2)
obs2
```

```
##
##           0         8.7         9.7        11.5        11.7        12.1
## 0.046895641 0.020475561 0.442536328 0.156538970 0.063408190 0.000660502
##        12.6          13        13.1        16.5        18.5          21
## 0.005284016 0.058784676 0.001321004 0.155878468 0.007926024 0.019815059
##          40
## 0.020475561
```

2.4 Manipulate Raster Data Layer

1. Exercise 1.7 - 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 "RasterScript.R" in that folder and run code directly from the script. Create an Ascii file from your raster grid in ArcMap 10.X if experimenting with your own raster using the following Toolbox in ArcMap 10.X:

ArcToolbox - Conversion Tools - From Raster - Raster to Ascii

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

```
#install.packages(c("adehabitatHR","maptools","raster", "rgdal"))

library(adehabitatHR)
library(raster)
library(rgdal)
library(maptools)
```

5. Now let's 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

```
#CRS of shapefile layers
crs <- CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0
  +y_0=0 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
#CRS of raster layers
crs2 <- CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0
  +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
utm.crs<-CRS("+proj=utm +zone=17 +ellps=WGS84")
```

6. Alternatively, you can set your working directory by opening a previously saved workspace by double clicking it in that folder which will also serve to set that folder as the working directory. All of the raster layers we are going to use will be located here

Note: Most of the exercises that follow are exploratory in nature and not the recommended way to bring raster data into R. I include these exercises only for those that may perhaps only have alternate rasters available (e.g., ASCII). I would recommend using .tif and not ASCII or .txt files as in the first few exercises below.

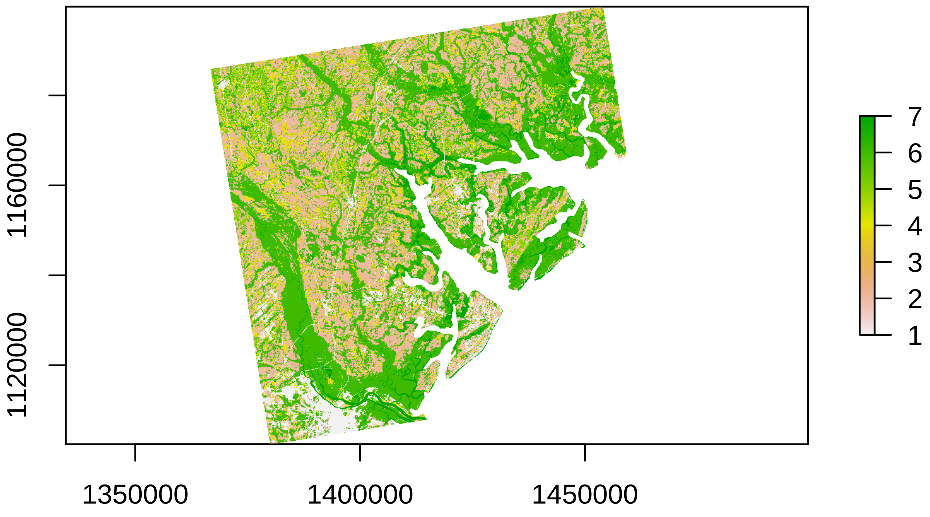
7. If you have troubles getting a raster to Ascii in ArcMap to actually show up as an Ascii file there is good reason. We need to rename the text file by replacing ".txt" with ".asc" in Windows Explorer. ArcMap will not save as an ".asc" file and I have no idea why!

8. Here is some code to import Ascii files (i.e., rasters) from ArcMap into R using one of several packages. Ascii files can be categorical (Vegetation/Habitat categories) or numeric (DEMs). Because adehabitat package has been discontinued, we will skip this section and move on to step 9.

```
#Import raster as text using the "raster" package
r <-raster("polyascii2.txt")
proj4string(r)#Note: No projection information was imported with the raster
```

```
## [1] NA
```

```
plot(r)
```



```r
#Assign projection information for imported text file
proj4string(r) <-CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0
  +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
r
```

```
## class      : RasterLayer
## dimensions : 3245, 3353, 10880485  (nrow, ncol, ncell)
## resolution : 30, 30  (x, y)
## extent     : 1366773, 1467363, 1102414, 1199764  (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +ellps=GRS8C
## source     : polyascii2.txt
## names      : polyascii2
```

```r
proj4string(r)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +ellps=GRS80 +units=
```

```r
#Import raster as an Ascii file (factor) using "adehabitat" package if available for
#file1, file2, and file3 in the 3 sections of code below:
#Path of the file to be imported
# file1 <-  paste("polyextract2.asc", sep = "\\")
# levelfile <- paste("TableExtract.txt", sep = "\\")
# asp <- import.asc(file1, lev = levelfile, type = "factor")
# image(asp)
# asp
#
# #Now let's look at the vegetation categories of the file
# ta <- table(as.vector(asp))
# names(ta) <- levels(asp)[as.numeric(names(ta))]
# ta
#
# file2 <-  paste("polyclip.asc", sep = "\\")
# levelfile2 <- paste("TableClip.txt", sep = "\\")
# asp2 <- import.asc(file2, lev = levelfile2, type = "factor")
# image(asp2)
# asp2
#
# #Shows 7 vegetation categories
#
# #Now let's look at the vegetation categories of the file
# ta2 <- table(as.vector(asp2))
# names(ta2) <- levels(asp2)[as.numeric(names(ta2))]
# ta2
```

```
#NOTE: R won't recognize double digit veg categories
#Import raster as an Ascii files (factor) using:
#Path of the file to be imported
# file3 <-  paste("polyascii2.asc")
# levelfile3 <- paste("TableCode.txt")
# asp3 <- import.asc(file3, lev = levelfile3, type = "factor")
# image(asp3)
# asp3

#Now let's look at the vegetation categories of the file
# ta3 <- table(as.vector(asp3))
# names(ta3) <- levels(asp3)[as.numeric(names(ta3))]
# ta3

#Or we can also use the "adehabitat" package to import DEM
# fileElev <-  paste("demascii.asc")
# elev <- import.asc(fileElev)
# image(elev)
# elev
```

9. We will primarily be using the “rgdal” package to import an ascii grid as a Spatial Grid Data Frame

```
habitat <- readGDAL("polyascii2.asc")
```

```
## Warning: GDAL support is provided by the sf and terra packages among others
```

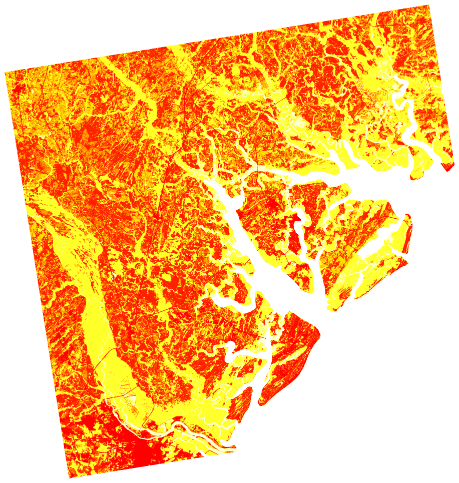
```
## polyascii2.asc has GDAL driver AAIGrid
## and has 3245 rows and 3353 columns
```

```
## Warning: GDAL support is provided by the sf and terra packages among others
```

```
proj4string(habitat)
```

```
## [1] NA
```

```
proj4string(habitat) <-CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_
  +y_0=0 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
image(habitat)
```



10. Now let's add some shapefiles to our raster

```r
#Load county shapefile
county<-readOGR(dsn=".",layer="BeaufortCoAlbers", verbose = FALSE)
proj4string(county)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NAD83 +units=
```

```r
#Now let's make county a SpatialPolygon class to simply remove data contained within it
polys <- as(county, "SpatialPolygons")
plot(polys,lwd=2)
text(coordinates(polys), labels="Beaufort")

#Load airport runway shapefile
run<-readOGR(dsn=".",layer="RunwayAlbers", verbose = FALSE)
proj4string(run)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NAD83 +units=
```

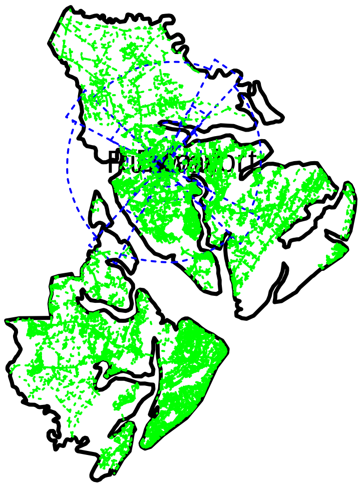
```r
polys2 <- as(run, "SpatialPolygons")
plot(polys2,add=T,lwd=2)
text(coordinates(polys2), labels="Runway")

#Load aircraft flight pattern shapefile
path<-readOGR(dsn=".",layer="FlightImage", verbose = FALSE)
proj4string(path)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NAD83 +units=
```

```r
polys3 <- as(path, "SpatialLines")
plot(polys3,add=T,lty="32", col="blue")

#Load aircraft flight pattern shapefile
road<-readOGR(dsn=".",layer="CountyRoadAlbers", verbose = FALSE)
proj4string(road)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NA
```

```r
polys4 <- as(road, "SpatialLines")
plot(polys4,add=T,lty="22", col="green")
```

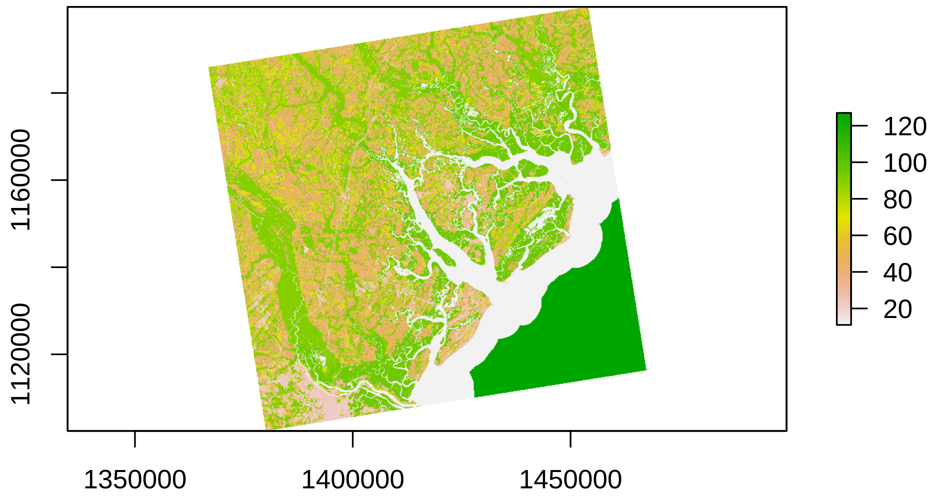


11. Plot out all the shapefiles overlayed on each other with and without the raster.

```r
plot(county)
plot(road, add=T)
plot(run, col="red",add=T)
plot(path, col="blue",add=T)
```

12. Let's reclassify layer to get fewer vegetation categories to make it easier to work with the raster.

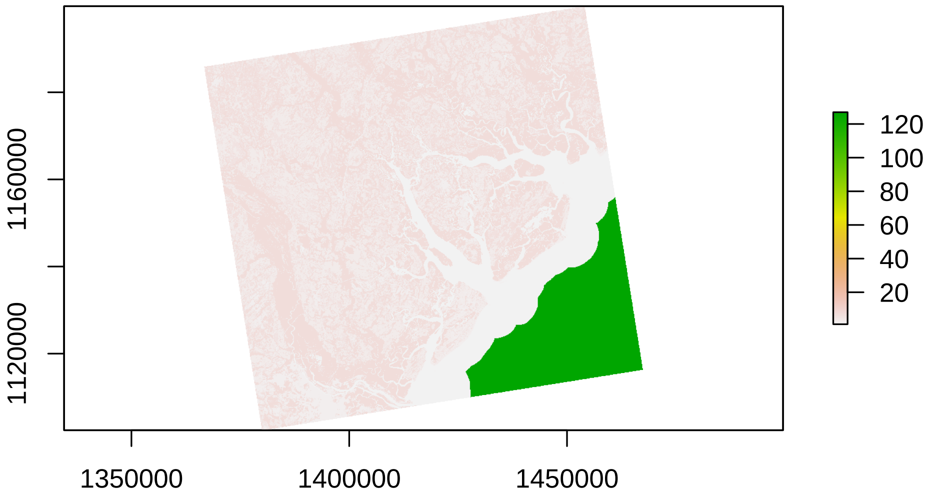
```r
veg <-raster("polydouble.txt")
plot(veg)
```



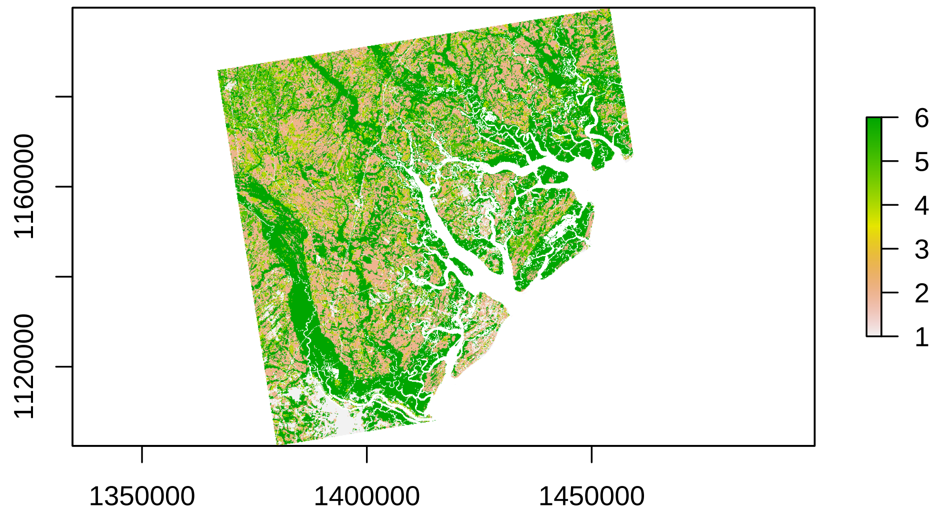
```r
veg
```

```
## class       : RasterLayer
## dimensions  : 3245, 3353, 10880485  (nrow, ncol, ncell)
## resolution  : 30, 30  (x, y)
## extent      : 1366773, 1467363, 1102414, 1199764  (xmin, xmax, ymin, ymax)
## crs         : NA
## source      : polydouble.txt
## names      : polydouble
```

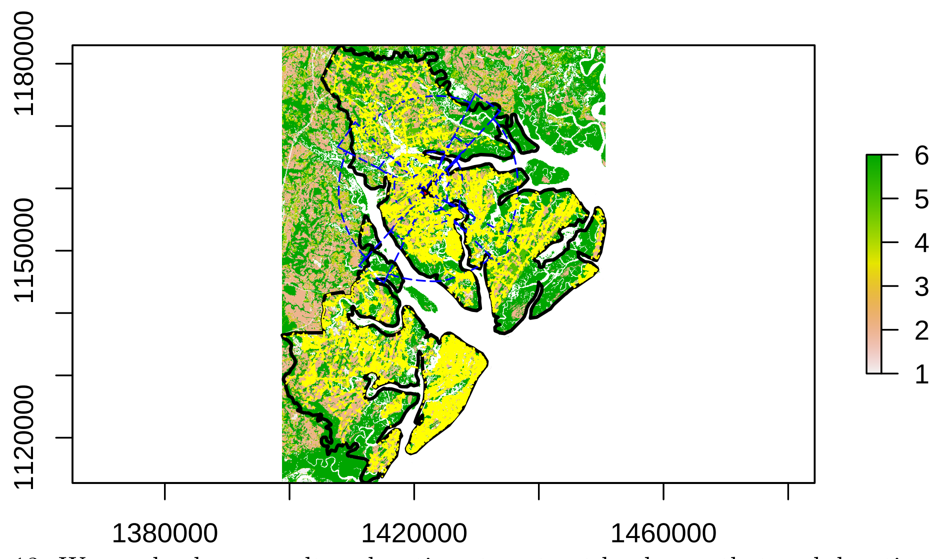
```
# reclassify the values into 7 groups
# all values between 0 and 20 equal 1, etc.
m <- c(0, 19, 1, 20, 39, 2, 40, 50, 3, 51, 68, 4, 69,79, 5, 80, 88, 6, 89, 99, 7)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(veg, rclmat)
plot(rc)
```



```
#Now, let's remove water that is coded 11 and No Data that is coded as 127
m1 <- c(0, 19, NA, 20, 39, 1, 40, 50, 2, 51,68, 3, 69, 79, 4, 80, 88, 5, 89, 99, 6, 10
rclmat1 <- matrix(m1, ncol=3, byrow=TRUE)
rc1 <- reclassify(veg, rclmat1)
plot(rc1)
```



```
#Clip the raster within the county polygon for a zoomed in view then plot
clip <- crop(rc1, polys)
plot(clip)
plot(polys,add=T,lwd=2)
plot(polys2,add=T,lwd=2, col="red")
plot(polys3,add=T,lty="62", col="blue")
plot(polys4,add=T,lty="22", col="yellow")
```



13. We can load some vulture locations to extract landcover that each location occurs in that will be considered "used" habitat in resource selection analysis.

```
#Import bird 49 locations to R
bv49 <-read.csv("Bird49.csv", header=T)#How many bird locations?

#Make a spatial points data frame of locations and convert to Albers
coords<-data.frame(x = bv49$x, y = bv49$y)
bvspdf <- SpatialPointsDataFrame(coords= coords, data = bv49, proj4string = utm.crs)
#utm.crs<-"+proj=utm +zone=17 +ellps=WGS84" #NOTE: If using a mac, remove the N in zon
plot(bvspdf, col="red")
#NOTE: Locations must be assigned to the UTM coordinate system prior to projection to
#so won't overlay on veg layer at this point because veg is in Albers
bv49Albers <-spTransform(bvspdf, CRS=crs2)
```

```
## Warning: PROJ support is provided by the sf and terra packages among others
```

```
class(bv49Albers)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

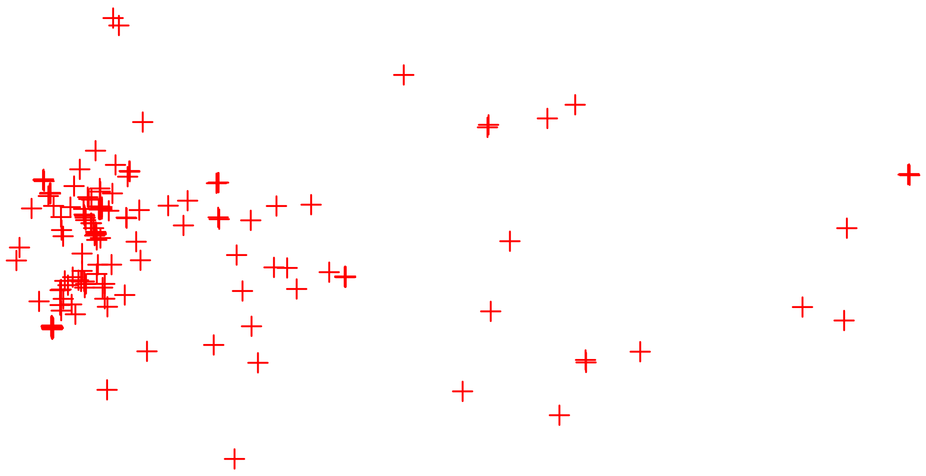
```
proj4string(bv49Albers)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +ellps=GR
```

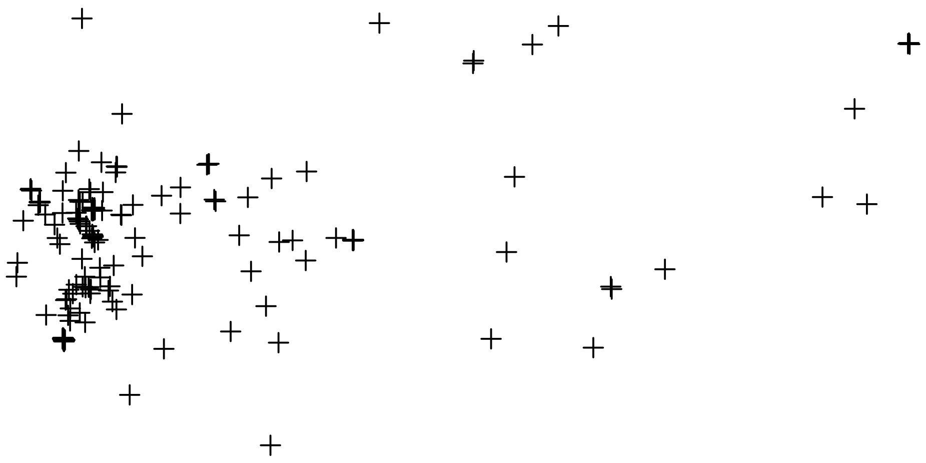
```
bv49Albers[1:5,]
```

```
## class       : SpatialPointsDataFrame
## features    : 5
## extent      : 1415830, 1418301, 1152353, 1156708  (xmin, xmax, ymin, ymax)
## crs         : +proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0
## variables   : 4
## names       :     Date, id,      x,       y
## min values  : 20061001, 49, 519067, 3587085
## max values  : 20061030, 49, 521821, 3591245
```

```
points(bv49Albers, col="red")
```



```
#Determine which of those points lie within a cell that contains data by using the extract functi
#The extract function will extract covariate information from the raster at a particular the xy c
veg.survey<-extract(clip, bv49Albers)
veg.survey<-subset(bv49Albers,!is.na(veg.survey))
plot(veg.survey, col="black")
```

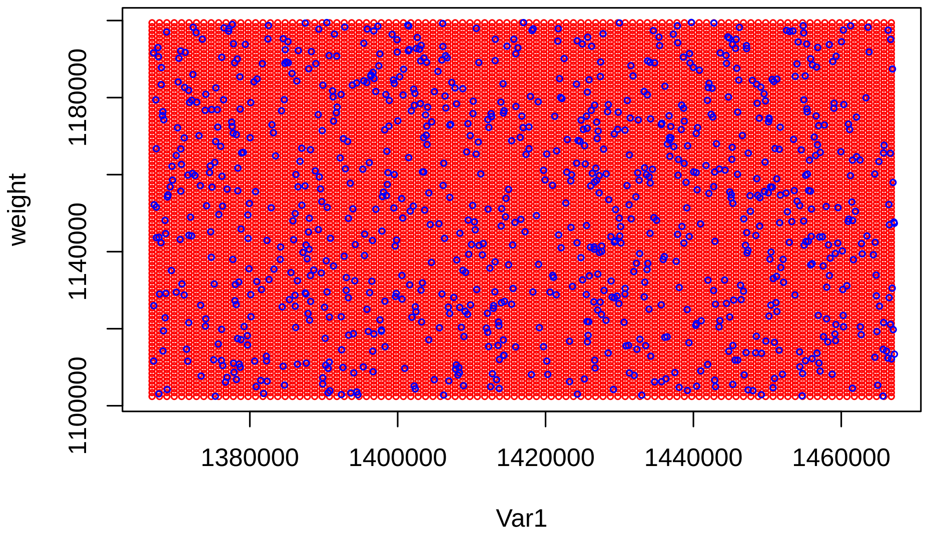


14. We can also create some random points within the extent of the area to be considered as “available” habitat.

```
##First we need to create a grid across the study site with sample points
Sample.points<-expand.grid(seq(veg@extent@xmin, veg@extent@xmax, by=1000),
  weight = seq(veg@extent@ymin, veg@extent@ymax, by=1000))
plot(Sample.points, bg="red", cex=.5,col="red")

#Now create some random points using the minimum and maximum coordinates of the raster
#determine the range of points from which to select x and y
x.pts<-sample(seq(veg@extent@xmin, veg@extent@xmax, by=10),1000) ##generate x coordina
y.pts<-sample(seq(veg@extent@ymin, veg@extent@ymax, by=10),1000)

#Now create a spatial points file from the randomly generated points
coords2<-data.frame(x = x.pts, y = y.pts)
head(coords2)
```

```
##         x       y
## 1 1409453 1123744
## 2 1373993 1120684
## 3 1413203 1195134
## 4 1373573 1195154
## 5 1377793 1111034
## 6 1398073 1155424
```

```
points(coords2, bg="red", cex=.5,col="blue")
```



```
#Determine which of those points lie within a cell that contains data by using the extract
#function agin.
veg.sample<-extract(rc1, coords2)
head(veg.sample)
```

```
## [1]  2 NA NA NA NA  6
```

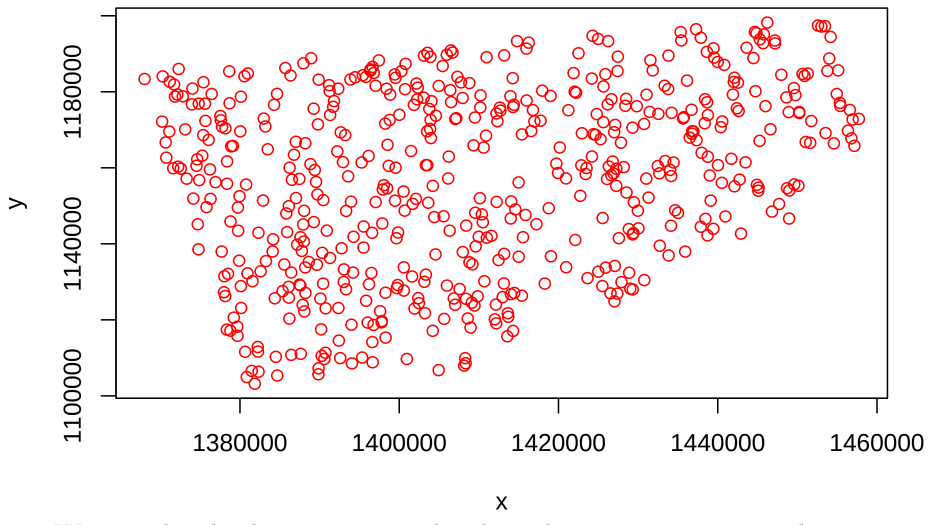
```
veg.sample<-subset(coords2,!is.na(veg.sample))
str(veg.sample)
```

```
## 'data.frame':    553 obs. of  2 variables:
##  $ x: num  1409453 1398073 1399703 1408823 1414253 ...
##  $ y: num  1123744 1155424 1128354 1135134 1183584 ...
```

```
head(veg.sample)
```

```
##          x       y
## 1  1409453 1123744
## 6  1398073 1155424
## 7  1399703 1128354
## 9  1408823 1135134
## 11 1414253 1183584
## 14 1380943 1132234
```

```
plot(veg.sample,col="red")
```



15. We can also do the same using the clipped vegetation raster to be more in

line with vulture locations or using the reclassified vegetation categories. For each locations, we can determine if a locations lies within a cell that contains data by using the extract function and this will extract covariate information from the raster at a each location.

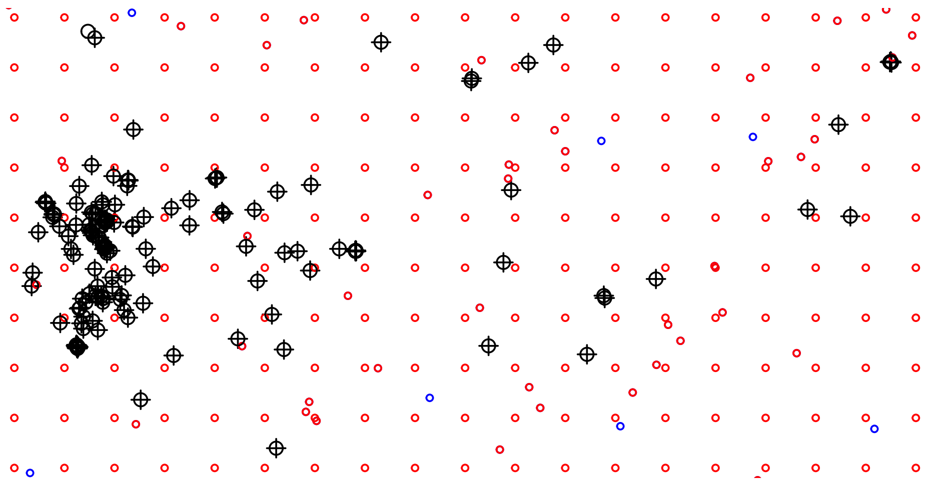
```
clip.survey<-extract(clip, bv49Albers)
clip.survey
```

```
##   [1]  1  2  2  6  6  6  6  6  6  6  6  6  6  6  6  6  6  4  6  6  6  6  2  6  6
##  [26]  6  6  6  6  6  6  6  6  2  2  6  6  6  4  2  6  6  6  6  6  6  6  6  6  2
##  [51]  6  6  6  2  2  6  6  6  6  1  6  6  6  6  6  6  6  6  6  6  6  6  6  6  2
##  [76]  6  6  6  6  6  6  6  6  3  6  6  2  2  6  6  6  6  6  6  3  6  2  2  6  6
## [101]  6  6  6  6  6  6  6  6  2  2  4  2  3  6  6  6  6  4  2  2  2  6  6  6  6
## [126]  1  2  6  1  2  6  6  6  6  6  6  6  6  6  6  6  4  3  2  6  2  6  6  2  4
## [151]  6  6  1  1  5  4  2  3  3  2  2  2  1  2  2  2  1  1  6  1  2  1  2  2  2
## [176]  2  2  6  2  6  2  6  2  1  2  2  2  2  2  1  1  4  4  4  4  4  4  1  4  6
## [201]  2  4  1  4  4  4  5  4  2 NA  4  3  3  6  6  6  2  6  2  2  5  2  4  5  2
## [226]  3  6  6  6  6  6  6  2  2  6  4  6  4  1  1  1  2  1  6  6  4  5  4  4  2
## [251]  2  2  1  2  4  4  2  4  2  2
```

```
clip.survey<-subset(bv49Albers,!is.na(clip.survey))
plot(clip.survey, col="black")

#Create a regular grid and do the same thing
Sample.points2<-expand.grid(seq(clip@extent@xmin, clip@extent@xmax, by=1500),
  weight = seq(clip@extent@ymin, clip@extent@ymax, by=1500))
points(Sample.points2, bg="red", cex=.5,col="red")

#Create random points using the minimum and maximum coordinates of the raster
x.pts2<-sample(seq(clip@extent@xmin, clip@extent@xmax, by=10),500)
y.pts2<-sample(seq(clip@extent@ymin, clip@extent@ymax, by=10),500)

#Now create a spatial points file from the randomly generated points
coords3<-data.frame(x = x.pts2, y = y.pts2)
points(coords3, bg="red", cex=.5,col="blue")

#Determine which of those points lie within a cell that contains data by using the ext
#function.
clip.sample<-extract(clip, coords3)
head(clip.sample)
```

```
## [1]  6  6 NA NA  5  6
```

```
clip.sample<-subset(coords3,!is.na(clip.sample))
points(clip.sample, cex=.5, col="red")
points(bv49Albers)
```



New code addition below (Update 1/7/2021) to use mapview package to treat plotting data layers as done with GIS software. This package enables user to zoom in and out and scroll around layer after calling a plot function

```
library(mapview)
mapview(clip)
mapview(polys) + mapview(bv49Albers, color="yellow")
mapview(polys) + bv49Albers + polys4
```

2.5 Creating a Hexagonal Polygon Grid Over a Study Area

1. Exercise 1.8 - 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 "GridScripts.Rmd" and run code directly from the script

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

```
library(rgdal)
library(rgeos)
library(raster)
```

5. Now let's 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

```
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
Albers.crs <-CRS("+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0
```

6. Also need to import several shapefiles for mule deer from Section 1.3

```
study.counties<-readOGR(dsn=".",layer="MDcounties", verbose = FALSE)
class(study.counties)#Shows class and package used
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

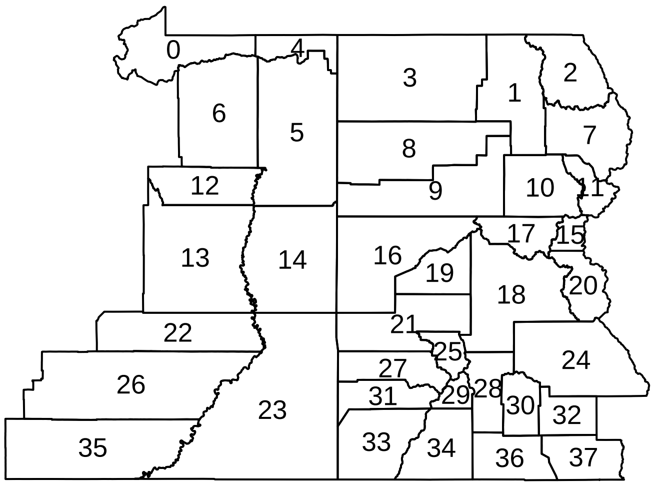
```
proj4string(study.counties)#Shows projection information
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

```
plot(study.counties)#plots study sites on map
study.counties@data$StateCO#Displays labels for counties
```

```
##  [1] "Summit County, UT"     "Routt County, CO"      "Jackson County, CO"
##  [4] "Moffat County, CO"     "Daggett County, UT"    "Uintah County, UT"
##  [7] "Duchesne County, UT"   "Grand County, CO"      "Rio Blanco County, CO"
## [10] "Garfield County, CO"   "Eagle County, CO"      "Summit County, CO"
## [13] "Carbon County, UT"     "Emery County, UT"      "Grand County, UT"
## [16] "Lake County, CO"       "Mesa County, CO"       "Pitkin County, CO"
## [19] "Gunnison County, CO"   "Delta County, CO"      "Chaffee County, CO"
## [22] "Montrose County, CO"   "Wayne County, UT"      "San Juan County, UT"
## [25] "Saguache County, CO"   "Ouray County, CO"      "Garfield County, UT"
## [28] "San Miguel County, CO" "Hinsdale County, CO"   "San Juan County, CO"
## [31] "Mineral County, CO"    "Dolores County, CO"    "Rio Grande County, CO"
## [34] "Montezuma County, CO"  "La Plata County, CO"   "Kane County, UT"
## [37] "Archuleta County, CO"  "Conejos County, CO"
```

```
#Labels each county with @plotOrder of each polygon (i.e., county)
text(coordinates(study.counties), labels=sapply(slot(study.counties, "polygons"),
  function(i) slot(i, "ID")), cex=0.8)
```



```
muleys <-read.csv("muleysexample.csv", header=T)

#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.50 & muleys$Lat > 37.8 & muleys$Long < -107)
muleys <- newmuleys
```

7. Identify the columns with coordinates then make a spatial data frame of locations after removing outliers

```
coords<-data.frame(x = muleys$Long, y = muleys$Lat)
head(coords)
```

```
##           x        y
## 1 -108.9784 37.86562
## 2 -108.9781 37.86521
## 3 -108.9794 37.86471
## 4 -108.9811 37.86176
## 5 -108.9803 37.86142
## 6 -108.9799 37.86124
```

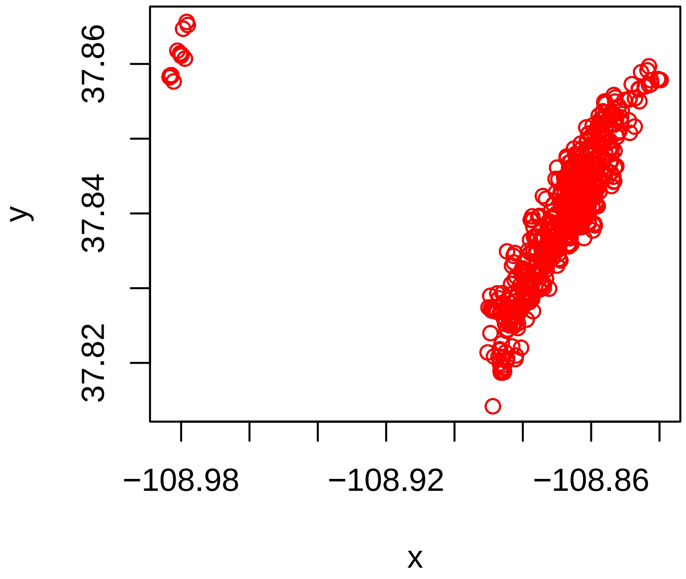
```
plot(coords)

deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys, proj4string = CRS(c
class(deer.spdf)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
proj4string(deer.spdf)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

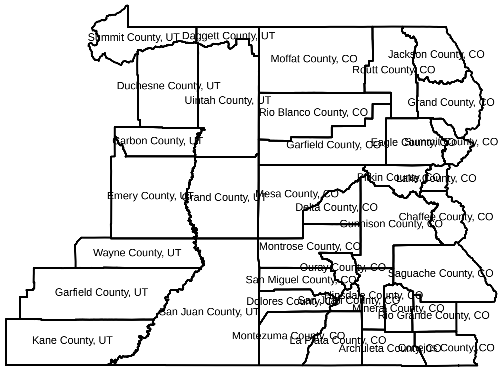
```
points(deer.spdf,col="red")
```



8. Rename labels by county name otherwise plot order would be used because

duplicate counties within each state (i.e., CO, UT) occured in original shapefile from ArcMap

```r
row.names(study.counties)<-as.character(study.counties$StateCO)
names.polygons<-sapply(study.counties@polygons, function(x) slot(x,"ID"))
#Now add labels of State and County to Map
plot(study.counties)
text(coordinates(study.counties), labels=sapply(slot(study.counties, "polygons"),
  function(i) slot(i, "ID")), cex=0.3)
```



9. Now lets extract counties within the extent of the mule deer locations

```r
int <- gIntersection(study.counties,deer.spdf)#requires rgeos library
clipped <- study.counties[int,]
MDclip <- as(clipped, "SpatialPolygons")
plot(MDclip,pch=16)
#Now add labels of State and County to Map
text(coordinates(MDclip), labels=sapply(slot(MDclip, "polygons"),
  function(i) slot(i, "ID")), cex=0.8)
```

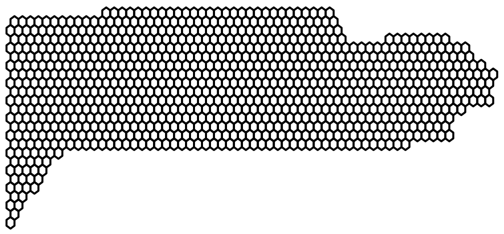
Dolores County, CO

```r
bbox(MDclip)
```

```
##          min       max
## x -109.04440 -107.8615
## y   37.47728   37.8964
```

10. We also can create a hexagonal grid across the study site

```
HexPts <-spsample(MDclip,type="hexagonal", n=1000, offset=c(0,0))
HexPols <- HexPoints2SpatialPolygons(HexPts)
#proj4string(HexPols) <- CRS(crs)
plot(HexPols)
```

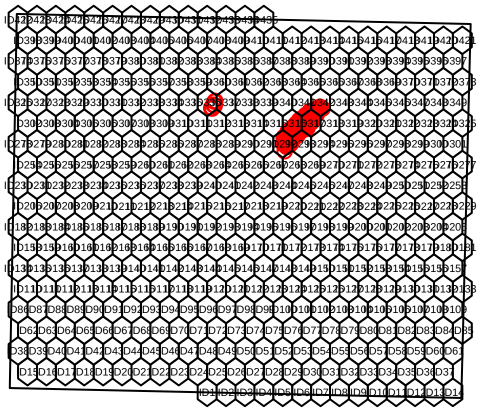


11. Create this hexagonal grid across our study site by zooming into deer locations from Section 1.3

```
#Import the study site zoomed in shapefile
study.zoom<-readOGR(dsn=".",layer="MDzoom")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/davidwalter/Library/CloudStorage/OneDrive-ThePennsylvaniaStateUnive
## with 1 features
## It has 1 fields
```

```
plot(study.zoom,pch=16)
points(deer.spdf,col="red")

#Create new hexagonal grid
HexPts2 <-spsample(study.zoom,type="hexagonal", n=500, offset=c(0,0))
HexPols2 <- HexPoints2SpatialPolygons(HexPts2)
proj4string(HexPols2) <- CRS(crs)
plot(HexPols2, add=T)
#Now add labels to each hexagon for unique ID
text(coordinates(HexPols2), labels=sapply(slot(HexPols2, "polygons"),
  function(i) slot(i, "ID")), cex=0.3)
```



12. We can intersect the mule deer locations with the polygon shapefile (i.e., county) they occured in if needed

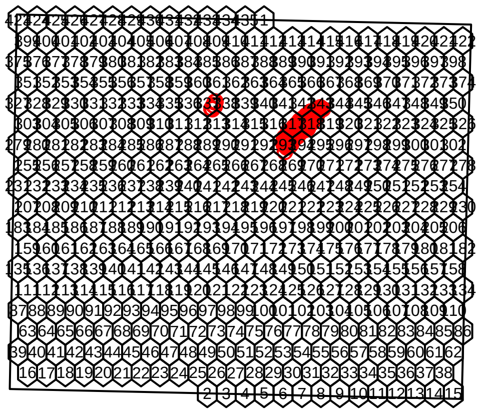
```
o = over(deer.spdf,study.counties)
new = cbind(deer.spdf@data, o)

#Used to rename labels by hexagonal grid ID only otherwise plot order with "IDxx" would be used
#and would throw an error (i.e., ID2, ID3)
row.names(HexPols2)<-as.character(HexPols2@plotOrder)
names.hex<-sapply(HexPols2@polygons, function(x) slot(x,"ID"))
```

13. As an aside, we can explore how to assign the area a location occurs in by intersecting points within the polygon shapefile.

```
o2 = over(deer.spdf,HexPols2)
new2 = cbind(deer.spdf@data,o2)
HexPols2
```

```
## class       : SpatialPolygons
## features    : 435
## extent      : -109.2448, -108.6473, 37.55422, 37.96259  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
```

```
#Now plot with new grid IDs
plot(study.zoom,pch=16)
points(deer.spdf,col="red")
plot(HexPols2, add=T)
#Now add labels of State and County to Map
text(coordinates(HexPols2), labels=sapply(slot(HexPols2, "polygons"),
  function(i) slot(i, "ID")), cex=0.5)
```



14. As an alternative to importing a polygon that we created in ArcMap, we can create a polygon in R using the coordinates of the boundary box of the area of interest. In our case here, the bounding box will be the mule deer locations.

```
proj4string(deer.spdf)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

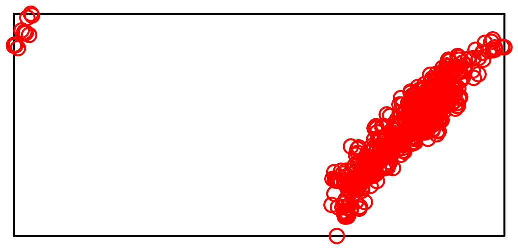
```
bbox(deer.spdf@coords)
```

```
##          min        max
## x -108.9834 -108.83966
## y   37.8142   37.86562
```

```
bb <- cbind(x=c(-108.83966,-108.83966,-108.9834,-108.9834, -108.83966),
  y=c(37.8142, 37.86562,37.86562,37.8142,37.8142))
SP <- SpatialPolygons(list(Polygons(list(Polygon(bb)),"1")), proj4string=CRS(proj4stri
plot(SP)
proj4string(SP)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

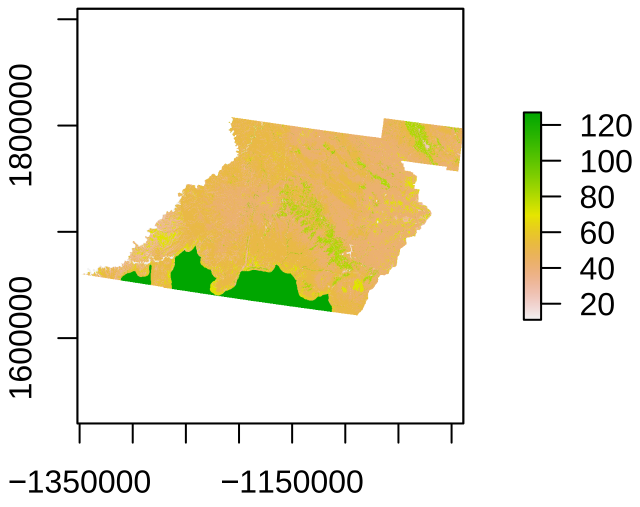
```
points(deer.spdf,col="red")
```



15. Now make practical use of the new bounding box we created by clipping a

larger raster dataset. A smaller raster dataset runs analyses faster, provides a zoomed in view of mule deer locations and vegetation, and is just easier to work with.

```
#Load vegetation raster layer textfile clipped in ArcMap
veg <-raster("extentnlcd2.txt")
plot(veg)
```



```
class(veg)
```

```
## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

```
#Clip using the raster imported with "raster" package
bbclip <- crop(veg, SP)
veg
```

```
#WON'T WORK because projections are not the same, WHY?
#Let's check projections of layers we are working with now.
proj4string(MDclip)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

```
proj4string(deer.spdf)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

```
proj4string(SP)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

```
proj4string(veg)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NA
```

16. We need to have all layers in same projection so project the deer.spdf to Albers and then clip vegetation layer with new polygon we created in the Albers projection.

```
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
class(deer.albers)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

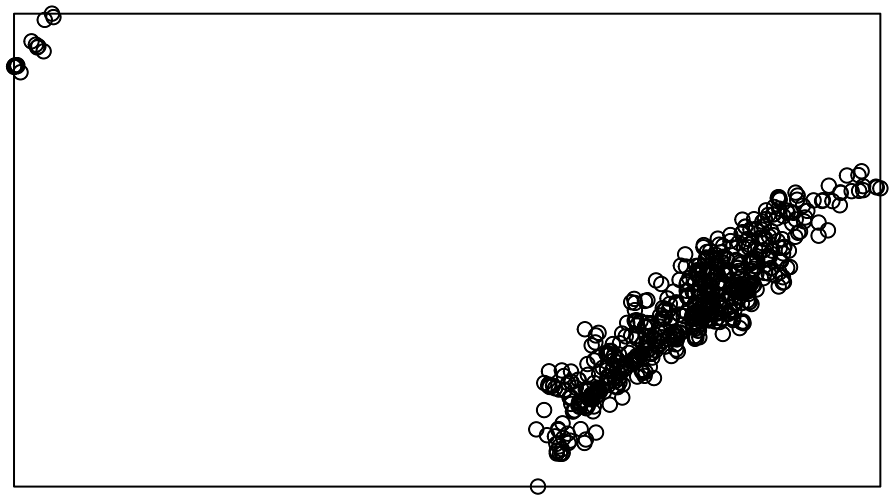
```
proj4string(deer.albers)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NA
```

```
bbox(deer.albers)
```

```
##        min      max
## x -1127964 -1115562
## y  1718097  1724867
```

```
bb1 <- cbind(x=c(-1115562,-1115562,-1127964,-1127964,-1115562),
  y=c(1718097, 1724867,1724867,1718097,1718097))
AlbersSP <- SpatialPolygons(list(Polygons(list(Polygon(bb1)),"1")),
  proj4string=CRS(proj4string(deer.albers)))
plot(AlbersSP)
points(deer.albers)
```



```
#Check to see all our layers are now in Albers projection
proj4string(veg)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NAD83 +units=
```

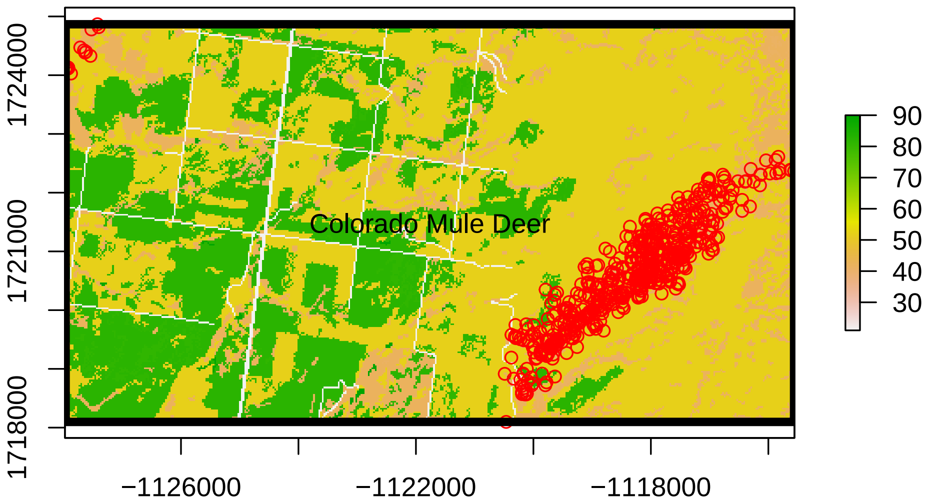
```
proj4string(deer.albers)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NAD83 +units=
```

```
proj4string(AlbersSP)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +datum=NAD83 +units=
```

```
#Clip using the raster imported with "raster" package
bbclip <- crop(veg, AlbersSP)
plot(bbclip)
points(deer.albers, col="red")
plot(AlbersSP, lwd=5, add=T)
text(coordinates(AlbersSP), labels="Colorado Mule Deer")
```



2.6 Creating a Square Polygon Grid Over a Study Area

1. Exercise 1.9 - 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 “GridSystem2Script.Rmd” and run code directly from the script

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

```
library(raster)
library(adehabitatMA)
```

5. Now let’s 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

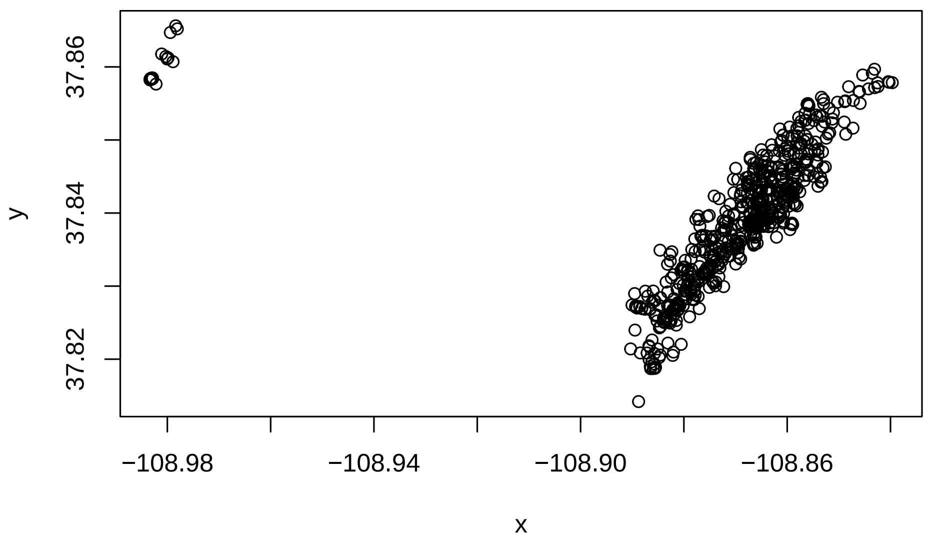
```
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
Albers.crs <-CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23
  +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
crs2<-"+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96
  +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
#NOTE: The difference between crs and Albers.crs is one
#is used to define projection that and the other to project a layer, respectively.
```

6. We need to have all layers in same projection so import, create, and remove outliers for mule deer locations then project all to the Albers projection as we did previously.

```
muleys <-read.csv("muleysexample.csv", header=T)
summary(muleys$id)
```

```
##    Length     Class      Mode
##      1091 character character
```

```
#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.50 & muleys$Lat >   37.8 & muleys$Long < -107)
muleys <- newmuleys

#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = muleys$Long, y = muleys$Lat)
plot(coords)
```



```
deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys, proj4string = CRS(crs))
proj4string(deer.spdf)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

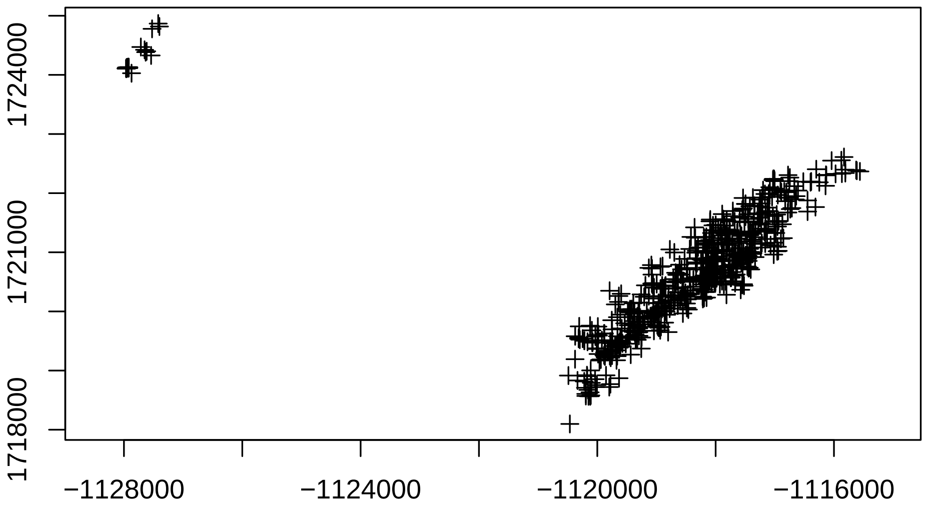
```
#Project deer.spdf to Albers as in previous exercise
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
proj4string(deer.albers)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +ellps=GRS
```

```
bbox(deer.albers)
```

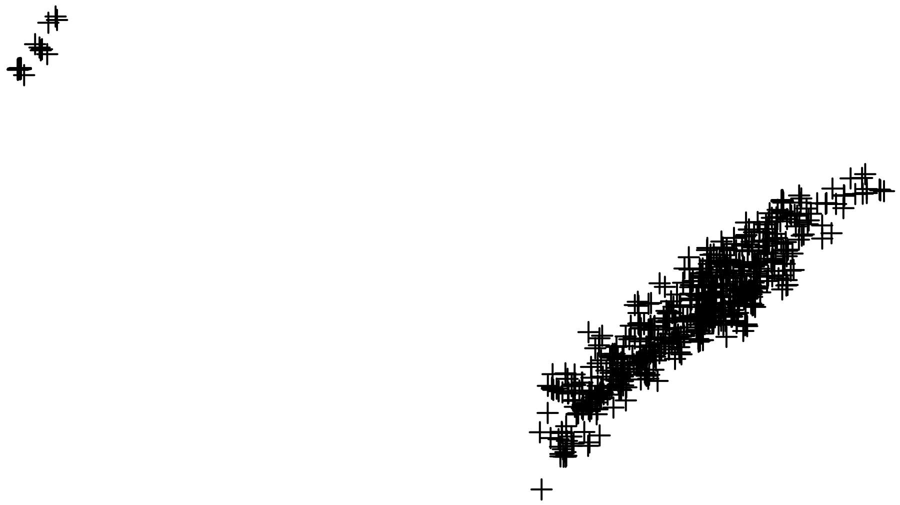
```
##         min      max
## x -1127964 -1115562
## y  1718097  1724867
```

```
plot(deer.albers,axes=T)
```



7. Create points for x and y from the bounding box of all mule deer locations with 1500 m spacing between each point.

```
plot(deer.albers)
```



```
#Create vectors of the x and y points
x <- seq(from = -1127964, to = -1115562, by = 1500)
y <- seq(from = 1718097, to = 1724867, by = 1500)
```

8. Create a grid of all pairs of coordinates (as a data.frame) using the "expand grid" function and then make it a gridded object.

```
xy <- expand.grid(x = x, y = y)
class(xy)
```

```
## [1] "data.frame"
```

```
#Make grid points into a Spatial Points Data Frame
grid.pts<-SpatialPointsDataFrame(coords= xy, data=xy, proj4string = CRS(crs2))
proj4string(grid.pts)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +ellps=GRS80 +units=
```

```
gridded(grid.pts)
```

```
## [1] FALSE
```

```
class(grid.pts)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
#Make points a gridded object (i.e., TRUE or FALSE)
gridded(grid.pts) <- TRUE
gridded(grid.pts)
```

```
## [1] TRUE
```

9. Make the grid of points into a Spatial Polygon then convert the spatial polygons to a SpatialPolygonsDataFrame.

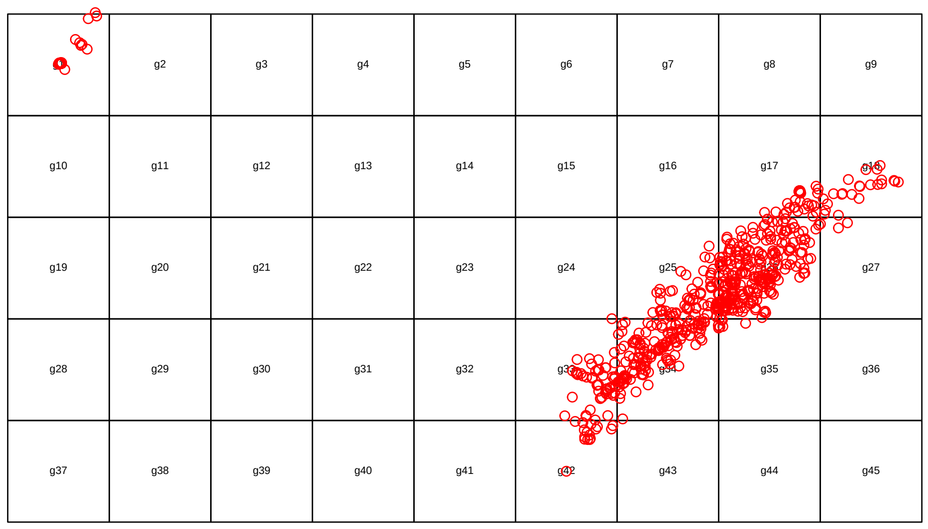
```
grid <- as(grid.pts, "SpatialPolygons")
plot(grid)
class(grid)
```

```
## [1] "SpatialPolygons"
## attr(,"package")
## [1] "sp"
```

```
summary(grid)
```

```
## Object of class SpatialPolygons
## Coordinates:
##         min      max
## x -1128714 -1115214
## y  1717347  1724847
## Is projected: TRUE
## proj4string :
## [+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0
## +ellps=GRS80 +units=m +no_defs]
```

```
gridspdf <- SpatialPolygonsDataFrame(grid,
  data=data.frame(id=row.names(grid), row.names=row.names(grid)))
names.grd <- sapply(gridspdf@polygons, function(x) slot(x,"ID"))
text(coordinates(gridspdf), labels=sapply(slot(gridspdf, "polygons"),
  function(i) slot(i, "ID")), cex=0.5)
points(deer.albers, col="red")
```



10. Similar to the hexagonal grid, identify the cell ID that contains each mule deer location.

```
o = over(deer.albers,gridspdf)
head(o)
```

```
##      id
## 20 <NA>
## 21   g1
## 22   g1
## 23   g1
## 24   g1
## 25   g1
```

```
new = cbind(deer.albers@data, o)
```

11. We get some NA errors because our grid does not encompass all mule deer locations so expand the grid then re-run the code over from xy through new2 again (i.e., Lines 62-86).

```
x <- seq(from = -1127964, to = -1115562, by = 1500)
y <- seq(from = 1718097, to = 1725867, by = 1500)
```

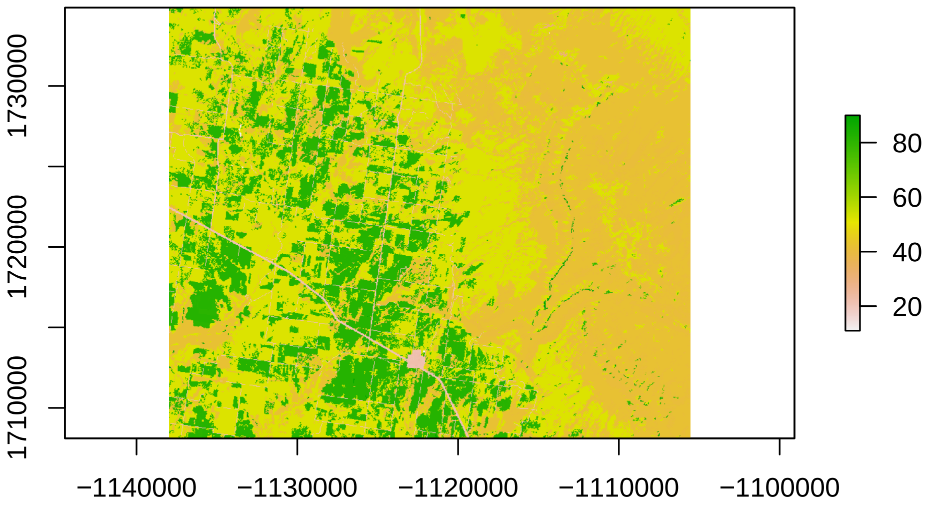
```
##BE SURE TO RUN CODE FROM XY CREATION THROUGH NEW2 AGAIN THEN LOOK AT DATA!!
o2 = over(deer.albers,gridspdf)
head(o2)
```

```
##      id
## 20 <NA>
## 21   g1
## 22   g1
## 23   g1
## 24   g1
## 25   g1
```

```
new2 = cbind(deer.albers@data, o2)#No more NAs causing errors!
```

12. Now we can load a vegetation raster layer textfile clipped in ArcMap to summarize vegetation categories within each polygon grid cell.

```
veg <-raster("cropnlcd.tif")
plot(veg)
```

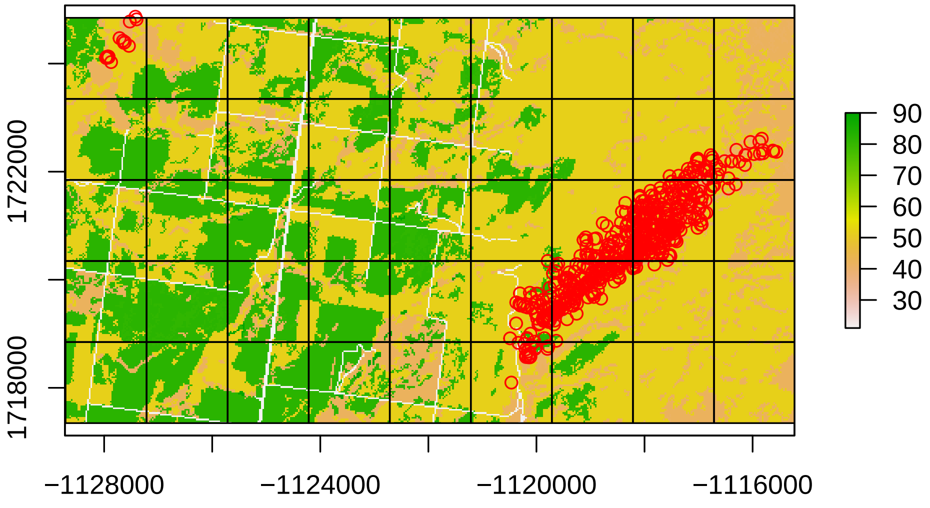


```
class(veg)
```

```
## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

13. Clip the raster within the extent of the newly created grid

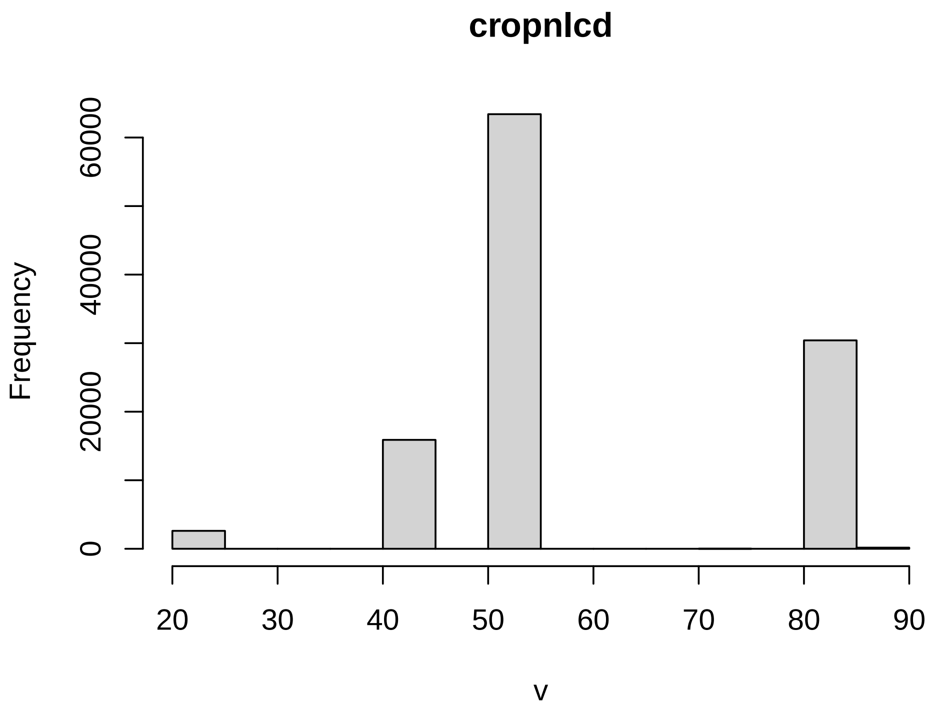
```
bbclip <- crop(veg, gridspdf)
plot(bbclip)
points(deer.albers, col="red")
plot(gridspdf, add=T)
```



```
#Cell size of raster layer
xres(bbclip)
```

```
## [1] 30
```

```
#Create histogram of vegetation categories in bbclip
hist(bbclip)
```



```
#Calculate cell size in square meters
ii <- calcperimeter(gridspdf)#requires adehabitatMA package
as.data.frame(ii[1:5,])
```

```
##       id perimeter
## g37 g37      6000
## g38 g38      6000
## g39 g39      6000
## g40 g40      6000
## g41 g41      6000
```

14. We can extract the vegetation characteristics within each polygon of the grid.

```
table = extract(bbclip,gridspdf)
str(table[1])
```

```
## List of 1
##  $ : num [1:2500] 52 82 82 82 82 82 82 52 52 52 ...
```

15. We can then tabulate area of each vegetation category within each polygon by extracting vegetation within each polygon by ID then appending the results back to the extracted table by running it twice but with different names. Summarizing the vegetation characteristics in each cell will be used in future resource selection analysis or disease epidemiology.

```
area = extract(bbclip,gridspdf)
combine=lapply(area,table)
combine[[1]]#Shows vegetation categories and numbers of cells in grid #1
```

```
##
##   21   42   52   81   82
##   86   75  923    1 1415
```

```
combine[[27]]
```

```
##
##   41   42   52
##  183  762 1555
```

2.7 Creating Buffers

For this exercise, we will again be working with the Colorado mule deer locations and rasters from earlier sections (1.3, 1.7). Creating buffers around locations of animals, plots, or some other variable may be necessary to determine what occurs around the locations. Often times,in resource selection studies, we may want to generate buffers that can be considered used habitat within the buffer as opposed to simply counting only the habitat that the location is found. Lets begin with loading the proper packages and mule deer locations from previous exercise. Because we are dealing with the raster layer projected in Albers, we will need to project our mule deer locations as we did above.

1. Exercise 1.10 - 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 "BufferScript.Rmd" and run code directly from the script

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

```
library(sp)
library(raster)
library(rgeos)
```

5. Now let's 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

```
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
Albers.crs <-CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23
  +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
```

6. Now open the script "BufferScript.Rmd" and run code directly from the script

```
muleys <-read.csv("muleysexample.csv", header=T)
summary(muleys$id)
```

```
##    Length     Class      Mode
##      1091 character character
```

```
#Let us subset data so there are fewer locations to work with
muley8 <- subset(muleys, id=="D8")

#Remove outlier locations if needed
summary(muley8$Long)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -111.8  -108.9  -108.9  -108.9  -108.9  -108.8
```

```
#NOTE: Min. of -111.8 is an outlier so remove
summary(muley8$Lat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   33.38   37.84   37.84   37.83   37.85   37.86
```

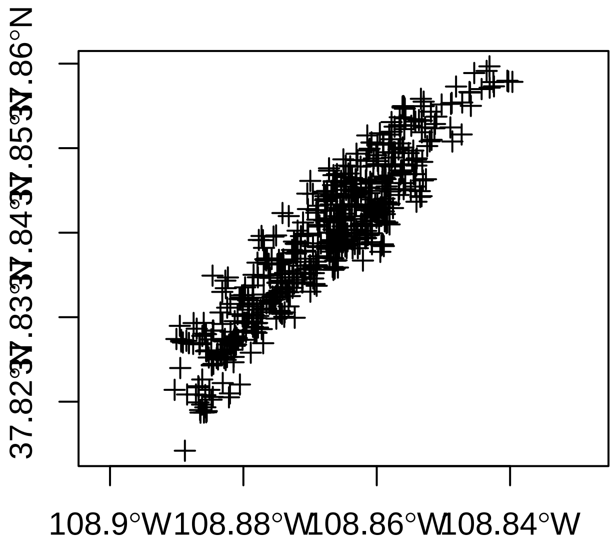
```
#NOTE: Min. of 33.38 is an outlier so remove
newmuley8 <-subset(muley8, muley8$Long > -111.7 & muley8$Lat > 37.80)
muley8 <- newmuley8

#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = muley8$Long, y = muley8$Lat)

deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muley8, proj4string = CRS(c
#head(deer.spdf)
class(deer.spdf)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
proj4string(deer.spdf)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

```
plot(deer.spdf, axes=T)
```



```
#Again let's project the deer.spdf to Albers
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
class(deer.albers)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
proj4string(deer.albers)
```

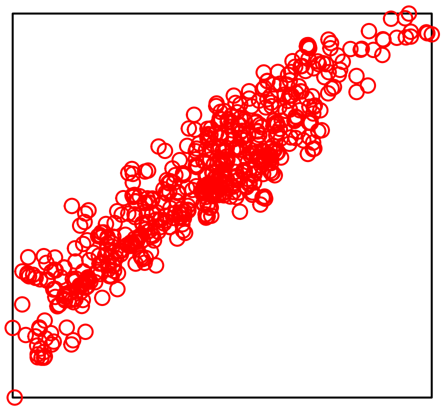
```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0 +ellps=GRS80 +units=
```

7. Clip the study.zoom so we can zoom in on mule deer 8 locations as we did in previous exercise but with a bounding box of only mule deer 8 locations.

```
bbox(deer.albers)
```

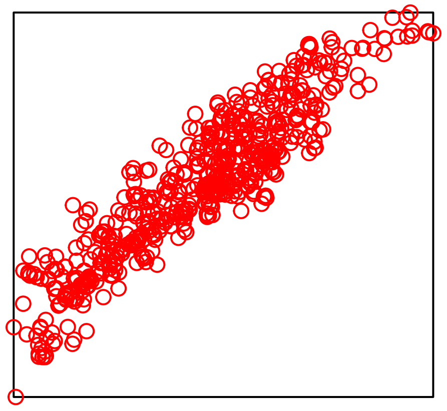
```
##        min      max
## x -1120488 -1115562
## y  1718097  1722611
```

```
bb1 <- cbind(x=c(-1115562,-1115562,-1120488,-1120488, -1115562),
  y=c(1718097,1722611,1722611,1718097,1718097))
AlbersSP <- SpatialPolygons(list(Polygons(list(Polygon(bb1)),"1")),
proj4string=CRS(proj4string(deer.albers)))
plot(AlbersSP)
points(deer.albers, col="red")
```



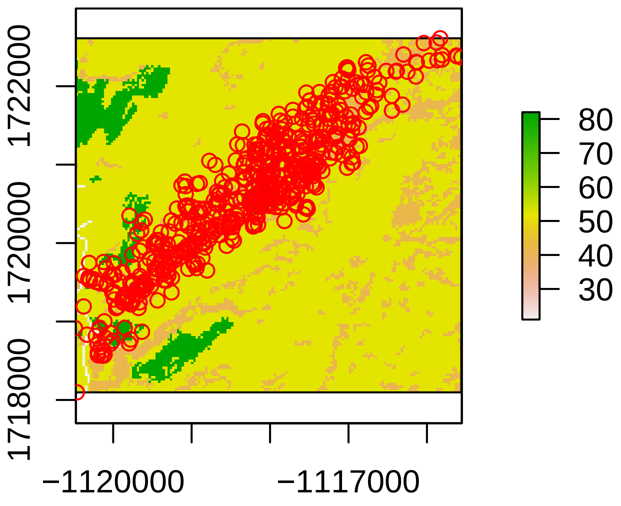
8. Alternatively, load the vegetation raster layer textfile clipped in ArcMap to be within several counties around the mule deer locations. Plot the points and bounding box over the vegetation layer and notice they are barely visible due to the large extent of the raster layer.

```r
veg <-raster("extentnlcd2.txt")
#plot(veg)
plot(AlbersSP)
points(deer.albers, col="red")
```



9. We can clip the vegetation raster and plot the bounding box polygon and locations on the raster. Notice that the locations are nearly off the extent of the raster.

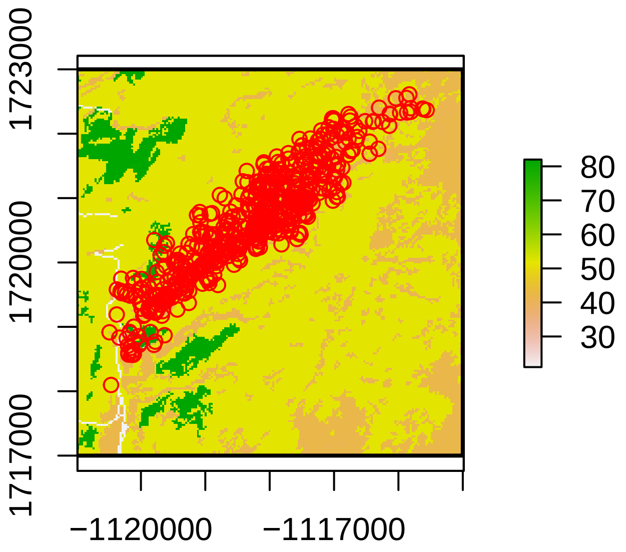
```r
bbclip <- crop(veg, AlbersSP)
plot(bbclip)
plot(AlbersSP,add=T)
points(deer.albers, col="red")
```



10. So let us create a new bounding box that encompass mule deer 8 locaitons but also extends beyond the periphery of the outermost locations. Then clip

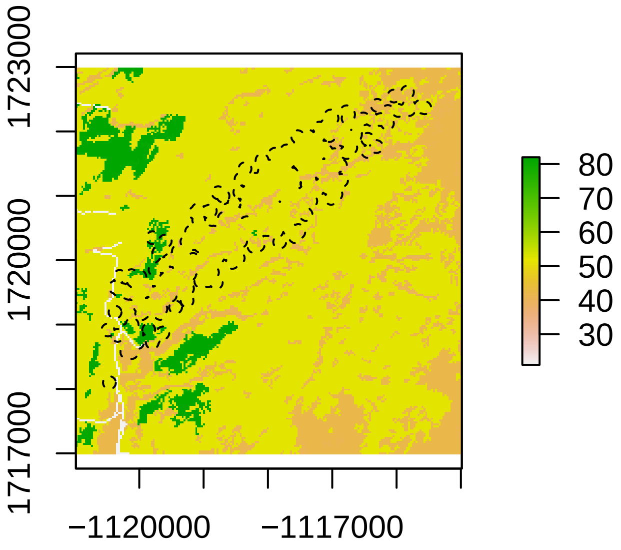
the large vegetation raster again so it is within the newly created bounding box polygon

```
bb1 <- cbind(x=c(-1115000,-1115000,-1121000,-1121000, -1115000),
  y=c(1717000,1723000,1723000,1717000,1717000))
AlbersSP <- SpatialPolygons(list(Polygons(list(Polygon(bb1)),"1")),
  proj4string=CRS(proj4string(deer.albers)))

bbclip <- crop(veg, AlbersSP)
plot(bbclip)
plot(AlbersSP,lwd=2, add=T)
points(deer.albers, col="red")
```



11. To conduct some analyses, let us create 100 m buffered circles around all the locations and extract vegetation that occurs in each buffered circle

```
settbuff=gBuffer(deer.albers,width=100)
plot(bbclip)
plot(settbuff, add=T, lty=2)
```



```
table(extract(bbclip,settbuff))
```

```
##
##   21   41   42   52   81   82
##   37  288  532 4725   56  157
```

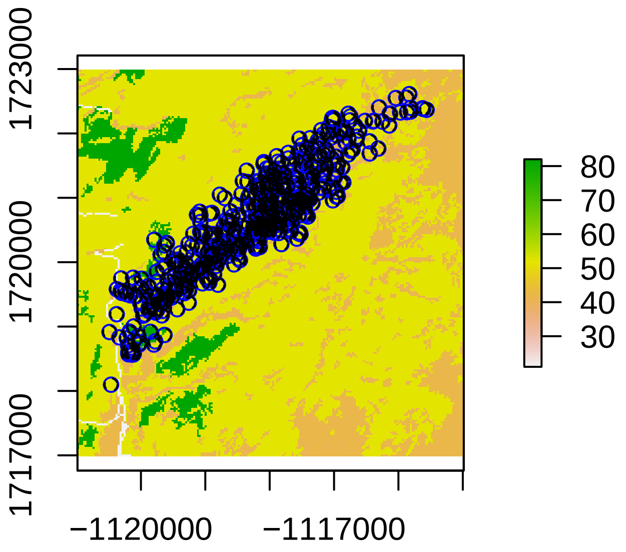
```
#Cell size of raster layer
res(bbclip)
```

```
## [1] 30 30
```

```
# 30^2
# 900*37
# (900*37)/1000000
```

12. Most efforts will want percent habitat or area of each habitat defined individually for each location (i.e., within each buffered circle). To do this we only need to specify in the gBuffer function to create unique buffered circles with the byid=TRUE command.

```
settbuff=gBuffer(deer.albers, width=100, byid=TRUE)
plot(bbclip)
points(deer.albers, col="blue")
plot(settbuff, add=TRUE,lty=8)
```



```
#Extract the amount of vegetation in each buffer and place it in a table by buffer ID
e= extract(bbclip,settbuff)
et=lapply(e,table)

#Example below identifies buffered circles number 328
et[[328]]
```

```
##
## 41 42 52
##  4  7 24
```

```
#Buffer ID 328 has 3 vegetation categories 41, 42, and 52 of 4, 7, and 24 cells, respectively
```

Chapter 3

Climate Data Interpretation

3.1 Cleaning Raw Climate Data

1. Exercise 2.3 - 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 "Read_FilesScript.Rmd" and run code directly from the script

4. No packages are needed for this exercise, these are base R functions

5. For each csv file, save as Excel Worsheet 1997-2003 if importing to NCSS or keep in csv or txt if using R.

6. Take out all non-Jan months for every year

7. Files will need to meet the following criteria but will be addressed in Exercise 2.4: Each weather station must have records for at least 10 of the 11 Januaries. Each weather station must have at least 95% of daily records for those Januaries. This means at least 325 days for 11 seasons and 295 days for 10 seasons.

Snow Depth (SNWD) 66 stations Maximum temp (TMAX) 69 stations Minimum temp (TMIN) 68 stations

8. The code that follows should have all files in the same folder but not the R script or any R files or code will not run. The code below brings in each text file and summarizes the data for each weather station as instructed in the code.

```
#setwd("/Users/wdw12/Library/CloudStorage/OneDrive-ThePennsylvaniaStateUniversity/Walt
setwd("/Users/davidwalter/Library/CloudStorage/OneDrive-ThePennsylvaniaStateUniversity/
# Vector of files names in working directory
files <- list.files(pattern = ".txt")

# Total number of files in working directory (for loop below)
n.files <- length(files)

# Container to hold text files
files.list <- list()

#Populate the container files.list with climate data sets
files.list <- lapply(files, read.table, header =T, sep="\t")

#Set up matrix for weather station summary data
m1 <- matrix(NA,ncol=8,nrow=n.files)

#Loop for running through all weather station files
for(i in 1:n.files){

    # Assign elevation
        m1[i,1] <- files.list[[i]][1,10]

    #Assign Lat
        m1[i,2] <- files.list[[i]][1,11]

    #Assign Long
        m1[i,3] <- files.list[[i]][1,12]

    #Calculate mean snow depth
        SNWD_mm <- mean(files.list[[i]][,7],na.rm=T)

    #Convert snow depth mean to inches
    SNWD_in <- SNWD_mm/25.4

    #Assign snow depth
    m1[i,4] <- SNWD_in

    #Calculate mean maximum temp
        TMAX_C <- mean(files.list[[i]][,8],na.rm=T)

    #Convert max temp to F
    TMAX_F <- TMAX_C*0.18 + 32

    #Assign max temp
    m1[i,5] <- TMAX_F

    #Calculate mean minimum temp
    TMIN_C <- mean(files.list[[i]][,9],na.rm=T)

    #Convert min temp to F
    TMIN_F <- TMIN_C*0.18 + 32

    #Assign min temp
    m1[i,6] <- TMIN_F

    #Reassign GHCN number
    GHCN <- toString(files.list[[i]][1,1])

    #Assign Station Name
    m1[i,7] <- GHCN

    #Reassign Station Name
    SN <- toString(files.list[[i]][1,2])

    #Assign Station Name
    m1[i,8] <- SN
}

colnames(m1) <- c("Elevation","Lat","Long","SNWD","TMAX","TMIN","GHCN","Station")
write.csv(m1,paste(".","\\output.csv",sep=""))

#Removes quotation marks in output table
m1 <-noquote(m1)
m1[1:5,]
```

```
#setwd("/Users/wdw12/Library/CloudStorage/OneDrive-ThePennsylvaniaStateUniversity/WalterRprojects
setwd("/Users/davidwalter/Library/CloudStorage/OneDrive-ThePennsylvaniaStateUniversity/WalterRpro
```

3.2 Using Data in R

This code is designed to process data downloaded from Climate Date Online. http://www.ncdc.noaa.gov/cdo-web/ This version looks at weather stations that provide snowfall data in and around Pennsylvania. The code pulls out the desired data from the downloaded aggregate weather-station file and calculates the mean annual snowfall per weather station for 12/1/94 - 3/31/05. The data is then exported to a text file for interpolation in ArcGIS. - Bill Kanapaux, PA Cooperative Fish & Wildlife Research Unit

Last modified: Jan. 13, 2014

1. Exercise 2.4 - 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 "Weather-Station-Code_Snowfall.Rmd" and run code directly from the script

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

```
library(adehabitatHR)
library(rgdal)
library(gstat)
library(plyr)
```

5. This code is designed to process data downloaded from NOAA Date. This version looks at weather stations that provide snowfall data in and around Pennsylvania. The code pulls out the desired data from the downloaded aggregate weather-station file and calculates the mean annual snowfall per weather station for 12/1/94 - 3/31/05. The data is then exported to a text file for interpolation in R or ArcGIS. First, we read weather station data - note the use of stringsAsFactors=FALSE and na.strings='-9999'

```
WS <-read.table('Weather_Station_Data-Sep_01Dec1994-31March2005.txt',
  stringsAsFactors=FALSE, na.strings='-9999',header=T)
```

```
#Check data
dim(WS)
head(WS)
summary (WS)
```

```
#Reformat DATE and create Year Month Day columns from NewDate column ###
WS$NewDate <- as.Date(as.character(WS$DATE), format("%Y%m%d"))
WS$Year = as.numeric(format(WS$NewDate, format = "%Y"))
WS$Month = as.numeric(format(WS$NewDate, format = "%m"))
WS$Day = as.numeric(format(WS$NewDate, format = "%d"))
```

6. Make a subset of WS that includes only the months of Dec-March with further manipulation of the data for desired output of project objectives.

```
Winter <- WS[WS$Month %in% c(1,2,3,12), ]

#For December, add 1 to Year so that Year matches Jan-March in that season ###
Winter <- within(Winter, Year[Month==12] <- Year[Month==12] +1)
```

```r
#Check subset, including random row to make sure only selected months included ###
dim(Winter)
head(Winter)
Winter[699,]
```

```r
#Create a matrix of unique STATION values (GHCND ) with Lat/Long values for later reference.
### Data contains some multiple versions of individual GHCND coordinates. Only want 1 set per.
PulledCoords <- Winter[!duplicated(Winter[,1]),]
CoordChart <- ddply(PulledCoords, c('STATION'), function(x) c(Lat=x$LATITUDE, Long=x$LONGITUDE))

#Get the number of snowfall records for each STATION for each year and name it RecordTotal.
#Note that NA is omitted from the length count
WinterRecords <- ddply(Winter, .(STATION,Year), summarize, RecordTotal = length(na.omit(SNOW)))

#Get the total amount of snowfall per STATION per year and name it YearlySnow
YearlySnow <- ddply(Winter, .(STATION,Year), summarize, Snow = sum(SNOW, na.rm=TRUE))

#Combine WinterRecords and YearlySnow into one matrix
AllWinters <- cbind(WinterRecords,YearlySnow)
AllWinters <- AllWinters[,-4:-5]

#Only include years that have more than 75% of days recorded
WinterDays <- 121
FullWinters <- AllWinters[AllWinters$RecordTotal/WinterDays > 0.75, ]

#Get the number of years with more than 75% of days recorded for each STATION
WinterYears <- ddply(FullWinters, c('STATION'), function(x) c(TotalYears=length(x$Year)))

#Get the total amount of snow for each station for all years
TotalWinterSnow <- ddply(FullWinters, c('STATION'), function(x) c(TotalWinterSnow=sum(x$Snow)))

#Combine WinterYears and TotalWinterSnow into one matrix
SnowCalc <- cbind(WinterYears,TotalWinterSnow)
SnowCalc <- SnowCalc[,-3]

#Get rid of the stations that don't have at least 10 years recorded at >75% of days ###
Complete.Records <- SnowCalc[SnowCalc$TotalYears > 9, ]

#Calculate average annual snowfall and round to nearest mm
Complete.Records$MeanAnnualSnowfall <- Complete.Records$TotalWinterSnow/Complete.Records$TotalYea
Complete.Records$MeanAnnualSnowfall <- round (Complete.Records$MeanAnnualSnowfall, digits = 0)

#Convert SnowDepth from mm to cm
Complete.Records$MeanAnnualSnowfall <- Complete.Records$MeanAnnualSnowfall/10
head(Complete.Records)
```

```
##              STATION TotalYears TotalWinterSnow MeanAnnualSnowfall
## 1 GHCND:USC00072730         11            3493               31.8
## 3 GHCND:USC00079605         10            4967               49.7
## 4 GHCND:USC00181530         11            7232               65.7
## 5 GHCND:USC00181750         11            4828               43.9
## 7 GHCND:USC00182282         11            7496               68.1
## 8 GHCND:USC00182336         11            8420               76.5
```

```
#Add a column to CoordChart showing whether each row matches  a STATION in Complete.Re
#Use "NA" for value if no match, then delete rows with "NA" value.
#Number of rows in CoordChart should now equal number of rows in Complete.Records
CoordChart$match <- match(CoordChart$STATION, Complete.Records$STATION, nomatch=NA)
CoordChart <- na.omit(CoordChart)

#Combine Complete.Records and CoordChart. Make sure each STATION matches in row
#Delete any rows that don't match. Shouldn't be any. If number of rows in Final.Values
#is less than number of rows in CoordChart, there is a problem (but note that # of col
Final.Values <- cbind(Complete.Records,CoordChart)
Final.Values$match2 <- match(Final.Values[  ,1], Final.Values[ ,5], nomatch=NA)
Final.Values <- na.omit(Final.Values)
dim(Final.Values)
```

```
## [1] 144   9
```

```
dim(CoordChart)
```

```
## [1] 144   4
```

```
#Take out unnecessary rows (2nd STATION, match, and match2) and round MeanSnow to 2 de
Final.Values[,5] <- Final.Values[,8] <- Final.Values[,9] <- NULL
```

7. Make data frame to get rid of lists (in R) so can export to text file to use to load weather station points into ArcGIS and skip to Section 2.5.

```
Final.Values <- as.data.frame(lapply(Final.Values,unlist))
```

```
write.table(Final.Values, "MeanSnowData_95-05.txt", sep="\t", row.names=F)
```

8. Alternatively we can conduct interpolation directly in R using the steps below

```
#Need to convert factors to numeric
Final.Values$Longitude <- as.numeric(as.character(Final.Values$Long))
Final.Values$Latitude <- as.numeric(as.character(Final.Values$Lat))
```

```
#Here we need to create Spatial points, attach ID and Date Time sorted
data.xy = Final.Values[c("Longitude","Latitude")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
proj4string(xysp) <- CRS("+proj=longlat +ellps=WGS84")

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

#Creates a spatial data frame of STATION
ID<-data.frame(Final.Values[1])
#Creates a spatial data frame of Mean Annual Snow Fall
MAS<-data.frame(Final.Values[4])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(ID,MAS)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
proj4string(merge) <- CRS("+proj=longlat +ellps=WGS84")

#Import a county layer for study site and check projections
counties<-readOGR(dsn=".",layer="PACountiesAlbers",verbose = FALSE)
proj4string(counties)
```

```
## [1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.3 +lat_2=45.3 +x_0=0 +y_0=0 +datum=NAD83 +units=
```

```
EPSG <- make_EPSG()
PAsp <- subset(EPSG, EPSG$code == "6564")
PAsp
```

```
##      code                            note
## 3741 6564 NAD83(2011) / Pennsylvania South
##
## 3741 +proj=lcc +lat_0=39.3333333333333 +lon_0=-77.75 +lat_1=40.9666666666667 +lat_2=39.9333333
##                          prj_method
## 3741 Lambert Conic Conformal (2SP)
```

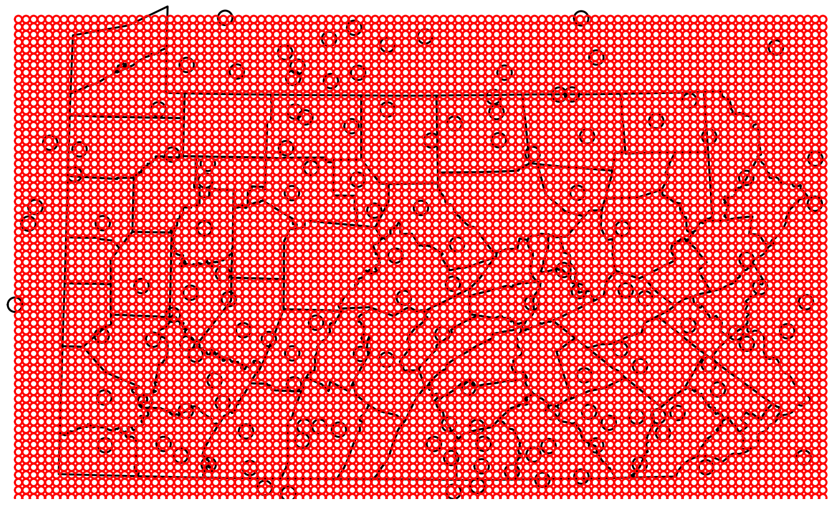
```
#Copy and Paste State Plane projection into code for StatePlane below
#Project Weather Stations and Counties to State Plane
StatePlane <- CRS("+proj=lcc +lat_0=39.3333333333333 +lon_0=-77.75 +lat_1=40.9666666666667 +lat_2

stations <- spTransform(merge, CRS=StatePlane)
counties <- spTransform(counties, CRS=StatePlane)

stations@data$MAS <- stations@data$MeanAnnualSnowfall
```

```
#Plot out the MAS across the study region
bubble(stations, zcol='MAS', fill=FALSE, do.sqrt=FALSE, maxsize=2, add=T)
```

```
#Create a grid onto which we will interpolate:
xx = spsample(stations, type="regular", cellsize=5000)
class(xx)
```

```
## [1] "SpatialPoints"
## attr(,"package")
## [1] "sp"
```

```
#Plot Weather Stations over counties
plot(counties)
points(stations)
points(xx, bg="red", cex=.5,col="red")
```



```
#Now expand to a grid with 5000 meter spacing
x.range <- as.integer(range(xx@coords[,1]))
y.range <- as.integer(range(xx@coords[,2]))

grd <- expand.grid(x=seq(from=x.range[1]-50000, to=x.range[2]+50000, by=5000),
                   y=seq(from=y.range[1]-50000, to=y.range[2]+50000, by=5000))

#Convert to SpatialPixel class
coordinates(grd) <- ~ x+y
gridded(grd) <- TRUE
proj4string(grd) <- StatePlane
```

Interpolate grid over sample points directly in R with gstat package

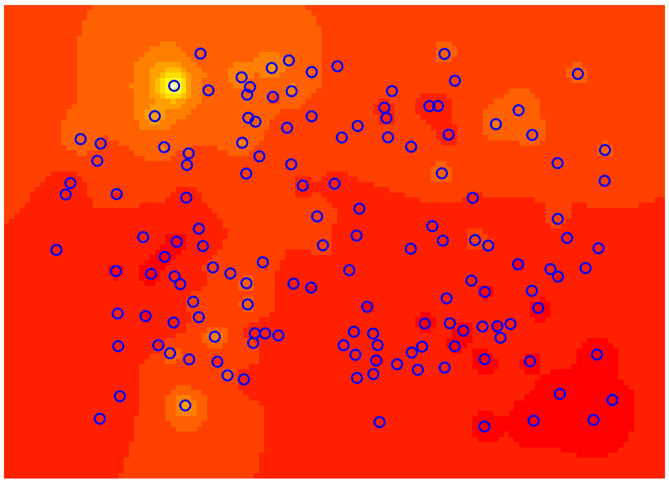
```
x <- krige(stations@data$MAS~1, stations, grd)
```

```
## [inverse distance weighted interpolation]
```

```
class(x)
```

```
## [1] "SpatialPixelsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
image(x)
points(stations, pch=1, col='blue', cex=0.7)
```



3.3 Importing Dynamically Downscaled Global Climate Data

This exercise will provide some code for manipulating climate change data from the Regional Climate Downscaling by copy the link into your browser: http://regclim.coas.oregonstate.edu/data-access/index.html or just select the link here: Regional Climate Downscaling . IMPORTANT: For each climate projection, must change name in first command and file name in last command.

1. Exercise 2.5 - 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 "NetCDF_Script.Rmd" and run code directly from the script

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

```
library(ncdf4)
```

5. Open netCDF and setting verbose=true provides details about the data in the netcdf file including the varid. You need to know the varid to select the variable you want to extract/summarize. Note: the dimensions x, y, time also get a varid so you will need to subtract 3 from the varid of interest to get the correct one.

```
dat <- nc_open("Monthly_AvgMinTemp_1995-99_MPI.nc", write=TRUE,  readunlim=TRUE, verbos

#Read data to load all the data from the downloaded variable into the tmin object
tmin <- dat$var[[1]]

####################################
# The following illustrates how to read the data
####################################
print(paste(tmin$name)) #in this case the 'field name' is TAMIN
```

```
## [1] "TAMIN"
```

```
# Grab data for TAMIN variable and place in object df1
df1 <- ncvar_get(dat, tmin)

#head(df1, n = 10L)# head(x, n = 6L, ...); head returns the first data  entries, x is
#n sets the number of entries displayed. tail returns  the last of the data entries
# Dimensions of df1 (x, y, time)
dim(df1)
```

```
## [1] 37 22 49
```

```
# Dimensions can also be examined one at a time
dim(df1)[1]     # number of x grids (37)
```

```
## [1] 37
```

```
dim(df1)[2]     # number of y grids (22)
```

```
## [1] 22
```

```
dim(df1)[3]     # number of months in file (49)
```

```
## [1] 49
```

```
#NOTE: FILE INCLUDES MONTHS OTHER THAN JANUARY (Jans are 1,13,25,37,49)

# Check first element
df1[1,1,1]
```

```
## [1] -2.499621
```

```
# Check first January for all x,y
df1[,,1]
```

```
#Create a new matrix which is monthly averages for each grid cell. Make the new  matrix the
#same size (i.e. same number of rows and columns as there are in the dataframe df1
sum1 <- array(data=NA, c(dim(df1)[1],dim(df1)[2] ))
dim(sum1)
```

```
## [1] 37 22
```

```
# Create January mean TAMIN for each x-y coordinate
for(i in 1:dim(df1)[1]){ # loop over x-coords
    for(j in 1:dim(df1)[2]){ # loop over y-coords
        sum1[i, j] <- (df1[i,j,1]+df1[i,j,13]+df1[i,j,25]+df1[i,j,37]+df1[i,j,49])/5
    }
}

############################################################
############################################################
# Create netcdf file from sum1 (contains matrix of new data)
############################################################
# Get x and y coordinates from original "dat" ncdf file
x  = ncvar_get(nc=dat,varid="x")
y  = ncvar_get(nc=dat,varid="y")

# Check dimensions
length(x)
```

```
## [1] 37
```

```
length(y)
```

```
## [1] 22
```

```
dim(sum1)
```

```
## [1] 37 22
```

```
## define the netcdf coordinate variables - note that these are coming from the dat
#file with actual values
dim1 = ncdim_def( "X","meters", as.double(x))
dim2 = ncdim_def( "Y","meters", as.double(y))

## define the EMPTY (climate) netcdf variable and define names that will be used in the
#var.def.ncdf function
# Define climate variable names
    new.name <- 'mintemp'
# Define units of measurement for variable
    units <- 'degreesC'
# Define long name for variable
    long.name <- 'Jan average min temperature'

varz = ncvar_def(new.name,units, list(dim1,dim2), -1,
          longname=long.name)

# associate the netcdf variable with a netcdf file
# put the variable into the file, and close

nc.ex = nc_create( "MPI1999-95.nc", varz )
ncvar_put(nc.ex, varz, sum1)
nc_close(nc.ex)
```

Chapter 4

Movement Methods

4.1 Importing datasets from a web source

Movebank.org is a cloud-based repository for relocation data from GPS-collared or VHF-collared animals. It provides a storage facility in the cloud that can serve as a backup for your data or a transfer portal to share data among colleagues or interested researchers. Similar to any email account, each user has a Movebank account that has a login and password to gain access to your data. Administration privileges can be given to anyone with an account for viewing and downloading data.

1. Exercise 3.1 - 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 "DMAdeerMovebank.Rmd" and run code directly from the script

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

```r
library(move)
library(RCurl)
library(circular)
```

5. Next we are going to the Movebank home page and explore what it has to offer. Need to create an account or select a dataset that does not require permission to use.

```r
login <- movebankLogin(username="wdwalter", password="Cervus54")
```

```
deer <- getMovebankData(study="DMA White-tailed Deer 2018 Pennsylvania USA",
                        login=login, moveObject=TRUE)
n.indiv(deer)
n.locs(deer)
#Plot the first deer in the stack
plot(deer[[1]])

#Now we will select a single deer to explore more
deer1 <- deer[['X20212_20242F']]
plot(deer1)

#Select and plot locations of the initial 2 deer in your list
deer2 <- deer[[c(1,2)]]
plot(deer2)
#Determine names of initial 2 deer selected above
namesIndiv(deer2)
```

4.2 Movement Trajectories

We will start with simply creating trajectories between successive locations. As stated previously, there are 2 types of trajectories but their are also 2 forms of Type II trajectories if we have time recorded. Depending on the duration between locations we can have uniform time lag between successive relocations termed regular trajectories and non-uniform time lag that results in irregular trajectories. We will begin this section with simply creating irregular trajectories from relocation data because, even though we set up a time schedule to collection locations at uniform times, climate, habitat, and satellites do not always permit such schedules of data collection.

1. Exercise 3.2 - 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 "MovementScript.Rmd" and run code directly from the script

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

```
library(adehabitatLT)
library(chron)
```

5. We are again going to be using more of the mule deer dataset than from the earlier exercises

```
muleys <-read.csv("DCmuleysedited.csv", header=T)
```

6. Check for duplicate locations in dataset. The reason for this is very important and will be apparent shortly.

```
#Check for duplicate locations in dataset
summary(duplicated(muleys))
```

```
##    Mode   FALSE
## logical    9752
```

```
#Sort data to address error in code if needed
#muleys <- muleys[order(muleys$id),]

### Conversion of the date to the format POSIX
#Date <- as.character(muleys$GPSFixTime)
#Date <- as.POSIXct(strptime(as.character(muleys$GPSFixTime),"%Y.%m.%d %H:%M:%S"))
#muleys$Date <- Date
```

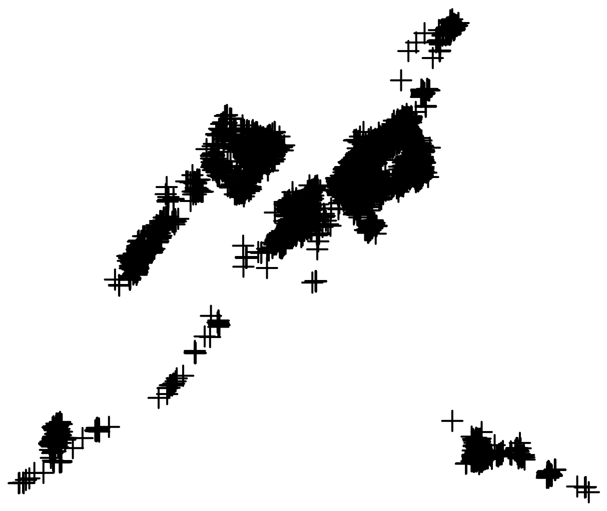
7. For trajectories of type II (time recorded), the conversion of the date to the format POSIX needs to be done to get proper digits of date into R.

```
da <- as.POSIXct(strptime(muleys$GPSFixTime,format="%Y.%m.%d %H:%M:%S"))
muleys$da <- da

timediff <- diff(muleys$da)*60
muleys <-muleys[-1,]
muleys$timediff <-as.numeric(abs(timediff))

newmuleys <-subset(muleys, muleys$X > 599000 & muleys$X < 705000 & muleys$Y > 4167000
  & muleys$timediff < 14401)
muleys <- newmuleys
```

8. Now create a Spatial Points Data Frame in UTM zone 12 adding ID, time diff, burst to xy coordinates

```
data.xy = muleys[c("X","Y")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)
#Creates a spatial data frame of ID
idsp<-data.frame(muleys[2])
```

```
#Creates a spatial data frame of dt
dtsp<-data.frame(muleys[24])
#Creates a spatial data frame of Burst
busp<-data.frame(muleys[23])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp,dtsp,busp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
plot(merge)
```

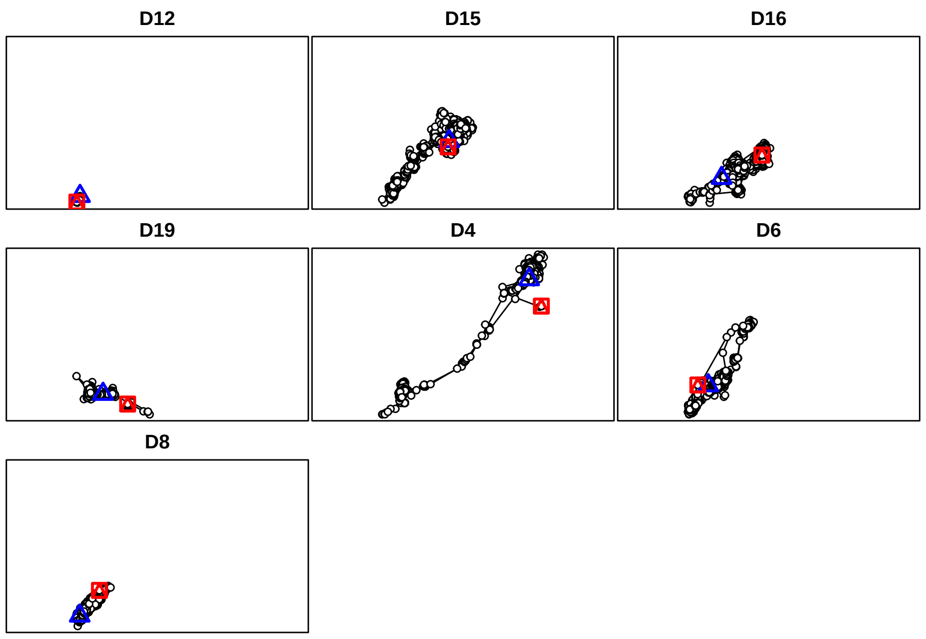


9. Now create an object of class “ltraj” by animal using the ID field and display by each individual (i.e., ltraj[1]).

```
ltraj <- as.ltraj(coordinates(merge),merge$da,id=merge$id)
head(ltraj[1])#Describes the trajectory
```

```
##
## *********** List of class ltraj ***********
##
## Type of the traject: Type II (time recorded)
## * Time zone unspecified: dates printed in user time zone *
## Irregular traject. Variable time lag between two locs
##
## Characteristics of the bursts:
##    id burst nb.reloc NAs          date.begin            date.end
## 1 D12   D12       98   0 2011-10-12 06:00:52 2011-10-24 21:00:48
##
##
##  infolocs provided. The following variables are available:
## [1] "pkey"
```

```
plot(ltraj)#plot all trajectories created
```



```r
#Plot each trajectory separately
plot(ltraj[1])
plot(ltraj[2])
plot(ltraj[3])
plot(ltraj[4])
plot(ltraj[5])
plot(ltraj[6])
plot(ltraj[7])
```

10. Create a histogram of time lag (i.e., interval) and distance between successive locations for each deer. This is a nice way to inspect the time lag between locations as you don't want to include a location if too much time has passed since the previous and it also shows why a trajectory is irregular.

```r
hist(ltraj[1], "dt", freq = TRUE)
hist(ltraj[1], "dist", freq = TRUE)
hist(ltraj[2], "dt", freq = TRUE)
hist(ltraj[2], "dist", freq = TRUE)
hist(ltraj[3], "dt", freq = TRUE)
hist(ltraj[3], "dist", freq = TRUE)
hist(ltraj[4], "dt", freq = TRUE)
hist(ltraj[4], "dist", freq = TRUE)
hist(ltraj[5], "dt", freq = TRUE)
hist(ltraj[5], "dist", freq = TRUE)
hist(ltraj[6], "dt", freq = TRUE)
hist(ltraj[6], "dist", freq = TRUE)
hist(ltraj[7], "dt", freq = TRUE)
hist(ltraj[7], "dist", freq = TRUE)
```

4.3 Distance Between Locations

Determining the distance between locations or between locations and respective habitat types can serve a variety of purposes. Several resource selection procedures require a description of the daily movement distance of an animal to determine the habitat available to an animal or when generating random locations around known locations. We will start here with a method to determine the average distance moved by mule deer in Colorado in a study to determine methods to alleviate depradation on sunflowers that have become a high commodity crop in the area.

1. Exercise 3.3 - 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 “DistanceUniqueBurst.Rmd” and run code directly from the script

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

```
library(adehabitatLT)
library(chron)
library(class)
```

5. Code to read in dataset then subset for an individual animal

```
muleys <-read.csv("DCmuleysedited.csv", header=T)
#Code to select an individual animal
muley15 <- subset(muleys, id=="D15")
table(muley15$id)
```

```
##
##  D15
## 2589
```

```
#Sort data to address error in code and then look at first 20 records of data to confirm
muley15 <- muley15[order(muley15$GPSFixTime),]
#Run code to display the first 20 records to look at what sorting did to data
```

6. Prepare data to create trajectories using the ltraj command in Adehabitat LT

```
######################################################
## Example of a trajectory of type II (time recorded) with conversion of the date to the
#format POSIX that nNeeds to be done to get proper digits of date into R then POSIXct uses
#library(chron)
da <- as.character(muley15$GPSFixTime)
da <- as.POSIXct(strptime(muley15$GPSFixTime,format="%Y.%m.%d %H:%M:%S"))
head(da)
```

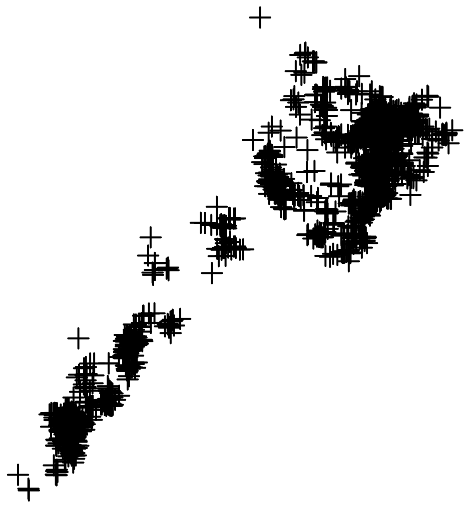
```
## [1] "2011-10-12 00:02:03 EDT" "2011-10-12 03:00:52 EDT"
## [3] "2011-10-12 06:00:52 EDT" "2011-10-12 09:00:38 EDT"
## [5] "2011-10-12 12:00:34 EDT" "2011-10-12 15:00:42 EDT"
```

```
#Attach da to muley15
muley15$da <- da

timediff <- diff(muley15$da)
muley15 <-muley15[-1,]
```

```
muley15$timediff <-as.numeric(abs(timediff))

#Clean up muley15 for outliers
newmuleys <-subset(muley15, muley15$X > 599000 & muley15$X < 705000 & muley15$Y > 4167
                   & muley15$timediff < 14401)
muley15 <- newmuleys
```

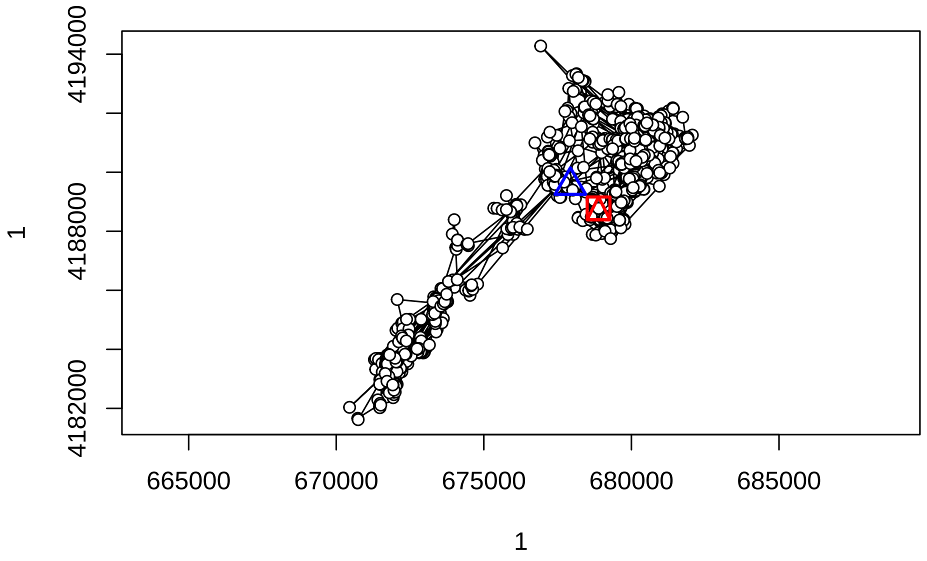
7. Create a spatial data frame of locations for muley 15 for use in creating trajectories that includes time difference between locations and dates in proper format (as.POSIXct)

```
data.xy = muley15[c("X","Y")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
proj4string(xysp) <- CRS("+proj=utm +zone=12 +ellps=WGS84")

#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)
#Creates a spatial data frame of ID
idsp<-data.frame(muley15[2])
#Creates a spatial data frame of dt
dtsp<-data.frame(muley15[24])
#Creates a spatial data frame of Burst
busp<-data.frame(muley15[23])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp,dtsp,busp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
plot(merge)
```



8. Create an object of class “ltraj” for muley15 dataset

```
ltraj <- as.ltraj(coordinates(merge),merge$da,id=merge$id)
plot(ltraj)
```



```
ltraj
```

```
##
## *********** List of class ltraj ***********
##
## Type of the traject: Type II (time recorded)
## * Time zone unspecified: dates printed in user time zone *
## Irregular traject. Variable time lag between two locs
##
## Characteristics of the bursts:
##    id burst nb.reloc NAs          date.begin            date.end
## 1 D15   D15     2588   0 2011-10-12 03:00:52 2012-08-31 09:00:51
##
##
##  infolocs provided. The following variables are available:
## [1] "pkey"
```

```
#Now let's look at time differences between locations before moving forward
summary(muley15$timediff)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.998   2.998   3.000   3.008   3.002   6.004
```

9. Need to create separate "bursts" for each trajectory based on the number of locations collected each day. In our case it was 8 (i.e., locations collected every 3 hours during a 24-hour period).

```
#We want to study the trajectory of the day at the scale of the day. We define one tra
#per day. The trajectory should begin at 2200 hours so the following function returns
#the date is time between 06H00 and 23H00 (i.e. results in 7-8 locations/day bursts)
foo <- function(date) {
da <- as.POSIXlt(date)
ho <- da$hour + da$min
return(ho>15.9&ho<23.9)
}
deer <- cutltraj(ltraj, "foo(date)", nextr = TRUE)
```

```
## Warning in cutltraj(ltraj, "foo(date)", nextr = TRUE): At least 3 relocations are n
##  345 relocations have been deleted
```

```
#Notice that the above code will remove 328 relocations that fall
#outside of your time criteria
#Warning message:
#In cutltraj(ltraj, "foo(date)", nextr = TRUE) :
#  At least 3 relocations are needed for a burst
# 328 relocations have been deleted
head(deer)
```

```
##
```

```
## *********** List of class ltraj ***********
##
## Type of the traject: Type II (time recorded)
## * Time zone unspecified: dates printed in user time zone *
## Irregular traject. Variable time lag between two locs
##
## Characteristics of the bursts:
##    id   burst nb.reloc NAs          date.begin            date.end
## 1 D15 D15.001        6   0 2011-10-12 03:00:52 2011-10-12 18:00:52
## 2 D15 D15.003        7   0 2011-10-13 00:00:35 2011-10-13 18:00:35
## 3 D15 D15.005        7   0 2011-10-14 00:00:42 2011-10-14 18:00:42
## 4 D15 D15.007        7   0 2011-10-15 00:00:35 2011-10-15 18:00:45
## 5 D15 D15.009        7   0 2011-10-16 00:00:39 2011-10-16 18:00:49
## 6 D15 D15.011        6   0 2011-10-17 00:01:07 2011-10-17 15:01:03
##
##
##  infolocs provided. The following variables are available:
## [1] "pkey"
```

10. Code to change ltraj to a data.frame to summarize distance between locations for each daily burst

```
dfdeer <- ld(deer)
head(dfdeer)

#Code to get mean distance moved for each burst
dfdeer <- subset(dfdeer, !is.na(dfdeer$dist))#remove NAs from last location of a burst
mean_dist <- do.call(data.frame, aggregate(dfdeer$dist, by=list(dfdeer$burst),
    function(x) c(mean = mean(x), sd = sd(x), n=abs(length(x)))))
head(mean_dist)
#Write.table gives csv output of Summary
write.table(mean_dist, file = "Distance.csv", sep =",", row.names = TRUE,
  col.names = TRUE, qmethod ="double")
```

4.4 First Passage Time (FPT)

The first passage time (FPT) is a parameter often used to describe the scale at which patterns occur in a trajectory. For a given scale r, it is defined as the time required by the animals to pass through a circle of radius r. The mean first passage time scales proportionately to the square of the radius of the circle for an uncorrelated random walk (Johnson et al. 1992). Johnson et al. (1992) used this property to differentiate facilitated diffusion and impeded diffusion, according to the value of the coefficient of the linear regression log(FPT) = a * log(radius) + b. Under the hypothesis of a random walk, a should be equal to 2 (higher for impeded diffusion, and lower for facilitated diffusion). Note however, that the value of a converges to 2 only for large values of radius. Another use of the FPT was proposed that, instead of computing the mean of FPT, use the variance of the log(FPT). This variance should be high for scales at which patterns occur in the trajectory (e.g. area restricted search; Fauchald and Tverra 2003). This method is often used to determine the scale at which an animal searches for food.

The value fpt computes the FPT for each relocation and each radius, and for each animal. This function returns an object of class “fipati” (i.e., a list with one component per animal). Each component is a data frame with each column corresponding to a value of radii and each row corresponding to a relocation. An object of class fipati has an attribute named “radii” corresponding to the argument radii of the function fpt. meanfpt and varlogfpt return a data frame giving respectively the mean FPT and the variance of the log(FPT) for each animal (rows) and rach radius (column). These objects also have an attribute “radii”.

1. Exercise 3.4 - 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 “FPTscript.Rmd” and run code directly from the script

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

```
library(adehabitatLT)
library(chron)
library(sp)
library(rgdal)
```

5. Load in our mule deer dataset from previous exercises

```
muleys <-read.csv("DCmuleysedited.csv", header=T)

#Code to look at number of relocations per animal
table(muleys$id)
```

```
##
##  D12  D15  D16  D19   D4   D6   D8
##  120 2589 2157 1156 1304 1455  971
```

```
#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.50 & muleys$Lat > 37.3 & muleys$Long < -1
muleys <- newmuleys
```

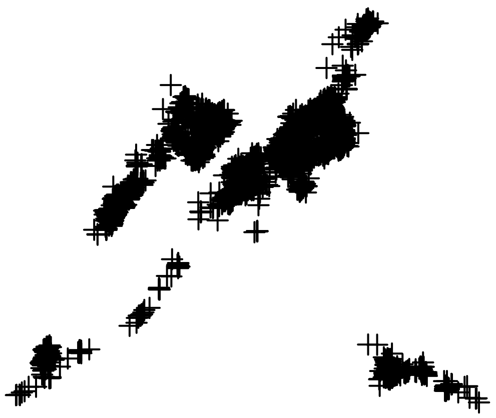
```
#Conversion of the date to the format POSIX as in previous exercise
da <- as.character(muleys$GPSFixTime)
da <- as.POSIXct(strptime(muleys$GPSFixTime,format="%Y.%m.%d %H:%M:%S"))
muleys$da <- da
```

6. Determine the time difference between each relocation for use later

```
timediff <- diff(muleys$da)*60*60
muleys <-muleys[-1,]
muleys$timediff <-as.numeric(abs(timediff))
```

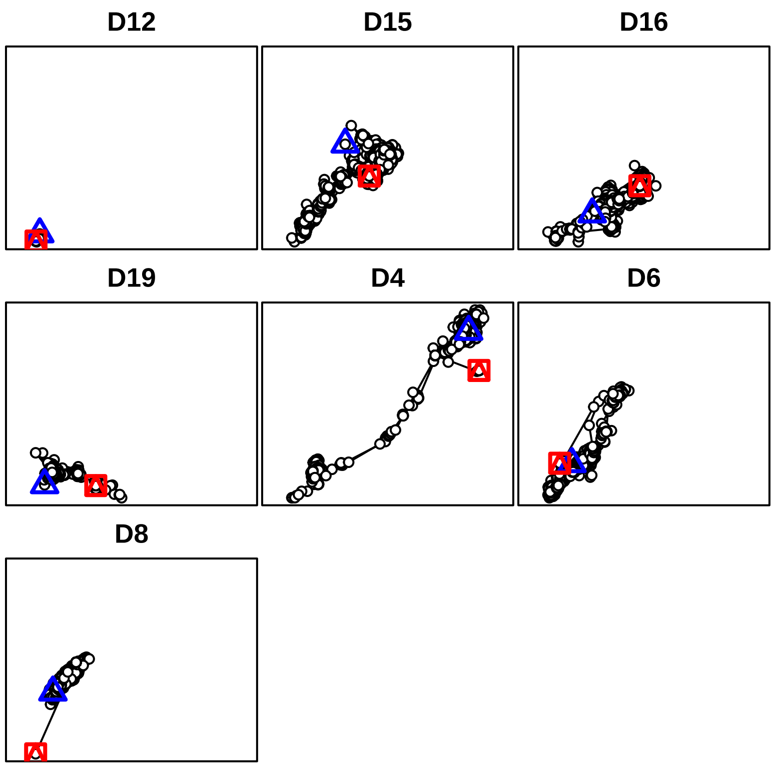
7. Create the spatial data from of xy coordinates and additional information

```
data.xy = muleys[c("X","Y")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)
#Creates a spatial data frame of ID
idsp<-data.frame(muleys[2])
#Creates a spatial data frame of dt
dtsp<-data.frame(muleys[24])
#Creates a spatial data frame of Burst
busp<-data.frame(muleys[23])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp,dtsp,busp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
plot(merge)
```



8. Create an object of class "ltraj" (i.e., trajectory) for all animals

```
ltraj <- as.ltraj(coordinates(merge),merge$da,id=merge$id)
plot(ltraj)
```



9. Code below actually creates First Passage Time and mean and variance of fpt

```
plot(ltraj[1])
i1 <- fpt(ltraj[1], seq(300,1000, length=30))
plot(i1, scale = 200, warn = FALSE)

plot(ltraj[2])
i2 <- fpt(ltraj[2], seq(300,1000, length=30))
plot(i2, scale = 500, warn = FALSE)

toto2 <- meanfpt(i2)
toto2
attr(toto2, "radii")

toto2 <- varlogfpt(i2)
toto2
attr(toto2, "radii")
```

```
plot(ltraj[3])
i3 <- fpt(ltraj[3], seq(300,1000, length=30))
plot(i3, scale = 500, warn = FALSE)

toto3 <- meanfpt(i3)
toto3
attr(toto3, "radii")

toto3 <- varlogfpt(i3)
toto3
attr(toto3, "radii")

plot(ltraj[4])
i4 <- fpt(ltraj[4], seq(300,1000, length=30))
plot(i4, scale = 500, warn = FALSE)

toto4 <- meanfpt(i4)
toto4
attr(toto4, "radii")

toto4 <- varlogfpt(i4)
toto4
attr(toto4, "radii")

plot(ltraj[5])
i5 <- fpt(ltraj[5], seq(300,1000, length=30))
plot(i5, scale = 500, warn = FALSE)

toto5 <- meanfpt(i5)
toto5
attr(toto5, "radii")

toto5 <- varlogfpt(i5)
toto5
attr(toto5, "radii")

plot(ltraj[6])
i6 <- fpt(ltraj[6], seq(300,1000, length=30))
plot(i6, scale = 500, warn = FALSE)

plot(ltraj[7])
i7 <- fpt(ltraj[7], seq(300,1000, length=30))
plot(i7, scale = 500, warn = FALSE)

toto7 <- meanfpt(i7)
```

```
toto7
attr(toto7, "radii")

toto7 <- varlogfpt(i7)
toto7
attr(toto7, "radii")
```

10. Code to export each trajectory as a shapefile if needed

```
toto1 <-ltraj2sldf(ltraj[1])
plot(toto1)
writeOGR(toto1,dsn=".",layer="D12", driver = "ESRI Shapefile",overwrite=TRUE)
summary(toto1)

#Write lines and points as a shapefile
toto2lines <-ltraj2sldf(ltraj[2],byid=TRUE)
toto2pts <- ltraj2spdf(ltraj[2])

#If we want to define projection before making a shapefile
proj4string <- CRS("+proj=utm +zone=13N +ellps=WGS84")
toto2lines@proj4string <- proj4string
toto2pts@proj4string <- proj4string

plot(toto2pts)
plot(toto2lines, add=T)

writeOGR(toto2pts,dsn=".",layer="D15pts", driver = "ESRI Shapefile",overwrite_layer=TRU
writeOGR(toto2lines, dsn=".", paste("traj_line_",sep=""),driver = "ESRI Shapefile",ove

toto3 <-ltraj2sldf(ltraj[3])
plot(toto3)
writeOGR(toto3,dsn=".",layer="D16", driver = "ESRI Shapefile",overwrite=TRUE)

toto4 <-ltraj2sldf(ltraj[4])
plot(toto4)
writeOGR(toto4,dsn=".",layer="D19", driver = "ESRI Shapefile",overwrite=TRUE)

toto5 <-ltraj2sldf(ltraj[5])
plot(toto5)
writeOGR(toto5,dsn=".",layer="D4", driver = "ESRI Shapefile",overwrite=TRUE)

toto6 <-ltraj2sldf(ltraj[6])
plot(toto6)
writeOGR(toto6,dsn=".",layer="D6", driver = "ESRI Shapefile",overwrite=TRUE)
```

```
toto7 <-ltraj2sldf(ltraj[7])
plot(toto7)
writeOGR(toto7,dsn=".",layer="D8", driver = "ESRI Shapefile",overwrite=TRUE)
```

4.5 Regular Trajectories

1. Exercise 3.5 - 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 "RegTrajScript.Rmd" and run code directly from the script

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

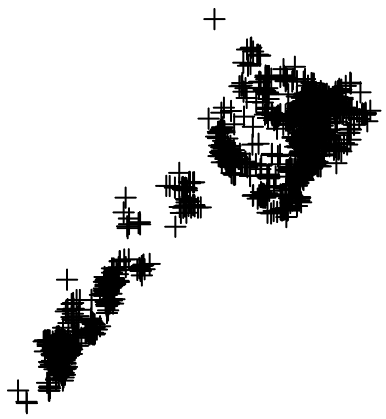
```
library(adehabitatLT)
library(chron)
library(sp)
```

5. Now read in dataset, extract a single animal and create ltraj as in previous exercise

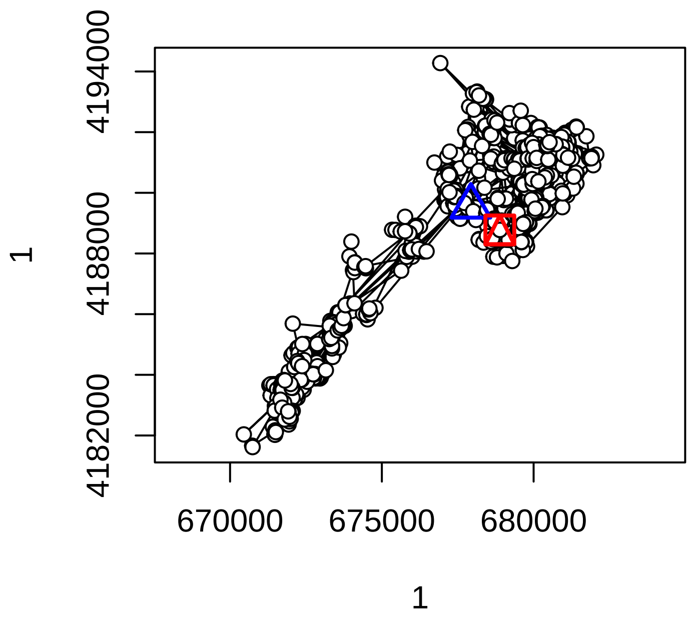
```
muleys <-read.csv("DCmuleysedited.csv", header=T)

#CODE FOR AN INDIVIDUAL ANIMAL
muley15 <- subset(muleys, id=="D15")
muley15$id <- factor(muley15$id)
table(muley15$id)
```

```
##
##  D15
## 2589
```

```
#Sort data to address error in code and then look at first 10 records of data to confirm
muley15 <- muley15[order(muley15$GPSFixTime),]

######################################################
## Example of a trajectory of type II (time recorded)
### Conversion of the date to the format POSIX
#Needs to be done to get proper digits of date into R then POSIXct
#uses library(chron)
da <- as.character(muley15$GPSFixTime)
da <- as.POSIXct(strptime(muley15$GPSFixTime,format="%Y.%m.%d %H:%M:%S"))
#Attach da to muley15
muley15$da <- da

timediff <- diff(muley15$da)
muley15 <-muley15[-1,]
muley15$timediff <-as.numeric(abs(timediff))

#Clean up muley15 for outliers
newmuleys <-subset(muley15, muley15$X > 599000 & muley15$X < 705000 & muley15$Y > 4167
  & muley15$timediff < 14401)
muley15 <- newmuleys

data.xy = muley15[c("X","Y")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)
#Creates a spatial data frame of ID
idsp<-data.frame(muley15[2])
#Creates a spatial data frame of dt
dtsp<-data.frame(muley15[24])
#Creates a spatial data frame of Burst
busp<-data.frame(muley15[23])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp,dtsp,busp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
plot(merge)
```



```
#Creation of an object of class "ltraj"
ltraj <- as.ltraj(coordinates(merge),merge$da,id=merge$id)
plot(ltraj)
```



6.We want to study the trajectory of the day at the scale of the day. We define one trajectory per day. The trajectory should begin at 2200 hours so the following function returns TRUE if the date is time between 06H00 and 23H00 (i.e. results in 7-8 locations/day bursts)

```
foo <- function(date) {
da <- as.POSIXlt(date)
ho <- da$hour + da$min
return(ho>18.0&ho<23.9)
}
deer <- cutltraj(ltraj, "foo(date)", nextr = TRUE)
```

```
## Warning in cutltraj(ltraj, "foo(date)", nextr = TRUE): At least 3 relocations are needed for a
##  27 relocations have been deleted
```

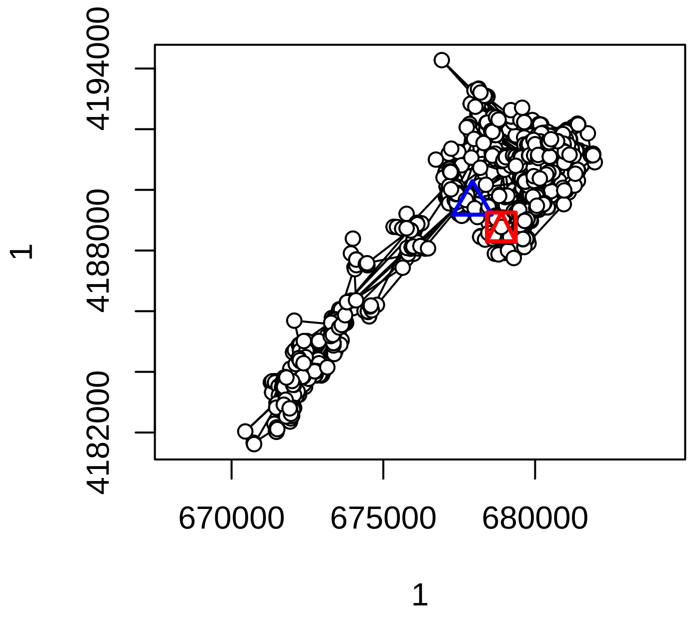
```
head(deer)
```

```
##
## *********** List of class ltraj ***********
##
## Type of the traject: Type II (time recorded)
## * Time zone unspecified: dates printed in user time zone *
## Irregular traject. Variable time lag between two locs
##
## Characteristics of the bursts:
##    id   burst nb.reloc NAs          date.begin            date.end
## 1 D15 D15.001        7   0 2011-10-12 03:00:52 2011-10-12 21:00:40
## 2 D15 D15.002        8   0 2011-10-13 00:00:35 2011-10-13 21:00:39
## 3 D15 D15.003        8   0 2011-10-14 00:00:42 2011-10-14 21:00:39
## 4 D15 D15.004        8   0 2011-10-15 00:00:35 2011-10-15 21:00:51
## 5 D15 D15.005        8   0 2011-10-16 00:00:39 2011-10-16 21:00:37
## 6 D15 D15.006        8   0 2011-10-17 00:01:07 2011-10-17 21:00:49
##
##
##  infolocs provided. The following variables are available:
## [1] "pkey"
```

```
## Remove the first and last burst if needed?
#deer2 <- deer[-c(1,length(deer))]

#Bind the trajectories
deer3 <- bindltraj(deer)
deer3
```

```
##
## *********** List of class ltraj ***********
##
## Type of the traject: Type II (time recorded)
## * Time zone unspecified: dates printed in user time zone *
## Irregular traject. Variable time lag between two locs
##
## Characteristics of the bursts:
##    id burst nb.reloc NAs          date.begin            date.end
## 1 D15   D15     2561   0 2011-10-12 03:00:52 2012-08-31 09:00:51
##
##
##  infolocs provided. The following variables are available:
## [1] "pkey"
```

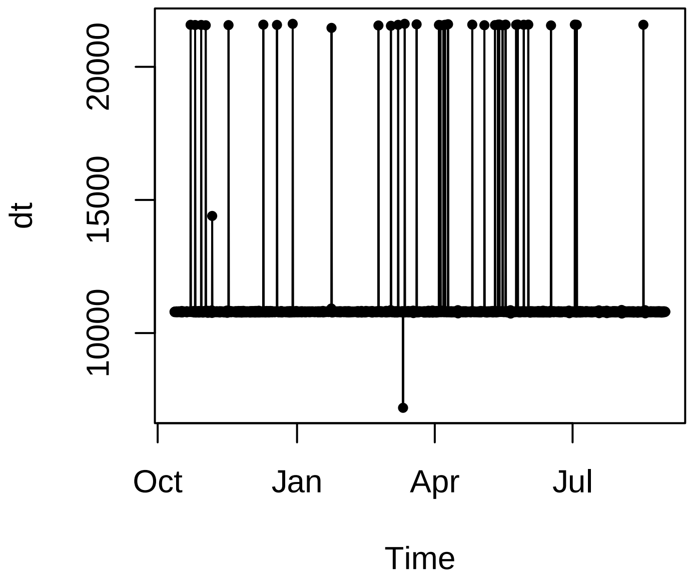
```
plot(deer3)
```



```r
is.regular(deer3)
```

```
## [1] FALSE
```

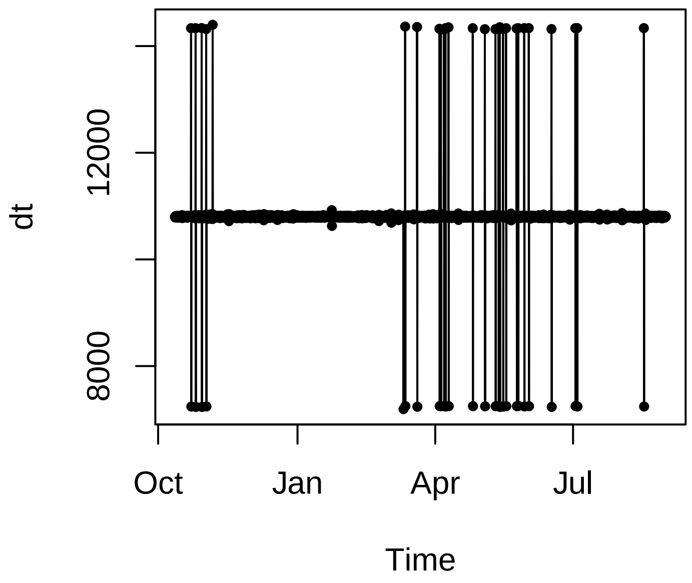
```r
plotltr(deer3, "dt")
```



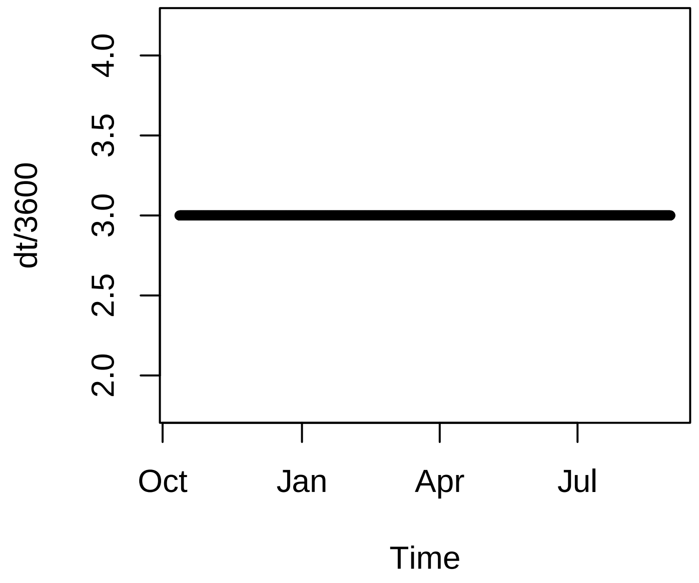
```r
## The relocations have been collected every 3 hours, and there are some
## missing data
## The reference date: the hour should be exact (i.e. minutes=0):
refda <- strptime("00:00", "%H:%M")
refda
```

```
## [1] "2023-12-05 EST"
```

```r
## Set the missing values
deerset <- setNA(deer3, refda, 3, units = "hour")
## now, look at dt for the bursts:
plotltr(deerset, "dt")
```



```r
## dt is nearly regular: round the date:
deerset1 <- sett0(deerset, refda, 3, units = "hour")
plotltr(deerset1, "dt/3600")
```



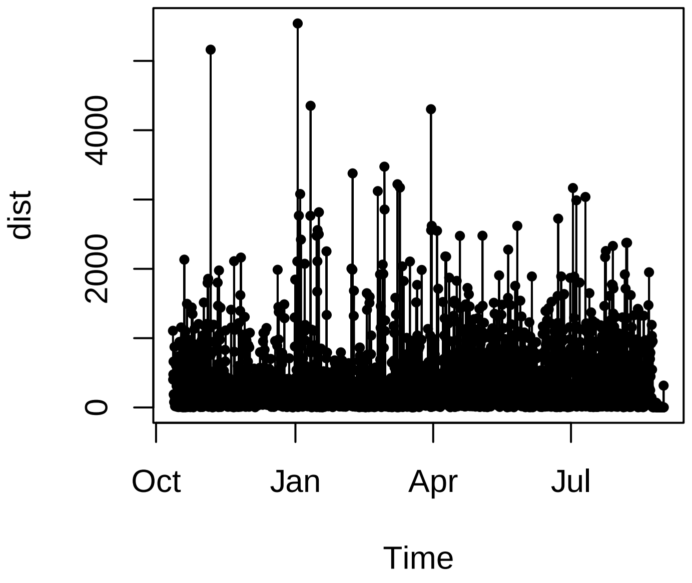
```
is.regular(deerset1)
```

```
## [1] TRUE
```

```
## deerset1 is now regular

## Is the resulting object "sd" ?
is.sd(deerset1)
```

```
## [1] TRUE
```

```
#Show the changes in the distance between successive relocations with the time
plotltr(deerset1, "dist")
```



```
deerset1#Is the trajectory regular now?
```

```
##
## *********** List of class ltraj ***********
##
## Type of the traject: Type II (time recorded)
## * Time zone unspecified: dates printed in user time zone *
## Regular traject. Time lag between two locs: 10800 seconds
##
## Characteristics of the bursts:
##    id burst nb.reloc NAs          date.begin            date.end
## 1 D15   D15     2595  34 2011-10-12 04:00:00 2012-08-31 10:00:00
##
##
##  infolocs provided. The following variables are available:
## [1] "pkey"
```

4.6 Net Squared Displacement

Net squared displacement (NSD) looks at the movement vectors of animals to determine their use of the landscape (Bunnefeld et al. 2011, Papworth et al. 2012). Bunnefeld et al. (2011) determined a novel method to identify movement patterns using NSD which is the straight line distance between an animals' starting location and subsequent locations. Movements were then categorized into one of 5 categories based on the top model that describes movement for each individual. In the code for this section, we have updated the code to include the 5 movement equations along with R code provided in Papworth et al. (2012) to enable determination of migratory, mixed migratory, disperser, home range, or nomadic movement behavior. Figure 1 in Bunnefeld et al. (2011) is helpful to interpret the output of this code that identifies the pattern of NSD over time that is determined by which of the 5 movement behaviors the animal follows. Those interested in this section should read the 2 papers cited for more details and specifics of the methods.

1. Exercise 3.6 - 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 "NSDScript.Rmd" and run code directly from the script

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

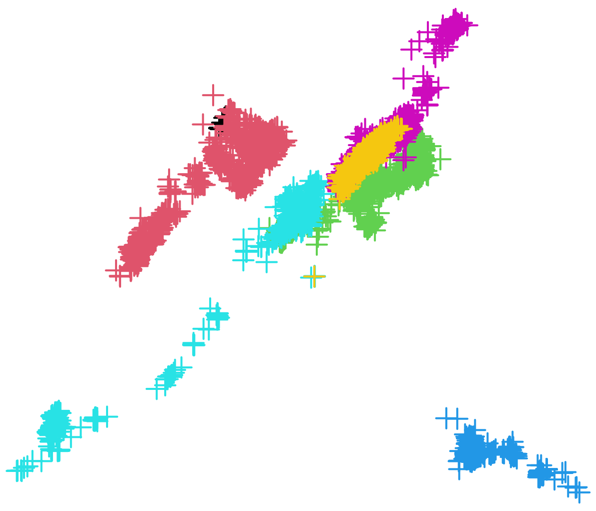
```
library(adehabitatHR)
library(trip)
library(lattice)
```

5. Now let's 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

```
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
utm.crs <-"+proj=utm +zone=12 +datum=WGS84"
```

6. Going to be using previous mule deer dataset from Colorado and clean up the data as in previous exercises

```
muleys<-read.csv("DCmuleysedited.csv", header=T, sep=",")

#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.50 & muleys$Lat > 37.3 & muleys$Long < -107)
muleys <- newmuleys

#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = muleys$Long, y = muleys$Lat)
deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys, proj4string = CRS(crs))
deer.spdf$id <- as.factor(deer.spdf$id)
plot(deer.spdf,col=deer.spdf$id)
```



```
muleys$NewDate<-as.POSIXct(muleys$GPSFixTime, format="%Y.%m.%d %H:%M:%S", origin="1970-

#TIME DIFF NECESSARY IN BBMM CODE
timediff <- diff(muleys$NewDate)*60*60
# remove first entry without any difference
muleys <- muleys[-1,]
muleys$timelag <-as.numeric(abs(timediff))
summary(muleys$timelag)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##     7182    10792    10800    40066    10809 27820757
```

```
#Remove locations greater than 24 hours apart in time
muleys <- subset(muleys, muleys$timelag < 18000)
```

7. The key to NSD is proper delineation of movement periods so we can explore a few alternatives here. First we will define deer and year based simply on the Year the location was recorded. Not very biologically meaningful but for simplicity we will start with calender year. Be sure to skip step 8 below and continue on with step 9 to the end of the exercise.

```
muleys$Year <- format(muleys$NewDate, "%Y")
muleys <- subset(muleys, muleys$Year != "NA")
muleys$YearBurst <- c(paste(muleys$id,muleys$Year,sep="_"))
muleys$YearBurst <- as.factor(muleys$YearBurst)
range(muleys$NewDate)
```

8. Alternatively, we could assign Year as when we would expect mule deer to disperse or migrate from summer to winter range. In our Colorado example,

the mule deer were captured on 30 September 2011 (winter range) so we will start there and separate out a summer season from May to August using the code below. Be sure to skip step 7 above and run this step instead but do not run both.

```
range(muleys$NewDate)
```

```
## [1] "2011-10-11 21:00:39 EDT" "2012-08-31 09:00:51 EDT"
```

```
muleys$Year2 <- NULL
muleys$Year2[muleys$NewDate > "2011-09-30 00:30:00" &
  muleys$NewDate < "2012-03-31 23:00:00"] <- 2011
muleys$Year2[muleys$NewDate > "2012-03-31 23:59:00" &
  muleys$NewDate < "2012-10-01 23:00:00"] <- 2012
muleys$Year2 <- as.factor(muleys$Year2)
muleys$YearBurst2 <- c(paste(muleys$id,muleys$Year2,sep="_"))
muleys$YearBurst2 <- as.factor(muleys$YearBurst2)
table(muleys$YearBurst2)
```

```
##
## D12_2011 D15_2011 D15_2012 D16_2011 D16_2012 D19_2011  D4_2011  D6_2011
##       98     1344     1178     1336      759     1106     1274     1323
##  D6_2012  D8_2011
##       84      956
```

```
#If needed we can remove deer that don't fit our minimum
#sample size requirement
muleys <- subset(muleys, table(muleys$YearBurst2)[muleys$YearBurst2] > 100)
muleys$YearBurst2 <- droplevels(muleys$YearBurst2)
```

9. Next we are going to rename our dataset to simply follow along with previous code I have created unless you want to change d1 to muleys throughout. This code will separate each deer by year which will make it easier to test NSD for each deer in our study

```
d1 <- muleys
#Code separate each animal into a shapefile or text file to use as a "List"
#Start with making an input file
indata <- d1
innames <- unique(d1$YearBurst2)
innames <- innames[1:8]#needs to be number of unique IDs
outnames <- innames
# begin loop to calculate home ranges
for (i in 1:length(innames)){
  data <- indata[which(indata$YearBurst2==innames[i]),]
  if(dim(data)[1] != 0){
    #data <-data[c(-21)]
    # export the point data into a shp file
    data.xy = data[c("X", "Y")]
    coordinates(data.xy) <- ~X+Y
    sppt <- SpatialPointsDataFrame(coordinates(data.xy),data)
    proj4string(sppt) <- CRS(utm.crs)
    #writePointsShape(sppt,fn=paste(outnames[i],sep="/"),factor2char=TRUE)
    #sppt <-data[c(-22,-23)]
    write.table(sppt, paste(outnames[i],"txt",sep="."), sep="\t", quote=FALSE, row.name
    write.table(paste(outnames[i],"txt",sep="."), sep="\t", quote=FALSE, row.names=FALS
      col.names=FALSE, "In_list.txt", append=TRUE)
#The write.table line above should only be run once to create the In_list.txt file oth
#it rights all animals each time

#Note: There are 3 lines of code that are not active that can #be activated to export f
#shapefiles for all resulting #animals but need to remove as.POSIX column first
}}
```

10. Code below will be needed to get NSD and best movement model for each animal

```
#Reads the List file of GPS datasets
List<-read.table("In_list.txt",sep="\t",header=F)
head(List) #List contains the filenames of the all the datasets in our study
```

```
##             V1
## 1 D15_2011.txt
## 2 D15_2012.txt
## 3 D16_2011.txt
## 4 D16_2012.txt
## 5 D19_2011.txt
## 6  D4_2011.txt
```

```
#(i.e., By YearBurst we created previoulsy)

# We will start by generating a vector of results we would like as the final
#output to our analyses
ID <- rep(0,nrow(List))
LOCS <- rep(0,nrow(List))
MIGR <- rep(0,nrow(List))
MIXM <- rep(0,nrow(List))
DISP <- rep(0,nrow(List))
```

```
HORA <- rep(0,nrow(List))
NOMA <- rep(0,nrow(List))
ID <- rep(0,nrow(List))
LOCS <- rep(0,nrow(List))
MIGR <- rep(0,nrow(List))
MIXM <- rep(0,nrow(List))
DISP <- rep(0,nrow(List))
HORA <- rep(0,nrow(List))
NOMA <- rep(0,nrow(List))
AICC_1 <- rep(0,nrow(List))
AICC_2 <- rep(0,nrow(List))
AICC_3 <- rep(0,nrow(List))
AICC_4 <- rep(0,nrow(List))
AICC_5 <- rep(0,nrow(List))

minAIC <- rep(0,nrow(List))
d_AICC_1 <- rep(0,nrow(List))
d_AICC_2 <- rep(0,nrow(List))
d_AICC_3 <- rep(0,nrow(List))
d_AICC_4 <- rep(0,nrow(List))
d_AICC_5 <- rep(0,nrow(List))
LL_AICC_1 <- rep(0,nrow(List))
LL_AICC_2 <- rep(0,nrow(List))
LL_AICC_3 <- rep(0,nrow(List))
LL_AICC_4 <- rep(0,nrow(List))
LL_AICC_5 <- rep(0,nrow(List))
sumLL_AICC <- rep(0,nrow(List))
wi_AICC_1 <- rep(0,nrow(List))
wi_AICC_2 <- rep(0,nrow(List))
wi_AICC_3 <- rep(0,nrow(List))
wi_AICC_4 <- rep(0,nrow(List))
wi_AICC_5 <- rep(0,nrow(List))
```

11. The remainder of the code will be within a loop to run all animals in our dataset individually. The vector above will be populated with our results each time an animal has finished running through all the code

```
for(i in 1:nrow(List)) {

coords<-read.table(as.character(List[i,]),sep="\t",header=T)
coords$DT<-as.POSIXct(coords$NewDate, format="%Y-%m-%d %H:%M:%S")

##Make a data.frame of coordinates. Here the raw values are divided
#by 1000 so that trajectories are calculated using km as the unit of measurement not meters
coord<-data.frame((coords$Y),(coords$X))
```

```
#Make ltraj: a trajectory of all the relocations
d2<-as.ltraj(coord,coords$DT,
coords$YearBurst2,        #separate your data by individual.
burst=coords$YearBurst2, #burst is used to create subdivisions within an individual.
typeII=TRUE)

#you can now make your trajectory regular
#first, create a reference start time
#refda <- strptime("00:00", "%H:%M")   #all relocations should be altered
#to occur at 30 seconds past each minute
#firstly create a reference start time
refda <- strptime("00:00:30", "%H:%M:%S")

#you can now make your trajectory regular, as radio tracks tend to lose
#a few seconds / minutes with each relocation
#firstly add "NA" for each missing location in your trajectory
d3<-setNA(d2,refda,
#as.POSIXct("2007-06-01 06:00:00 EDT"), #any time before earliest timedate
10800,           #stating there should be a location every 3 hours
tol=10800,       #how many time units to search each side of expected location
units="sec")   #specifying the time units

#you can now make your trajectory regular

#NOTE: The refda and d3 code above was not run as in Papworth because
#it results in too many relocations as "NA" that get removed below. Not
#quite sure the reason behind it being included?

#You can now make your trajectory regular
d4<-sett0(d3, refda,
10800,                        #stating the interval at which relocations should be
correction.xy =c("none"),   #if "cs" performs location correction based on the
#assumption the individual moves at a constant speed
tol=10800,      #how many time units to search either side of an expected location
units = "sec")  #specifying the time units

#to view your regular trajectory of points with NA's
summary(d4)
#now calculating NSD for each point
datansd<-NULL
for(n in 1:length(summary(d4)[,1])) #stating that NSD should be
#calculated separately for each burst
{
nsdall<-d4[[n]][,8]              #extracting the NSD for each location
nsdtimeall<-d4[[n]][,3]          #extracting the time for each location
nsdtimestartzero<-d4[[n]][,3]-d4[[n]][1,3]
#extracting the time since trip start for each location
nsdid<-rep(as.vector(summary(d4)[n,1]),
length.out=summary(d4)[n,3])
#extracting the individual associated with each location
nsdtrip<-rep(as.vector(summary(d4)[n,2]),length.out=summary(d4)[n,3])
#extracting the trip associated with each location
datansd1<-data.frame(nsdall,nsdtimeall,nsdtimestartzero,nsdid,nsdtrip)
#joining all these variables together in a data frame
datansd<-rbind(datansd,datansd1)
#joining all the data frames together
}
datansd$zero1<-as.numeric(unclass(datansd$nsdtimestartzero))
# making seconds since trip start numeric
datansd$zerostart<-datansd$zero1/60
#changing the time since trip start from seconds to minutes
datansd$minslitr2<-as.numeric(strftime(as.POSIXlt(datansd$nsdtimeall),
format="%M"))
#making a vector of the hour of the day a location occured
datansd$hdaylitr2<-as.numeric(strftime(as.POSIXlt(datansd$nsdtimeall),
format="%H"))
#making a vector of the minute in an hour a location occured
datansd$minsday<-((datansd$hdaylitr2*60)+datansd$minslitr2)
#calculating the minute in the day a location occured

summary(datansd)
datansd1<-na.omit(datansd)            #remove NA's

datansd1$coordinates<-coord           #add the coordinates for each point
#you now have the dataframe you need (datansd) to start analysis

#NSD
#table(datansd1$nsdid)

#Now you can start modelling NSD using nlme.
#Equations are from Bunnefeld at al (2011) A model-driven approach to quantify migration
#patterns:individual, regional and yearly differences.
#Journal of Animal Ecology 80:466-476

#First we are going to model the data using nls, a least squares method,
#the simplest method and first method in Bunnefeld et al. 2011 (i.e., MIGRATION)
#that uses a double sigmoid or s-shaped function.

###########################
##
```

```
##  MIGRATION
##
###########################

m1<-tryCatch(nls(nsdall ~  asym /(1+exp((xmidA-zerostart)/scale1)) +
(-asym / (1 + exp((xmidB-zerostart)/scale2))), #Equation 1 in Bunnefeld et al. 2011
start = c(asym=15000000,xmidA=200000,xmidB=450000,scale1=1000,scale2=1000)
#these are the starting values for each parameter of the equation
,data=na.omit(datansd1)),error=function(e)99999)   #this is the data
#summary(m1)        #this will print a summary of the converged model
#NOTE: The error function is simply to prevent the loop from crashing
#if model does not converge

###########################
##
##  MIXED MIGRATORY
##
###########################

m2 <-tryCatch(nls(nsdall ~  asymA /(1+exp((xmidA-zerostart)/scale1)) +
(-asymB / (1 + exp((xmidB-zerostart)/scale2))), #Equation 2 in Bunnefeld et al. 2011
start = c(asymA=15000000,asymB=10000000, xmidA=200000,xmidB=450000,scale1=1000,scale2=
#these are the starting values for each parameter of the equation
,data=na.omit(datansd1)),error=function(e)99999)   #this is the data
#summary(m2)

###########################
##
##  DISPERSAL
##
###########################

m3 <-tryCatch(nls(nsdall ~  asym /(1+exp((xmid-zerostart)/scale)),
start = c(asym=15000000,xmid=200000,scale=1000)#Equation 3 in Bunnefeld et al. 2011
#these are the starting values for each parameter of the equation
,data=na.omit(datansd1)),error=function(e)99999)   #this is the data
#summary(m3)

###########################
##
## HOME RANGE
##
###########################

m4 <- tryCatch(nls(nsdall ~ intercept, data=na.omit(datansd1),start = list(intercept =
  error=function(e)99999) #Equation 4 in Bunnefeld et al. 2011
#where c is a constant
#summary(m4)

###########################
##
## NOMADIC
##
###########################

m5 <- tryCatch(nls(nsdall ~ beta*zerostart,start=c(beta=1), data=na.omit(datansd1)),
  error=function(e)99999) #Equation 5 in Bunnefeld et al. 2011 where beta is a constant
#and t the number of days since initial start date (i.e., 1 June of each year)
#summary(m5)

#Below we are going to set up the AIC table
ID[i] <- paste(unique(as.factor(datansd$nsdid)))
LOCS[i] <- nrow(coords)
MIGR[i] <- print(tryCatch(AIC(m1),error=function(e)0))
MIXM[i] <- print(tryCatch(AIC(m2),error=function(e)0))
DISP[i] <- print(tryCatch(AIC(m3),error=function(e)0))
HORA[i] <- print(tryCatch(AIC(m4),error=function(e)0))
NOMA[i] <- print(tryCatch(AIC(m5),error=function(e)0))

AICC_1[i] <- print(tryCatch(AIC(m1),error=function(e)99999))
AICC_2[i] <- print(tryCatch(AIC(m2),error=function(e)99999))
AICC_3[i] <- print(tryCatch(AIC(m3),error=function(e)99999))
AICC_4[i] <- print(tryCatch(AIC(m4),error=function(e)99999))
AICC_5[i] <- print(tryCatch(AIC(m5),error=function(e)99999))

minAIC[i] <- min(AICC_1[i],AICC_2[i],AICC_3[i],AICC_4[i],AICC_5[i])

d_AICC_1[i] <- (AICC_1[i] - minAIC[i])
d_AICC_2[i] <- (AICC_2[i] - minAIC[i])
d_AICC_3[i] <- (AICC_3[i] - minAIC[i])
d_AICC_4[i] <- (AICC_4[i] - minAIC[i])
d_AICC_5[i] <- (AICC_5[i] - minAIC[i])

LL_AICC_1[i] <- exp(-0.5*d_AICC_1[i])
LL_AICC_2[i] <- exp(-0.5*d_AICC_2[i])
LL_AICC_3[i] <- exp(-0.5*d_AICC_3[i])
LL_AICC_4[i] <- exp(-0.5*d_AICC_4[i])
LL_AICC_5[i] <- exp(-0.5*d_AICC_5[i])

sumLL_AICC[i] <- sum(LL_AICC_1[i],LL_AICC_2[i],LL_AICC_3[i],LL_AICC_4[i],LL_AICC_5[i])
```

```r
wi_AICC_1[i] <- LL_AICC_1[i]/sumLL_AICC[i]
wi_AICC_2[i] <- LL_AICC_2[i]/sumLL_AICC[i]
wi_AICC_3[i] <- LL_AICC_3[i]/sumLL_AICC[i]
wi_AICC_4[i] <- LL_AICC_4[i]/sumLL_AICC[i]
wi_AICC_5[i] <- LL_AICC_5[i]/sumLL_AICC[i]

filename<-paste(substr(List[i,],1,8),"png",sep=".")
#NOTE:Numbers after "List[i,] need to encompass possible lengths of output name
#(i.e., D19.txt is 6 characters)
png(filename,height=20,width=30,units="cm",res=600)
#graphical exploration of the data will help you find sensible starting values
#for each of the parameters asym, xmidA, xmidB, scale1 and scale2.
#to graph nsd against time, use:
xyplot(nsdall~zerostart|nsdtrip,data=datansd)
#str(nsdtest)
#now plot the data with the predicted curve
nsdplot <- xyplot(nsdall ~ zerostart/3600, data=datansd1,
col="grey",    #color for the observed locations
type='b',      # 'b' shows the locations as dots, with a line connecting
#successive locations. Can also be 'p' for just the locations, or 'l' for just
#the line between locations
ylab=expression(paste('Net squared displacement ',' ', (km^2))), #y axis label
xlab="Hours after trip start")

plot(nsdplot)

dev.off()
}

#Create table of AIC values with lower AIC identifying best model
RESULT<-cbind(ID,LOCS,MIGR,MIXM,DISP,HORA,NOMA)
colnames(RESULT)<- c("ID","LOCS","MIGR","MIXM","DISP","HORA","NOMA")
#write.table(RESULT,"OUT_NSDresults.txt",sep="\t")

#Create table of raw values to calculate AICweights
Migratory <- rbind(ID,AICC_1,d_AICC_1,LL_AICC_1,wi_AICC_1)
MixedMig <- rbind(ID,AICC_2,d_AICC_2,LL_AICC_2,wi_AICC_2)
Disperser <- rbind(ID,AICC_3,d_AICC_3,LL_AICC_3,wi_AICC_3)
HomeRange <- rbind(ID,AICC_4,d_AICC_4,LL_AICC_4,wi_AICC_4)
Nomadic <- rbind(ID,AICC_5,d_AICC_5,LL_AICC_5,wi_AICC_5)
RESULT2 <- rbind(Migratory,MixedMig,Disperser,HomeRange,Nomadic)
write.csv(RESULT2,"OUT_NSDresults.csv")
```

4.7 Movement Trajectory Animation

1. Exercise 3.7 - 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 “TrajDynScript.Rmd” and run code directly from the script

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

```
library(adehabitatLT)
library(chron)
library(raster)
```

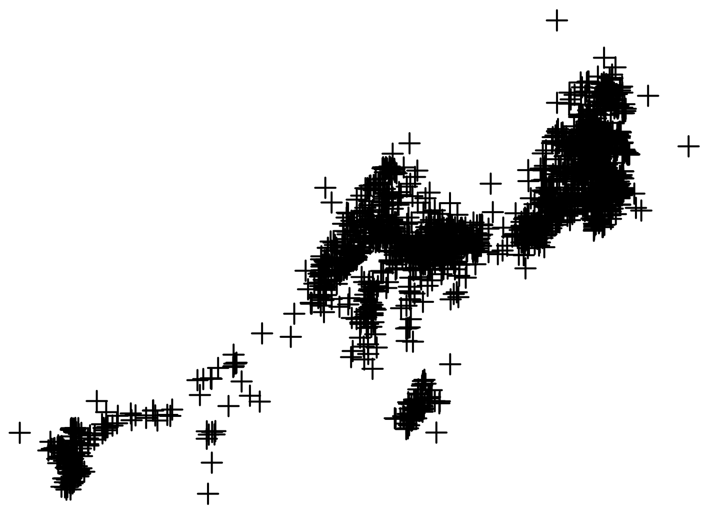
5. Now let’s have a separate section of code to include projection information we will use throughout the exercise. In previous versions, these lines of code were within each block of code

```
utm.crs <- "+proj=utm +zone=12 +ellps=WGS84"
Albers.crs <-CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0
    +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
```

6. We will be using the mule deer dataset for this exercise

```
muleys <-read.csv("DCmuleysedited.csv", header=T)

#CODE FOR AN INDIVIDUAL ANIMAL
muley16 <- subset(muleys, id=="D16")
muley16$id <- factor(muley16$id)
summary <- table(muley16$UTM_Zone,muley16$id)

#Sort data to address error in code and then look at first 10 records of data to confirm
muley16 <- muley16[order(muley16$GPSFixTime),]

######################################################
## Example of a trajectory of type II (time recorded)
### Conversion of the date to the format POSIX
#Needs to be done to get proper digits of date into R then POSIXct
#uses library(chron)
da <- as.character(muley16$GPSFixTime)
da <- as.POSIXct(strptime(muley16$GPSFixTime,format="%Y.%m.%d %H:%M:%S"))
#Attach da to muley15
muley16$da <- da
```

```
timediff <- diff(muley16$da)
muley16 <-muley16[-1,]
muley16$timediff <-as.numeric(abs(timediff))

#Clean up muley15 for outliers
newmuleys <-subset(muley16, muley16$X > 599000 & muley16$X < 705000 & muley16$Y > 4167
    & muley16$timediff < 14401)
muley16 <- newmuleys

data.xy = muley16[c("X","Y")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
proj4string(xysp) <- CRS("+proj=utm +zone=12 +ellps=WGS84")
#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)
#Creates a spatial data frame of ID
idsp<-data.frame(muley16[2])
#Creates a spatial data frame of dt
dtsp<-data.frame(muley16[24])
#Creates a spatial data frame of Burst
busp<-data.frame(muley16[23])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp,dtsp,busp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
plot(merge)
```



```
#Give dataset projection information then project to Albers
proj4string(merge) <- CRS(utm.crs)
deer.albers <-spTransform(merge, CRS=Albers.crs)
```

```
## Warning: PROJ support is provided by the sf and terra packages among others
```

7. Need to create a movement trajectory as we did in previous exercises

```
ltr.albers <- as.ltraj(coordinates(deer.albers),merge$da,id=merge$id)
```

8. Now let's have a little fun with these mule deer locations and explore

```
#Load vegetation raster layer textfile clipped in ArcMap
veg <-raster("extentnlcd2.txt")
```

9. Code below is used to just zoom in on all of our locations and crop within it so just select a study area around your locations using the drawExtent function below

```
plot(deer.albers)
e <- drawExtent()#click on top left of crop box and bottom right of crop box to create
#a polygon around all locations
newclip <- crop(veg,e)
plot(newclip)
points(deer.albers, col="red")
vegspdf <- as(newclip,"SpatialPixelsDataFrame")
plot(ltr.albers, spixdf=vegspdf)
```

10. #Or zoom in even closer on a few areas by repeating the drawExtent function above to a specific area

```
e2 <- drawExtent()
newclip2 <- crop(newclip,e2)
plot(newclip2)
points(deer.albers)
zoomspdf <- as(newclip2,"SpatialPixelsDataFrame")
zoom.ltr <- crop(deer.albers,zoomspdf)
ltr.zoom <- as.ltraj(coordinates(zoom.ltr),zoom.ltr$da,id=zoom.ltr$id)
plot(ltr.zoom, spixdf=zoomspdf)
```

11. We are first going to randomly select one location day for the calendar year our deer is monitored. This will result in fewer locations to plot overall. Then create an ltraj of the subset locations or all locations if you skipped lines 112-120 below

```
deer.albers$Year <- format(deer.albers$da, "%Y")
deer.albers <- subset(deer.albers,deer.albers$Year != "NA")
deer.albers$YearBurst <- c(paste(deer.albers$id,deer.albers$Year,sep="_"))
deer.albers$YearBurst <- as.factor(deer.albers$YearBurst)
range(deer.albers$da)

deer.albers$subDate <-  as.POSIXct(as.factor(deer.albers$da), format="%Y-%m-%d", tz="ES
deer.albers$Oneperday <- paste(deer.albers$YearBurst,deer.albers$subDate,sep="_")
deer.albers2 <- do.call(rbind, lapply(split(deer.albers,deer.albers$Oneperday) ,
function(deer.albers) deer.albers[sample(nrow(deer.albers), 1) , ] ))
ltr.year <- as.ltraj(coordinates(deer.albers2),deer.albers2$da,id=deer.albers2$id)
```

12. Now we can use the function to create movements of our deer over the landscape

```
windows() #NOTE: a new window is needed in Rstudio
#Line of code below plots trajectory one location at a time
trajdyn(ltr.year,spixdf=vegspdf)
```

Chapter 5

Home Range Estimation

5.1 Kernel Density Estimation (KDE) with reference bandwidth selection (href)

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 (href) was able to produce KDE in both Home Range Tools and adehabitat or adehabitatHR (Calenge 2007, 2011). Estimation with (href) 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 4.2 - 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 "HlscvScript.Rmd" and run code directly from the script

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

```
library(adehabitatHR)
```

5. Now let's 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=17 +ellps=WGS84")
```

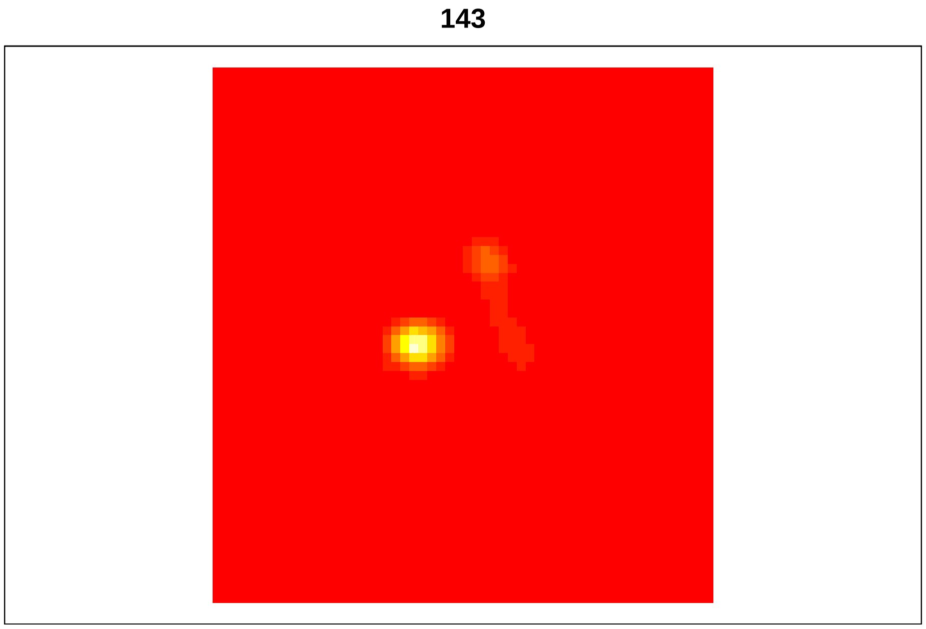
6. Now we can run fixed kernel home range with href bandwidth selection. The code below uses the original adehabitat package to run home range so skip if unable to load package

```
panther<-read.csv("pantherjitter.csv", header=T)
panther$CatID <- as.factor(panther$CatID)
#Note below the code uses the original adehabitat package to run home range
library(adehabitat)
loc <- panther[, c("X", "Y")]
## Estimation of UD for each animal separately
id <- panther[, "CatID"]
udbis <- kernelUD(loc, id, h = "href")
ud <- kernelUD(loc, id = id, h = "href", grid = 40, same4all = FALSE, hlim = c(0.1, 1.
    kern = c("bivnorm"), extent = 0.5)
image(ud) ## Note that the contours are under the locations

## Calculation of the 95 percent home range
ver <- getverticeshr(ud, 95)
plot(ver)

## Look at the estimates of home range by contour
cuicui1 <- kernel.area(loc, id)
plot(cuicui1)
cuicui1
# write output
write.table(cuicui1,"output.csv", row.names=TRUE, sep=" ", col.names=TRUE, quote=TRUE,
  na = "NA")


##########################################################
# 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 for each animal at each level
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=17 +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)
library(rgdal)
writeOGR(spdfTmp,"HREF","FP048HREF", "ESRI Shapefile")


kvtmp <- getverticeshr(udbis, lev = 99)
str(kvtmp)
plot(kvtmp[[2]])
```

```
kv$KHR99<- kvtmp[[2]]
kvtmp <- getverticeshr(udbis, lev = 95)
kv$KHR95<- kvtmp[[2]]
kvtmp <- getverticeshr(udbis, lev = 90)
kv$KHR90<- kvtmp[[2]]
kvtmp <- getverticeshr(udbis, lev = 75)
kv$KHR75<- kvtmp[[2]]
kvtmp <- getverticeshr(udbis, lev = 50)
kv$KHR50<- kvtmp[[2]]
kvtmp <- getverticeshr(udbis, lev = 25)
kv$KHR25<- kvtmp[[2]]

spolTmp <- kver2spol(kv,"+proj=utm +zone=17 +ellps=WGS84")
dfTmp <- data.frame(Isopleth=c("99","95","90","75","50","25"),row.names=c("KHR99","KHR
    "KHR90","KHR75","KHR50","KHR25"))
spdfTmp <- SpatialPolygonsDataFrame(spolTmp, dfTmp, match.ID = TRUE)
writeOGR(spdfTmp,"HREF","FP094HREF", "ESRI Shapefile")
```

7. Using the adehabitatHR package requires dataset to be formatted differently than previous package

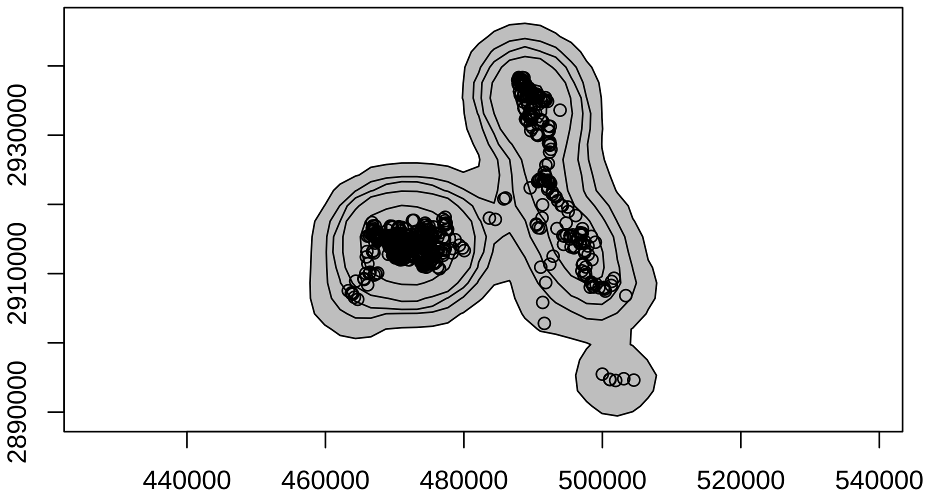
```
#Let's select only one animal
panther<-read.csv("pantherjitter.csv", header=T)
panther <- subset(panther, panther$CatID == "143")
panther$CatID <- factor(panther$CatID)
loc <- data.frame("x"=panther$X,"y"=panther$Y)
cats <- SpatialPointsDataFrame(loc,panther, proj4string = utm.crs)
udbis <- kernelUD(cats[,1], 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
```

```
## Object of class "SpatialPolygonsDataFrame" (package sp):
##
## Number of SpatialPolygons:  1
##
## Variables measured:
##      id     area
## 143 143 97882.18
```

```
plot(ver99, col="grey",axes=T);plot(ver95, add=T);plot(ver90, add=T);plot(ver80, add=T)
plot(ver50, add=T)
points(cats)
```



5.2 Kernel Density Estimation (KDE) with least squares cross validation (lscv)

Both the least squares cross-validation (hlscv) and bias crossed validation (hbcv) have been suggested instead of href in attempts to prevent over-smoothing of KDE (Rodgers and Kie 2010). However, (hlscv) and (hbcv) 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 (hplug-in; for description see section 3.3) at identifying distributions with tight clumps but risk of failure increases with hlscv 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 Session - Set Working Directory...

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

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

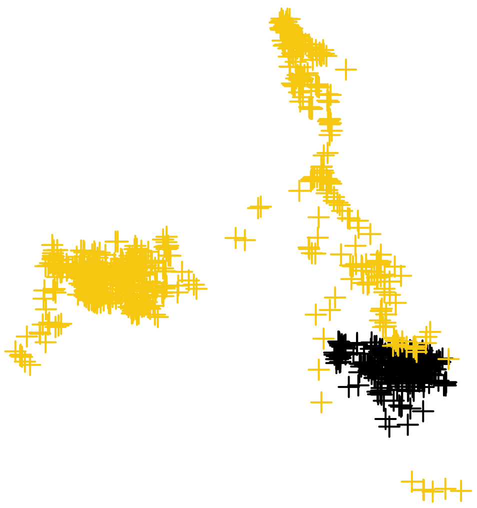
```r
library(adehabitatHR)
```

5. Now let's 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

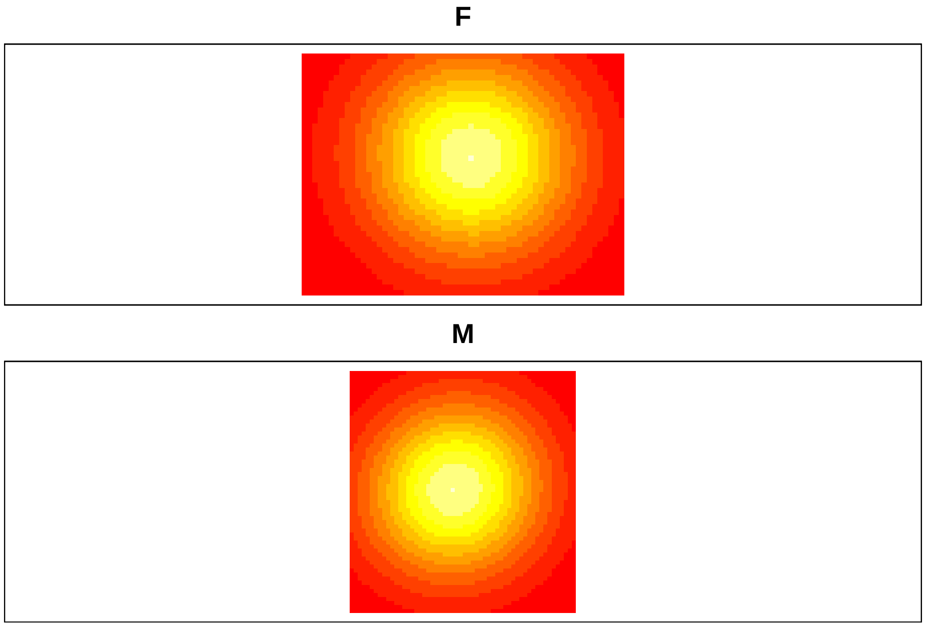
```
utm.crs <- CRS("+proj=utm +zone=17 +ellps=WGS84")
```

6. Now we can run fixed kernel home range with hlscv

```
panther <-read.csv("pantherjitter.csv", header=T)
loc <- data.frame("x"=panther$X,"y"=panther$Y)
pantherspdf <- SpatialPointsDataFrame(loc,panther, proj4string = utm.crs)
plot(pantherspdf, col=pantherspdf$CatID)
```

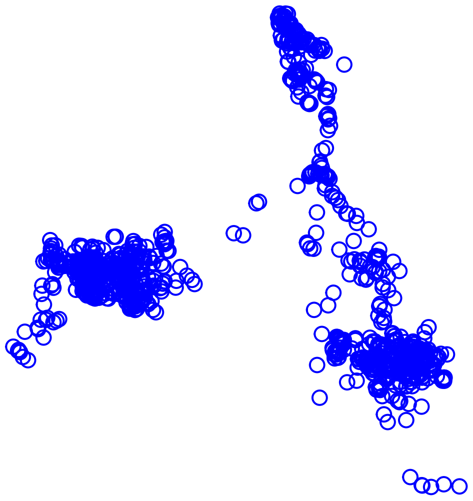


```
## Example of estimation using LSCV
udbis2 <- kernelUD(pantherspdf[,2], h = "LSCV", hlim = c(10,50),extent=1)
image(udbis2)
```

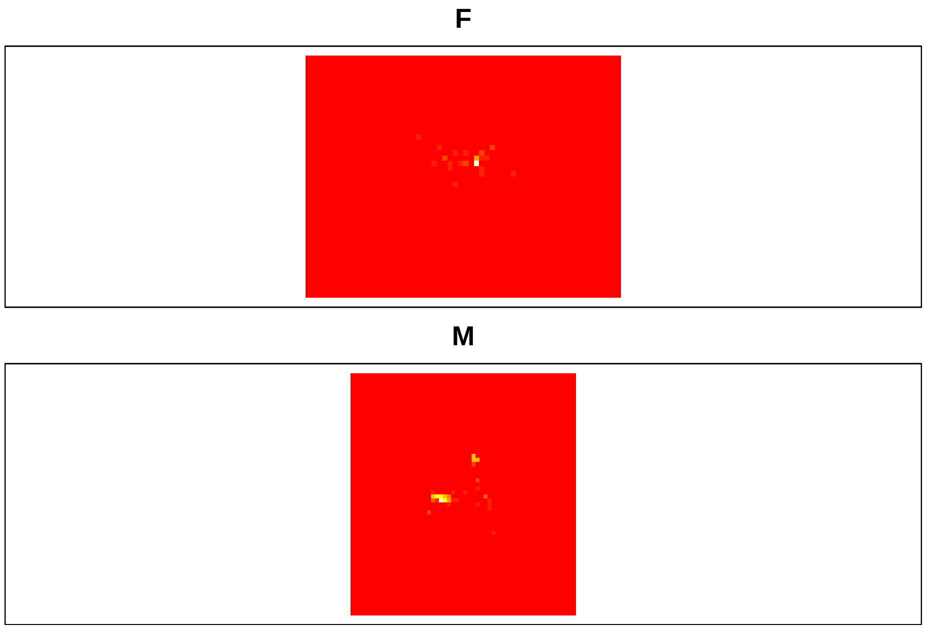


7. Note that regardless of change hlim or extent, LSCV will not converge for these animals so we can try a trick here. I believe LSCV is a poor estimator with GPS locations being too numerous and very close together compared to traditional VHF datasets which LSCV were originally evaluated. So we will 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=17 +ellps=WGS84")
jitterspdf <- SpatialPointsDataFrame(locjitter,panther, proj4string = proj4string)
plot(jitterspdf, col=pantherspdf$id)
points(pantherspdf, col="blue")
```



```
udbis3 <- kernelUD(jitterspdf[,2], h = "LSCV")#, hlim = c(1, 5),extent=1)
image(udbis3)
```



7. Now rerun with jitter factor = 100 instead of 50 and see what happens? Then rerun with jitter factor = 500 instead of 100 and see what happens?

5.3 KDE with plug-in bandwidth selection (hplug-in)

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

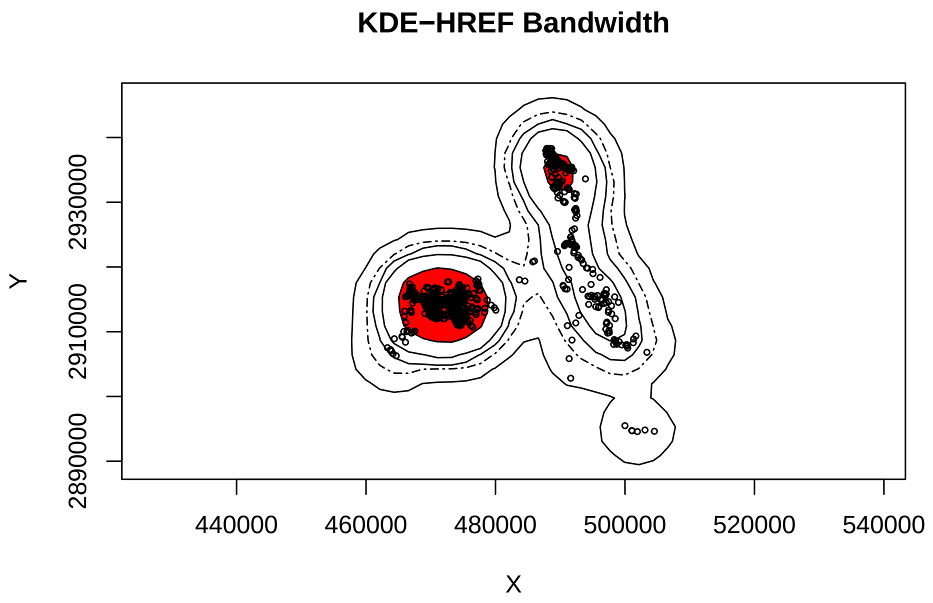
```
utm.crs <- CRS("+proj=utm +zone=17 +ellps=WGS84")
```

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.

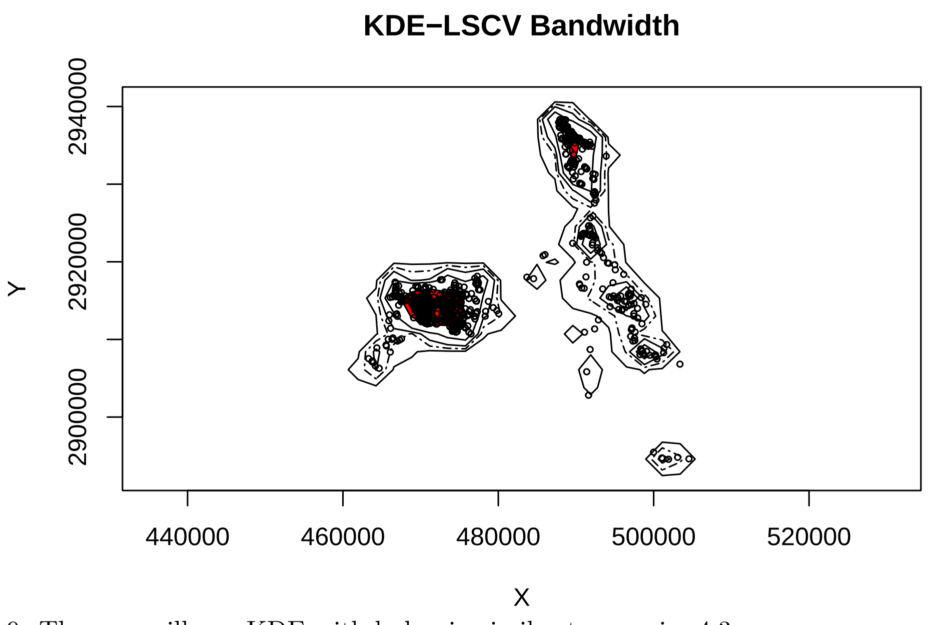
```
loc <- data.frame("x"=cat143$X,"y"=cat143$Y)

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

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)
```



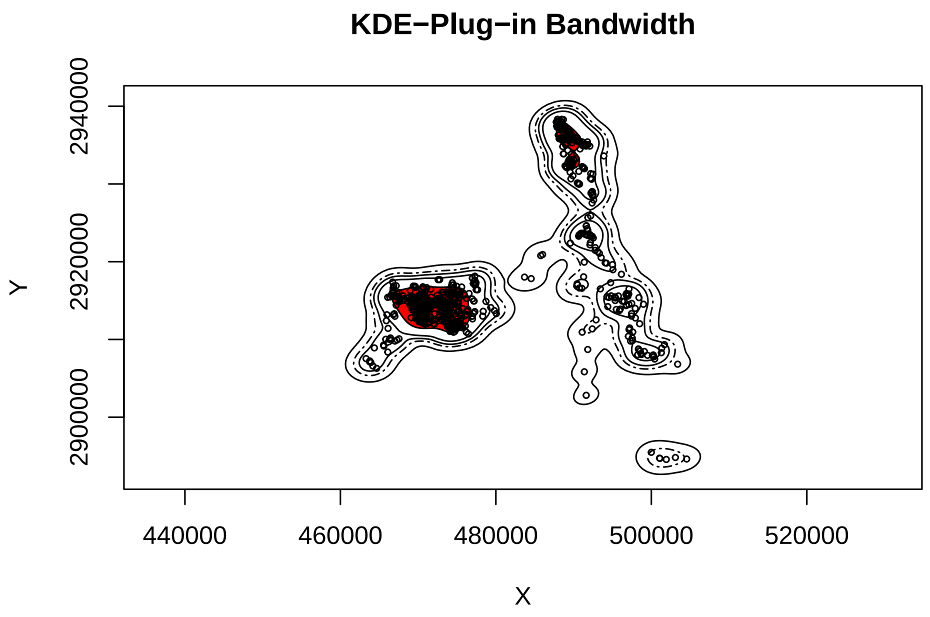
9. Then we will run KDE with hplug-in similar to exercise 4.3.

```
##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
```

```
##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
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
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=
cell.size=100)

bbmm.summary(BBMM)

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

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

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

# Convert to SpatialPolygonsDataFrame and export as ESRI Shapefile
```

```r
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)
# 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, ce
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, ax
# 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
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, axe
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, ce
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
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")
```

5.4 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 Session - Set Working Directory…

3. Now open the script “BBMMscript.Rmd” and run code directly from the script

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

```
#library(BBMM)
library(stringr)
library(rgdal)
library(maptools)
library(PBSmapping)
```

5. Now let’s 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

```
utm17.crs <- CRS("+proj=utm +zone=17 +ellps=WGS84")
```

6. Load panther dataset

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

7. First, we need to get Date and time into proper format for R because 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),]
```

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

```
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)
```

9. 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=
bbmm.summary(BBMM)
```

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

10. 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 hplug-in (Fig. 4.3) and href (Fig. 4.4). We start by creating a data.frame indicating cells within the contour desired and export as Ascii Grid

```r
# 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$probability)
contours = bbmm.contour(BBMM, levels=c(seq(50, 90, by=10), 95, 99), locations=cat143, plot=TRUE)
print(contours)
# Pick a contour for export as Ascii
bbmm.50 = bbmm.contour[bbmm.contour$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)
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)
data50 <- data.frame(PID = 1)
diss.map.p50 <- SpatialPolygonsDataFrame(diss.map.p50, data = data50)
plot(diss.map.p50)

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

# 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)
```

```r
plot(diss.map.p80)

# 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)
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)
plot(diss.map.p90)

# 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)
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)
plot(diss.map.p95)

# 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)
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)
plot(diss.map.p99)

# writeOGR(diss.map.p99, dsn = ".", layer="contour99", driver = "ESRI Shapefile")
# map.99 <- readOGR(dsn=".", layer="contour99")
# plot(map.99)
```

11. 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$timelag, location.error=34, cell.size=100)
bbmm.summary(BBMM2)
bbmm.contour1 = data.frame(x = BBMM2$x, y = BBMM2$y, probability = BBMM2$probability)
contours1 = bbmm.contour(BBMM2, levels=c(seq(50, 90, by=10), 95, 99), locations=loc, plot=TRUE)
```

12. Or we could exclude the extreme 1% of locations from cat143 based on time difference. This will result in subsetting data and only calculate BBMM with

timediff cutoff of <2940 minutes

```
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$percent[1]
for (j in 2:dim(freq)[1]){
freq$sum[j] <- freq$percent[j]+freq$sum[j-1]
}
indicator <- which(freq$sum>99)
cutoff <- as.numeric(as.character(freq$Var1[min(indicator)]))
cutoff

loc2 <- subset(cat143, cat143$timelag < 2940)
str(loc2)
BBMM3 = brownian.bridge(x=loc2$X, y=loc2$Y, time.lag=loc2$timelag, location.error=34, c
bbmm.summary(BBMM3)
bbmm.contour2 = data.frame(x = BBMM3$x, y = BBMM3$y, probability = BBMM3$probability)
contours2 = bbmm.contour(BBMM3, levels=c(seq(50, 90, by=10), 95, 99), locations=loc2, p
```

13.

```
#Plot results for all contours
contours2 = bbmm.contour(BBMM3, levels=c(seq(50, 90, by=10), 95, 99), locations=loc2, p

#Be sure to change bbmm.contours and rerun Steps 7-11 each time before running each se

raw.df <- data.frame("x"=cat143$X,"y"=cat143$Y)
##Define the projection of the coordinates
##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)

loc.df <- data.frame("x"=loc$X,"y"=loc$Y)
##Define the projection of the coordinates
proj4string <- CRS("+proj=utm +zone=17 +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)

loc2.df <- data.frame("x"=loc2$X,"y"=loc2$Y)
##Define the projection of the coordinates
proj4string <- CRS("+proj=utm +zone=17 +ellps=WGS84")
##Make SpatialPointsDataFrame using the XY, attributes, and projection
spdf2 <- SpatialPointsDataFrame(loc2.df, loc2, 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)
```

5.5 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 data with time intervals, we changed to 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 Session - Set Working Directory…

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

4. 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)#for function "crop"
library(stringr)
library(FedData)
```

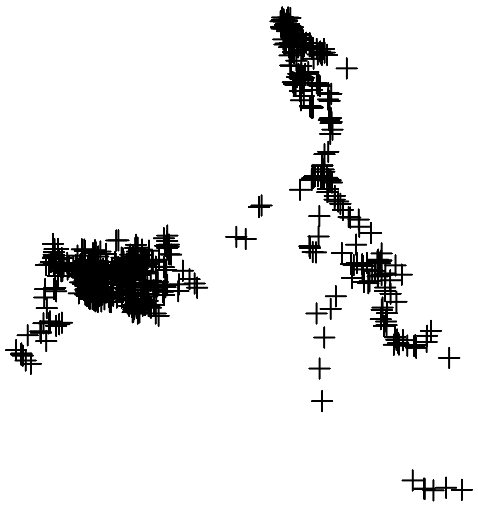
5. Now let's have a separate section of code to include projection information we will use throughout the exercise. In previous versions, these lines of code were within each block of code

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

6. Now open the script "MKDEscript.R" and run code directly from the script. We need to import our locations and get them in the form we need for this exercise before we can move forward

```
panther<-read.csv("pantherjitter.csv",header=T)
panther$CatID <- as.factor(panther$CatID)
#To run BBMM we first need to use the original dataset to calculate time between locat
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 <- droplevels(cat143$CatID)

data.xy = cat143[c("X","Y")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)

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

#Creates a spatial data frame of ID
```

```
idsp<-data.frame(cat143[2])
#Creates a spatial data frame of dt
dtsp<-data.frame(cat143[20])
#Creates a spatial data frame of Burst
busp<-data.frame(cat143[19])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp,dtsp,busp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
proj4string(merge) <- utm.crs
plot(merge)
```



```
# e <- drawExtent()
# e.poly <- as(e, "SpatialPolygons")
# e.poly@proj4string <- utm.crs
#select a polygon around locations to use later
#or read in raster created for this exercise later and skip to section 8
```

NOTE: Two issues at this step held me 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 ourexperiences 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.

7. 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).

```
FLnlcd <- get_nlcd(template=e.poly, 2006, label = 'Florida',force.redo = T)
habitat = as(FLnlcd, "SpatialPixelsDataFrame")
#writeRaster(FLnlcd,"FL_nlcd.tif",format="GTiff",datatype = 'INT1U',overwrite=T)
```

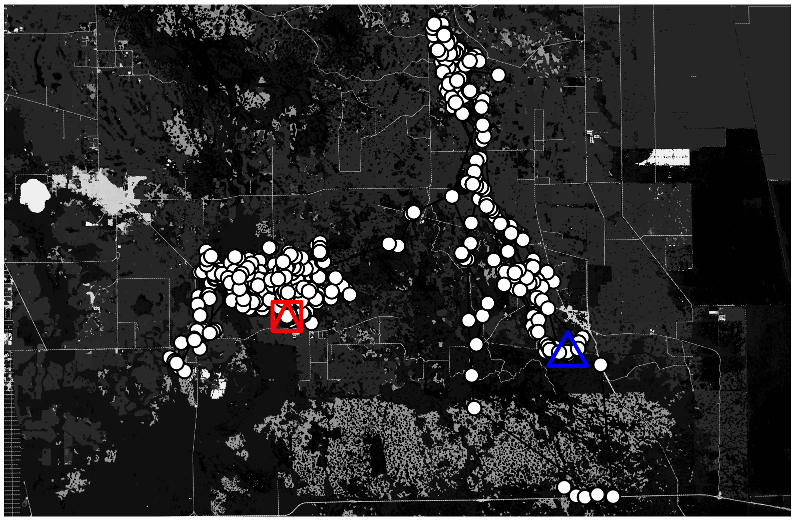
8. With the same projections for our 2 data layers, we can move forward. First we need to create ltraj as in Chapter 3 and use some additional code to overlay the trajectory onto the Spatial Pixels Data Frame using the command "spixdf" as in the code below that results in Fig. 4.11. Basically, if this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

```
FLnlcd <- raster("FL_nlcd.tif")
habitat = as(FLnlcd, "SpatialPixelsDataFrame")
merge.proj <- spTransform(merge, CRS=proj4string(FLnlcd))
```

```
## Warning: PROJ support is provided by the sf and terra packages among others
```

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

```
## Found more than one class "ltraj" in cache; using the first, from namespace 'spacet

## Also defined by 'trip'

## Found more than one class "ltraj" in cache; using the first, from namespace 'spacet

## Also defined by 'trip'
```



9. Now identify habitats that can and can not be used by panthers

```
#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]]))

#This step is needed to convert SpatialGrid to SpatialPixels for use in "ud" estimation
#if needed
#"habitat" in "grid=habitat" must be of class SpatialPixels
fullgrid(hab) <- FALSE
class(hab)
```

```
## [1] "SpatialPixelsDataFrame"
## attr(,"package")
## [1] "sp"
```

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

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

```
vv

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

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

```
ver1_99 <- getverticeshr(ud, percent=99, standardize = TRUE, whi = id(ltraj))
plot(ver1_99)
#ver1_95 <- getverticeshr(ud, percent=95, standardize = TRUE, whi = id(ltraj))
#ver1_90 <- getverticeshr(ud, percent=90, standardize = TRUE, whi = id(ltraj))
#ver1_80 <- getverticeshr(ud, percent=80, standardize = TRUE, whi = id(ltraj))
ver1_50 <- getverticeshr(ud, percent=50, standardize = TRUE, whi = id(ltraj))
plot(ver1_99)
#plot(ver1_95, add=T)
plot(ver1_50, add=T)
```

13. 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
m <- c(0, 19, 1, 20, 39, 2, 40, 50, 3, 51, 68, 4, 69,79, 5, 80, 88, 6, 89, 99, 7)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(FLnlcd, rclmat)
habitat2 <- as(rc, "SpatialPixelsDataFrame")

#CODE TO CONDUCT BRB
#Assign 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
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

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

```
names(ud2) <- id(ltraj)

ver2_99 <- getverticeshr(ud2, percent=99, standardize = TRUE, whi = id(ltraj))
#ver2_95 <- getverticeshr(ud2, percent=95, standardize = TRUE, whi = id(ltraj))
#ver2_90 <- getverticeshr(ud2, percent=90, standardize = TRUE, whi = id(ltraj))
#ver2_80 <- getverticeshr(ud2, percent=80, standardize = TRUE, whi = id(ltraj))
ver2_50 <- getverticeshr(ud2, percent=50, standardize = TRUE, whi = id(ltraj))
```

14. 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

```
#4 habitats instead of the 7 above with water, open, and forested
m1 <- c(0,39, 1, 40, 68, 2, 69,88, 3, 89, 99, 4)
rclmat1 <- matrix(m1, ncol=3, byrow=TRUE)
rc1 <- reclassify(FLnlcd, rclmat1)
habitat3 <- as(rc1, "SpatialPixelsDataFrame")

#Diffusion component for each habitat type using plug-in method
vv3<- BRB.D(ltraj, Tmax = tmax, Lmin = lmin,  habitat = habitat3)
vv3

ud3 <- BRB(ltraj, D = vv3, Tmax = tmax, Lmin = lmin, hmin=hmin, habitat = habitat3, b=TRUE, exter

names(ud3) <- id(ltraj)

ver3_99 <- getverticeshr(ud3, percent=99, standardize = TRUE, whi = id(ltraj))
#ver3_95 <- getverticeshr(ud3, percent=95, standardize = TRUE, whi = id(ltraj))
#ver3_80 <- getverticeshr(ud3, percent=80, standardize = TRUE, whi = id(ltraj))
ver3_50 <- getverticeshr(ud3, percent=50, standardize = TRUE, whi = id(ltraj))
```

15. Now we will plot these for comparison with the habitat layer differing by each analysis.

```
par(mfrow=c(1,3))
plot(ver1_99,main="mKDE no habitat",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
#plot(ver1_95, lty=6, add=TRUE)
#plot(ver1_90, add=TRUE)
#plot(ver1_80, add=TRUE)
plot(ver1_50, col="red",add=TRUE)
points(merge, pch=1, cex=0.5)

plot(ver2_99,main="mKDE 7 habitats",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
#plot(ver2_95, lty=6, add=TRUE)
```

```r
#plot(ver2_90, add=TRUE)
#plot(ver2_80, add=TRUE)
plot(ver2_50,col="red", add=TRUE)
points(merge, pch=1, cex=0.5)

plot(ver3_99,main="mKDE 4 habitats",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
#plot(ver3_95, lty=6, add=TRUE)
#plot(ver3_80, add=TRUE)
plot(ver3_50,col="red", add=TRUE)
points(merge, pch=1, cex=0.5)
```

5.6 Dynamic Brownian Bridge Movement Models (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 href/hplug-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). Estimating a movement path over the entire trajectory of data, however, should be separated into behavioral movement patterns (i.e., resting, feeding) prior to estimating the variance of the Brownian motion. Overestimating the variance 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 Session - Set Working Directory…

3. Now open the script “CatdBBMM.Rmd” and run code directly from the script

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

```r
library(adehabitatLT)
library(move)
library(rgdal)
library(adehabitatHR)
library(maptools)
library(PBSmapping)
library(stringr)
```

5. Now let's 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=17 +ellps=WGS84")
```

6. Read in panther dataset we have used previously

```
#Creates a Spatial Points Data Frame for 2 animals by ID
panther<-read.csv("pantherjitter.csv",header=T)
panther$CatID <- as.factor(panther$CatID)
#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 <- droplevels(cat143$CatID)
cat143.xy <- data.frame(x=cat143$X, y=cat143$Y)
cat143.spdf <- SpatialPointsDataFrame(coords=cat143.xy, data = cat143, proj4string = utm.crs)
```

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

```
loc <- move(x=cat143$X, y=cat143$Y, time=as.POSIXct(cat143$DT,
     format="%Y %m %d %H%M"), proj=utm.crs,data=cat143,
     animal=cat143$CatID)
min(timeLag(x=loc,units="mins"))
```

```
## [1] 418
```

8. Now create a dBBMM object

```
cat_dbbmm <- brownian.bridge.dyn(object=loc, location.error=34, window.size=19, margin=7,
dimSize=500,time.step=180)
plot(cat_dbbmm)
```

9. We can then explore the isopleth sizes and manipulate package move output to plot isopleths like previous exercises and write them out as shapefiles if needed

```r
contour(cat_dbbmm, levels=c(.5,.9,.95,.99))
show(cat_dbbmm)

#Plot the movement of the animal
plot(loc, type="o", col=3, lwd=2, pch=20, xlab="location_east",ylab="location_north")

#Code below will get area of each isopleth
cat_cont <- getVolumeUD(cat_dbbmm)
cat_cont50 <- cat_cont<=.50
cat_cont95 <- cat_cont<=.95
area50 <- sum(values(cat_cont50))
area50
area95 <- sum(values(cat_cont95))
area95

##Cast the data over to an adehabitatHR estUD
dbbmm.px <- as(cat_dbbmm, "SpatialPixelsDataFrame")
image(dbbmm.px)
dbbmm.ud <- new("estUD",dbbmm.px)
dbbmm.ud@vol = FALSE
dbbmm.ud@h$meth = "dBBMM"

shp99 <- getverticeshr(dbbmm.ud, percent=99, standardize=TRUE)
plot(shp99, add=TRUE)
map.ps99 <- SpatialPolygons2PolySet(shp99)
diss.map.99 <- as.PolySet(map.ps99, 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)
plot(diss.map.p99)
# writeOGR(diss.map.p99, dsn = ".", layer="catcontour99", driver = "ESRI Shapefile")
# catmap.99 <- readOGR(dsn=".", layer="catcontour99")

shp95 <- getverticeshr(dbbmm.ud, percent=95, standardize=TRUE)
plot(shp95, add=TRUE)
map.ps95 <- SpatialPolygons2PolySet(shp95)
diss.map.95 <- as.PolySet(map.ps95, 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)
plot(diss.map.p95, add=T)
#writeOGR(diss.map.p95, dsn = ".", layer="catcontour95", driver = "ESRI Shapefile")
#catmap.95 <- readOGR(dsn=".", layer="catcontour95")
```

```
shp90 <- getverticeshr(dbbmm.ud, percent=90, standardize=TRUE)
map.ps90 <- SpatialPolygons2PolySet(shp90)
diss.map.90 <- as.PolySet(map.ps90, 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)
plot(diss.map.p90,add=T)
# writeOGR(diss.map.p90, dsn = ".", layer="catcontour90", driver = "ESRI Shapefile")
# catmap.90 <- readOGR(dsn=".", layer="catcontour90")

shp80 <- getverticeshr(dbbmm.ud, percent=80, standardize=TRUE)
map.ps80 <- SpatialPolygons2PolySet(shp80)
diss.map.80 <- as.PolySet(map.ps80, 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)
plot(diss.map.p80,add=T)
# writeOGR(diss.map.p80, dsn = ".", layer="catcontour80", driver = "ESRI Shapefile")
# catmap.80 <- readOGR(dsn=".", layer="catcontour80")

shp50 <- getverticeshr(dbbmm.ud, percent=50, standardize=TRUE)
map.ps50 <- SpatialPolygons2PolySet(shp50)
diss.map.50 <- as.PolySet(map.ps50, projection = 'UTM', zone = '17')
diss.map.p50 <- PolySet2SpatialPolygons(diss.map.50, close_polys = TRUE)
data50 <- data.frame(PID = 1)
diss.map.p50 <- SpatialPolygonsDataFrame(diss.map.p50, data = data50)
plot(diss.map.p50,add=T)
# writeOGR(diss.map.p50, dsn = ".", layer="catcontour50", driver = "ESRI Shapefile")
# catmap.50 <- readOGR(dsn=".", layer="catcontour50")
```

10. We can plot out the 50-99% isopleths to compare to previous estimators in size and shape around locations

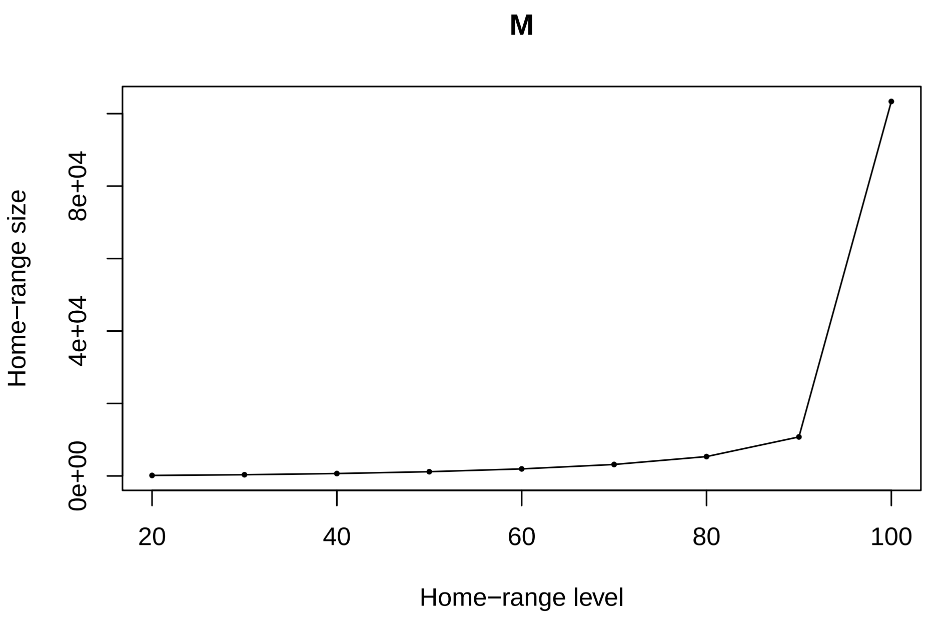
```
plot(diss.map.p99)
plot(diss.map.p95, add=T)
plot(diss.map.p90,add=T)
plot(diss.map.p80,add=T)
plot(diss.map.p50,add=T)
points(cat143.spdf,pch=1, cex=0.5)
```

11. Now we will shift towards polgyon-based estimators of home range to compare them to dBBMM. We will start with Characteristic Hull Polygons (CHP) in adehabitatHR package using the CharHull function.

```
data.xy = cat143[c("X","Y")]

#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
proj4string(xysp) <- utm.crs

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

#Creates a spatial data frame of ID
idsp<-data.frame(cat143[2])

#Merges ID data frame with GPS locations data frame
#Data frame is called "idsp" comparable to the "relocs" from puechabon dataset
coordinates(idsp)<-sppt

#Home Range estimation
res <- CharHull(idsp[,1])
class("res")
```

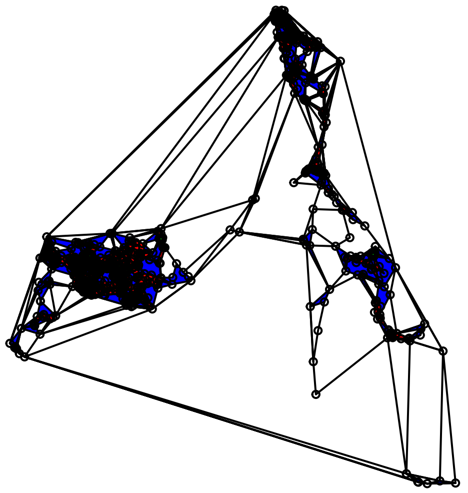
```
## [1] "character"
```

```
#Computes the home range size for 20-100 percent
MCHu2hrsize(res)
```



```
##                M
## 20      144.4545
## 30      340.6498
## 40      664.8405
## 50     1173.5487
## 60     1949.0595
## 70     3170.9746
## 80     5344.8134
## 90    10767.9219
## 100 103361.1336
```

```
#OR use

res_ver99 <- getverticeshr(res, percent=99)
res_ver95 <- getverticeshr(res, percent=95)
res_ver90 <- getverticeshr(res, percent=90)
res_ver80 <- getverticeshr(res, percent=80)
res_ver50 <- getverticeshr(res, percent=50)

plot(res_ver99)
plot(res_ver95,add=T)
plot(res_ver90, add=TRUE, col="blue")
plot(res_ver80, add=TRUE, col="red")
plot(res_ver50, add=T, col="green")
points(cat143.spdf,pch=1, cex=0.5)
```

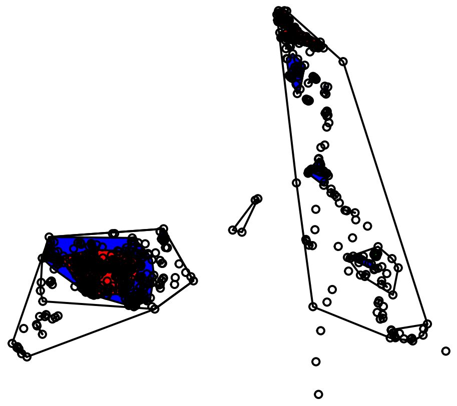


11. Next we will estimate home range with the Single-linkage Cluster (SLCA) using the clusthr function

```
uu <- clusthr(idsp)
class(uu)
```

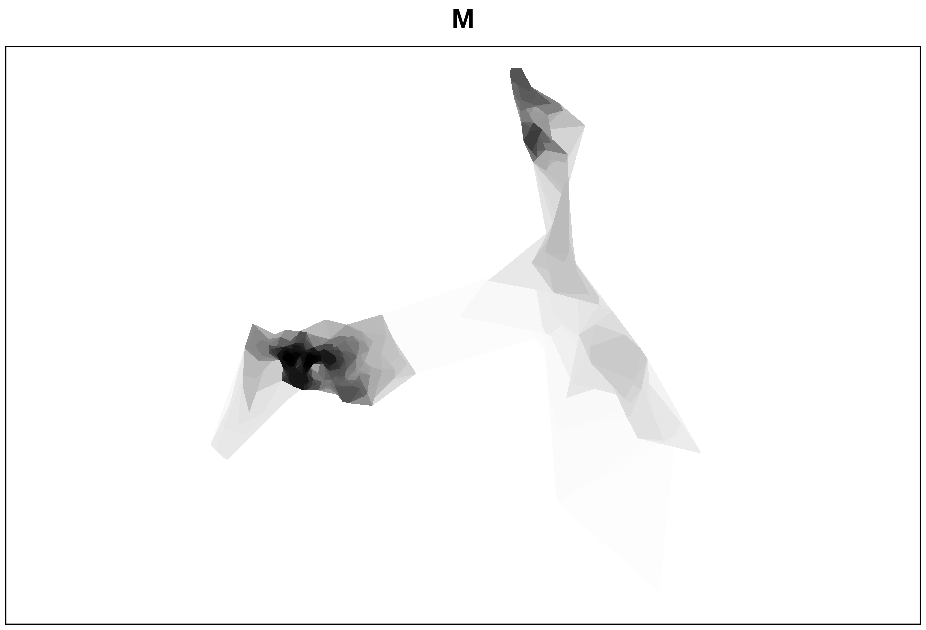
```
## [1] "MCHu"
```

```
uu_ver99 <- getverticeshr(uu, percent=99)
uu_ver95 <- getverticeshr(uu, percent=95)
uu_ver90 <- getverticeshr(uu, percent=90)
uu_ver80 <- getverticeshr(uu, percent=80)
uu_ver50 <- getverticeshr(uu, percent=50)

plot(uu_ver99)
plot(uu_ver95,add=T)
plot(uu_ver90, add=TRUE, col="blue")
plot(uu_ver80, add=TRUE, col="red")
plot(uu_ver50, add=T, col="green")
points(cat143.spdf,pch=1, cex=0.5)
```

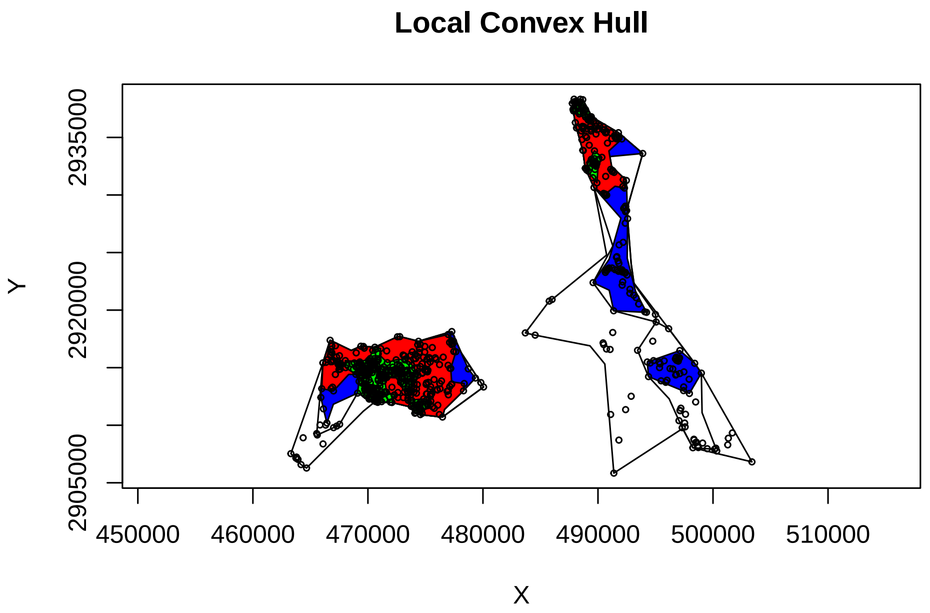


12./ Next we will explore Local Convex Hull (LoCoH)

```
## Exams the changes in home-range size for various values of k
## Be patient! the algorithm can be very long
#LoC.area <- LoCoH.k.area(idsp, k=c(5:40))
#NOTE: The line of code above does not run for this animal

## the k-LoCoH method:
nn <- LoCoH.k(idsp[,1], k=30)
## Graphical display of the results
plot(nn, border=NA)
```



```r
## the object nn is a list of objects of class

#Save shapefiles of resulting home range
ver <- getverticeshr(nn)

#writeOGR(ver,dsn="FixedK",layer="FixedK24", driver = "ESRI Shapefile", overwrite=TRUE)
##Overwrite will not work so must edit path so "FixedK" folder is created with code below.
nn_ver50 <-getverticeshr(nn, percent=50)
#writeOGR(ver50,dsn="FixedK",layer="50FixedK24", driver = "ESRI Shapefile",overwrite=TRUE)
nn_ver80 <-getverticeshr(nn, percent=80)
#writeOGR(ver80,dsn="FixedK",layer="80FixedK24", driver = "ESRI Shapefile",overwrite=TRUE)
nn_ver90 <-getverticeshr(nn, percent=90)
#writeOGR(ver90,dsn="FixedK",layer="90FixedK24", driver = "ESRI Shapefile",overwrite=TRUE)
nn_ver95 <-getverticeshr(nn, percent=95)
#writeOGR(ver95,dsn="FixedK",layer="95FixedK24", driver = "ESRI Shapefile",overwrite=TRUE)
nn_ver99 <-getverticeshr(nn, percent=99)
#writeOGR(ver99,dsn="FixedK",layer="99FixedK24", driver = "ESRI Shapefile",overwrite=TRUE)

plot(nn_ver99,main="Local Convex Hull",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(nn_ver95,add=T)
plot(nn_ver90, add=TRUE, col="blue")
plot(nn_ver80, add=TRUE, col="red")
plot(nn_ver50, add=T, col="green")
points(cat143.spdf,pch=1, cex=0.5)
```



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

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

plot(diss.map.p99,main="dynamic BBMM",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(diss.map.p95, add=T)
plot(diss.map.p90,add=T)
plot(diss.map.p80,add=T)
plot(diss.map.p50,add=T)
points(cat143.spdf,pch=1, cex=0.5)

plot(res_ver99,main="Characteristic Hull Polygons",xlab="X", ylab="Y", font=1, cex=0.8
plot(res_ver95,add=T)
plot(res_ver90, add=TRUE, col="blue")
plot(res_ver80, add=TRUE, col="red")
plot(res_ver50, add=T, col="green")
points(loc, pch=1, cex=0.5)
points(cat143.spdf,pch=1, cex=0.5)

plot(uu_ver99,main="Single-linkage Cluster",xlab="X", ylab="Y", font=1, cex=0.8, axes=
plot(uu_ver95,add=T)
plot(uu_ver90, add=TRUE, col="blue")
plot(uu_ver80, add=TRUE, col="red")
plot(uu_ver50, add=T, col="green")
points(cat143.spdf,pch=1, cex=0.5)

plot(nn_ver99,main="Local Convex Hull",xlab="X", ylab="Y", font=1, cex=0.8, axes=T)
plot(nn_ver95,add=T)
plot(nn_ver90, add=TRUE, col="blue")
plot(nn_ver80, add=TRUE, col="red")
plot(nn_ver50, add=T, col="green")
points(cat143.spdf,pch=1, cex=0.5)
```

5.7 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 Session - Set Working Directory…

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

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

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

5. Now let's 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

```
utm17.crs <- CRS("+proj=utm +zone=17 +ellps=WGS84")
```

6. Load panther dataset

```
#Creates a Spatial Points Data Frame for 2 animals by ID
twocats <-read.csv("pantherjitter.csv", header=T)
data.xy = twocats[c("X","Y")]

#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
proj4string(xysp) <- CRS("+proj=utm +zone=17 +ellps=WGS84")
```

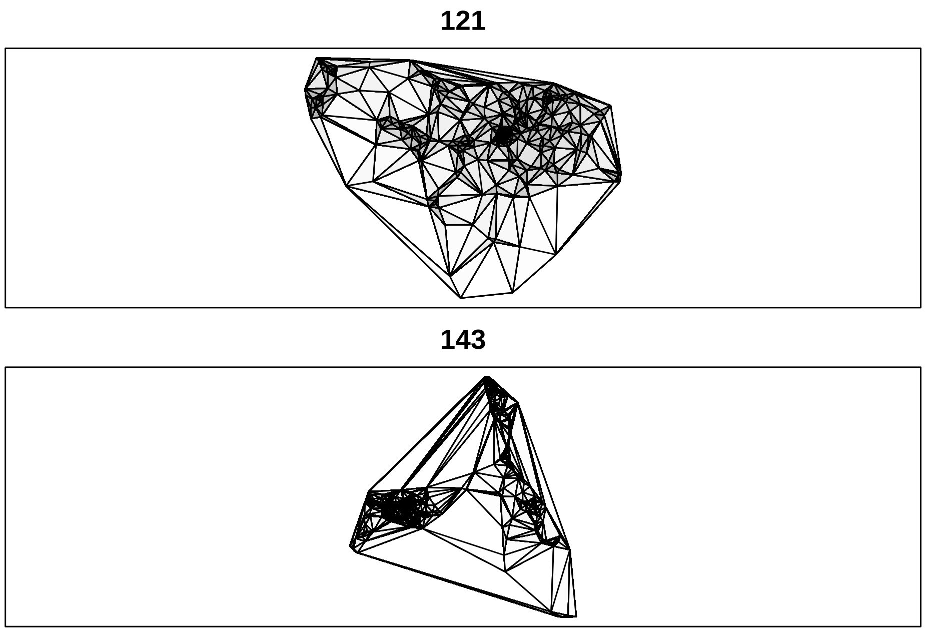
```
#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)

#Creates a spatial data frame of ID
idsp<-data.frame(twocats[1])

#Merges ID data frame with GPS locations data frame
#Data frame is called "idsp" comparable to the "relocs" from puechabon dataset
coordinates(idsp)<-sppt

head(as.data.frame(idsp))
```

```
##   CatID        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
```

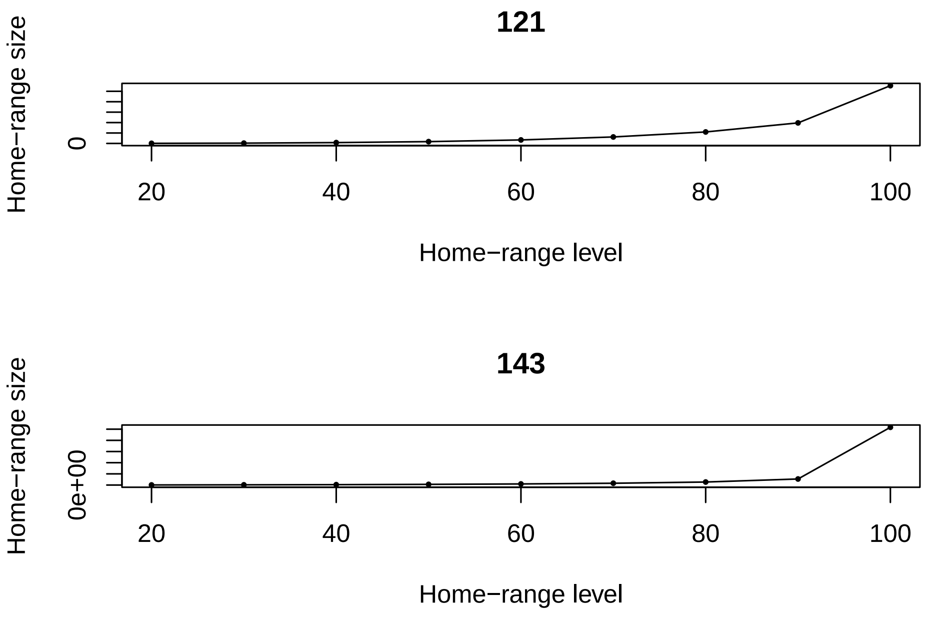
```
#Results for above code

#Home Range estimation
res <- CharHull(idsp[,1])
class("res")
```

```
## [1] "character"
```

```
#Displays the home range
plot(res)
```

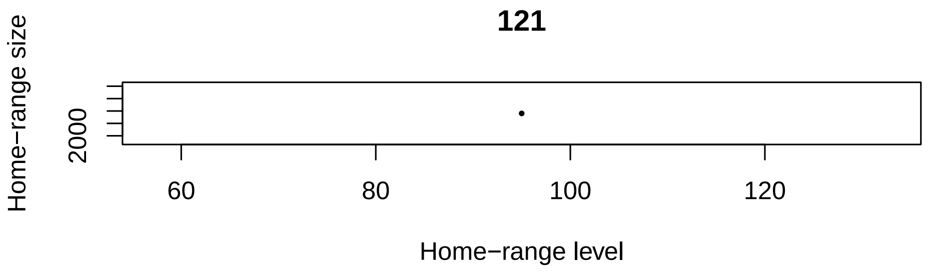


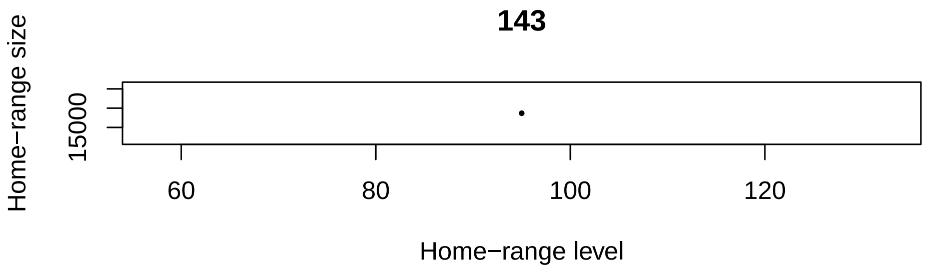
```
#Computes the home range size for 20-100 percent
MCHu2hrsize(res)
```



```
##           121      143
## 20   9.813887 145.4998
## 30    28.240966    340.6394
## 40    79.150785    666.6799
## 50   172.953172   1176.7365
## 60   333.989391   1957.3329
## 70   619.054096   3201.4047
## 80  1100.273001   5366.1080
## 90  1969.957499  10812.4950
## 100 5538.224019 103361.1336
```

```
#OR

#Computes the home range size for 95 percent
MCHu2hrsize(res, percent=95)
```





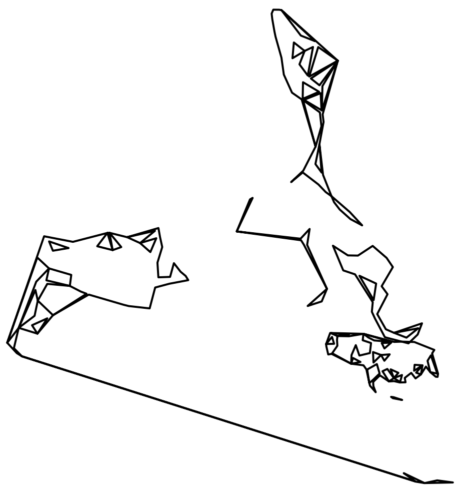
```
##          121      143
## 95 2903.147 18678.06
```

```
#OR use

ver <- getverticeshr(res, percent=90)
ver
```

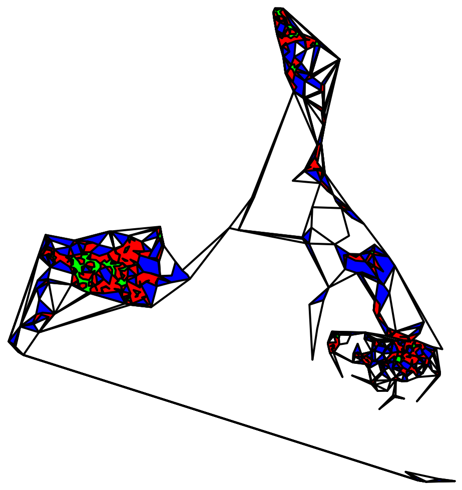
```
## Object of class "SpatialPolygonsDataFrame" (package sp):
##
## Number of SpatialPolygons:  2
```

```
##
## Variables measured:
##    id      area
## 1 121  1952.205
## 2 143 10745.761
```

```r
plot(ver)
```



```r
ver50 <- getverticeshr(res, percent=50)
ver80 <- getverticeshr(res, percent=80)
ver90 <- getverticeshr(res, percent=90)
ver95 <- getverticeshr(res, percent=95)
plot(ver95)
plot(ver90, add=TRUE, col="blue")
plot(ver80, add=TRUE, col="red")
plot(ver50, add=T, col="green")
```

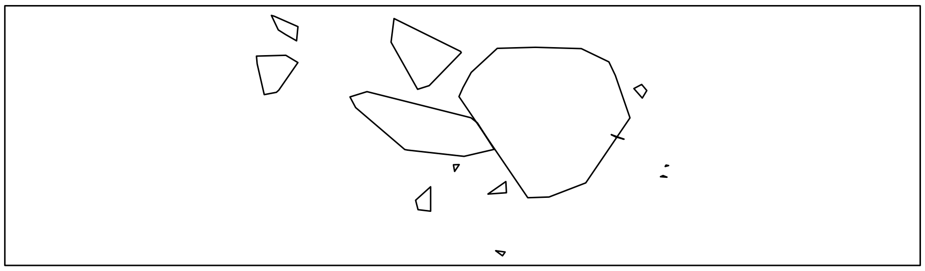


```
#The object uu below is the single-linkage cluster analysis that estimates
#home range returned as a list of SpatialPolygonDataFrame objects (one per animal)

uu <- clusthr(idsp)
class(uu)
```

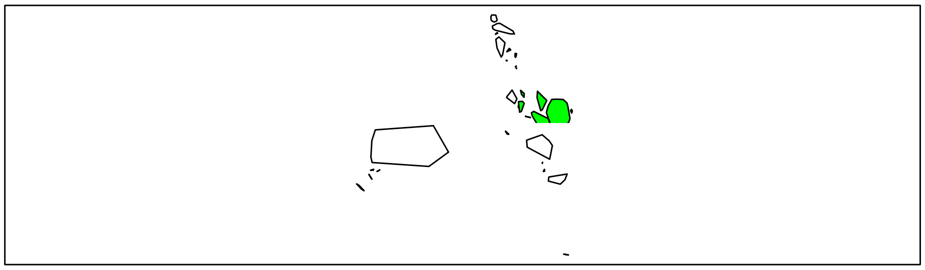
```
## [1] "MCHu"
```

```
#[1] "MCHu"

plot(uu, percent=95)

#Returns home range of Cat143 and 95% HR
plot(uu[[1]][250,], add=T, col="green")
```

121



143



5.8 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 4.8 - 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 “CodeHRscript.Rmd” and run code directly from the script

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

```r
library(adehabitatHR)
library(rgdal)
```

5. Now let's 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

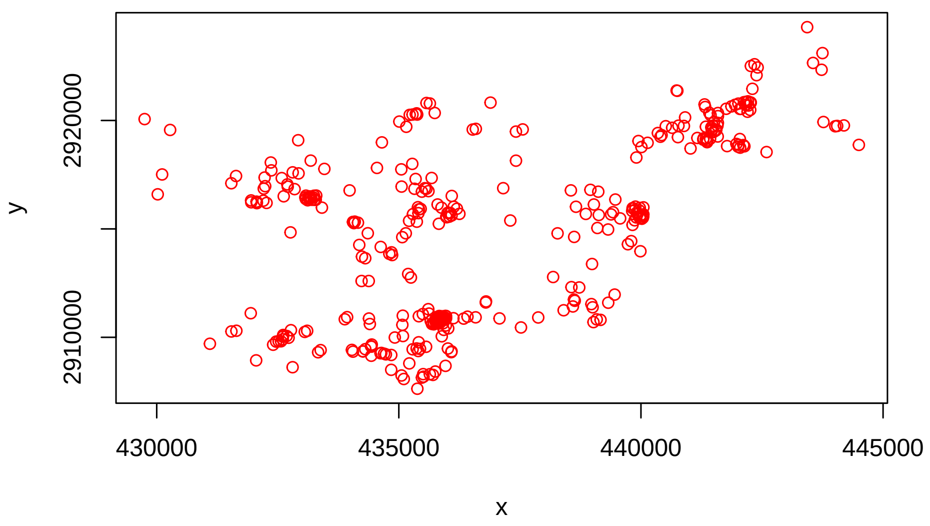
```r
utm17.crs <- CRS("+proj=utm +zone=17 +ellps=WGS84")
```

6. Run code directly from the script

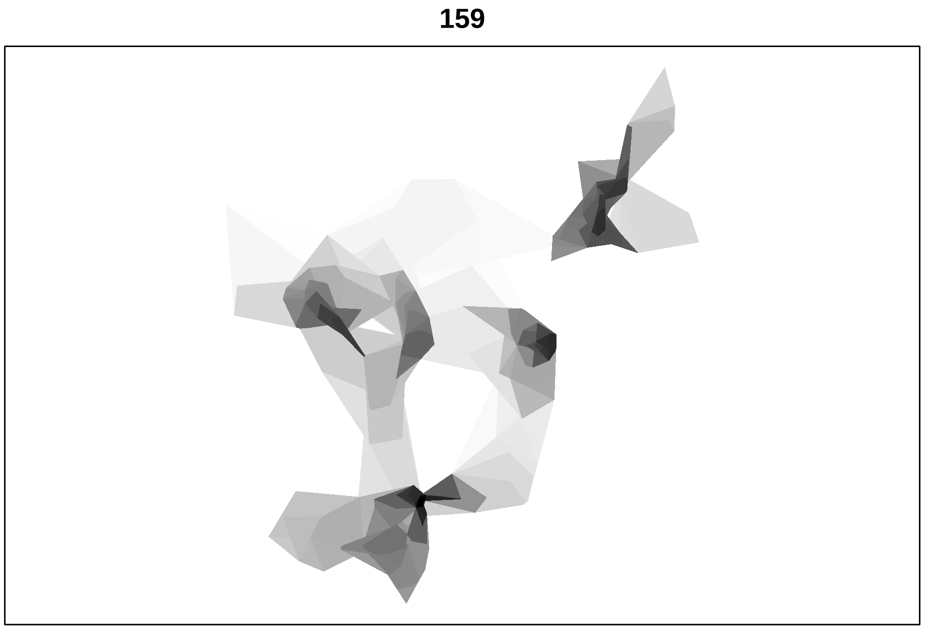
```r
#Get input file
panther <- read.csv("pantherjitter2.csv")
panther$CatID <- as.factor(panther$CatID)

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

```
##      CatID        x       y
## 4310   159 435794.4 2910781
## 4311   159 435756.7 2910734
## 4312   159 435840.8 2910796
## 4313   159 435880.1 2910750
## 4314   159 435845.9 2910839
## 4315   159 435962.5 2910821
```

```r
## Shows the relocations
plot(data.xy, col="red")
```



```r
## Examinates the changes in home-range size for various values of k
## Be patient! the algorithm can be very long
#LoC.area <- LoCoH.k.area(idsp, k=c(18:24))
## 24 points seems to be a good choice (rough asymptote for all animals)
## the k-LoCoH method:
nn <- LoCoH.k(idsp[,1], k=19)
## Graphical display of the results
plot(nn, border=NA)
```



```
## the object nn is a list of objects of class
## SpatialPolygonsDataFrame
length(nn)
```

```
## [1] 1
```

```
names(nn)
```

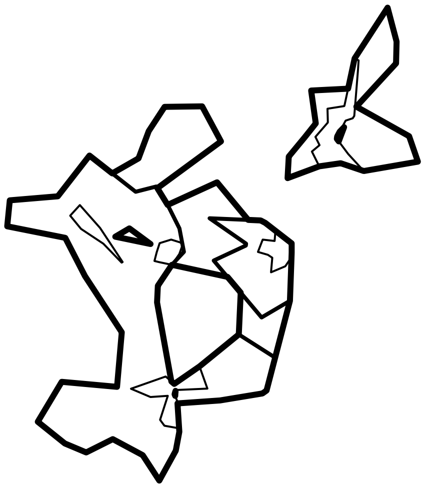
```
## [1] "159"
```

```
class(nn[[1]])
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
## 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.
plot(nn[[1]][339,],lwd=2)

#The 50% home range code is on line 146
```

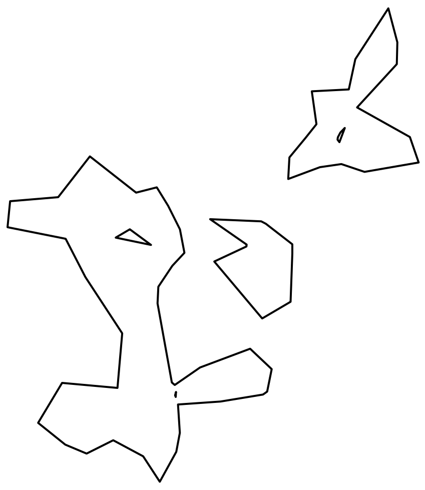
```
plot(nn[[1]][146,],add=TRUE)

#The 99% home range code is on line 359
plot(nn[[1]][359,],lwd=3, add=TRUE)
```



```
#Save shapefiles of resulting home range
ver <- getverticeshr(nn)
ver
```

```
## Object of class "SpatialPolygonsDataFrame" (package sp):
##
## Number of SpatialPolygons:  1
##
## Variables measured:
##    id     area
## 1 159 5412.973
```

```
plot(ver)
```



```
#writeOGR(ver,dsn="FixedK",layer="FixedK24", driver = "ESRI Shapefile", overwrite=TRUE
##Overwrite will not work so must edit path so "FixedK" folder is created with code be
ver50 <-getverticeshr(nn, percent=50)
#writeOGR(ver50,dsn="FixedK",layer="50FixedK24", driver = "ESRI Shapefile",overwrite=T
ver95 <-getverticeshr(nn, percent=95)
#writeOGR(ver95,dsn="FixedK",layer="95FixedK24", driver = "ESRI Shapefile",overwrite=T
ver99 <-getverticeshr(nn, percent=99)
#writeOGR(ver99,dsn="FixedK",layer="99FixedK24", driver = "ESRI Shapefile",overwrite=T
```

Chapter 6

Landscape Metrics

6.1 7.1 and 7.2 Fragstats Metrics within Polygons

Some research designs may just need landscape metrics for a single area or several study areas and that is what the SDMToolsl package is able to estimate in the code that follows. While the single area can be defined by the extent of the raster we imported as in previous chapters, the ability of the SDMToolsl package to determine patch and class statistics depends on the area defined by the user from that could be study site, within polygons such as counties or townships, or within buffers around locations.

1. Exercise 7.1 and 7.2 - 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 "Frag_Auto.Rmd" and run code directly from the script

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

```
#library(SDMTools)
library(raster)
library(plyr)
library(rgdal)
#install.packages("landscapemetrics")
library(landscapemetrics)
```

5. Now let's 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

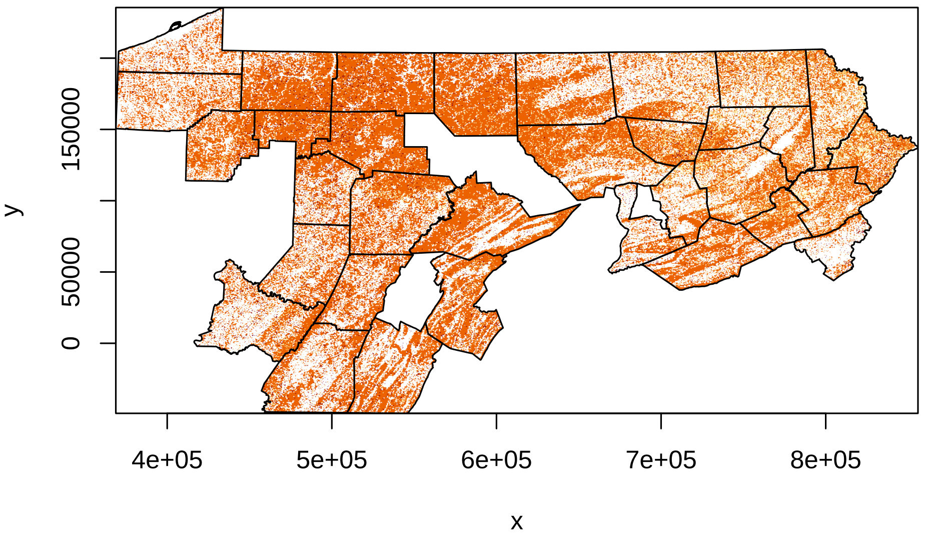
```
new.crs <-CRS("+proj=lcc +lat_1=41.95 +lat_2=40.88333333333333 +lat_0=40.1666666666666
 +lon_0=-77.75 +x_0=600000 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_def
```

6. Load vegetation raster layer clipped in ArcMap

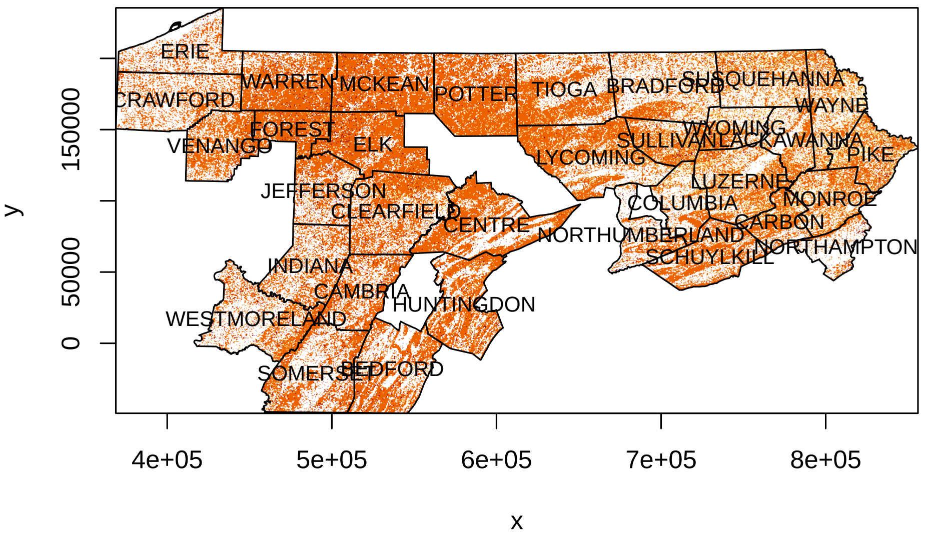
```
crops <-raster("crop2012utm12.tif")
plot(crops)
class(crops)

# reclassify the values into 9 groups
# all values between 0 and 20 equal 1, etc.
m <- c(-Inf,0,NA,2, 7, 2, 20, 60, 3, 60, 70, 4, 110, 132, 5, 133, 150, 6, 151, 191, 7,
  192,Inf,NA)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(crops, rclmat)
plot(rc)
as.matrix(table(values(rc)))

check_landscape(rc)
#Now we get into Landscape Metrics with the SDTM Tool
#Calculate the Patch statistics
ps.data = lsm_p_enn(rc)
ps.data

#Calculate the Class statistics
cl.data = calculate_lsm(rc)
cl.data
```

7. Some research designs may need landscape metrics for several areas that may be available as a shapefile or some other polygon layer.

```
#Load raster file into R
raster <- raster("county_hab")

#Load PA shapefile into R
HareCounties <- readOGR(dsn=".", layer="Hare_Counties", verbose=FALSE)
image(raster)
plot(HareCounties, add=T)
```



```
#Let's project Counties to the same projection as the habitat raster
county <- spTransform(HareCounties, CRS=new.crs)
HareCounties <- county
#Matching projections successful!
image(raster)
plot(HareCounties, add=T)
row.names(HareCounties)<-as.character(HareCounties$COUNTY_NAM)
names.polygons<-sapply(HareCounties@polygons, function(x) slot(x,"ID"))
text(coordinates(HareCounties), labels=sapply(slot(HareCounties, "polygons"), function(i)
  slot(i, "ID")), cex=0.8)
```



8. Now we want to export by County name (i.e., COUNTY_NAM) as indi-

vidual shapefiles. We will only select the first 2 counties for processing to save time

```
indata <- HareCounties
innames <- unique(HareCounties@data$COUNTY_NAM)
innames <- innames[1:2]#Place the number of unique polygons in your shapefile here
outnames <- innames

# set up output table
#output <- as.data.frame(matrix(0,nrow=length(innames),ncol=38))

# begin loop to create separate county shapefiles
for (i in 1:length(innames)){
  data <- indata[which(indata$COUNTY_NAM==innames[i]),]
  if(dim(data)[1] != 0){
    writeOGR(data, dsn = ".", layer=paste(outnames[i],sep="/"), driver = "ESRI Shapefil
    write.table(innames, "List.txt", col.names=FALSE, quote=FALSE,
  row.names=FALSE)
}
}

#Read in a list of shapefiles files from above
Listshps<-read.table("List.txt",sep="\t",header=F)
Listshps

shape <- function(Listshps) {
file <- as.character(Listshps[1,])
shp <- readOGR(dsn=".", layer=file)
mask <- mask(raster,shp)
### Calculate the Class statistics in each county
cl.data <- lsm_c_enn(mask)
}

results <- ddply(Listshps, 1, shape)
results
#write.table(results, "FragCounty.txt")
```

6.2 Fragstats Metrics within Buffers

Some research designs may just need landscape metrics for a single area or several study areas and that is what the SDMToolsl package is able to estimate in the code that follows. While the single area can be defined by the extent of the raster we imported as in previous chapters, the ability of the SDMToolsl package to determine patch and class statistics depends on the area defined by the user from that could be study site, within polygons such as counties or townships, or within buffers around locations.

1. Exercise 7.1 and 7.2 - 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 "PatchAnalystScript.Rmd" and run code directly from the script

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

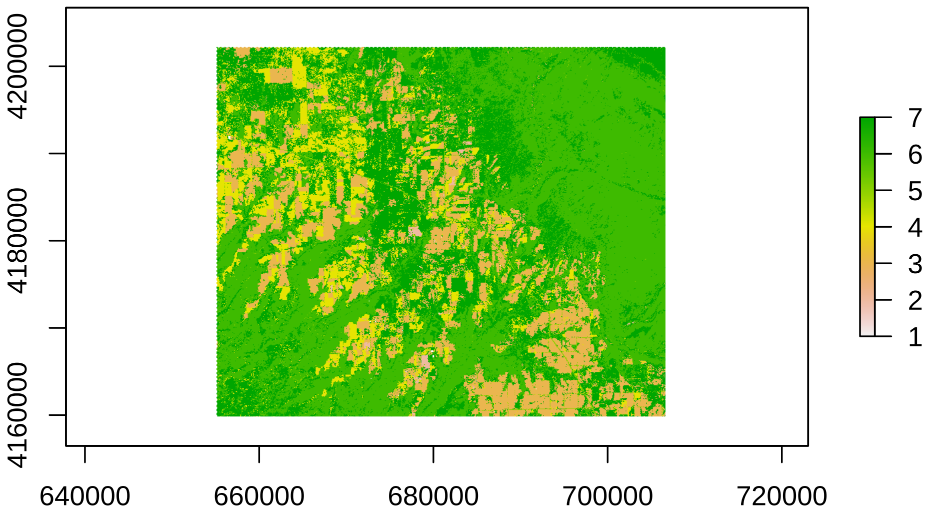
```
#library(SDMTools)
library(raster)
library(rgeos)
library(plyr)
library(landscapemetrics)
```

5. Now let's 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

```
utm12.crs <- CRS("+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +datum=GRS80
  +towgs84=0,0,0")
```

6. Load vegetation raster layer textfile clipped in ArcMap

```
crops <-raster("crop2012utm12.tif")

# reclassify the values into 9 groups
# all values between 0 and 20 equal 1, etc.
m <- c(-Inf,0,NA,2, 7, 2, 20, 60, 3, 60, 70, 4, 110, 132, 5, 133, 150, 6, 151, 191, 7,
  192,Inf,NA)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(crops, rclmat)
plot(rc)
```



7.What if we wanted to compare difference in patch statistics among all deer with all locations combined?

We will start by creating buffers around individual locations for our mule deer dataset

```
muleys <-read.csv("muleysexample.csv", header=T)

#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.80)
muleys <- newmuleys
newmuleys <-subset(muleys, muleys$Long < -107)
muleys <- newmuleys
table(muleys$id)
```

```
##
## D12  D8
## 101 968
```

```
muleys$GPSFixTime<-as.POSIXct(muleys$GPSFixTime, format="%Y.%m.%d%H:%M:%S")

#Only use the 5 lines of code below to subsample for demonstration purposes!
onemuley <-muleys[1:5,]
twomuley <-muleys[102:106,]
shortmd <- rbind(onemuley, twomuley)
shortmd$id <- factor(shortmd$id)#removed deer D12 because to few locations
muleys <- shortmd
```

8. Next we need to create a function to extract Fragstats metrics within individual polygons

```
buff3rd <- function(muleys) {
  coords<-data.frame(x = muleys$X, y = muleys$Y)
  deer.spdf <- SpatialPointsDataFrame(coords=coords, data = muleys, proj4string = utm12.crs)
  settbuff <- gBuffer(deer.spdf, width=1000, byid=FALSE)
  buffclip <- mask(rc, settbuff)
  buff.data <- calculate_lsm(buffclip)
  newline <- muleys$id
  bind <-cbind(newline[1], buff.data)
}

results <- ddply(muleys, .(id), buff3rd)
results
class(results)
```

9. Code above looks at patch and class metrics for each deer by combining all buffers into one polygon for each deer (i.e., comparable to defining available habitat in 3rd order selection. However, what if we wanted to compare difference in patch statistics among all deer by averaging metrics across buffers?

```
coords<-data.frame(x = muleys$X, y = muleys$Y)
deer.spdf <- SpatialPointsDataFrame(coords=coords, data = muleys, proj4string = utm12.crs)
setbuff <- gBuffer(deer.spdf, width=1000, byid=TRUE)
muleys$newID <- paste(muleys$id, setbuff@plotOrder, sep="_")
muleys$newID <- as.factor(muleys$newID)

buff3rdA <- function(muleys) {
  bufclip <- mask(rc, setbuff)
  buf.data <- PatchStat(bufclip)
}

results2 <- ddply(muleys, .(newID), buff3rdA)
results2
```

Chapter 7

Resource Selection

7.1 Minimum Convex Polygon (MCP)

Minimum Convex Polygon (MCP) estimation was considered a home range originally described for use with identifying animals recaptured along a trapping grid (Mohr 1947). The reason we removed this from the Home Range Section is because MCP can be used to describe the extent of distribution of locations of an animal but NOT as an estimation of home range size. In fact, reporting size of home range using MCP should be avoided at all costs unless you can justify its use as opposed to the plethora of other estimators we have learned in the previous section. We may use MCP within resource selection function analysis as it has been suggested as a method to describe the extent of area occupied by a species that would be available to animals using either second or third order selection of habitat (Johnson 1980), although this should also be avoided unless specifically justified as to why MCP is better than an alternate home range estimator. The extent of an area an animal uses (i.e., habitat available) should be determined for each species and the most appropriate estimator should be used.

1. Exercise 8.1 - 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 "MCPscript.Rmd" and run code directly from the script

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

```
library(adehabitatHR)
library(rgdal)
```

5. Now we will 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

```
utm17.crs <- CRS("+proj=utm +zone=17 +ellps=WGS84")
```

6. Load in a mule deer dataset we have used in previous exercises

```
muleys <-read.csv("muleysexample.csv", header=T)
newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.8)
muleys <- newmuleys
newmuleys <-subset(muleys, muleys$Long < -107)
muleys <- newmuleys
```

7. Create Spatial Points for all relocations and assign IDs to each location

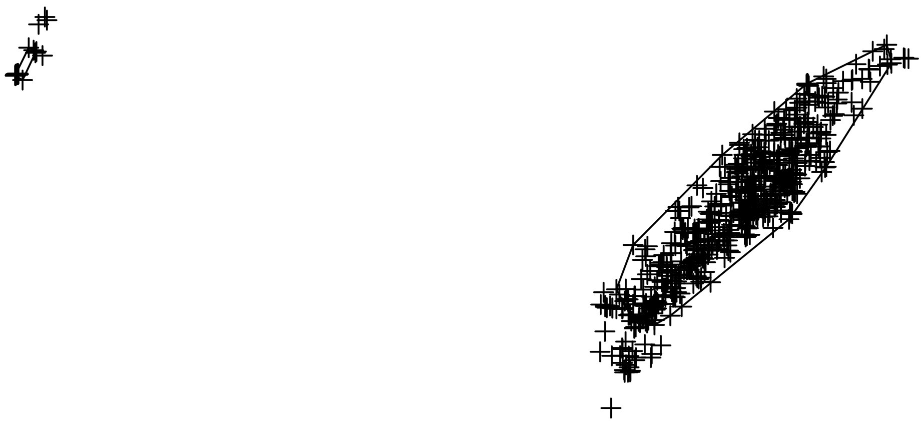
```
data.xy = muleys[c("X","Y")]
#Creates class Spatial Points for all locations
xysp <- SpatialPoints(data.xy)
proj4string(xysp) <- CRS("+proj=utm +zone=17 +ellps=WGS84")

#Creates a Spatial Data Frame from
sppt<-data.frame(xysp)
#Creates a spatial data frame of ID
idsp<-data.frame(muleys[2])
#Merges ID and Date into the same spatial data frame
merge<-data.frame(idsp)
#Adds ID and Date data frame with locations data frame
coordinates(merge)<-sppt
```

8. We are now ready to create MCPs for our new dataset “merge” by individual animal ID (Fig. 8.1).

```
cp <- mcp(merge[,1], percent=95)#(95% is the default)
## The size of the bounding polygon
as.data.frame(cp)
```

```
##      id      area
## D12 D12   5.29995
## D8   D8 515.42075
```

```
## Plot the home ranges
plot(cp)
## ... And the relocations
plot(merge, add=TRUE)
```

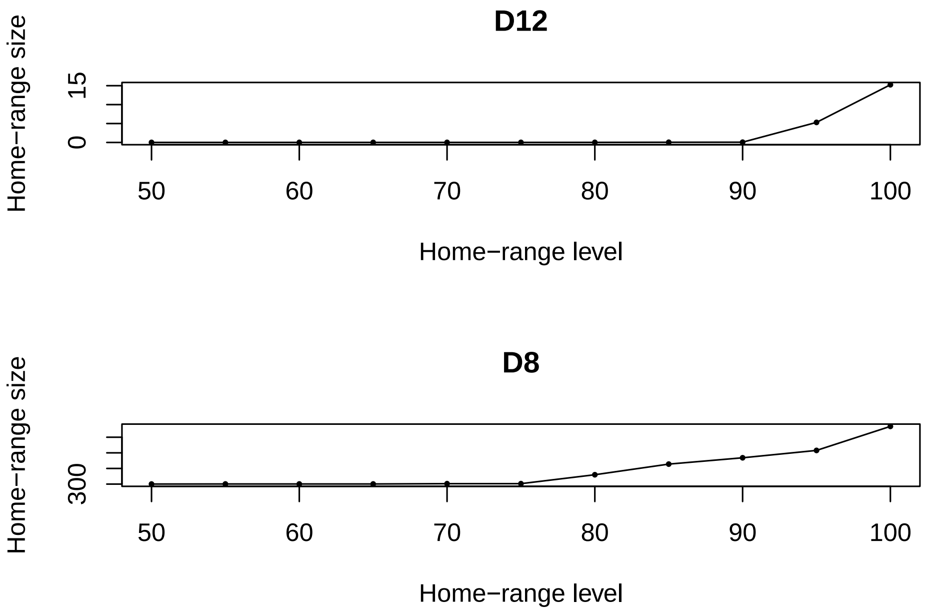


9. Use the rgdal package to write to a shapefile

```
writeOGR(cp, dsn = ".", layer="MCPhomerange", driver = "ESRI Shapefile")
```

10. We have chosen to exclude 5% of the most extreme relocations, but we could have made another choice. We may compute the MCP for various choices of the number of extreme relocations to be excluded, using the function mcp.area:

```
hrs <- mcp.area(merge[,1], percent=seq(50, 100, by = 5))
```



```
hrs
```

```
##          D12       D8
## 50   0.01405 300.3191
## 55   0.01480 300.6203
## 60   0.01600 300.6898
## 65   0.01670 300.7083
## 70   0.01725 302.6114
## 75   0.02005 302.6438
## 80   0.02340 359.7976
## 85   0.04850 427.8537
## 90   0.06580 468.4115
## 95   5.29995 515.4207
## 100 15.24355 669.3478
```

7.2 Preparing Linear Measures

First we will begin with determining the distance between several features. In our first example, we want to measure distance from each mule deer location to the nearest stream if it is determined a priori that water or riparian habitats influence mule deer distribution in our study area. While this may not seem like a very complicated process, there are numerous steps needed to achieve this feat. We will need to use the package spatstat that will help us in creating individual segments with nodes for linear features such as roads and streams/rivers.

1. Exercise 8.2 - 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 LinearDistscript.Rmd" and run code directly from the script

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

```
library(spatstat)
library(maptools)#Needed for spatstat class "owin"
library(polyCub)#replaces gpclib function
library(rgdal)
library(raster)
library(rgeos)
library(OneR)#bin function
```

5. Now we will 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

```
utm12.crs<-"+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0"
Albers.crs <-CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0
  +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
```

6. We will use the mule deer dataset we used in previous exercises

```
muleys <-read.csv("muleysexample.csv", header=T)

#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.80)
muleys <- newmuleys
newmuleys <-subset(muleys, muleys$Long < -107)
muleys <- newmuleys

#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = muleys$X, y = muleys$Y)
deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys, proj4string = CRS(utm12.crs))
```

7. Load the necessary road and rivers shapefiles already in Albers projection to match previous vegetation raster.

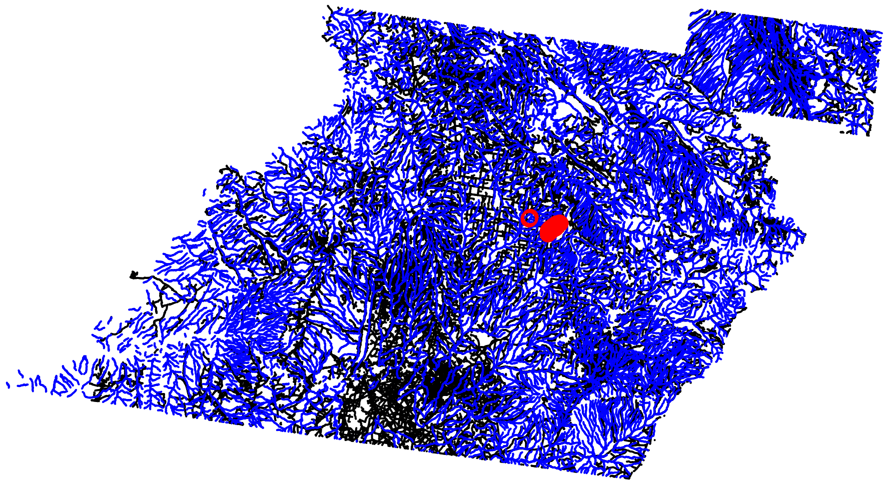
```
roads<-readOGR(dsn=".",layer="AlbersRoads",verbose=FALSE)
rivers<-readOGR(dsn=".",layer="AlbersRivers",verbose=FALSE)
```

8. We need to project the deer.spdf from utm to Albers to match other road and river shapefiles

```
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
```

```
## Warning: PROJ support is provided by the sf and terra packages among others
```

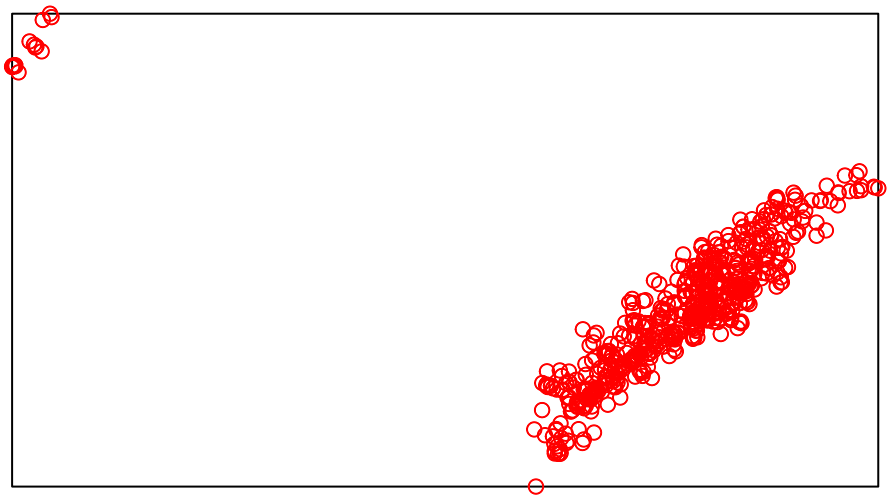
```
plot(roads)
plot(rivers,add=T, col="blue")
points(deer.albers, col="red")
```


9.
Determine boundary box around mule deer locations to create a layer to clip and zoom in.

```r
bbox(deer.albers)
```

```
##         min      max
## x -1127964 -1115562
## y  1718097  1724868
```

```r
bb1 <- cbind(x=c(-1127964,-1127964,-1115562,-1115562,-1127964),
  y=c(1718097, 1724868, 1724868,1718097,1718097))
AlbersSP <- SpatialPolygons(list(Polygons(list(Polygon(bb1)),"1")),
            proj4string=CRS(proj4string(deer.albers)))
plot(AlbersSP)
points(deer.albers, col="red")
```

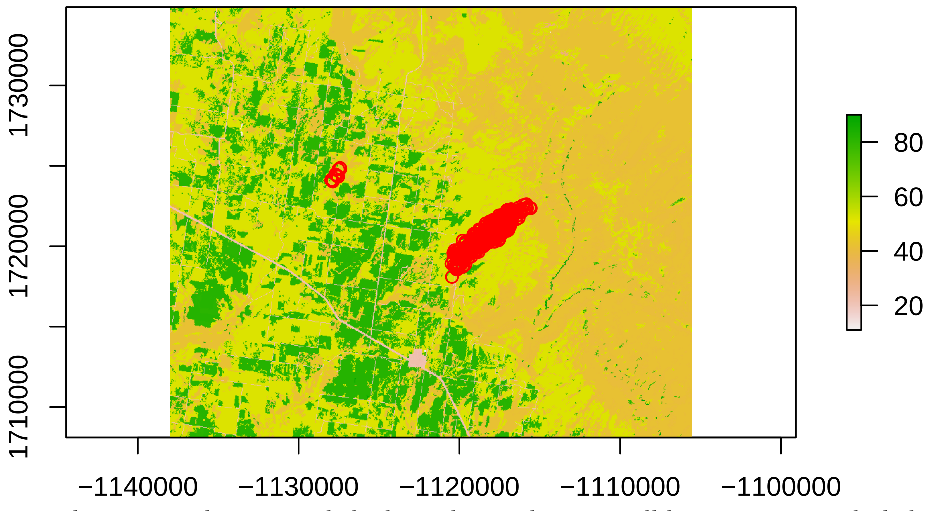


10. Load vegetation raster layer tif that came in the Albers projection from the online source.

```
veg <-raster("cropnlcd.tif")
```

```
#Check to see all our layers are now in Albers projection
proj4string(veg)
proj4string(deer.albers)
proj4string(AlbersSP)
```

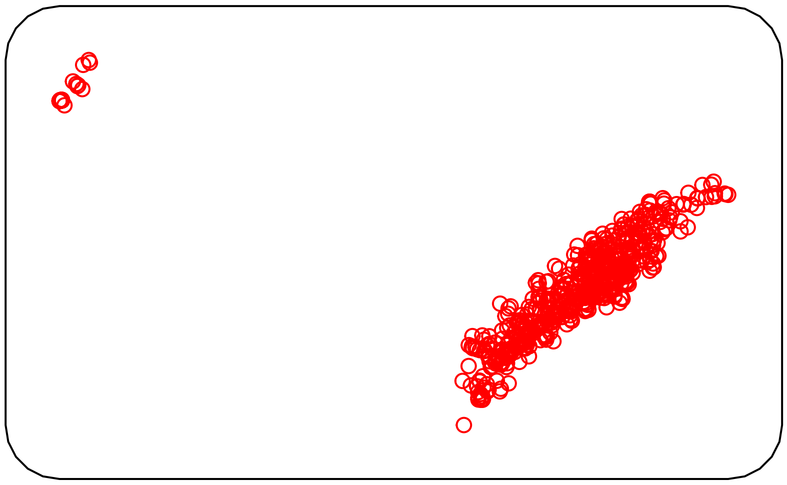
```
plot(veg)
points(deer.albers, col="red")
```



11. Then we need to expand the bounding polygon so all locations are included.

We can then make the bounding polygon (AlbersSP) a class owin in order to proceed with functions in package spatstat.

```r
buffSP <- gBuffer(AlbersSP,width=1000)
plot(buffSP)
points(deer.albers,col="red")
```



12. Code below will be for use with the spatstat package to convert segments of line layers (e.g., roads, rivers) to lines to enable distance to feature from deer locations. Most calculations with spatstat require 3 new classes so most code is created to achieve this goal:

“owin” Observation windows “ppp” Planar point patterns “psp” Planar segment patterns

```r
#Replace AlbersSP with buffSP
AlbersSPDF <- as(buffSP, "SpatialPolygonsDataFrame")
bdy.owin <- as.owin(AlbersSPDF)
is.owin(bdy.owin)
#It is TRUE so now we can move forward with the analysis
```

12. Now clip the raster using the buffered bounding box (buffSP) created in step 5.

```r
bbclip <- crop(veg, buffSP)
cliproads <- gIntersection(roads, buffSP, byid=TRUE)
cliprivers <- gIntersection(rivers, buffSP, byid=TRUE)

plot(bbclip)
points(deer.albers, col="red")
```

```
plot(cliproads, add=T)
plot(cliprivers, col="blue",add=T)
```

13. We will start with the road layer by converting a single line to a set of segments packaged as a function.

```
foo <- function(cliproads){
x <- cliproads@Lines[[1]]@coords
cbind(
head(x,-1),
tail(x,-1))}
#The function can be applied successively to each line in the list we extracted from roads.
#Results are output as a list, then converted to a matrix.

segs.lst <- lapply(cliproads@lines,foo)
segs <- do.call(rbind,segs.lst)

segs.x <- c(segs[,c(1,3)])
segs.y <- c(segs[,c(2,4)])
segs.owin <- as.owin(c(range(segs.x),range(segs.y)))#create a new "owin" class because
#roads occur outside our bdy.owin created above

#The segments as a planar segment pattern:
segs.psp <- as.psp(segs, window=segs.owin)
plot(segs.psp)
points(deer.albers)
segs.psp[1:5]
#lengths.psp(segs.psp[1:10])

#We can cut road segments into distances we control
dist <- pointsOnLines(segs.psp, eps=1000)
```

14. We first need to handle the mule deer locations. We need to make mule deer xy coordinates a planar point pattern (i.e., ppp) for use in package spatstat.

```
deer2 <-as.data.frame(deer.albers)
newdeer <-cbind(deer2$x,deer2$y)
xy.ppp <- as.ppp(newdeer,W=bdy.owin)
plot(xy.ppp)
```

15. Now we can determine the distance from mule deer locations (xy.ppp) to the nearest road

```
roaddist <- nncross(xy.ppp, segs.psp)$dist
#Or identify segment number closest to each point
v <- nearestsegment(xy.ppp,segs.psp)#Identifies segment number not a distance
plot(segs.psp)
plot(xy.ppp[101], add=TRUE, col="red")
plot(segs.psp[v[101]], add=TRUE, lwd=5, col="red")
```

16. Now we do the same to a river layer by converting a single line to a set of segments packaged as a function.

```
foo <- function(cliprivers){
x <- cliprivers@Lines[[1]]@coords
cbind(
head(x,-1),
tail(x,-1))}
#The function can be applied successively to each line in the list we extracted from r
#Results are output as a list, then converted to a matrix.
rivs.lst <- lapply(cliprivers@lines,foo)
rivs <- do.call(rbind,rivs.lst)
rivs.x <- c(rivs[,c(1,3)])
rivs.y <- c(rivs[,c(2,4)])
rivs.owin <- as.owin(c(range(rivs.x),range(rivs.y)))

#The segments as a planar segment pattern:
rivs.psp <- as.psp(rivs, window=rivs.owin)
plot(rivs.psp)
points(deer.albers)
is.psp(rivs.psp)
#All is TRUE so now we can move forward with the analysis
```

17. Now we can determine the distance from mule deer locations (xy.ppp) to the nearest river.

```
rivdist <- nncross(xy.ppp, rivs.psp)$dist

#Or identify segment number closest to each point
riv <- nearestsegment(xy.ppp,rivs.psp)
plot(rivs.psp, lwd=1)
plot(xy.ppp[1], add=TRUE, col="red")
plot(rivs.psp[riv[1]], add=TRUE, lwd=5, col="red")
plot(xy.ppp[290], add=TRUE, col="blue")
plot(rivs.psp[riv[290]], add=TRUE, lwd=5, col="blue")
points(deer.albers)
```

18. We can then summarize the distances in some meaningful way for analysis. Instead of representing distance to road as individual numerical values we can bin the distances in some categories we determine appropriate for our research objective.

```
br <- seq(0,1000,200)
lbl <- paste(head(br,-1),tail(br,-1),sep="-")
road.tbl <- table(cut(roaddist,breaks=br,labels=lbl))
Rdresults <- road.tbl/sum(road.tbl)
Rdresults

br1 <- seq(0,4000,500)
lbl1 <- paste(head(br1,-1),tail(br1,-1),sep="-")
river.tbl <- table(cut(rivdist,breaks=br1,labels=lbl1))
Rivresults <- river.tbl/sum(river.tbl)
Rivresults
```

19. Or we can place each distance into a category or Bin for each deer

```
BinRoad <- bin(roaddist, nbins=5, method='content', labels=c('1','2','3','4','5'))
BinRoad2 <- cut(roaddist, 5, method='intervals', include.lowest=TRUE, labels=c('1','2','3'
  ,'4','5'))
table(BinRoad)

BinRivers <- bin(rivdist, nbins=5, method='content', labels=c('1','2','3','4','5'))
BinRivers <- cut(rivdist, 5, method='intervals', include.lowest=TRUE, labels=c('1','2','3'
    ,'4','5'))
table(BinRivers)

#Now use cbind function to add binned distances to muleys dataset.
Dist <- cbind(BinRoad,BinRivers)
muleys <- cbind(muleys, Dist)
```

7.3 Preparing Additional Covariates

We may often be interested in assessing various covariates that may influence resource selection of our study animals. If we have a priori knowledge that elevation or slope may influence selection for or use of portions of the landscape then we need to create these layers for analysis. While this may not seem like a very complicated process because it is routinely done in ArcMap, those same available layers can be used and manipulated in R as in Chapter 1. We can then create slope, aspect, hillshade or other variables within R using concepts in earlier chapters and extract those covariates for use in modeling all within the R environment.

1. Exercise 8.3 - 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 MD_DataPrep.Rmd" and run code directly from the script

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

```
library(rgdal)
library(rgeos)#gBuffer
library(raster)#to use "raster" function
library(adehabitatHR)
library(FedData)
```

5. Now we will 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

```
utm12.crs<-"+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,
# #Albers.crs <-CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_
#   +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
Albers.crs <- CRS("+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=
                  +datum=NAD83 +units=m +no_defs")
```

6. Load the mule deer dataset we used in the previous exercise

```
muleys <-read.csv("muleysexample.csv", header=T)
#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.80)
muleys <- newmuleys
newmuleys <-subset(muleys, muleys$Long < -107)
muleys <- newmuleys
```

```
#Only use the line below for example exercise so fewer locations are used.
muleys <- muleys[sample(nrow(muleys), 100),]

#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = muleys$X, y = muleys$Y)
utm.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys, proj4string = CRS(utm12.crs))
deer.spdf <-spTransform(utm.spdf, CRS=Albers.crs)
```

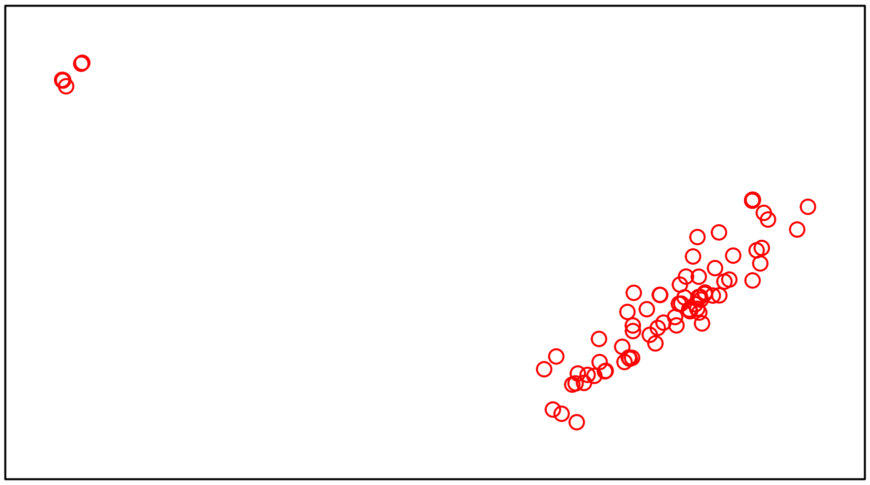
```
## Warning: PROJ support is provided by the sf and terra packages among others
```

7. We can create a bounding box around locations and crop rasters later using coordinates of box.

```
bbox(deer.spdf)
```

```
##         min      max
## x -1127934 -1116142
## y  1718721  1724399
```

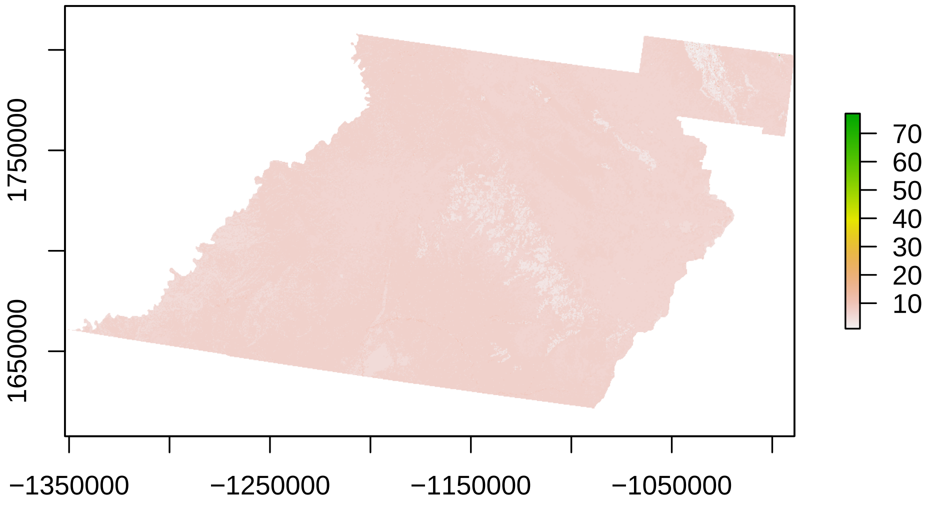
```
bb1 <- cbind(x=c(xmin(deer.spdf)-900,xmin(deer.spdf)-900,xmax(deer.spdf)+900,xmax(deer.spdf)+900,
            xmin(deer.spdf)-900), y=c(ymin(deer.spdf)-900, ymax(deer.spdf)+900,ymax(deer.spdf)+90
                                      ymin(deer.spdf)-900,ymin(deer.spdf)-900))

AlbersSP <- SpatialPolygons(list(Polygons(list(Polygon(bb1)),"1")),
 proj4string=CRS(proj4string(deer.spdf)))

plot(AlbersSP)
points(deer.spdf, col="red")
```



8. Now it is time to import some raster layers of the covariates we are interested

in for our analysis. Start with raster of vegetation from the 2012 NRCS Crop data that is a nice dataset that is crop specific for each year. Crop data can be found at the NRCS webpage Cropland Data Layer that can be accessed for each county of each state.

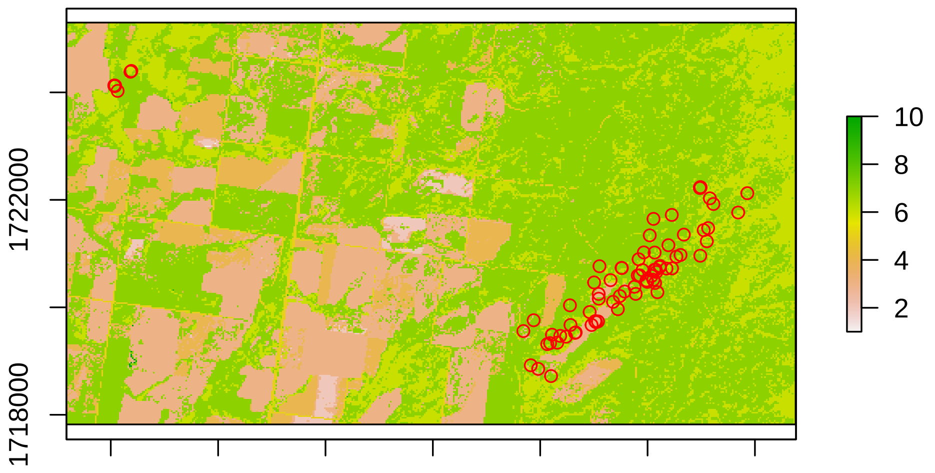
```r
crops <-raster("crop12clip.tif")

# Reclassify crops raster from above into 9 groups
# all values between 0 and 20 equal 1, etc.
m <- c(-Inf,0,NA,2, 7, 2, 20, 60, 3, 60, 70, 4, 110, 132, 5, 133, 150, 6, 151, 172, 7,
       180, 183, 8, 189, 191, 9,192,205,10)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(crops, rclmat)
plot(rc)
```



```r
#Crop using AlbersSP polygon created earlier to reduce size of raster (if needed).
bbclip <- crop(rc, AlbersSP)

as.matrix(table(values(bbclip)))#Identifies the number of cells in each category
```

```
##      [,1]
## 1       5
## 2    1097
## 3   21410
## 4    9504
## 5    2097
## 6   19298
## 7   59363
## 9       9
## 10     14
```

```
plot(bbclip)
points(deer.spdf,col="red")
plot(AlbersSP, add=T)
```



9. We also want to look at elevation and covariates related to elevation (e.g., slope, aspect). These can be created directly in R using the terrain function in the raster package. We will use the FedData package to get Digital Elevation Models for the large study area

```
DEM <- get_ned(template=rc, label = 'PAdem',force.redo = T)
image(DEM, col=terrain.colors(10))
contour(DEM, add=TRUE)

#Or read in previously created DEM
DEM <- raster("dem_albers.txt")

slope = terrain(DEM,opt='slope', unit='degrees')
aspect = terrain(DEM,opt='aspect', unit='degrees')

elevation <- projectRaster(DEM, bbclip,method='ngb')
slope <- projectRaster(slope, bbclip,method='ngb')
aspect <- projectRaster(aspect, bbclip,method='ngb')

demclip <- crop(elevation, AlbersSP)
sloclip <- crop(slope, AlbersSP)
aspclip <- crop(aspect, AlbersSP)
```

10. Cast over all 4 layers to a Spatial Grid Data Frame to permit combining into one layer.

```
nlcd <- as.data.frame(as(bbclip, "SpatialGridDataFrame"))
elev <- as.data.frame(as(demclip, "SpatialGridDataFrame"))
slo <- as.data.frame(as(sloclip, "SpatialGridDataFrame"))
asp <- as.data.frame(as(aspclip, "SpatialGridDataFrame"))

#Now check to be sure the number of cells in each layer are the same before proceeding
#next step of combining layers.
str(elev)
str(slo)
str(asp)
str(nlcd)
```

11. Combine elevation, slope, and aspect into one layer.

```
layers = cbind(nlcd, elev, asp, slo)
head(layers)
#Remove xys that are repeated in each layer
layers = layers[,-c(2,3,5,6,8,9)]
names(layers) = c("nlcd", "elevation", "aspect", "slope","x", "y")

# turn aspect into categorical
aspect_categorical = rep(NA, nrow(layers))
aspect_categorical[layers$aspect < 45 | layers$aspect >= 315] = "N"
aspect_categorical[layers$aspect >= 45 & layers$aspect < 135] = "E"
aspect_categorical[layers$aspect >= 135 & layers$aspect < 225] = "S"
aspect_categorical[layers$aspect >= 225 & layers$aspect < 315] = "W"
table(aspect_categorical)
table(is.na(aspect_categorical))
layers$aspect_categorical = aspect_categorical
head(layers)
#write.table(layers,"layer1.txt",sep=",",col.names=TRUE, quote=FALSE)

layers2 <- layers #change to layers2 simply to avoid confusion with "layer" term in fu
#below
```

#NOTE: Script may contains Demonstration code that will subset number of locations to speed up processing of data during a course exercise. To prevent this, skip this line of code above (Line 44): #muleys <- #muleys[sample(nrow(muleys), 100),]

12. We can now begin the task of sampling each of our locations using the code below. This code was created by Ryan Nielsen of West Inc. and was very helpful

in this exercise. Alternatively, we could have extracted each covariate layer by layer and included it in our dataset.

```
# grab values for points created above
grab.values = function(layer, x, y){
    # layer is data.frame of spatial layer, with values 'x', 'y', and ____?
    # x is a vector
    # y is a vector
    if(length(x) != length(y)) stop("x and y lengths differ")
    z = NULL
    for(i in 1:length(x)){
        dist = sqrt((layer$x - x[i])^2 + (layer$y-y[i])^2)
        #Could adjust this line or add another line to calculate moving window or
        #distance to nearest feature
        z = rbind(z, layer[dist == min(dist),][1,])
    }
    return(z)
}

#Grab all values from muleys for each layer in r
test = grab.values(layers2, muleys$X, muleys$Y)
head(test)

##NOTE that all values are the same but this is not correct.
##What is the problem here and how do we fix it?

#Need to grab Albers XY not UTM as in muleys above
muleys <- as.data.frame(deer.spdf)

# grab all values for used and available points based on combined layer data set
# can take 5+ minutes
used = grab.values(layers2, muleys$x, muleys$y)
used$x = muleys$x
used$y = muleys$y
used$animal_id = muleys$id
used$use = 1
head(used)
```

13. We also need to get some measure of what is available for our mule deer population (2nd order selection) or for each mule deer (3rd order selection). We really do not understand the need for 2nd order selection unless you are looking at deer across different landscapes but hardly seems necessary for deer occupying similar areas such as our mule deer in southwestern Colorado. Below we will focus on 3rd order selection with used locations for each deer being compared to available locations randomly determined within each deer's MCP.

```
#Create MCP for all locations for each deer by ID (3nd order selection).
cp = mcp(deer.spdf[,2],percent=100)
as.data.frame(cp)

#Determine the habitat available using all code below
#First create random sample of points in each polygon
random <- sapply(slot(cp, 'polygons'), function(i) spsample(i, n=50, type='random', of
plot(cp) ; points(random[[2]], col='red', pch=3, cex=.5)#The number in double brackets
#polygons stack into a single SpatialPoints object
random.merged <- do.call('rbind', random)
#Extract the original IDs
ids <- sapply(slot(cp, 'polygons'), function(i) slot(i, 'ID'))
#Determine the number of ACTUAL sample points generated for each polygon
newpts <- sapply(random, function(i) nrow(i@coords))
newpts #Nice check of how many points generated per polygon
# generate a reconstituted vector of point IDs
pt_id <- rep(ids, newpts)

# promote to SpatialPointsDataFrame
random.final <- SpatialPointsDataFrame(random.merged, data=data.frame(poly_id=pt_id))

random.final

# make 'random.final' a data.frame
random.df = as.data.frame(random.final)
names(random.df) = c("ID","x","y")
# can take 5+ minutes
available = grab.values(layers2, random.df$x, random.df$y)
available$x = random.df$x
available$y = random.df$y
available$animal_id = pt_id
available$use = 0
head(available)
```

14. Bind together mule deer locations with covariates extracted (used) and random locations within each polygon by deer ID (available) into a master dataset for modeling (data). The (use) column identifies 1 as (used) and 0 as (available)

```
data = rbind(available, used)
##A quick check of the data to determine if correct number of records.
#''data.frame': 200 obs. of  9 variables:
#100 locations used +
#100 locations available (2 animals X 50 random locations)
#= 100 #Confirmed in code below
```

```
# nlcd              : num  7 6 7 7 7 7 7 7 7 6 ...
# elevation         : int  2058 2058 2068 2068 2070 2072 2076 2062 2071 2071 ...
# aspect            : num  105 278 105 80 135 ...
# slope             : num  2.72 3.37 4.68 4.11 6.05 ...
# x                 : num  -1127639 -1127610 -1127864 -1127805 -1127862 ...
# y                 : num  1724257 1724271 1724091 1724218 1724174 ...
# aspect_categorical: chr  "E" "W" "E" "E" ...
# animal_id         : chr  "D12" "D12" "D12" "D12" ...
# use               : num  0 0 0 0 0 0 0 0 0 0 ...
```

15. The above code is for 3rd order selection within home range of each deer. We could also look at 3rd order selection within a buffered circle around each mule deer location that is common in Discrete Choice Models. The code is similar except the initial steps of creating buffered polygons and obviously includes a lot more polygons than simply MCPs for each deer. Determining the daily distance moved was done in Chapter 3 but new code is available to estimate for each deer or all deer combined.

```
settbuff=gBuffer(deer.spdf, width=500, byid=TRUE)

#Determine the habitat available using all code below
#First create random sample of points in each polygon
ranbuff <- sapply(slot(settbuff, 'polygons'), function(i) spsample(i, n=3, type='random',
  offset=c(0,0)))
plot(settbuff) ; points(ranbuff[[100]], col='red', pch=3, cex=.5)#The number in double
#brackets changes polygons
#stack into a single SpatialPoints object
ranbuff.merged <- do.call('rbind', ranbuff)

#Extract the original IDs
buff_ids <- sapply(slot(settbuff, 'polygons'), function(i) slot(i, 'ID'))
buff_ids <- paste(settbuff$id, buff_ids, sep="_")
#Determine the number of ACTUAL sample points generated for each polygon
buffpts <- sapply(ranbuff, function(i) nrow(i@coords))
buffpts[1:20] #Nice check of how many points generated per polygon
# generate a reconstituted vector of point IDs
buffpt_id <- rep(buff_ids, buffpts)

# promote to SpatialPointsDataFrame
buff.final <- SpatialPointsDataFrame(ranbuff.merged, data=data.frame(poly_id=buffpt_id))

#Plot buff.final on buffered circles
plot(settbuff) ; points(buff.final, col="red",pch=3, cex=0.5)

# make 'buff.final' a data.frame
```

```
buffer.df = as.data.frame(buff.final)
names(buffer.df) = c("ID","x", "y")
head(buffer.df)
str(random.df)
str(buffer.df)

# can take 5+ minutes
buff_avail = grab.values(layers2, buffer.df$x, buffer.df$y)
buff_avail$x = buffer.df$x
buff_avail$y = buffer.df$y
buff_avail$animal_id = buffpt_id
buff_avail$use = 0

data2 = rbind(buff_avail, used)

#Save workspace so all analysis are available
save.image("RSF_dataprep.RData")

#Before closing, let's save the "used" and available data set to use in the next exerc
write.table(used, "MD_used.txt")
write.table(available, "MD_avail.txt")
```

16. We are going to focus the remainder of this chapter on Selection Ratios and Resource Selection Functions (RSFs) because Selection Ratios identify a general use of habitat given what is available that can be further explored and studied through use of RSFs. Resource Selection Functions are spatially-explicit models that predict the (relative) probability of use by an animal at a given area/location during a given time, based on the environmental conditions that influence or account for selection. There are numerous types of RSFs that can be performed based on the availability of data collected during the study and there are volumes of literature devoted to the topic of resource selection and sampling designs for radiotelemetry studies (Manly et al. 2002, Cooper and Millspaugh 2001, Erickson et al. 2001, Leban et al. 2001).

7.4 Selection Ratios

We are going to focus the remainder of this chapter on Selection Ratios and Resource Selection Functions (RSFs) because Selection Ratios identify a general use of habitat given what is available that can be further explored and studied through use of RSFs. Resource Selection Functions are spatially-explicit models that predict the (relative) probability of use by an animal at a given area/location during a given time, based on the environmental conditions that influence or account for selection. There are numerous types of RSFs that can be performed based on the availability of data collected during the study and there are volumes of literature devoted to the topic of resource selection and sampling designs for radiotelemetry studies (Manly et al. 2002, Cooper and Millspaugh 2001, Erickson et al. 2001, Leban et al. 2001).

Selection Ratio basic functions

widesI may be used to explore resource selection by animals when designs I occur (i.e., habitat use and availability are measured at the population level because individual animals are not identified). The Manly selectivity measure (selection ratio = used/available) is computed and preference/avoidance is tested for each habitat, and the differences between selection ratios are computed and tested (Manly et al. 2002).

widesII computes the selection ratios with design II data (i.e., the same availability for all animals, but use is measured for each one). An example would be to place a minimum convex polygon around all animal locations throughout a study site and define this as "available" to all animals.

widesIII computes the selection ratios for design III data (i.e., use and the availability are measured for each animal with use and availability unique to each individuals movements and habitat use).

Note that all these methods rely on the following hypotheses: (i) independence between animals, and (ii) all animals are selecting habitat in the same way (in addition to "traditional" hypotheses in these kinds of studies: no territoriality, all animals having equal access to all available resource units, etc. (Manly et al. 2002).

1. Exercise 8.3 - 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 MuleDeerSR.Rmd" and run code directly from the script

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

```
library(adehabitatHS)
```

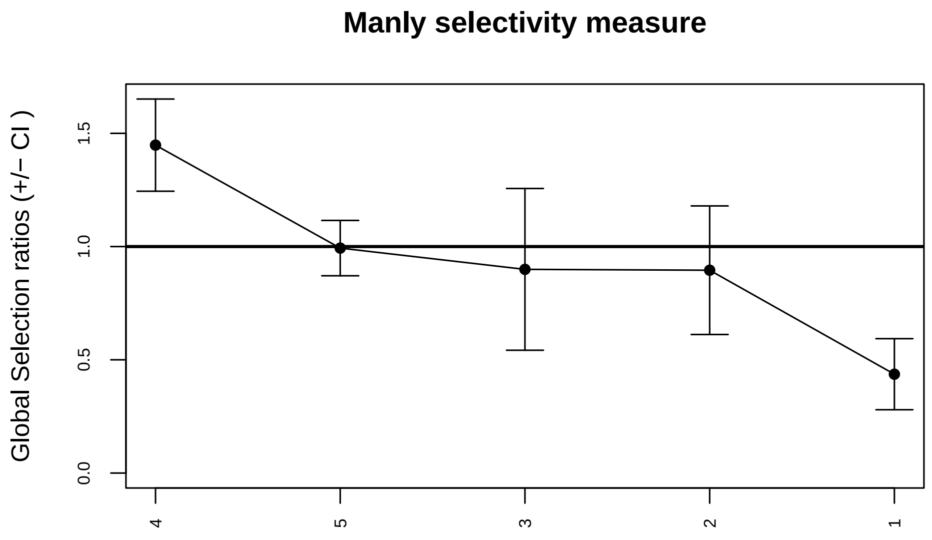
5. Load the mule deer dataset we used in the previous exercise with 5 habitat categories: 1 = Sunflower,summer crops, random crops, grassland 2 = Winter crops 3 = Alfalfa 4 = Forest 5 = Shrubland

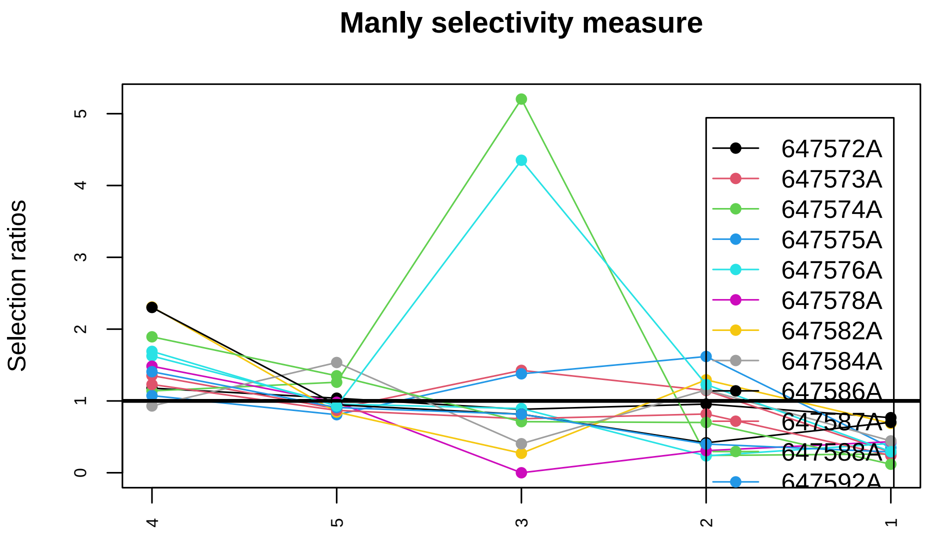
```r
MDsr <- read.csv("MD_winter12.csv",header=T)
#Remove deer that cause errors in plot function later
MDsr <- subset(MDsr,MDsr$animal_id !="647579A")
MDsr$animal_id <- factor(MDsr$animal_id)
used <- subset(MDsr, MDsr$use == 1)
used <- used[c(-2,-3,-5:-6,-8:-15)]
used <- xtabs(~used$animal_id + used$crop, used)
used <- as.data.frame.matrix(used[1:13, 1:5])

rand <- subset(MDsr, MDsr$use == 0)
rand <- rand[c(-2,-3,-5:-6,-8:-15)]
rand <- xtabs(~rand$animal_id + rand$crop, rand)
rand <- as.data.frame.matrix(rand[1:13, 1:5])

# PVT Code for VegRSF #
pvt.W <- widesIII(used,rand,avknown = FALSE, alpha = 0.1)
pvt.W
```

```
##
##
## ************** Manly's Selection ratios for design III ********
##
## 1. Test of habitat selection for each animal:
##
##             Khi2Lj df       pvalue
## 647572A  12.91033  4 1.172236e-02
## 647573A 152.03711  4 0.000000e+00
## 647574A 294.32661  4 0.000000e+00
## 647575A 121.30788  4 0.000000e+00
## 647576A 159.37964  4 0.000000e+00
## 647578A  30.86599  3 9.071081e-07
## 647582A 165.65100  4 0.000000e+00
## 647584A 201.84871  4 0.000000e+00
## 647586A 186.57206  4 0.000000e+00
## 647587A  32.81041  4 1.306135e-06
## 647588A 400.29363  4 0.000000e+00
## 647592A 119.00251  4 0.000000e+00
## 647593A 391.90801  4 0.000000e+00
##
##
## 2. Test of overall habitat selection:
```

```
##    Khi2L       df   pvalue
## 2268.914   51.000    0.000
##
##
## Table of selection ratios:
##          Wi         SE ICl ower ICupper
## 1 0.4364081 0.07639517  0.2587  0.6141
## 2 0.8955675 0.13824727  0.5740  1.2172
## 3 0.8993044 0.17389108  0.4948  1.3038
## 4 1.4478687 0.09908524  1.2174  1.6784
## 5 0.9932287 0.05949948  0.8548  1.1316
##
##
## Bonferroni classement
## Based on 90 % confidence intervals on the differences of Wi :
##
## habitat  4  5  3  2  1
## 4       ---
## 5          ---------
## 3          ---------
## 2          ---------
## 1                   ---
```

```
plot(pvt.W)
```





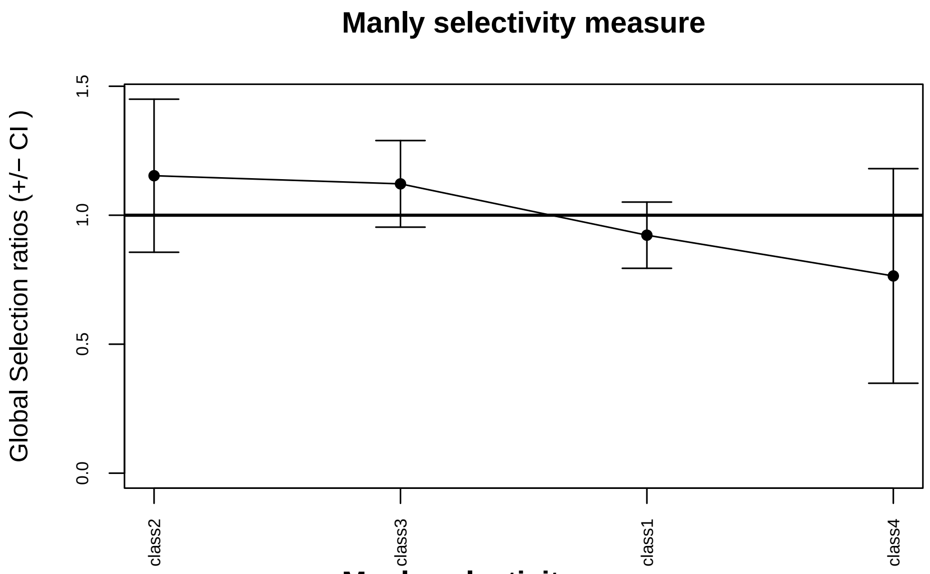
Next we will run on distance to road binned into 10 categories

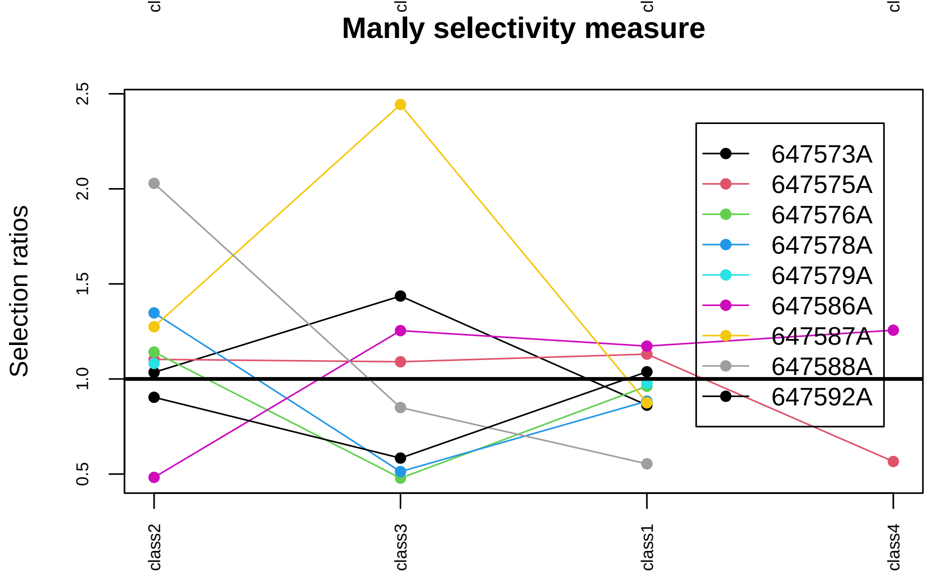
```
#Now run on distance to roads binned into 10 categories
MDsr_road <- read.csv("MD_winter12.csv",header=T)
#Delete deer that have limited data and will result in errors in code below
MDsr_road <- subset(MDsr_road,MDsr_road$animal_id !="647582A" & MDsr_road$animal_id
!="647584A" & MDsr_road$animal_id !="647572A" & MDsr_road$animal_id !="647574A" &
MDsr_road$animal_id !="647593A" )
MDsr_road$animal_id <- factor(MDsr_road$animal_id)

#Bin roads into 4 categories instead of 10
MDsr_road$NewRoad <- as.factor(MDsr_road$BinRoad)
levels(MDsr_road$NewRoad)<-list(class1=c("0-200","200-400"), class2=c("400-600","600-8
class3=c("800-1000","1000-12000","1200-1400"),class4=c("1400-1600","1600-1800","1800-2

used_road <- subset(MDsr_road, MDsr_road$use == 1)
used_road <- used_road[c(-2,-3,-5:-6,-8:-12)]
used_road <- xtabs(~used_road$animal_id + used_road$NewRoad, used_road)
used_road <- as.data.frame.matrix(used_road[1:9, 1:4])

rand_road <- subset(MDsr_road, MDsr_road$use == 0)
rand_road <- rand_road[c(-2,-3,-5:-6,-8:-12)]
rand_road <- xtabs(~rand_road$animal_id + rand_road$NewRoad, rand_road)
rand_road <- as.data.frame.matrix(rand_road[1:9, 1:4])

pvt.road <- widesIII(used_road,rand_road,avknown = FALSE, alpha = 0.1)
pvt.road
```

```
##
```

```
##
## ************** Manly's Selection ratios for design III ********
##
## 1. Test of habitat selection for each animal:
##
##              Khi2Lj df       pvalue
## 647573A  23.156014  2 9.369910e-06
## 647575A  43.618294  3 1.818935e-09
## 647576A  19.130089  2 7.013808e-05
## 647578A  14.413804  2 7.414506e-04
## 647579A   1.697957  1 1.925553e-01
## 647586A  99.947539  3 0.000000e+00
## 647587A  31.950959  2 1.153287e-07
## 647588A 361.725781  2 0.000000e+00
## 647592A   7.312625  2 2.582758e-02
##
##
## 2. Test of overall habitat selection:
##    Khi2L       df   pvalue
## 602.9531  19.0000   0.0000
##
##
## Table of selection ratios:
##                Wi         SE IClower ICupper
## class1 0.9225643 0.06548381  0.7758  1.0693
## class2 1.1531010 0.15136762  0.8138  1.4924
## class3 1.1215628 0.08562584  0.9296  1.3135
## class4 0.7645326 0.21223118  0.2888  1.2402
##
##
## Bonferroni classement
## Based on 90 % confidence intervals on the differences of Wi :
##
## habitat  class2  class3  class1  class4
## class2  ----------------
## class3  ----------------
## class1                  ----------------
## class4                  ----------------
```

```
plot(pvt.road)
```





7.5 Logistic Regression

Resource selection requires “used” and “available” habitats and the study designs would take up an entire course all on there own. In this section, we hope to show how we can go about this approach all in R and not need to involve excel spreadsheets with multiple columns of data. More details on methods to estimate resource selection functions (RSFs) or resource selection probability functions (RSPFs) can be found in the literature (Manly et al. 2002, Millspaugh et al. 2006, Johnson et al. 2006). We do not expect you to be experts in RSFs after this section but we want you to be able to implement these methods in R after determining study design, data collection protocol, and methodology to best achieve your research objectives.

8.4.1 Logistic regression

As we move forward in this section, we are going to assume that your study design and data assessment prior to this section addresses any collinearity in predictor variables and a priori hypothesis were used to generate your models used in logistic regression. There are several ways to to calculate RSFs in R using logistic functions that can assess population level or intra-population variation. The use of General Linear Models with various function using the lme4 package is often used for estimating population-level models only. Alternatively, we can assess intra-population variation using the glmer function. Assessing intra-population variation is a mixed-model approach that provides a powerful and flexible tool for the analysis of balanced and unbalanced grouped data that are often common in wildlife studies that have correlation between observations within the same group or variation among individuals at the same site (Gillies et al. 2006).

1. Exercise 8.5 - 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 LogisticRSF.Rmd" and run code directly from the script

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

```
library(lme4)
library(AICcmodavg)
library(adehabitatHR)
```

5. Load files for each season in the mule deer dataset We may need to identify some numerical data as factors for analysis prior to implementing resource selection analysis.

```
data_1 <- read.csv("MD_winter12.csv",header=T)

############################### MODELING ###############################
data_1$crop=as.factor(data_1$crop)
#dataset$SEASON=factor(dataset$SEASON)
data_1[,2:3]=scale(data_1[,2:3],scale=TRUE)#standardize data to mean of zero
```

6. We may need to use code that changes Reference Categories of our data. For our analysis we are going to define reference category of *used* habitat as crop= 1. Crop category 1 is sunflower which was the crop of interest but was not selected for based on Selection Ratios in Exercise 8.4.

```
fit1 = glmer(use ~ relevel(crop,"1")+(1|animal_id), data=data_1, family=binomial(link=
  "logit"),nAGQ = 0)#Sunflower and cover model
fit2 = glmer(use ~ d_cover+(1|animal_id), data=data_1, family=binomial(link="logit"),n
  = 0)#Distance to cover only model
fit3 = glmer(use ~ d_roads+(1|animal_id), data=data_1, family=binomial(link="logit"),n
  = 0)#Distance to roads only model
fit4 = glmer(use ~ d_cover+d_roads+(1|animal_id), data=data_1, family=binomial(link="l
  nAGQ = 0)#Distance to cover and roads model
fit5 = glmer(use ~ 1|animal_id, data=data_1, family=binomial(link="logit"),nAGQ = 0)#I
```

7. We can view the results of our modeling procedure to select the best model using Akaike's Information Criteria (AIC; Burnham and Anderson 2002).

```
fit1
```

```
## Generalized linear mixed model fit by maximum likelihood (Adaptive
##   Gauss-Hermite Quadrature, nAGQ = 0) [glmerMod]
##  Family: binomial  ( logit )
## Formula: use ~ relevel(crop, "1") + (1 | animal_id)
##    Data: data_1
##       AIC       BIC    logLik  deviance  df.resid
##  31610.18  31659.99 -15799.09  31598.18     29800
## Random effects:
##  Groups    Name        Std.Dev.
##  animal_id (Intercept) 0.3106
## Number of obs: 29806, groups:  animal_id, 14
## Fixed Effects:
##         (Intercept)  relevel(crop, "1")2  relevel(crop, "1")3
##              0.2707               0.6636               0.7483
## relevel(crop, "1")4  relevel(crop, "1")5
##              1.2252               0.9075
```

```
fit2
```

```
## Generalized linear mixed model fit by maximum likelihood (Adaptive
##   Gauss-Hermite Quadrature, nAGQ = 0) [glmerMod]
##  Family: binomial  ( logit )
## Formula: use ~ d_cover + (1 | animal_id)
##    Data: data_1
##       AIC       BIC    logLik  deviance  df.resid
##  31649.75  31674.66 -15821.88  31643.75     29803
## Random effects:
##  Groups    Name        Std.Dev.
##  animal_id (Intercept) 0.345
## Number of obs: 29806, groups:  animal_id, 14
## Fixed Effects:
## (Intercept)      d_cover
##      1.1655      -0.3278
```

```
fit3
```

```
## Generalized linear mixed model fit by maximum likelihood (Adaptive
##   Gauss-Hermite Quadrature, nAGQ = 0) [glmerMod]
##  Family: binomial  ( logit )
## Formula: use ~ d_roads + (1 | animal_id)
##    Data: data_1
##       AIC       BIC    logLik  deviance  df.resid
##  32194.01  32218.92 -16094.00  32188.01     29803
## Random effects:
##  Groups    Name        Std.Dev.
##  animal_id (Intercept) 0.2799
## Number of obs: 29806, groups:  animal_id, 14
## Fixed Effects:
## (Intercept)      d_roads
##     1.14853      0.05144
```

```
fit4
```

```
## Generalized linear mixed model fit by maximum likelihood (Adaptive
##   Gauss-Hermite Quadrature, nAGQ = 0) [glmerMod]
##  Family: binomial  ( logit )
## Formula: use ~ d_cover + d_roads + (1 | animal_id)
##    Data: data_1
##       AIC       BIC    logLik  deviance  df.resid
##  31651.72  31684.93 -15821.86  31643.72     29802
## Random effects:
##  Groups    Name        Std.Dev.
##  animal_id (Intercept) 0.3457
## Number of obs: 29806, groups:  animal_id, 14
## Fixed Effects:
## (Intercept)      d_cover      d_roads
##    1.165420    -0.328110    -0.002653
```

```
fit5
```

```
## Generalized linear mixed model fit by maximum likelihood (Adaptive
##   Gauss-Hermite Quadrature, nAGQ = 0) [glmerMod]
##  Family: binomial  ( logit )
## Formula: use ~ 1 | animal_id
##    Data: data_1
##       AIC       BIC    logLik  deviance  df.resid
##  32202.55  32219.16 -16099.28  32198.55     29804
## Random effects:
##  Groups    Name        Std.Dev.
##  animal_id (Intercept) 0.2867
## Number of obs: 29806, groups:  animal_id, 14
## Fixed Effects:
## (Intercept)
##       1.147
```

```
AIC(fit1,fit2,fit3,fit4,fit5)
```

```
##      df      AIC
## fit1  6 31610.18
## fit2  3 31649.75
## fit3  3 32194.01
## fit4  4 31651.72
## fit5  2 32202.55
```

```
mynames <- paste("fit", as.character(1:5), sep = "")
myaicc <- aictab(list(fit1,fit2,fit3,fit4,fit5), modnames = mynames)
print(myaicc, LL = FALSE)
```

```
##
## Model selection based on AICc:
##
##      K     AICc Delta_AICc AICcWt Cum.Wt
## fit1 6 31610.18       0.00      1      1
## fit2 3 31649.75      39.57      0      1
## fit4 4 31651.73      41.54      0      1
## fit3 3 32194.01     583.83      0      1
## fit5 2 32202.55     592.37      0      1
```

8. Our top model (fit 1) has all the support in this case indicating that during winter 2012 the mule deer were selecting for each habitat over sunflower. Considering sunflower is not available during the winter months this makes perfect sense. Looking at parameter estimates and confidence intervals for the additional habitat categories in fit 1 we see that forest (category 4) is most selected habitat followed by shrub (category 5). This is only a simply way to look at habitat, however, we used more animals that were on the air for several years and also could look at distance to habitat instead of representing habitat as categorical data.

```
per1_se <- sqrt(diag(vcov(fit1)))
# table of estimates with 95% CI
tab_per1 <- cbind(Est = fixef(fit1), LL = fixef(fit1) - 1.96 * per1_se, UL = fixef(fit1)
  + 1.96 * per1_se)
tab_per1
```

```
##                              Est         LL        UL
## (Intercept)          0.2706915 0.08792365 0.4534593
## relevel(crop, "1")2 0.6636420 0.54049466 0.7867894
## relevel(crop, "1")3 0.7482707 0.61547197 0.8810694
## relevel(crop, "1")4 1.2251593 1.12604248 1.3242761
## relevel(crop, "1")5 0.9074680 0.81258760 1.0023483
```

10. We can then create a surface of predictions from our top model indicating where in our study site we might find the highest probability of use. To do this, we need to export a text file from our "layer" created in Exercise 8.3.

```
layer1 <- read.table("layer1.txt",sep=",")
names(layer1) = c("crop", "d_cover", "d_roads","x", "y")
#Need to standardize the raw distance rasters first to match what we modeled
layer1[,2:3]=scale(layer1[,2:3],scale=TRUE)
head(layer1)
layer1$crop <- as.factor(layer1$crop)

# predictions based on best model
predictions = predict(fit1, newdata=layer1, re.form=NA, type="link")#based on the scale of the
  #linear predictors
predictions = exp(predictions)
range(predictions)

#-------------------------------------------------------------------------------------
# create Ascii grid of raw predictions if needed
layer1$predictions = predictions
#preds = layer1
#preds = SpatialPixelsDataFrame(points=preds[c("x", "y")], data=preds)
#preds = as(preds, "SpatialGridDataFrame")
#names(preds)
#writeAsciiGrid(preds, "predictions.asc", attr=13)#attr should be column number for 'predictions

#-------------------------------------------------------------------------------------
# assign each cell or habitat unit to a 'prediction class'.
# classes have (nearly) equal area, if the cells or habitat units have equal areas.
# output is a vector of class assignments (higher is better).
F.prediction.classes <- function(raw.prediction, n.classes){
  # raw.prediction = vector of raw (or scaled) RSF predictions
  # n.classes = number of prediction classes.
  pred.quantiles = quantile(raw.prediction, probs=seq(1/n.classes, 1-1/n.classes,
  by=1/n.classes))
  ans = rep(n.classes, length(raw.prediction))
  for(i in (n.classes-1):1){
    ans[raw.prediction < pred.quantiles[i]] = i
  }
  return(ans)
}

layer1$prediction.class = F.prediction.classes(layer1$predictions, 5)
table(layer1$prediction.class)

##############################################
# create map of RSF prediction classes in R
# m = SpatialPixelsDataFrame(points = layer1[c("x", "y")], data=layer1)
# names(m)
# par(mar=c(0,0,0,0))
# image(m, attr=7, col=c("grey90", "grey70", "grey50", "grey30", "grey10"))
# par(lend=1)
# legend("bottomright", col=rev(c("grey90", "grey70", "grey50", "grey30", "grey10")),
#        legend=c("High", "Medium-high", "Medium", "Medium-low", "Low"),
#        title="Prediction Class", pch=15, cex=1.0,bty != "n", bg="white")

# create Ascii grid of prediction classes
#m = as(m, "SpatialGridDataFrame")
#names(m)
#writeAsciiGrid(m, "PredictionClassess.asc", attr=7)
```

7.6 Negative Binomial

1. Exercise 8.6 - 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 NegBinomial.Rmd" and run code directly from the script

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

```
library(plyr)
library(adehabitatHR)
library(rgeos)
library(raster)
library(rgdal)
library(zoo)
library(MASS)#For nb models
```

5. Now let's 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

```
utm12.crs<-"+proj=utm +zone=12 +datum=WGS84"
Albers.crs<-"+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0
  +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
```

6. Load files for the mule deer dataset and clean up the dataset as we have done in previous exercises

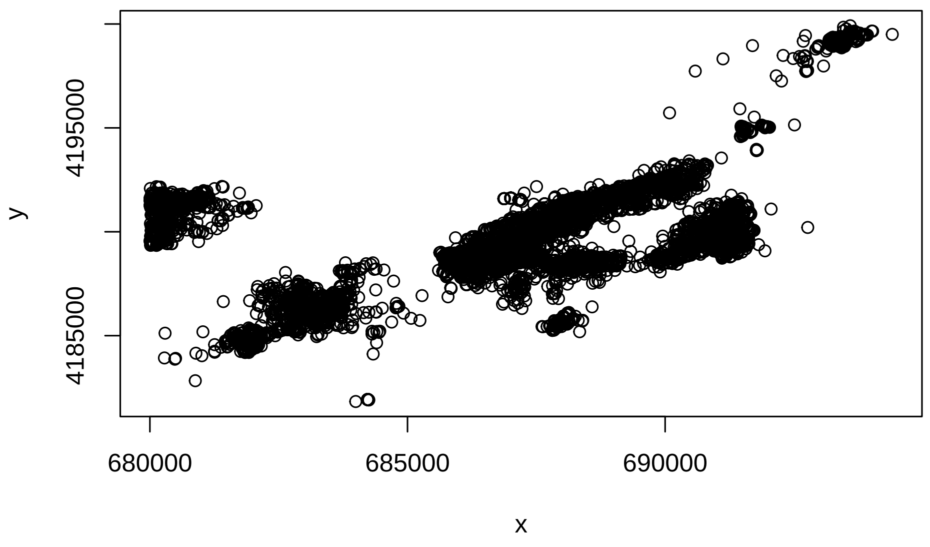
```
#muleys<-read.csv("muleysexample.csv", header=T, sep=",")
muleys<-read.csv("DCmuleysedited.csv", header=T, sep=",")

muleys$NewDate<-as.POSIXct(muleys$GPSFixTime, format="%Y.%m.%d %H:%M:%S", origin="1970-01-01")
muleys <- subset(muleys, muleys$id != "D19")

##Sort Data
muleys <- muleys[order(muleys$id, muleys$NewDate),]

fix_rate <- function(x){
  print(paste("Individual:", x$id[1]))
  dates=range(x$NewDate)
  print(paste("Range:", dates))
  days=as.numeric(round(diff(range(x$NewDate)), digits=0))
  print(paste("Number of monitoring days:", days))
  sched=as.numeric(round(median(abs(diff(sapply(x$NewDate[2:nrow(x)], difftime, time1 = x$NewDate
  print(paste("Scheduled fix rate (min):", sched))
  expected=as.numeric(days)*round(1440/(as.numeric(sched)), digits=0)#1440 minutes in 
  print(paste("Expected number of positions:", expected))
  success <- nrow(x)
  print(paste("Number of recorded positions", success))
  percentfix <- round(success/expected*100)
  print(paste("Fix rate success = ", percentfix,"%", sep=" "))
}

fix_deer <- ddply(muleys, .(as.factor(id)), fix_rate)
```

```
## [1] "Individual: D12"
## [1] "Range: 2011-08-12 15:01:53" "Range: 2011-10-24 21:00:48"
## [1] "Number of monitoring days: 73"
## [1] "Scheduled fix rate (min): 180"
## [1] "Expected number of positions: 584"
## [1] "Number of recorded positions 120"
## [1] "Fix rate success =  21 %"
## [1] "Individual: D15"
## [1] "Range: 2011-10-12 00:02:03" "Range: 2012-08-31 09:00:51"
## [1] "Number of monitoring days: 324"
## [1] "Scheduled fix rate (min): 180"
## [1] "Expected number of positions: 2592"
## [1] "Number of recorded positions 2589"
## [1] "Fix rate success =  100 %"
## [1] "Individual: D16"
## [1] "Range: 2011-10-11 21:00:36" "Range: 2012-07-08 12:01:46"
## [1] "Number of monitoring days: 271"
## [1] "Scheduled fix rate (min): 180"
## [1] "Expected number of positions: 2168"
## [1] "Number of recorded positions 2157"
## [1] "Fix rate success =  99 %"
## [1] "Individual: D4"
## [1] "Range: 2010-09-13 21:08:35" "Range: 2012-03-22 15:01:40"
## [1] "Number of monitoring days: 556"
## [1] "Scheduled fix rate (min): 180"
## [1] "Expected number of positions: 4448"
## [1] "Number of recorded positions 1304"
## [1] "Fix rate success =  29 %"
## [1] "Individual: D6"
## [1] "Range: 2011-10-11 21:00:39" "Range: 2012-04-11 15:00:48"
## [1] "Number of monitoring days: 183"
## [1] "Scheduled fix rate (min): 180"
## [1] "Expected number of positions: 1464"
## [1] "Number of recorded positions 1455"
## [1] "Fix rate success =  99 %"
## [1] "Individual: D8"
## [1] "Range: 2010-09-13 22:07:35" "Range: 2012-03-01 00:02:08"
## [1] "Number of monitoring days: 534"
## [1] "Scheduled fix rate (min): 180"
## [1] "Expected number of positions: 4272"
## [1] "Number of recorded positions 971"
## [1] "Fix rate success =  23 %"
```

```
##TIME DIFF NECESSARY IN BBMM CODE
timediff <- diff(muleys$NewDate)*60
## remove first entry without any difference
muleys <- muleys[-1,]
muleys$timelag <-as.numeric(abs(timediff))
##Remove locations greater than 5.5 hours apart in time
muleys <- subset(muleys, muleys$timelag < 19800)
summary(muleys$timelag)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3581   10792   10800   10795   10808   14401
```

```
muleys <- subset(muleys, muleys$X > 680000)# & muleys$GPS.UTM.Easting != "NA")
muleys$id <- factor(muleys$id)

##Make a spatial data frame of locations after removing outliers
muleysSPDF<-data.frame(x = muleys$X, y = muleys$Y)
plot(muleysSPDF, axes=T)#To visualize all locations
```



```
utm.spdf <- SpatialPointsDataFrame(coords = muleysSPDF, data = muleys, proj4string =
  CRS(utm12.crs))

##change muleysSPDF from UTM to Albers
muleys.spdf <-spTransform(utm.spdf, CRS=Albers.crs)
```

```
## Warning: PROJ support is provided by the sf and terra packages among others
```

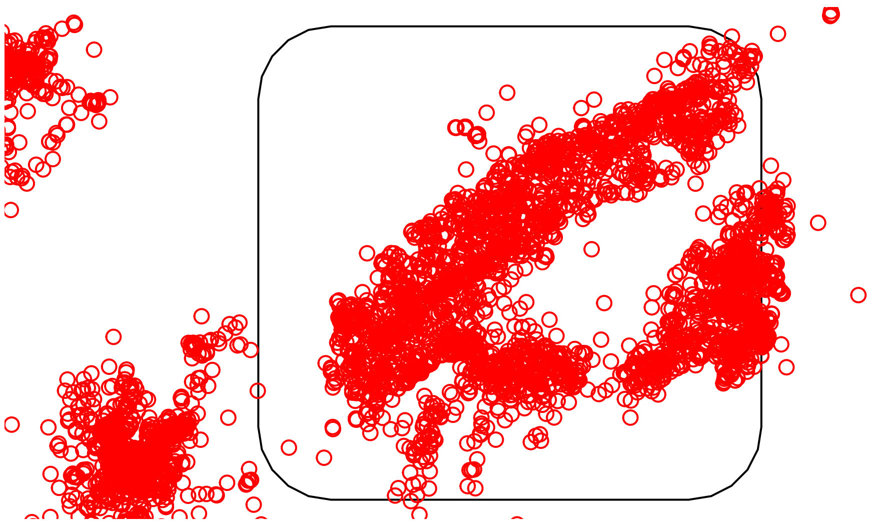
```
#Make data of albers x and y and replace with original muleys dataframe
muleys <- as.data.frame(muleys.spdf)

##We can create and clip with a rectangular grid
bbox(muleys.spdf)
```

```
##         min      max
## x -1126432 -1110147
## y  1712932  1729463
```

```
bb1 <- cbind(x=c(-1120488,-1120488,-1115562,-1115562,-1120488),
   y=c(1718097, 1722611,1722611,1718097,1718097))
muleysAlbersSP <- SpatialPolygons(list(Polygons(list(Polygon(bb1)),"1")),
  proj4string=CRS(proj4string(muleys.spdf)))

##Let's buffer around the bounding box to be sure it
#encompasses all locations
muleysbuffSP <- gBuffer(muleysAlbersSP,width=1000)
```

```
## Warning: GEOS support is provided by the sf and terra packages among others
```

```
plot(muleysbuffSP)
points(muleys.spdf,col="red")
```



```
#Subset locations by year for season-specific RSFs if needed
winter2012 <- muleys
# winter2012 <- crop(muleys.spdf,muleysbuffSP)
# winter2012$id <- droplevels(winter2012$id)
```

7. Now we need to set up our sample circles across our study area keeping in mind our discussions on "available" habitats. For this exercise, we will keep it simple by only including a polygon around our mule deer locations. This is for demonstration only, the appropriate study area should be specific to your study design and objectives. We also need to determine what is the appropriate size of our sample circles. In this case, we will use the mean daily movement distance for mule deer we determined to be 628 meters. This will be the radius of our sample circles.

```
#Determine the boundary around our mule deer locations that we will consider to be
#available to all deer in our analysis
bbox(muleysbuffSP)
```

```
##         min      max
## x -1121488 -1114562
## y  1717097  1723611
```

```
#  min       max
#x -1121488 -1114562
#y  1717097  1723611

#Create a square as we did in Exercise 1.9. Start by creating vectors of the x and y points
```

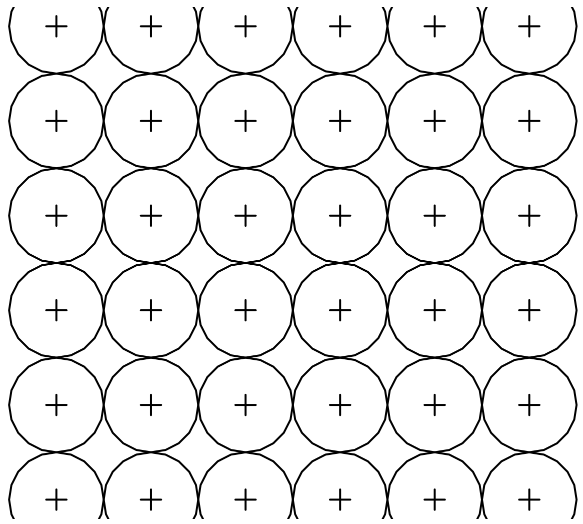
```
x <- seq(from = -1121488, to = -1114562, by = 1256)#628 m daily move distance times 2
y <- seq(from = 1717097, to =  1723611, by = 1256)

#Create a grid of all pairs of coordinates (as a data.frame)
#Also the restart point for later!!
xy <- expand.grid(x = x, y = y)

#Identify projection before creating Spatial Points Data Frame
grid.pts<-SpatialPointsDataFrame(coords= xy, data=xy, proj4string = CRS(Albers.crs))
plot(grid.pts)
gridded(grid.pts)
```

```
## [1] FALSE
```

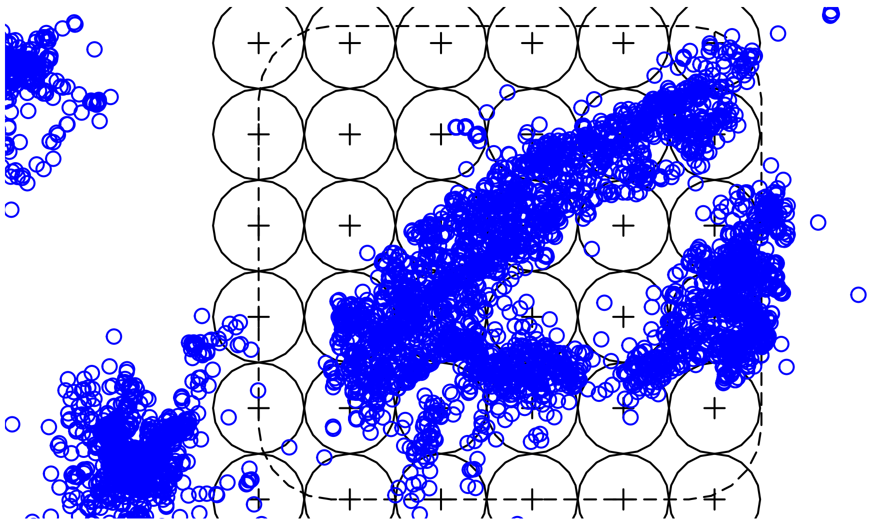
```
griddata <- as.data.frame(grid.pts)

#Create buffers around grid points
mbuffer <- gBuffer(grid.pts,width=628,byid=TRUE)
```

```
## Warning: GEOS support is provided by the sf and terra packages among others
```

```
mbuff.spdf <- SpatialPolygonsDataFrame(mbuffer, data=data.frame(id=row.names(mbuffer),
  row.names=row.names(mbuffer)))
plot(grid.pts)
plot(mbuff.spdf,add=T)
```



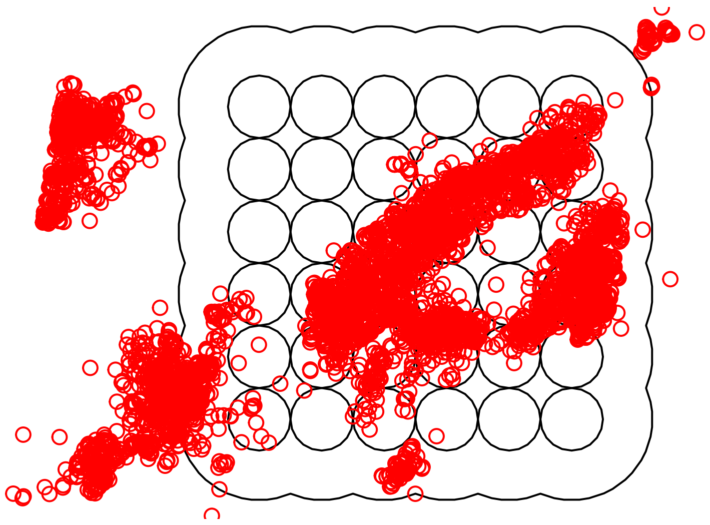
```r
plot(muleysbuffSP,lty=5)
plot(mbuff.spdf,add=T)
points(grid.pts, pch=3)
points(muleys.spdf, col="blue")
```



```r
#encompasses all locations
muleysbuffSP2 <- gBuffer(mbuff.spdf,width=1000)
```

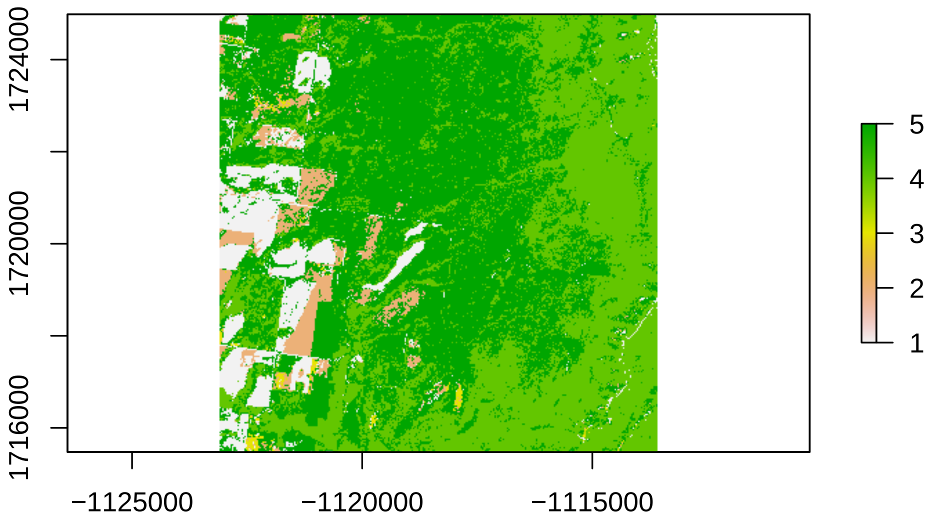
```
## Warning: GEOS support is provided by the sf and terra packages among others
```

```r
plot(muleysbuffSP2)
plot(mbuff.spdf,add=T)
points(muleys.spdf,col="red")
```

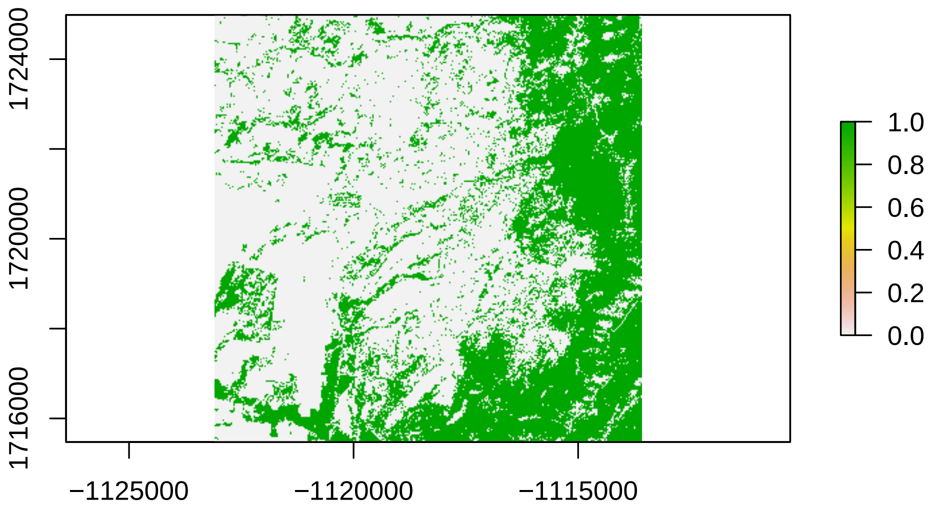


8. We will start here by creating covariate layers specifically for the year of interest, in this case crop data from NRCS for 2011

```
# Then clip buffer from crop11 layer
crop11 <- raster("crop11clip")
mulcrop11clip<-crop(crop11, muleysbuffSP2)
#Crop categories
#1 = Sunflower,summer crops, random crops, grassland
#2 = Winter crops
#3 = Alfalfa
#4 = Forest
#5 = Shrubland

#Reclassify into 5 habitat categories
m11 <- c(-Inf,0,NA, 5.5, 6.5, 1, 22.5, 24.5, 2, 26.5, 27.5, 2, 29.5, 30.5, 2, 35.5, 36
  3, 3.5, 4.5, 1, .5, 1.5, 1, 11.5, 12.5, 1, 27.5, 28.5, 1, 31.5, 33.5, 1,  42.5, 43.5
  47.5, 49.5, 1, 58.5, 59.5, 1, 60.5, 61.5, 1, 65.5, 69.5, 1, 76.5, 77.5, 1, 110.5,
  111.5, 1, 120.5, 124.5, 1, 130.5, 131.5, 1, 189.5, 190.5, 1, 194.5, 195.5, 1, 228.5,
  229.5, 1, 140.5, 143.5, 4, 170.5, 171.5, 1, 180.5, 181.5, 1, 36.5, 37.5, 1, 151.5,
  152.5, 5, 41.5, 42.5, 1, 204.5, 205.5, 1, 230,Inf,NA)
rclmat11 <- matrix(m11, ncol=3, byrow=TRUE)
crop11rc <- reclassify(mulcrop11clip, rclmat11)
plot(crop11rc)
```

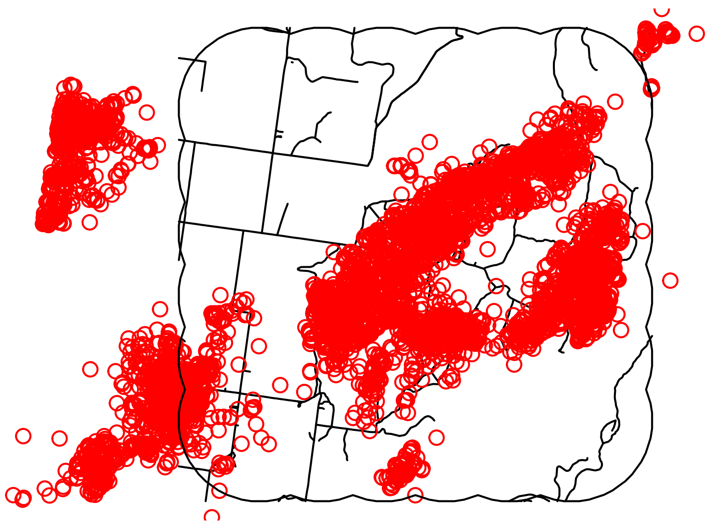


```
##Create cover layer
cov <- c(-Inf,0,NA, 1, 140, 0, 140.5, 143.5, 1, 151.5,
  205.5, 0, 230,Inf,NA)#cover=forest only
rclmatcov <- matrix(cov, ncol=3, byrow=TRUE)
cover <- reclassify(mulcrop11clip, rclmatcov)
plot(cover)
```



```
coverdf <- as.data.frame(as(cover,"SpatialGridDataFrame"))
coveronly <- subset(coverdf,coverdf$crop11clip=="1")
coveronly <- coveronly[c(-1)]
d_cover <- distanceFromPoints(crop11rc,coveronly)
```

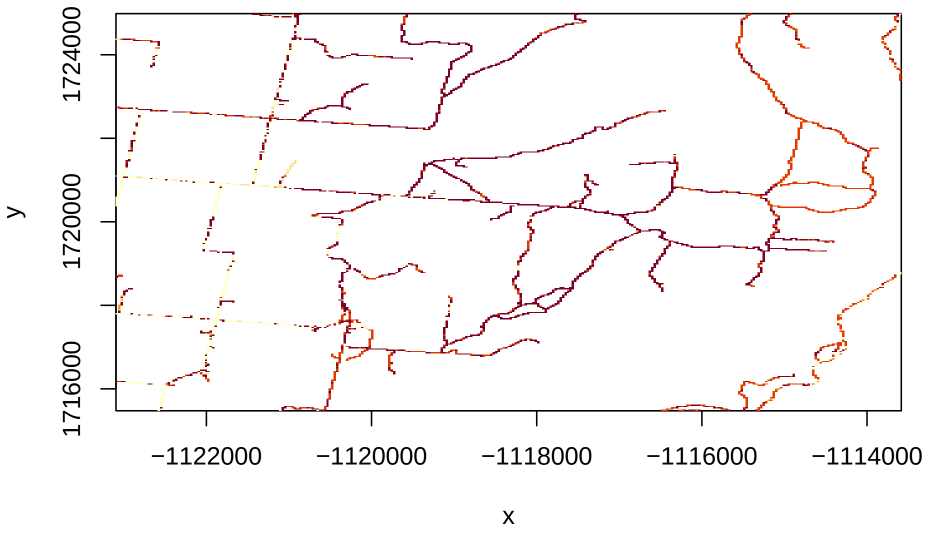
```
##Bring in roads layer
roads<-readOGR(dsn=".",layer="AlbersRoads", verbose = FALSE)

#Clip using muleysbuffSP
roadclip <- crop(roads, muleysbuffSP2)
cliproads <- gIntersection(roads, muleysbuffSP2, byid=TRUE)

plot(roadclip)
points(muleys.spdf, col="red")
plot(muleysbuffSP2, add=TRUE)
```



9. The downside of creating distance to roads with spatstat in Exercise 8.2 is that it is not in a raster so we need to create a raster of distance to roads for every raster cell in layer1 before grabbing values or making our predictive surface.

We will start by using the Rasterize function to create a raster of the road shapefile with crop data used as a mask. A mask will give the spatial resolution and projection information to the raster you plan to create.

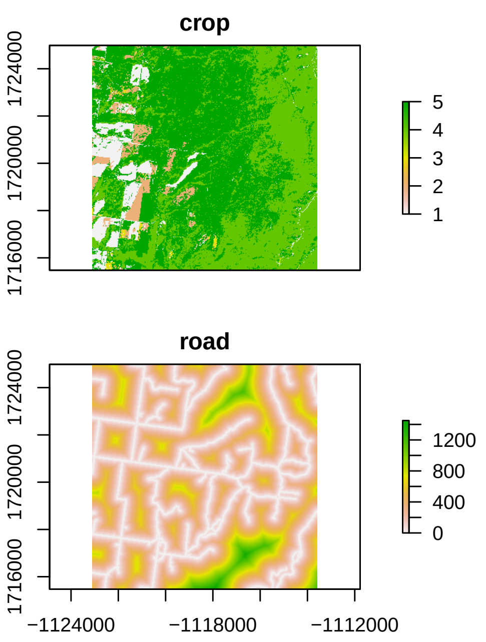
```
roadrast <- rasterize(roadclip,crop11rc, mask=TRUE)
image(roadrast)
```



```
#Now make a dataframe of all raster cell locations for each road segment
roadrastdf <- as.data.frame(as(roadrast, "SpatialGridDataFrame"))

#Remove the non-xy column of the data frame
roadrastdf <- roadrastdf[c(-1)]
#Use raster package to create distance from each raster
#cell to each road layer raster cell.
d_roadrast <- distanceFromPoints(crop11rc, roadrastdf)

##MAKE ALL RASTER LAYERS DATAFRAMES TO COMBINE LATER
crop11df <- as.data.frame(as(crop11rc, "SpatialGridDataFrame"))
#Distance to cover
d_covdf <- as.data.frame(as(d_cover, "SpatialGridDataFrame"))
#Distance to roads
final_roaddf <- as.data.frame(as(d_roadrast, "SpatialGridDataFrame"))

#Combine data frames for Crop and Distance to Cover and Roads
layers1 = cbind(crop11df, d_covdf,final_roaddf)
layers1 = layers1[,-c(2,3,5,6)]
names(layers1) = c("crop","d_cover","d_roads","x", "y")
#write.table(layers1,"layer1.txt",sep=",",col.names=TRUE, quote=FALSE)
#Now we need to extract all raster layers, grid and create a stack of all rasters
r <- stack(list(crop=crop11rc, cover=d_cover, road=d_roadrast))

nlayers(r)
```

```
## [1] 3
```

```
plot(r)
```



```
names(r)
```

```
## [1] "crop"  "cover" "road"
```

```
ext <- extract(r, mbuff.spdf, weights=TRUE, fun = mean)

#NOTE above that for each grid cell in the sampling grid layer (i.e., grid), the "extr
```

10. Code below extracts by land cover category and determines how many cells of each type were in each sample circle.

```
ex_crop <- (extract(crop11rc, mbuff.spdf, byid=TRUE))

#Land Cover category and number of cells (30x30m raster cells)
tab <- lapply(ex_crop, table)
tab[[1]]
```

```
##
##   1   2   3   4   5
## 364 208 100 159 519
```

```
#  1   2   3   4   5
#163  79  37   1  48
tab[[18]]
```

```
##
##   1   4   5
##   6 464 887
```

```
#1   4   5
#6 464 887

#What issue do you notice with the above habitat categories?

############################################
#Code here thanks to Tyler Wagner, PA Coop Unit, for creating this loop to summarize
#proportions of habitat within each grid cell
############################################
##Created land use categories
lus <- 1:5
##### Loop through and append missing land use categories to each grid cell
ex_crop_new <- list()
for(i in 1:length(mbuff.spdf)[1] ){
  # Land use cats in a given cell
  temp1 <- unique(ex_crop[[i]])
  # Give missing category 999 value
  ma1 <- match(lus, temp1, nomatch = 999, incomparables = NULL)
  # Get location (category of missing land use type)
  miss <- which(ma1%in%999)
  ex_crop_new[[i]] <- c(ex_crop[[i]], miss)
}

# New summary of land use in a grid cell
tab2 <- lapply(ex_crop_new, table)
tab2[[18]]
```

```
##
##   1   2   3   4   5
##   6   1   1 464 887
```

```
tab[[18]]
```

```
##
##   1   4   5
##   6 464 887
```

```
# Proportions of all land cover types per grid cell
prop <- list()
for(i in 1:length(mbuff.spdf)[1] ){
  prop[[i]] <- round((margin.table(tab2[[i]],1)/margin.table(tab2[[i]])),digits = 6)
}
#Function coredata is from the zoo package to convert the proportions from a list
#to a matrix
M <- coredata(do.call(cbind, lapply(prop, zoo)))
colnames(M) <- NULL
#Transpose matrix so land cover become separate columns of data
matrix <- t(M)
#Now convert the matrix to a data frame so it is easier to manipulate
dfland <- as.data.frame(matrix)
#Assing column names to land cover
colnames(dfland) <- c("sunflower","wintercrop","alfalfa","forest","shrub")
#Write out csv with new nlcd circle percents
#write.csv(dfland,paste(".", "circl_perc_nlcd.csv",sep=""))
```

11. Now that we have Land Cover in a similar format as the distance-to-derived data, we want to convert ext(the combined extracted rasters) into a data frame so it is easier to manipulate as well. The "extract" function in the raster package is supposed to be able to do this but does not work for some reason.

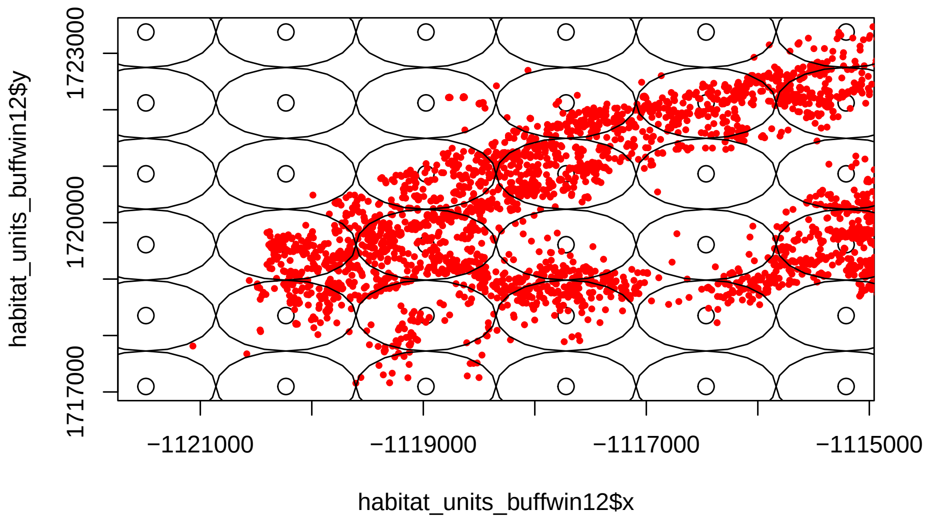
```
a <- as.data.frame(ext)
habitat_units_buffwin12 <- cbind(dfland, a)
```

12. Now we need to convert to a data frame for nb modeling. Read in animal_locations.txt or convert to data frame from above

```
#locations = read.table("deer_locations.txt", sep='\t', header=T)
locations.spdf <- crop(muleys.spdf,muleysbuffSP)
locations.df = as.data.frame(locations.spdf)
#locations.df = as.data.frame(winter2012)
locations <- locations.df[c(-1,-3:-24)]

#Add xy columns of circle centroids
mbuff.xy <- as.data.frame(grid.pts)
str(mbuff.xy)
```

```
## 'data.frame':    36 obs. of  4 variables:
##  $ x  : num  -1121488 -1120232 -1118976 -1117720 -1116464 ...
##  $ y  : num  1717097 1717097 1717097 1717097 1717097 ...
##  $ x.1: num  -1121488 -1120232 -1118976 -1117720 -1116464 ...
##  $ y.1: num  1717097 1717097 1717097 1717097 1717097 ...
```

```r
habitat_units_buffwin12$x <- mbuff.xy$x
habitat_units_buffwin12$y <- mbuff.xy$y

plot(habitat_units_buffwin12$x, habitat_units_buffwin12$y, type='p', cex=1.5)
points(locations$x, locations$y, col="red", cex=0.5, pch=19)
plot(mbuff.spdf, add=T)
```



13. Calculate number of animal locations in each sampled habitat unit(see code in “count_locations.R”).

```r
# Source code file containing functions used below.
source("count_locations.R")

pooled.locations = locations
colnames(pooled.locations) <- c("ID","x","y")
pooled.locations$ID = 1
NB = F.count.relocations(locations.df = pooled.locations,
    habitat.units.df = habitat_units_buffwin12,
    habitat.unit.size = 628)
# List of column names:
names(NB)
```

```
##  [1] "sunflower"   "wintercrop" "alfalfa"    "forest"     "shrub"
##  [6] "crop"        "cover"      "road"       "x"          "y"
## [11] "animal.ID"   "n.locations" "total"
```

```r
#Look at the range in number of locations in our sample circles
summary(NB$n.locations)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00   18.00   82.97  131.00  521.00
```

```
#Now run a population-level model for a few covariates (forest, road). NOTE: If you ru
nb = glm.nb(n.locations ~ offset(log(total)) + forest + road, data=NB)
summary(nb)
```

```
##
## Call:
## glm.nb(formula = n.locations ~ offset(log(total)) + forest +
##     road, data = NB, init.theta = 0.1862816104, link = log)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.702164   0.770412  -3.507 0.000452 ***
## forest      -0.147529   1.566934  -0.094 0.924989
## road        -0.003929   0.002155  -1.823 0.068282 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(0.1863) family taken to be 1)
##
##     Null deviance: 38.910  on 35  degrees of freedom
## Residual deviance: 37.642  on 33  degrees of freedom
## AIC: 322.4
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  0.1863
##           Std. Err.:  0.0462
##
##  2 x log-likelihood:  -314.3950
```

```
#-------------------------------------------------------------------------------
# Proportion of 0 counts in data
sum(NB$n.locations == 0)/nrow(NB)
```

```
## [1] 0.3888889
```

```
nb.density = structure(
function # Probability mass function for NB2

# Description: This function gives the probability that a discrete random variable, X,
#is exactly equal to some value  according to a NB2 distribution.
# Returns: Pr(X=k)

(k,
### value at which to estimate probability
### Pr(X=k)
mu,
### NB2 estimate of mu
theta
### NB2 estimate of theta
){

    (gamma(theta+k)/(gamma(theta)*factorial(k))*
        (mu^k)*(theta^theta)/
        ((mu+theta)^(theta+k))

})
# Expected proportion under NB2 model
nb.density(k=0, mu=mean(NB$n.locations), theta=0.1861)
```

```
## [1] 0.321222
```

```
             # (Note: use estimated theta of the model output found in summary statement above)
# The value above can be interpreted as:
# "A NB2 distribution with theta=0.1861 and mu=0.9196 should have an average of 32% zero values"

#Observed
zero = NB$n.locations == 0
sum(zero)   #total number of zeros
```

```
## [1] 14
```

```
mean(zero)  #proportion that are zeros
```

```
## [1] 0.3888889
```

```
#Expected based on NB distribution and our observed over-dispersions
theta = mean(NB$n.locations)^2 / (var(NB$n.locations) - mean(NB$n.locations))
check = rnegbin(n=10000, mu=mean(NB$n.locations), theta=theta)
check.zeros = check == 0
mean(check.zeros)
```

```
## [1] 0.0929
```

```
#saveRDS(habitat_units_buffwin12, "Exercise.8.6.rds")
```