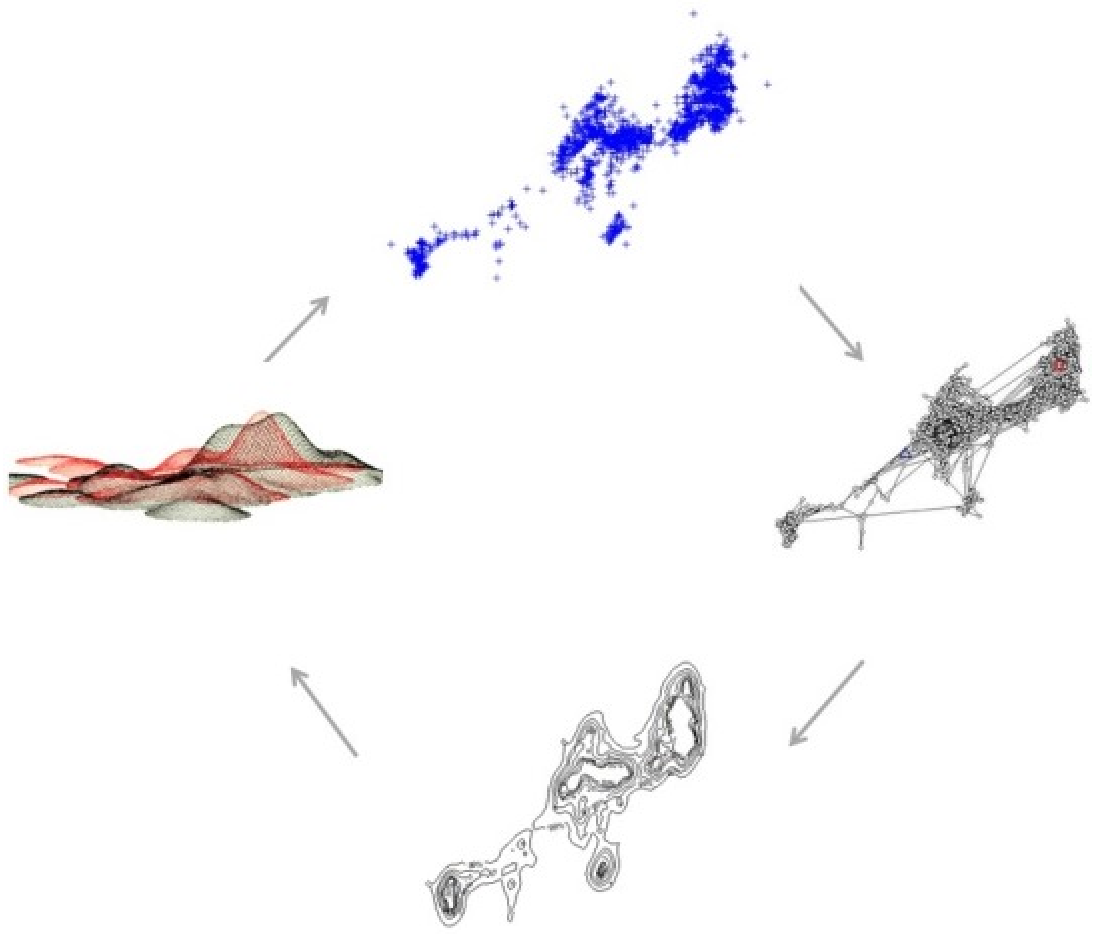


Manual of Applied Spatial Ecology

W. David Walter
U.S. Geological Survey, Pennsylvania Cooperative Fish and
Wildlife Research Unit, Pennsylvania State University, University Park, PA 16802, USA

and

Justin W. Fischer
United States Department of Agriculture, Animal and Plant Health Inspection
Service, Wildlife Services, National Wildlife Research Center, 4101 LaPorte Avenue, Fort
Collins, CO 80521, USA

1 February 2016

Contents

List of Figures

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. and J.W. Fischer. 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@west-inc.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., Pennyslvania State University, University Park, PA 16802*; twagner@psu.edu

Chapter 1

Data Manipulation and Management

Contents

Figures

1.1 Load R software and packages

1. Install R
   To assure proper functioning of downloaded packages, install R to a subdirectory for which you have complete read/write access. We used subdirectory `c:\Program Files\R`. Path statements that follow reflect this location.

2. Some quick code to set CRAN mirror and load some packages

```
install.packages(c("gpclib","ade4","adehabitat","adehabitatHR","chron","raster",
     "rgdal", "shapefiles"), dependencies=TRUE,
     repos="http://lib.stat.cmu.edu/R/CRAN/")

#Load needed libraries
library(adehabitatHR) #package needed for home range estimation
library(adehabitat) #package needed import.asc function
library(sp) #package needed to import and manipulate raster datasets
library(rgdal) #package needed to import ascii files into R
library(raster) #package needed to manipulate raster files
```

1.2 Geographic coordinate systems

Geographic Coordinate Systems uses a three-dimensional spherical surface to define locations on the earth. Points on the earth's surface are referenced by longitude (north-south vertical lines) and latitude (east-west horizontal lines) measured in degrees (or in grads) as angles from the earth's center. Although longitude and latitude can locate an exact position on the earth's surface, they are not uniform units of measure. For example, latitude gets gradually smaller as one leaves the equator and approaches the poles.

Datums define the position of the spheroid relative to the center of the earth by defining the origin and orientation of latitude and longitude lines. Because local datums are aligned with a particular area of the earth's surface, a datum for Europe (ED 1950) can't be used to reference locations in North America (NAD) and vice versa. There are numerous datums and dates of datums primarily due to the improvements of satellite data with WGS 1984 serving as the framework for locational measurements worldwide.

1.3 Projected coordinate systems

Projected Coordinate Systems are defined on a flat, two-dimensional surface with different projections causing different types of distortions. Various projected coordinate systems have been developed for different regions that provides a common framework to perform spatial analysis. Choosing the projection for data analysis requires knowledge of the spatial distribution and extent of GPS points (see for example Fig. 1.1; Walter et al. 2011).

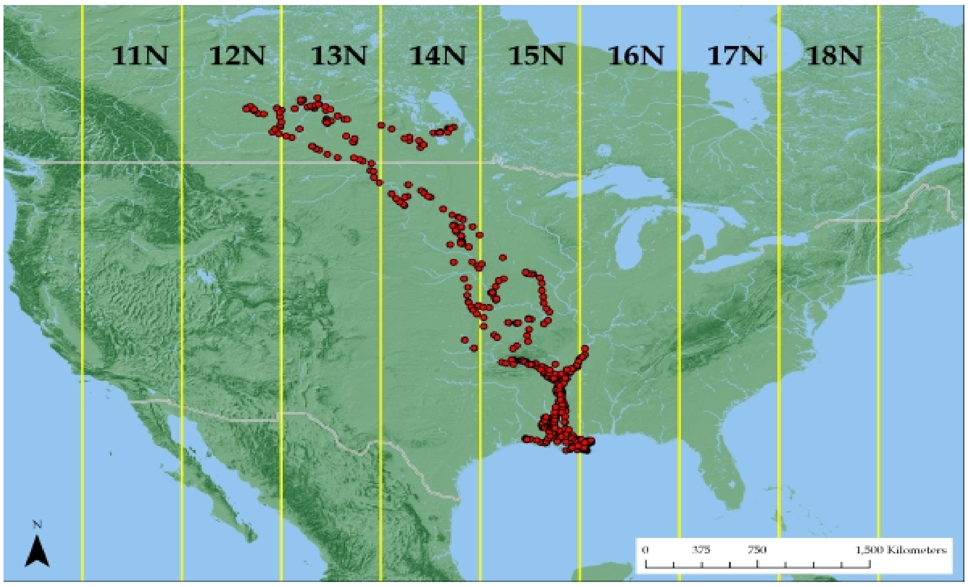


Figure 1.1: American White Pelican with locations spanning 5 UTM zones

Use of the geographic coordinate system (i.e. latitude, longitude) is recommended in cases of long distance movements and is often the default geographic collection method for

GPS collar data. However, some home range software (e.g. BBMM package in R) requires input coordinate data to be in meters so Albers Equal Area Conic or Universal Transverse Mercator could be used (Fig. 1.2).

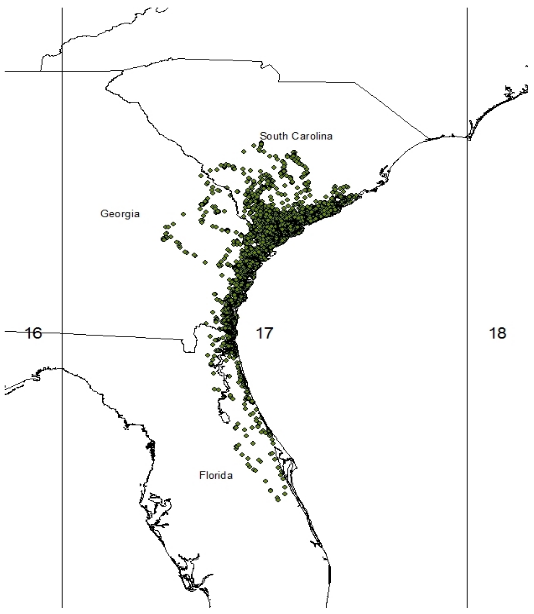


Figure 1.2: Turkey vulture locations only occurring in one UTM zone

1.4 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.

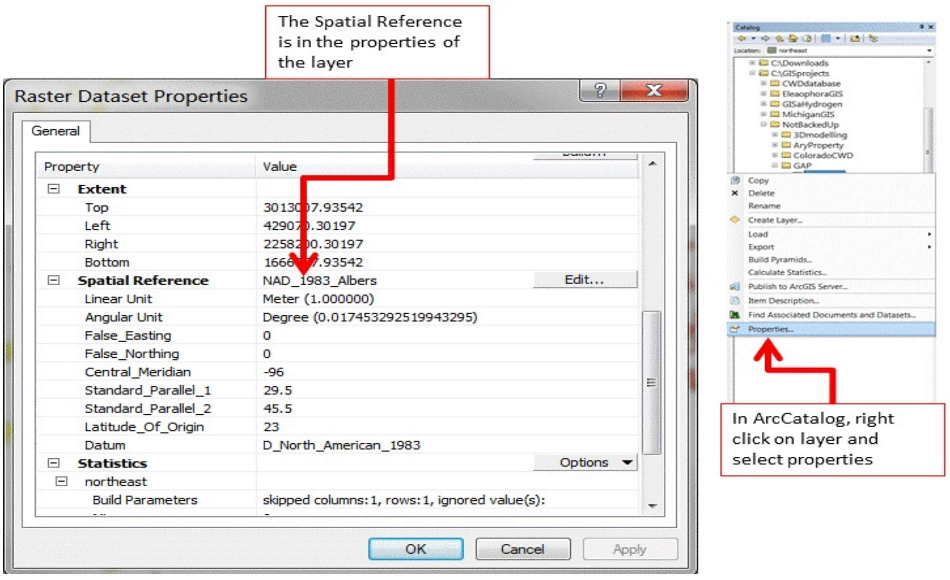
1. What coordiate system were the data collected in?

   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; Fig. 1.3). 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



Property	Value
Extent	
Top	3013[illegible]7.93542
Left	42907[illegible].30197
Right	2258[illegible]0.30197
Bottom	166[illegible]7.93542
Spatial Reference	NAD_1983_Albers
Linear Unit	Meter (1.000000)
Angular Unit	Degree (0.0174532925199433295)
False_Easting	0
False_Northing	0
Central_Meridian	-96
Standard_Parallel_1	29.5
Standard_Parallel_2	45.5
Latitude_Of_Origin	23
Datum	D_North_American_1983
Statistics	
northeast	
Build Parameters	skipped columns:1, rows:1, ignored value(s):

Figure 1.3: How to find coordinates system of GIS layer in ArcCatalog or Table of Contents in ArcMap

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 File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(rgdal)
   ```

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

   ```
   study.states<-readOGR(dsn=".",layer="MDcounties")

   #OGR data source with driver: ESRI Shapefile
   #Source: ".", layer: "MDcounties"
   #with 38 features and 8 fields
   #Feature type: wkbPolygon with 2 dimensions

   plot(study.states, col="grey")

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

   OGR data source with driver: ESRI Shapefile
   Source: ".", layer: "MDzoom"
   with 1 features and 1 fields
   Feature type: wkbPolygon with 2 dimensions

   plot(study.zoom, col="grey")
   ```

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

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

6. 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)
crs<-"+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0"
coords
plot(coords)
```

7. Remove outlier locations

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

8. 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)
crs<-"+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0"
coords
plot(coords)
```

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

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

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

```
new.crs <-CRS("+proj=utm +zone=12 +datum=WGS84")
MDzoomUTM12 <-spTransform(study.zoom, CRS=new.crs)
par(new=TRUE)
plot(MDzoomUTM12, col="bisque")
class(MDzoomUTM12)
proj4string(MDzoomUTM12)
summary(MDzoomUTM12)

#projection for mule deer locations
deer.crs <-CRS("+proj=utm +zone=12 +datum=WGS84")
deerUTM12 <-spTransform(deer.spdf, CRS=deer.crs)
points(deerUTM12, col="red")
class(deerUTM12)
proj4string(deerUTM12)
deerUTM12[1:5,]

#See new projected coordinates in UTM 12N for the first 5 locations
```

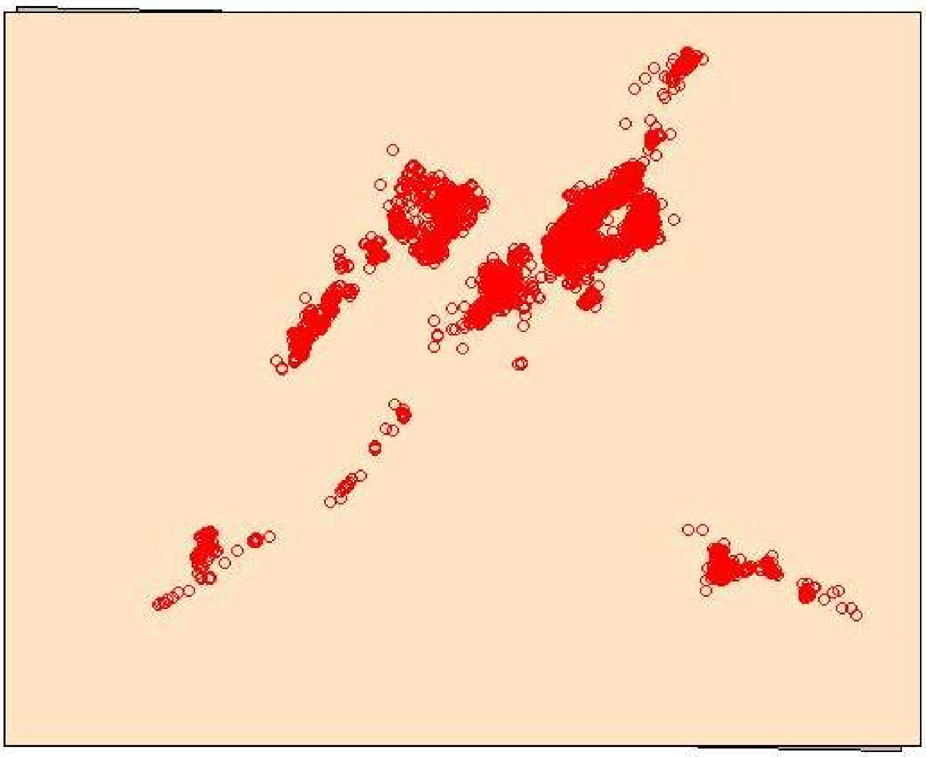


Figure 1.4: Study area projected into UTM Zone 12N with original WGS84 underneath

```
coordinates(deerUTM12)[1:5,]
            x       y
[1,] 677825.2 4192832
[2,] 677853.8 4192787
[3,] 677736.3 4192728
[4,] 677595.9 4192398
[5,] 677666.2 4192362

#plot coordinates in Lat Long over coordinates in UTM 12N
plot(coords)
par(new=TRUE)
plot(deerUTM12, col="red")

windows()
plot(study.zoom)
par(new=TRUE)
plot(deer.spdf, col="red")

windows()
plot(MDzoomUTM12,col="bisque")
par(new=TRUE)
plot(deerUTM12, col="red")
```

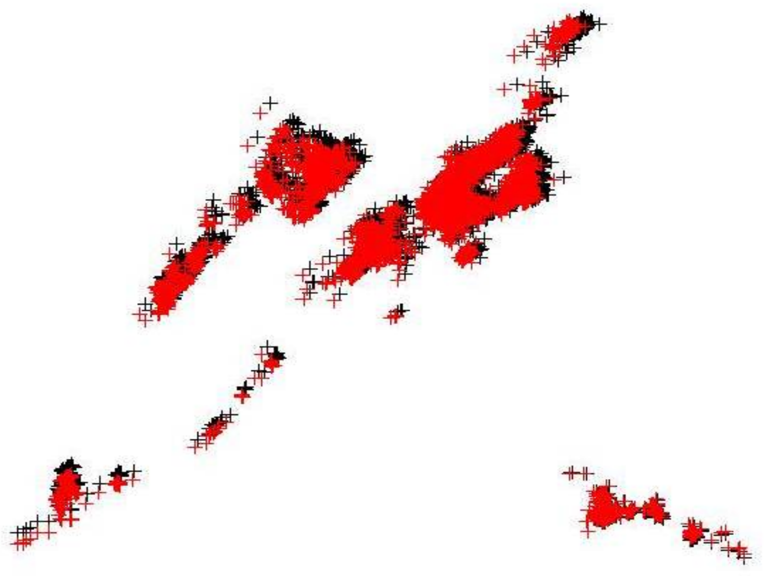


Figure 1.5: Mule deer locations projected into UTM Zone 12N (red) with original locations in WGS84 (black)

1.5 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 File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(chron)
   library(rgdal)
   library(RAtmosphere)
   ```

4. Now open the script "TimeLagScript.R" and run code directly from the script
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$time,format=c(dates="m/d/y",times="h:m:s"))
   # calc diff in minutes
   timediff <- diff(temp$date_time)*24*60
   # remove first entry without any difference
   ```

```
temp <- temp[-1,]
# assign timediff column
temp$timediff <- as.numeric(timediff)
```

The above code will result in a dataset that includes "timediff" that is the time between successive GPS points

```
Location      date_time      difference
1
2      (07/13/05 10:00:00)      7
3      (07/13/05 10:30:00)      30
4      (07/13/05 11:00:00)      30
5      (07/13/05 11:30:00)      30
6      (07/13/05 12:00:00)      30
7      (07/13/05 12:30:00)      30
8      (07/13/05 13:00:00)      30
9      (07/13/05 13:30:00)      30
10     (07/13/05 14:00:00)      30
11     (07/13/05 15:00:00)      60
12     (07/13/05 15:30:00)      30
13     (07/13/05 16:00:00)      30
14     (07/13/05 16:31:00)      31
15     (07/13/05 17:01:00)      30
16     (07/13/05 17:30:00)      29
```

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

```
write.table(temp,"TimeDiffdata.txt", row.names=TRUE, sep=" ",
col.names=TRUE, quote=TRUE, na = "NA")
```

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 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 your 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.crs)

#ll.crs <- CRS("+proj=longlat +ellps=WGS84")
#datall <-spTransform(utm.spdf, CRS=ll.crs)
#str(datall)
#temp <- as.data.frame(datall)
#str(temp)
```

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

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

  ##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.

  ## 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),sep="-")
str(temp)

#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")
str(dat1)

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)
str(dat1)
dat1[1:50,]
```

1.6 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 File - Change dir...
3. First we need to load the packages needed for the exercise

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

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

```
soils<-readOGR(dsn=".", layer="Soil_Properties")
soils@proj4string ###get projection
plot(soils)
names(soils) ###get attribute data

#Rename ArcMap 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 #= 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

```
##Select the areas that Percent Clay polygons are over 30%
plot(soils, col=grey(1-soils$Clay > 30))
plot(soils)
high.clay<- soils[soils$Clay>30,]
plot(high.clay, border="red", add=TRUE)

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

##Select 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 tested for CWD. Note: Locations have been offset or altered so do not reflect actual locations of samples

```
mule <-read.csv("MDclip.csv", header=T)
str(mule)
coords<-data.frame(x = mule$x, y = mule$y)
crs<-"+proj=utm +zone=13 +datum=WGS84 +no_defs +towgs84=0,0,0"
coords

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")
samples@proj4string

#Plot them to see if it worked or to create output figures
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

```
#Match points with soil polygons they occur in
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

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

10. Convert the counts to proportions:

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

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

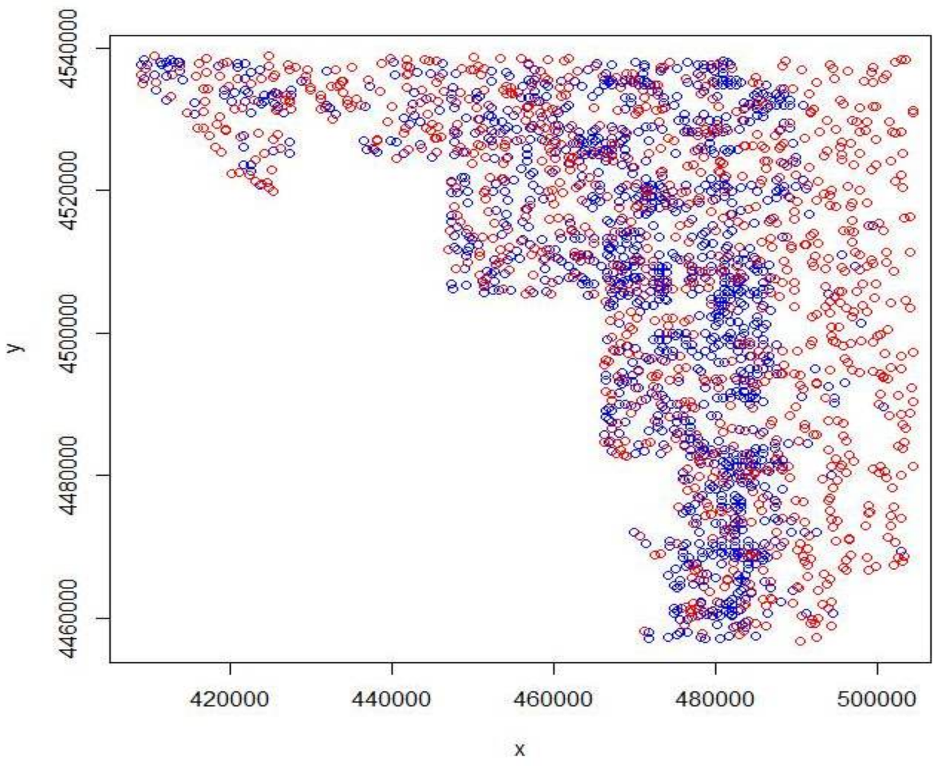


Figure 1.6: Overlay of mule deer locations with random locations generated in R

1.7 Manipulate raster 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 File - Change dir...
3. First we need to load packages to work with raster datasets

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

4. 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
   ```

5. 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.
   ```

6. 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!
7. 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).

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

   ##Or import raster as an Ascii files (factor) using "adehabitat" package
   ##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
str(asp)

NOTE: "Levelfile" refers to a text file created from exporting the raster
value attribute table from ArcMap by opening in table of contents and
exporting as a text file

#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
str(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
```

8. R won't recognize double digit veg categories with this method so reclassify in ArcMap then import raster as an Ascii files (factor) using:

```
file3 <-  paste(polyascii2.asc")
levelfile3 <- paste("TableCode.txt")
asp3 <- import.asc(file3, lev = levelfile3, type = "factor")
image(asp3)
asp3
str(asp3)

NOTE: "Levelfile" refers to a text file created from exporting the raster
value attribute table from ArcMap by opening in table of contents and
exporting as a text file and then edited to look like this:
"VALUE","COUNT","VEGCLASS"
1,464368,DEVELOPED
2,186853,FOREST
3,185059,SHRUB
4,509415,GRASS
5,341023,CROP
6,251492,WET
7,350491,NON

#Using "asp" results in the following:
Raster map of class "asc":  Cell size: 30   Number of rows: 3245
                            Number of columns: 3353  Type: factor
```

```
#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
```

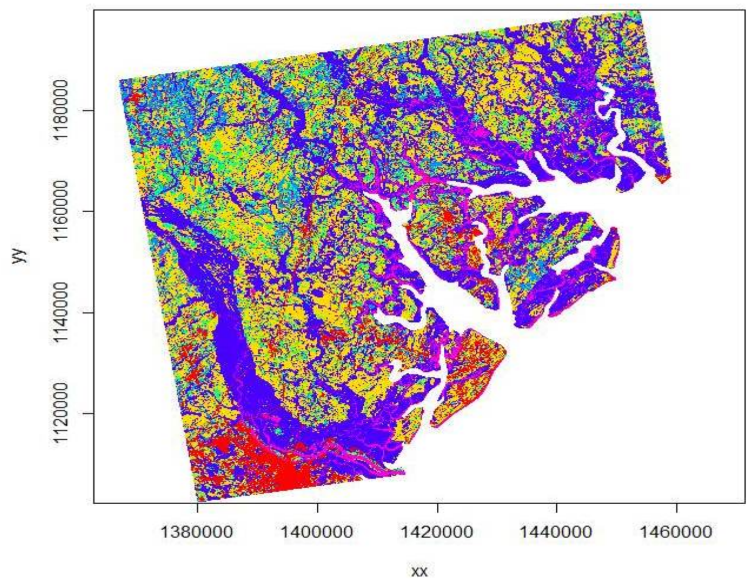


Figure 1.7: Imported raster dataset showing coastline and tributaries.

9. Or import raster as an Ascii files (numeric like a DEM) using:

```
fileElev <-  paste("demascii.asc", sep = "\\"))
    elev <- import.asc(fileElev)
    image(elev)
    plot(elev, col=terrain.colors(10))
```

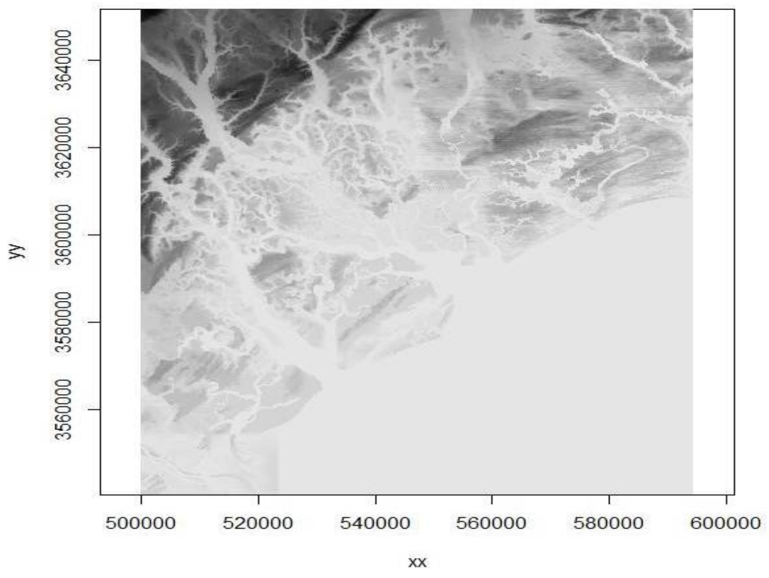


Figure 1.8: Imported Digital Elevation Model using adehabitat package showing coastline and tributaries.

10. We can also use the "rgdal" package to import an ascii grid as a Spatial Grid Data Frame

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

```
proj4string(habitat) <-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")
image(habitat)
str(habitat)
```

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

```
#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")

#Load county shapefile
county<-readOGR(dsn=".",layer="BeaufortCoAlbers")
proj4string(county)
#Now let's make county a SpatialPolygon class to simply data contained within it
polys <- as(county, "SpatialPolygons")
plot(polys,add=T,lwd=2)
polys
text(coordinates(polys), labels="Beaufort")
proj4string(polys)

#Load airport runway shapefile
run<-readOGR(dsn=".",layer="RunwayAlbers")
proj4string(run)
polys2 <- as(run, "SpatialPolygons")
plot(polys2,add=T,lwd=2)
polys2
proj4string(polys2)

#Load aircraft flight pattern shapefile
path<-readOGR(dsn=".",layer="FlightImage")
proj4string(path)
polys3 <- as(path, "SpatialLines")
plot(polys3,add=T,lty="32", col="blue")
polys3
proj4string(polys3)

#Load roads shapefile for Beaufort County
road<-readOGR(dsn=".",layer="CountyRoadAlbers")
proj4string(road)
polys4 <- as(road, "SpatialLines")
plot(polys4,add=T,lty="22", col="green")
polys4
proj4string(polys4)
```

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

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

13. Clip the raster within the county polygon for a zoomed in view then plot

```
#Clip using the raster imported with "raster" package
clip <- crop(r, 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="green")
```

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

```
#Load vegetation layer
veg <-raster("polydouble.txt")
plot(veg)
veg

# Reclassify the values into 7 groups with all values between 0 and 20 equal
# 1, 21 to 40 equal 2, 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)
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,
     100, 150, NA)
rclmat1 <- matrix(m1, ncol=3, byrow=TRUE)
rc1 <- reclassify(veg, rclmat1)
plot(rc1)
rc1
```

15. 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)
str(bv49)#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)
crs<-"+proj=utm +zone=17N +ellps=WGS84"
coords

bvspdf <- SpatialPointsDataFrame(coords= coords, data = bv49,
     proj4string = CRS(crs))
str(bvspdf)
bvspdf[1:5,]
points(bvspdf, col="red")
```

```
#NOTE: Locations must be assigned to the UTM coordinate system prior to projection
#to Albers so won't overlay on veg layer at this point because veg is in Albers
bv49Albers <-spTransform(bvspdf, 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"))
class(bv49Albers)
proj4string(bv49Albers)
bv49Albers[1:5,]
points(bv49Albers, col="red")
#Determine which of those points lie within a cell that contains data by using
     the extract function. The extract function will extract covariate information
     from the raster at a particular point.
veg.survey<-extract(veg, bv49Albers)
veg.survey
veg.survey<-subset(bv49Albers,!is.na(veg.survey))
plot(veg.survey, col="black", add=T)
```

16. 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))
points(Sample.points, bg="red", cex=.5,col="red")

#Now create some random points using the minimum and maximum coordinates of
   the raster to 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 coordinates for random points
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)
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"
coords2
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. The extract function will extract covariate information
     from the raster at a particular point.
veg.sample<-extract(veg, sample.pts)
head(veg.sample)
veg.sample<-subset(sample.pts,!is.na(veg.sample))
head(veg.sample)
points(veg.sample, col="green")
```

17. 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.

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

#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)
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"
coords3
points(coords3, bg="red", cex=.5,col="blue")

#Determine which of those points lie within a cell that contains data by using
 the extract function.
str(coords3)#Note number of locations
clip.sample<-extract(clip, coords3)
clip.sample
clip.sample<-subset(coords3,!is.na(clip.sample))
str(clip.sample)#Again note number of locations after subset function
points(clip.sample, cex=.5, col="red")
points(bv49Albers)
```

1.8 Creating a hexagonal polygon grid over a study area

Numerous research objectives require the need for creating a grid system of equal size over a study site such as studies on resource selection and spatial epidemiology. Grid systems overlayed on a study site typically are shapefiles that can either be created and imported from GIS software or created in R. Considering we have already learned how to import shapefiles, we will explore how to create grids in R for this section. Grids can be of any size and shape but should be based on something biologically meaningful to the animal or system you are studying. For example, spatial epidemiology studies often base the size of the grid cell on the dailiy movement distance or home range of the study animal if that data is known (Farnsworth et al. 2006, Rees et al. 2011).

1. Exercise 1.8 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(sp)
   library(lattice)
   ```

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

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

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

```
study.counties<-readOGR(dsn=".",layer="MDcounties")
str(study.counties) #Identifies 5 slots for the shapefile (data, polygons, order,
                                bbox, and proj4string)
class(study.counties)#Shows class and package used
proj4string(study.counties) #Shows projection information
plot(study.counties)#plots study sites on map
study.counties@data$StateCO #Displays labels for counties in plot
#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)
#NOTE: This can be any column or label within your shapefile

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

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

6. 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)
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
coords
deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys,
     proj4string = CRS(crs))
deer.spdf[1:5,]
class(deer.spdf)
proj4string(deer.spdf)
points(deer.spdf,col="red")
```

7. 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

```
row.names(study.counties)<-as.character(study.counties$StateCO)
str(study.counties@polygons[3], max.level=3)

#Now add labels of State and County to Map
text(coordinates(study.counties), labels=sapply(slot(study.counties, "polygons"),
     function(i) slot(i, "ID")), cex=0.3)
```

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

```
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)
```

9. 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, add=T)
```

10. Let's 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")
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)
```

11. 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) #By county locations occurs in
new = cbind(deer.spdf@data, o)
head(o)
head(deer.spdf)
head(new)

#Used to rename labels by hexagonal grid ID only for visualization only!
row.names(HexPols2)<-as.character(HexPols2@plotOrder)
```

12. As an aside, let's explore how to assign the area a location occurs in by intersecting points within the polygon shapefile.

```
o2 = over(deer.spdf,HexPols2)
o2
new2 = cbind(deer.spdf@data,o2)
head(new2)
new2
deer.spdf@data[1:10,]
HexPols2

#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=0.3)
```

13. 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.

```
#First we need to create the polygon within the extent of our mule deer locations
proj4string(deer.spdf)
bbox(deer.spdf@coords)
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(proj4string(MDclip)))
plot(SP)
proj4string(SP)
points(deer.spdf,col="red")
```

14. Now let's 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 from ArcMap
veg <-raster("extentnlcd2.txt")
plot(veg)
class(veg)

#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)
proj4string(deer.spdf)
proj4string(SP)
proj4string(veg)
```

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

```
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")
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
points(deer.albers, col="red")
class(deer.albers)
proj4string(deer.albers)
head(deer.spdf)
head(deer.albers)
```

```
#Now determine the new coordinates and create a new polygon to clip the raster.
bbox(deer.albers)
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)))

#Check to see all our layers are now in Albers projection
plot(AlbersSP)
proj4string(veg)
proj4string(deer.albers)
proj4string(AlbersSP)

plot(veg)
points(deer.albers, col="red")

#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")
```

1.9 Creating a square polygon grid over a study area

Recently researchers have been creating grids for analyses of various shapes. We already explored how to create a hexagonal grid but now we will learn how to create a square grid within the extent of a pre-defined study area. This method requires a few more steps but square polygon grids and the resulting adjacency matrix are common in disease epidemiology and will be used in future exercises.

1. Exercise 1.9 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(sp)
   library(rgdal)
   library(raster)
   library(adehabitatMA)
   ```

4. Now open the script "GridSystem2Script.R" and run code directly from the script
5. 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)
   str(muleys)

   #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)
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
head(coords)
plot(coords)

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

#Project the deer.spdf to Albers as in previous exercise
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")
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
proj4string(deer.albers)
bbox(deer.albers)
#       min      max
#x -1145027 -1106865
#y  1695607  1729463
```

6. 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 = -1145027, to = -1106865, by = 1500)
y <- seq(from = 1695607, to = 1729463, by = 1500)
```

7. 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)
str(xy)

#Identifiy projection before creating Spatial Points Data Frame
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"
grid.pts<-SpatialPointsDataFrame(coords= xy, data=xy, proj4string = CRS(crs2))
plot(grid.pts)
gridded(grid.pts)
class(grid.pts)

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

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

```
grid <- as(grid.pts, "SpatialPolygons")
plot(grid)
str(grid)
class(grid)
summary(grid)
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=0.3)
points(deer.albers, col="red")
str(gridspdf@polygons)
```

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

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

10. 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.

```
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 BEFORE CODE BELOW!!

o2 = over(deer.albers,gridspdf)
head(o2)
new2 = cbind(deer.albers@data, o2)#No more NAs causing errors!
new2[1:15,]
```

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

```
veg <-raster("ExtentNLCD2.txt")
plot(veg)
class(veg)
```

12. 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)#shows raster cell size

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

```
#Calculate the size of each cell in your square polygon grid
ii <- calcperimeter(gridspdf)#requires adehabitatMA package
as.data.frame(ii[1:5,])#Identifies size of only the first 5 grid cells
```

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

    ```
    table = extract(bbclip,gridspdf)
    ```

14. 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.

    ```
    table = extract(bbclip,gridspdf)
    str(table)

    area = extract(bbclip,gridspdf)
    combine=lapply(area,table)
    combine

    combine[[1]]#Shows vegetation categories and numbers of cells in grid #1
      21   22   31   42   52   82   90
      38    7   23  392 1883   11  146

     combine[[27]]
      21   42   52   81   82
     101   69  279    5 2046
    ```

1.10 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 in. Let's 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 File - Change dir...

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

   ```
   library(sp)
   library(lattice)
   library(rgdal)
   library(rgeos)
   library(raster)
   ```

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

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

```
summary(muleys$id)

#Let's subset data so there are fewer locations to work with
muley8 <- subset(muleys, id=="D8")
str(muley8)
summary <- table(muley8$UTM_Zone,muley8$id)
summary(muley8$id)
muley8$id <- factor(muley8$id)

#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)
str(newmuley8)
muley8 <- newmuley8

#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = muley8$Long, y = muley8$Lat)
crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
head(coords)

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

#Again let's project the deer.spdf to Albers
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")
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
class(deer.albers)
proj4string(deer.albers)
head(deer.spdf)
head(deer.albers)
```

5. Clip around locations 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)
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")
```

6. 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.

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

7. 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.

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

8. So let's 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.

   ```
   bbox(deer.spdf)
   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)))

   #Clip the vegetation raster within the boundaries of the new "AlbersSP."
   bbclip <- crop(veg, AlbersSP)

   #Plot the clipped raster, bounding polygon, and mule deer 8 locations
   plot(bbclip)
   plot(AlbersSP,lwd=2,add=T)
   points(deer.albers, col="red")
   ```

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

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

   #Cell size of raster layer
   res(bbclip)
    30^2 #30 x 30 m resolution of the raster
   [1] 900
   > 900*37 #Times the number of cells in category 21 (i.e., developed habitat)
   [1] 33300
   > (900*37)/1000000 #Divide to convert square meters to square kilometers
   0.0333 km2 #habitat 21
   ```

10. 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 (Fig. 1.9a,b).

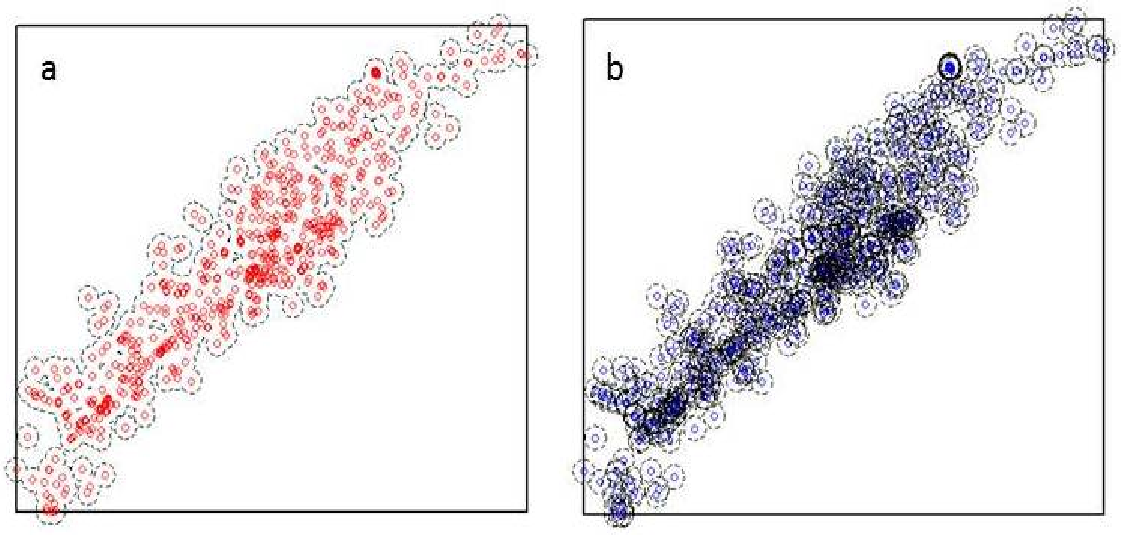


Figure 1.9: Buffers around mule deer locations using the a) byid=FALSE (default) and b) byid=TRUE for the gBuffer function using package rgeos.

```
settbuff=gBuffer(deer.albers, width=100, byid=TRUE)
plot(bbclip)
points(deer.albers, col="blue")
plot(settbuff, add=TRUE, lty=2)

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

Chapter 2

Climate Data Interpolation

Contents

The purpose of this chapter is to bring in climate data from weather stations or aquatic monitoring stations that are often in a text files or from spreadsheet software. These files often may be numerous separate files that you only want to summarize by station, annually, monthly, or some other time period. We are going to show you how to bring these files into R, clean up data, and create your own GIS layer for use in modeling efforts. The example used is from a project that attempted to predict snowshoe hare (*Lepus americanus*) presence or absence in Pennsylvania and determination of how habitats occupied may change due to changes in climate and temperature.

We obtained shapefiles from a variety of sources (see Section 2.1 for details) and weather station data from the National Oceanic and Atmospheric Administration's National Climatic Data Center and cleaned up data outside of R (see Section 2.2 for details). Data were collected from 102 weather stations in and around Pennsylvania in order to determine mean snow depths (SNWD), mean maximum temperatures (TMAX) and mean minimum temperatures (TMIN) for the month of January from 1995 to 2005. Data were used only from stations with records for at least 10 of the 11 Januaries covered by the time range. For each weather station, records for each of the three climate variables were included only if the data covered at least 95 percent of total January days. These criteria resulted in data from 66 stations for snow depth, 69 stations for maximum temperature, and 68 stations for minimum temperature. This data was edited outside of R and the resulting text files were then combined in R (Section 2.3). Alternatively, we can obtain data directly from NOAA (Section 2.4) and clean it up in R prior to moving forward thus eliminating the need for Section 2.3.

We entered spatial coordinates for the weather stations along with climate data that met the selection criteria into program R a geographic information system (Section 2.5; ArcMap 9.3). Data from these stations were then used to create maps that showed the range of mean snow depth, mean maximum temperature, and mean minimum temperature for the month of January across the state. This was accomplished with interpolation using the *kriging* method in ArcMap. The interpolations included data from a total of 46 weather stations outside Pennsylvania in order to avoid errors associated with boundary issues. The interpolated maps were then used to assign the appropriate climate data for each location sampled for presence of snowshoe hare in 2004.

These data were then used to select a model for predicting occupancy probability of snowshoe hare. Tested models examined occupancy as a function of habitat type and one of four additional correlated variables (elevation, minimum Jan temp, max Jan temp, and Jan snow depth). The model including mean minimum temp (TMIN) was found to be the best one, and the co-efficients for that model, which varied by habitat type, were included as a shapefile (county param) to calculate occupancy probability based on habitat type and TMIN across the range of snowshoe hare counties (Figure 4).

2.1 Incorporating background spatial layers

There is no formal exercise here so no R script. This is just some background information to assist with the remaining exercises in this section

1. PA_Counties and Snowshoe_Counties: Basic PA shapefile with county boundaries and a clip of only the counties with hare harvest reported.
2. Data_TMIN, Data_TMAX, and Data_SNWD: These are the files that result from assigning interpolation values to points. Attribute table for each will have TMIN, TMAX or SNWD values, which are then added to the study's database.
3. Weather_Stations_All: This shapefile is created from the data produced in R, namely the mean January values for TMIN, TMAX, and SNWD for each station from Jan. 1995 to Jan. 2005.
4. Weather_Stations_TMIN, Weather_Stations_TMAX, Weather_Stations_SNWD: These are the files that are used for interpolating weather variables across the state. The Weather_Stations_All shapefile is split into three shapefiles, one each for TMIN, TMAX and SNWD. Not every weather station has data for each of the three weather variables, and 0 values must be taken out in order for the interpolation's to be correct.
5. 2004_Study_Locs: Shapefile contains point data for all locations in 2004 snowshoe hare study. These points are used to assign interpolation values for new shapefiles (Data_TMIN, Data_TMAX, Data_SNWD).
6. pa_krig_tmin, pa_krig_tmax, pa_krig_snwd: These are the interpolation shapefiles, based on Weather_Stations_TMIN, Weather_Stations_TMAX, Weather_Stations_SNWD.
7. counties_hab: Contains the four forest types for the snowshoe hare counties: Coniferous, Mixed, Deciduous, Transitional.
8. county_param: Assigns values to each forest type based on Duane's model. These values will be used to determine occupancy probability when the model is plugged into map algebra.

2.2 Accessing climate data

There is no formal exercise here so no R script. All files downloaded in this section are available for Exercise 2.3 so only accessing climate data for practice or to download data specific to your study area and objectives

1. Let's find some weather data and explore the format available by copying the NOAA data link into a web browser `http://www.ncdc.noaa.gov/cdo-web/` or just select the link here: NOAA Data

2. Select the Mapping Tool Application box then All Maps tab, then choose "GHCN-Daily" all in a separate window.

3. Use polygon to select a defined area, check "GHCND" under "GHCN Daily", and scroll/zoom with the Polygon tool that allows for selecting areas outside PA boundaries for inclusion. This is done in order to create a better interpolation for SnowDepth, TMIN TMAX

4. Selection will bring up all weather stations in area. Look under "Period of Record to determine whether it has data for Jan. 1 1995 Jan. 31 2005 If 1 year out of the 11 is missing, save it. Greater than 1, skip it. Select all that match dates and click "Get Selected Data"

5. On next page, make "Output date range" as 01 Jan 1995 to 01 Jan 2005 and select the "Custom GHCN-Daily CSV" box and select continue.
   Then select:
   SnowDepth (under Precipitation)
   TMIN (under Air Temperature)
   TMAX (under Air Temperature)
   Be sure to include Include Station Name and Geographic Location (Lat Long Elev)

6. Wait for email with link then click and Save file as .csv

   Below are specific instructions when retrieving data for a specific station with known GHCND code:

   1. Go to NOAA Climate Data Online NOAA Data
   2. Select data search Enter GHCND code (but don't include "GHCND:" portion of code in search term) Select "Daily GHCND" as data set
   3. If dates match desired range (see 4), select station and hit continue If not, remain on page and enter new GHCND code in search box 4. Start date for this project: Jan. 1, 1995 End date: Jan. 31, 2005 5. Select "Custom GHCN Daily CSV" format option, hit continue
   6. Under Precipitation, select PRCP, SNOW, SNWD
   7. If have Air Temperature, select TMAX, TMIN
   8. Select Station Name in "Additional Output Options" and hit continue
   9. You will receive two emails - one confirming order and one with link to data
   10. Click link, save data as .csv file with Station Name as file name
   11. Click on Data Search, repeat process until have data for all weather stations

2.3 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 File - Change dir...

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

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

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 meets 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.

```
# 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 weather 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","Station")
write.csv(m1,paste(".","\\output.csv",sep=""))

#Removes quotes from output table below
m1 <-noquote(m1)

#Results of code that summarizes the 6 weather stations
m1
    Elevation  Lat       Long    SNWD    TMAX   TMIN   GHCN        Station
[1,] 520       42.249  -77.758  15.558  31.969 13.299 USC00300085 ALFRED
[2,] 457.2     42.1    -78.749   7.698  30.175 14.466 USC00300093 ALLEGANYSP
[3,] 452       42.303  -78.018   5.293  30.872 11.687 USC00300183 ANGELICA
[4,] 341.4     42.348  -77.347   4.122  32.247 13.549 USC00300448 BATH
[5,] 80.2      40.833  -75.083   NaN    36.457 19.621 USC00280734 BELVIDERE
```

2.4 Using data in R

1. Exercise 2.4 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(adehabitatHR)
   ```

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

4. Now open the script "Weather-Station-Code_Snowfall.R" and run code directly from the script

5. This code is designed to process data downloaded from NOAA Data This version looks at weather stations that provide snowfall data in and around Pennyslvania. 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.

```
### remove other objects from the working session ###
rm(list=ls())
```

6. 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"))

head(WS)
```

7. 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)

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

#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]),]
head(PulledCoords)
dim(PulledCoords)
```

```
CoordChart <- ddply(PulledCoords, c('STATION'), function(x) c(Lat=x$LATITUDE,
     Long=x$LONGITUDE))
head(CoordChart)

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

#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))
head(YearlySnow)
tail(YearlySnow)
dim(YearlySnow)

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

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

#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)))
head(WinterYears)
tail(WinterYears)
dim(WinterYears)

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

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

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

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

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

#Add a column to CoordChart showing whether each row matches  a STATION in
#Complete.Records.  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)
head(CoordChart)
dim(CoordChart)
dim(Complete.Records)

#Combine Complete.Records and CoordChart. Make sure each STATION matches in row
#Delete any rows that don't match. Shouldn't be any. If # of rows in Final.Values
#is less than number of rows in CoordChart, there is a problem (but note that
#number of cols does change).
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)
dim(CoordChart)

head(Final.Values)

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

8. 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)
```

9. 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")
proj4string(counties)
plot(counties)

#Project Weather Stations to match Counties
Albers.crs <-CRS("+proj=aea +lat_1=29.3 +lat_2=45.3 +lat_0=23 +lon_0=-96
     +x_0=0 +y_0=0 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
stations <- spTransform(merge, CRS=Albers.crs)

#Plot Weather Stations over counties
points(stations)

stations@data$MAS <- stations@data$MeanAnnualSnowfall
str(stations)

str(counties)

## create a grid onto which we will interpolate:
xx = spsample(counties, type="regular", cellsize=10000)
class(xx)

points(xx, bg="red", cex=.5,col="red")

#Convert to a SpatialPixels class
gridded(xx) <- TRUE
class(xx)

plot(xx)
```

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

#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:
## first get the range in data
 x.range <- as.integer(range(stations@coords[,1]))
 y.range <- as.integer(range(stations@coords[,2]))

## now expand to a grid with 500 meter spacing:
grd <- expand.grid(x=seq(from=x.range[1], to=x.range[2], by=5000),
     y=seq(from=y.range[1], to=y.range[2], by=5000) )

## convert to SpatialPixel class
 coordinates(grd) <- ~ x+y
 gridded(grd) <- TRUE

## test it out:
plot(grd, cex=0.5)
points(stations, pch=1, col='red', cex=0.7)
title("Interpolation Grid and Sample Points")

x <- krige(stations@data$MAS~1, stations, xx)
class(x)
image(x)
points(stations, pch=1, col='blue', cex=0.7)
```

2.5 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 File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(ncdf4)
   ```

4. Now open the script "NetCDF_Script.R" and run code directly from the script
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_openf("Monthly_AvgMinTemp_1995-99_MPI.nc", write=TRUE,
   ```

```
     readunlim=TRUE, verbose=TRUE)

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

#####################################
# The following illustrates how to read the data
#####################################
print(paste(tmin$name))  #in this case the 'field name' is 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 the object, n sets the
# number of entries displayed. tail returns  the last of the data entries

# Dimensions of df1 (x, y, time)
dim(df1)

# Dimensions can also be examined one at a time
dim(df1)[1] # number of x grids (36)
dim(df1)[2] # number of y grids (21)
dim(df1)[3] # number of months in file (49)
# NOTE: FILE INCLUDES MONTHS OTHER THAN JANUARY (Jans are 1,13,25,37,49)

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

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

# Create a new matrix which is monthly averages for each grid cell. Make the new
#matix 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)

# 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
}
}

# head(sum1)  ## useful for large files
sum1

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

```
############################################################
# 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)
length(y)
dim(sum1)

## 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 3

Movement Methods

Contents

Figures

Movement methods can serve a variety of purposes from determining mean daily distance moved by an animal to describing the scale at which an animal uses the landscape. Understanding movements can often shed some light on how an animal uses the landscape based on differences in turning angles and clustering of paths in each defined habitat type. Trajectories can be created from relocations and are also the precursor to several home range estimation methods we will go over later in the course.

Notes from the package AdehabitatLT manual (Calenge 2012): Two types of trajectories can be stored in objects of class ltraj: trajectories of type I correspond to trajectories where the time of relocations is not recorded. It may be because it could not be noted at the time of sampling (e.g. sampling of animals' tracks in the snow) or because it was decided that they did not want to take it into account, i.e. to study only its geometrical properties. In this case, the variable date in each burst of the object contains a vector of integer giving the order of the relocations in the trajectory (i.e. 1, 2, 3, ...). Trajectories of type II correspond to trajectories for which the time is available for each relocation. It is stored as a vector of class POSIXct in the column date of each burst of relocations. The type of trajectory should be defined when the object of class ltraj is defined, with the argument type II.

Concerning trajectories of type II, in theory, it is expected that the time lag between two relocations is constant in all the bursts and all the ids of one object of class ltraj (i.e., do not mix animals located every 10 minutes and animals located every day in the same object). Indeed, some of the descriptive parameters of the trajectory do not have any sense when the time lag varies. For example, the distribution of relative angles (angles between successive moves) depends on a given time scale; the angle between two during 10-min moves of a whitestork does not have the same biological meaning as the angle between two 1-day moves. If the time lag varies, the underlying process varies too. For this reason, most functions of adehabitatLT have been developed for "regular" trajectories, i.e. trajectories with a constant time lag (see *help(sett0)*).

3.1 Importing datasets from a web source

Movebank.org is a new storage warehouse 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 File - Change dir...
3. First we need to load the packages needed for the exercise

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

4. Now open the script "MBvultureCode.R" and run code directly from the script
5. Let's aslo go to the Movebank home page and explore what it has to offer
6. Login to Movebank

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

7. Access a stored dataset in Movebank of vulture data

   ```
   getMovebankAnimals(study="Turkey Vulture South Carolina USA",login=login)

   #Give the dataset a name while identifying each bird by ID in dataset
   turkey <- getMovebankData(study="Turkey Vulture South Carolina USA",
   animalName=c("Bird51","Bird52","Bird54","Bird55a","Bird56","Bird59a","Bird60a"),
   login=login, moveObject=TRUE)

   #Check to see the number of locations for each bird
   n.locs(turkey)
   # Bird51  Bird52  Bird54 Bird55a  Bird56 Bird59a Bird60a
   #   9655   11456    2378    2228    5876    8311    8594
   ```

3.2 Movement trajectories

We will start with simply creating trajectories between successive locations. As stated above, 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 File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(adehabitatLT)
   library(chron)
   library(spatstat)#for "duplicate" function
   ```

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

   ```
   #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)
   str(muleys)
   ```

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

   ```
   summary(duplicated(muleys))

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

6. 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$GPSFixTime,format="%Y.%m.%d %H:%M:%S"))
   head(da)
   muleys$da <- da #attach "da" to muleys dataset
   str(muleys)

   #Create time lag between successive locations to censor data if needed.
   timediff <- diff(muleys$da)
   muleys <-muleys[-1,]
   muleys$timediff <-as.numeric(abs(timediff))
   str(muleys)#check to see timediff column was added to muleys

   #Look at number of locations by animal ID
   summary(muleys$id)

   #Remove outlier locations or known outliers collected too far apart in time
   newmuleys <-subset(muleys, muleys$Long > -110.50 & muleys$Lat > 37.3 &
      muleys$Long < -107)
   muleys <- newmuleys
   str(muleys)
   ```

7. Let's 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)
#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(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)#visualize data
str(merge)
```

8. 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)
plot(ltraj)
plot(ltraj[1])
head(ltraj[1])#Describes the trajectory for the first deer

#> head(ltraj[1])#Describes the trajectory
*********** List of class ltraj ***********
#Type of the traject: Type II (time recorded)
#Irregular traject. Variable time lag between two locs
#
#Characteristics of the bursts:
#   id burst nb.reloc NAs          date.begin            date.end
#1 D12   D12      100   0 2011-10-12 06:00:52 2011-10-24 21:00:48
#
 #infolocs provided. The following variables are available:
#[1] "pkey"

plot(ltraj[2])
plot(ltraj[3])
plot(ltraj[4])
plot(ltraj[5])
plot(ltraj[6])
plot(ltraj[7])
```

9. Let's 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*.

```
hist(ltraj[1], "dt", freq = TRUE)
windows()#opens a new window to show both figures
hist(ltraj[1], "dist", freq = TRUE)

windows()
hist(ltraj[2], "dt", freq = TRUE)
windows()
hist(ltraj[2], "dist", freq = TRUE)
windows()

hist(ltraj[3], "dt", freq = TRUE)
windows()
hist(ltraj[3], "dist", freq = TRUE)
windows()

hist(ltraj[4], "dt", freq = TRUE)
windows()
hist(ltraj[4], "dist", freq = TRUE)
windows()

hist(ltraj[5], "dt", freq = TRUE)
windows()
hist(ltraj[5], "dist", freq = TRUE)
windows()

hist(ltraj[6], "dt", freq = TRUE)
windows()
hist(ltraj[6], "dist", freq = TRUE)
windows()

hist(ltraj[7], "dt", freq = TRUE)
windows()
hist(ltraj[7], "dist", freq = TRUE)
```

3.3 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 File - Change dir...
3. First we need to load the packages needed for the exercise

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

```
library(class)
library(Rcmdr)
```

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

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

5. Code to subset dataset for an individual animal

```
muley15 <- subset(muleys, id=="D15")
str(muley15)
summary <- table(muley15$UTM_Zone,muley15$id)
summary
muley15$id <- factor(muley15$id)

#Sort data to address error in code and then look at first 10 records
of data to confirm

muley15 <- muley15[order(muley15$GPSFixTime),]
muley15[1:10,]#code displays the first 20 records
```

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

```
######################################################
## 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"))
head(da)

#Attach da to muley15
muley15$da <- da

#Creates a column of time difference between each location
timediff <- diff(muley15$da)
muley15 <-muley15[-1,]
muley15$timediff <-as.numeric(abs(timediff))
str(muley15)

#Clean up muley15 for outliers
newmuleys <-subset(muley15, muley15$X > 599000 & muley15$X < 705000 &
     muley15$Y > 4167000 & muley15$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)
str(merge)
```

8. Creation of an object of class "ltraj" for muley15 dataset

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

#CAN BE USED TO REMOVE TIME FROM DATE IN GPSFIXTIME COLUMN if needed
#Date <- as.character(muleys$GPSFixTime)
#Date <- as.POSIXct(strptime(muleys$GPSFixTime,"%Y.%m.%d"))
#muleys$Date <- Date
#str(muleys)
```

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 trajectory per day. The trajectory should begin
## at 22H00
## The following function returns TRUE if the date is comprised between
## 06H00 and 23H00 (i.e. results in 3 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)

#Notice that the above code will remove 345 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
# 345 relocations have been deleted

deer

#Shows results of cutting the traj into individual bursts
```

```
#NOTE the "Irregular traject" line below because we will revisit this later!
#*********** List of class ltraj ***********

#Type of the traject: Type II (time recorded)
#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
#7   D15 D15.014        7   0 2011-10-18 00:00:34 2011-10-18 18:00:48
#8   D15 D15.016        7   0 2011-10-19 00:00:36 2011-10-19 18:00:40
#9   D15 D15.018        7   0 2011-10-20 00:00:53 2011-10-20 18:00:40
#10  D15 D15.020        7   0 2011-10-21 00:00:39 2011-10-21 18:00:37
```

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

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

str(dfdeer)
'data.frame':   2243 obs. of  13 variables:
 $ x        : num  677932 679037 679429 679750 679453 ...
 $ y        : num  4189551 4189493 4189406 4189053 4188461 ...
 $ date     : POSIXct, format: "2011-10-12 03:00:52" "2011-10-12 06:00:52"
 $ dx       : num  1105 392 321 -297 163 ...
 $ dy       : num  -58 -87 -353 -592 -89 NA -189 756 395 95 ...
 $ dist     : num  1107 402 477 662 186 ...
 $ dt       : num  10800 10786 10796 10808 10810 ...
 $ R2n      : num  0 1224389 2262034 3553128 3501541 ...
 $ abs.angle: num  -0.0524 -0.2184 -0.8328 -2.0358 -0.4998 ...
 $ rel.angle: num  NA -0.166 -0.614 -1.203 1.536 ...
 $ id       : Factor w/ 1 level "D15": 1 1 1 1 1 1 1 1 1 1 ...
 $ burst    : Factor w/ 325 levels "D15.001","D15.003",..: 1 1 1 1 1 1 2  ...
 $ pkey     : Factor w/ 2588 levels "D15.2011-10-12 03:00:52",..: 1

#Code to get mean distance moved for each burst
summary <- numSummary(dfdeer[,"dist"],groups=dfdeer$burst, statistics=
     c("mean","sd"))
summary

#Convert matrix from data.frame to a matrix to export as a .csv file
mean <- as.matrix(summary$table)

#Write.table gives csv output of Summary.  Be sure to specify the directory
```

```
     and the output files will be stored there
write.table(mean, file = "Distance.csv", sep =",", row.names = TRUE,
     col.names = TRUE, qmethod ="double")
```

3.4 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 differenciate 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 seaches for food.

The value fpt computes the FPT for each relocation and each radius, and for each animals. 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 File - Change dir...

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

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

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

5. In this example we are going to look at mule deer in southwestern Colorado. We can eliminate poor locations in the original dataset or code can be used after examining the data.

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

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

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

```
############################################################################
## Example of a trajectory of type II (time recorded) ##that must be converted
## to the format POSIX that needs to be done to get proper digits of date for
##  use with the adehabitatLT package
##############################################################################
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$da)
muleys <-muleys[-1,]
muleys$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)
#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])
#Creates a spatial data frame of dt
dtsp<-data.frame(muleys[24])
#Creates a spatial data frame of Burst
busp<-data.frame(muleys[25])
#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)
str(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)

#Or we can plot trajectories for each specific animal

plot(ltraj[1])
plot(ltraj[2])
plot(ltraj[3])
plot(ltraj[4])
plot(ltraj[5])
plot(ltraj[6])
plot(ltraj[7])
```

9. Code to plot histograms of distance distribution for each deer

```
windows()
```

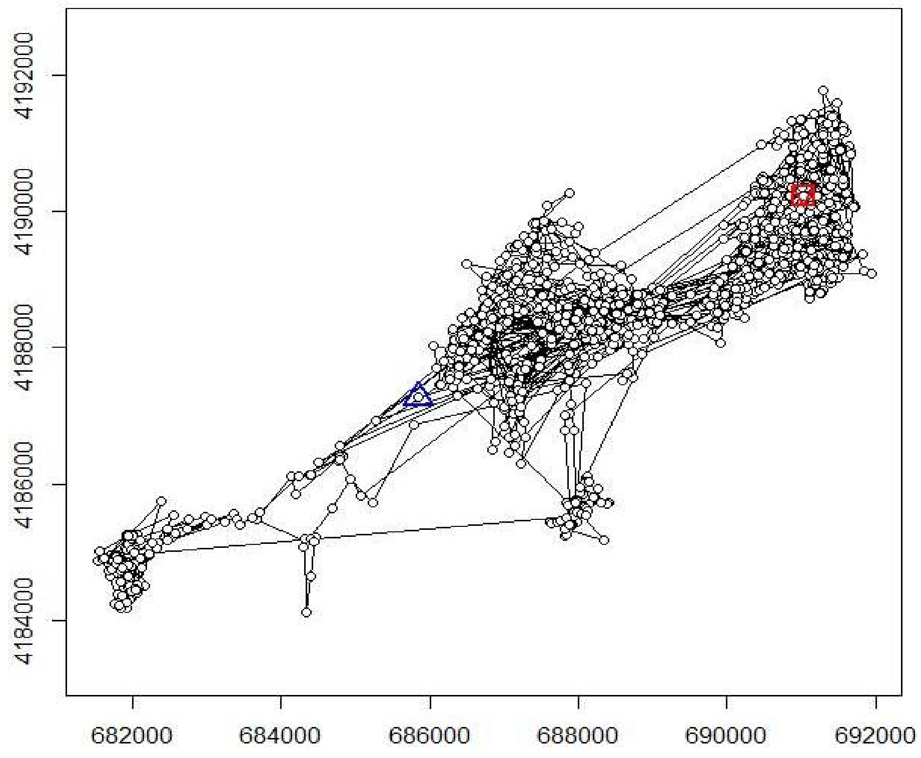


Figure 3.1: Example of a trajectory created using adehabitatLT for a mule deer in Colorado.

```
hist(ltraj[1], "dist", freq = FALSE)
windows()
hist(ltraj[2], "dist", freq = FALSE)
windows()
hist(ltraj[3], "dist", freq = FALSE)
windows()
hist(ltraj[4], "dist", freq = FALSE)
windows()
hist(ltraj[5], "dist", freq = FALSE)
windows()
hist(ltraj[6], "dist", freq = FALSE)
windows()
hist(ltraj[7], "dist", freq = FALSE)
```

10. 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")
```

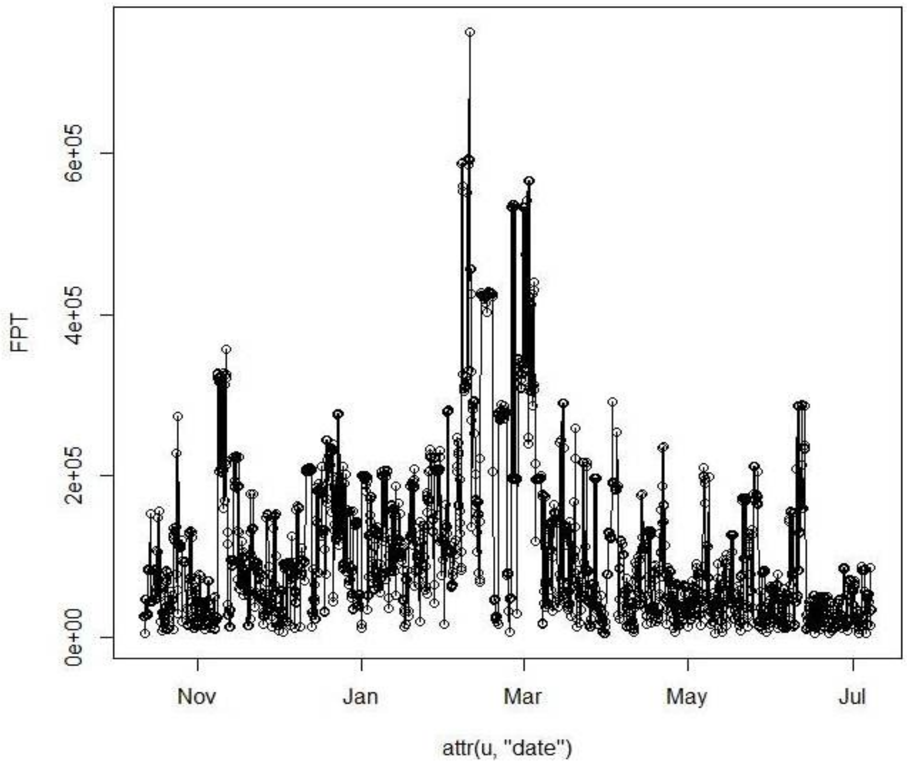


Figure 3.2: Plot of a First Passage Time for a mule deer in Colorado identifying mean FPT by month.

```
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")

is.regular(ltraj[1])
plotltr(ltraj[1], "dt")
windows()
plotltr(ltraj[1], "dist")

is.regular(ltraj[2])
plotltr(ltraj[2], "dt")
windows()
plotltr(ltraj[2], "dist")
ltraj[2]
```

11. 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)

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

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

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

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

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)
```

12. Another look at the time and distance between relocations of each animal. If GPS collars are programmed to collect 1 locations per 24 hours or a location every 4 hours, what would we expect Figure 3.3b to look like? The code can be used to produce these figures and can be used initially to inspect the data if there is concern about consistancy in location fixes or distance between each location.

```
is.regular(ltraj[3])
plotltr(ltraj[3], "dt")
plotltr(ltraj[3], "dist")
```

3.5 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 File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(adehabitatLT)
```

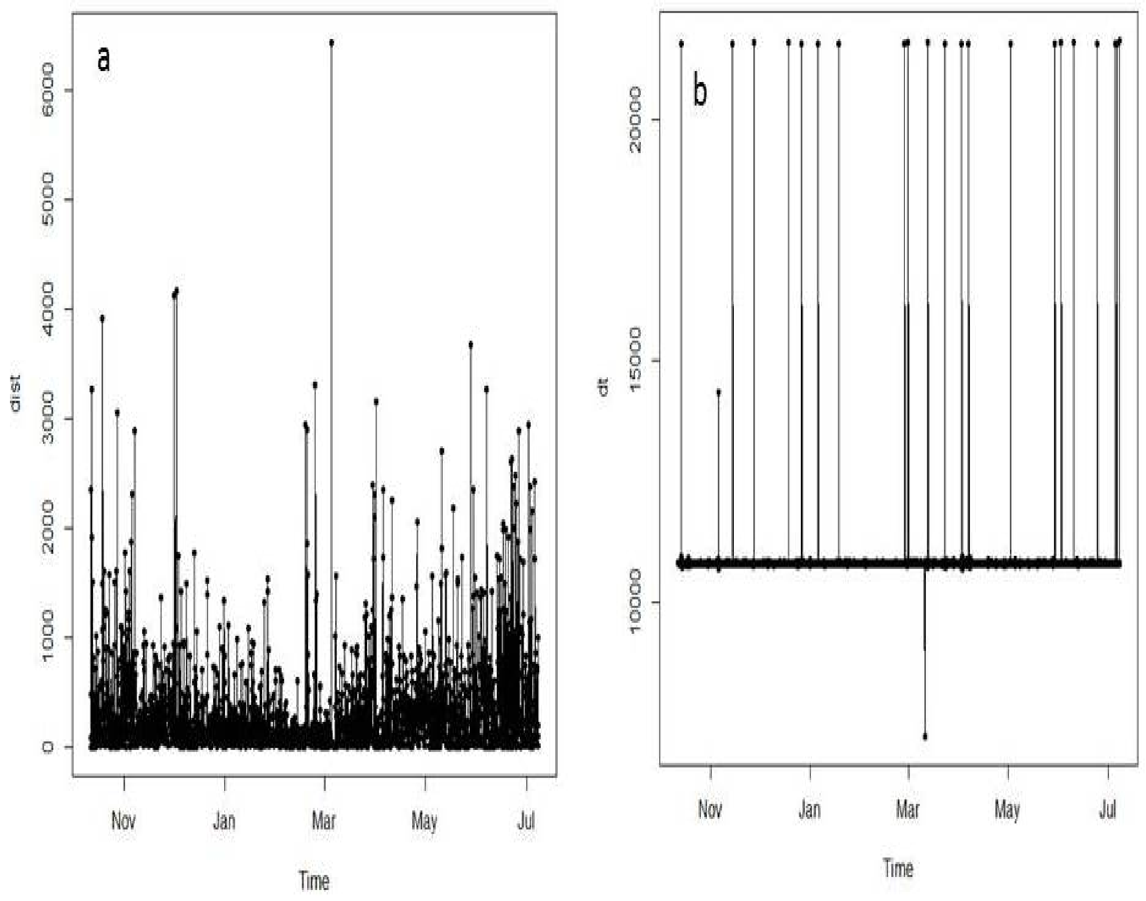


Figure 3.3: Summaries of distance and time (dt) between relocations for mule deer D16.

```
library(chron)
library(raster)
library(sp)
```

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

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

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

#Sort data to address error in code
muley15 <- muley15[order(muley15$GPSFixTime),]
muley15[1:10,]#code displays the first 10 records to look at sorting results
str(muley15)

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

```
## 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
da <- as.character(muley15$GPSFixTime)
da <- as.POSIXct(strptime(muley15$GPSFixTime,format="%Y.%m.%d %H:%M:%S"))
muley15$da <- da

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

newmuleys <-subset(muley15, muley15$X > 599000 & muley15$X < 705000 &
     muley15$Y > 4167000 & muley15$timediff < 14401)
muley15 <- newmuleys

data.xy = muley15[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(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)
str(merge)

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

#CAN BE USED TO REMOVE TIME FROM DATE IN GPSFIXTIME COLUMN
#Date <- as.character(muleys$GPSFixTime)
#Date <- as.POSIXct(strptime(muleys$GPSFixTime,"%Y.%m.%d"))
#muleys$Date <- Date
#str(muleys)

## 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 22H00
## The following function returns TRUE if the date is comprised between
## 06H00 and 23H00 (i.e. results in 3 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)
deer
## Remove the first and last burst if needed?
#deer2 <- deer[-c(1,length(deer))]

#bind the trajectories
deer3 <- bindltraj(deer)
deer3
plot(deer3)
is.regular(deer)
FALSE
plotltr(deer3, "dist")

## 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
## Set the missing values
deerset <- setNA(deer3, refda, 3, units = "hour")
## now, look at dt for the bursts:
plotltr(deerset, "dt")
## dt is nearly regular: round the date:
deerset1 <- sett0(deerset, refda, 3, units = "hour")
plotltr(deerset1, "dt")
is.regular(deerset1)
## deerset1 is now regular

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

## Show the changes in the distance between
## successive relocations with the time
windows()
plotltr(deerset1, "dist")
## Segmentation of the trajectory based on these distances
lav <- lavielle(deerset1, Lmin=2, Kmax=20)
## Choose the number of segments
chooseseg(lav)
## 20 segments seem a good choice
## Show the partition
kk <- findpath(lav, 20)
kk

##Results of code
```

```
*********** List of class ltraj ***********

#Type of the traject: Type II (time recorded)
#Regular traject. Time lag between two locs: 10800 seconds

#Characteristics of the bursts:
#    id      burst nb.reloc NAs          date.begin            date.end
#1  D15  Segment.1      199  27 2011-10-12 04:00:00 2011-11-05 22:00:00
#2  D15  Segment.2        2   0 2011-11-06 01:00:00 2011-11-06 03:00:00
#3  D15  Segment.3      455  64 2011-11-06 06:00:00 2012-01-02 00:00:00
#4  D15  Segment.4        1   0 2012-01-02 03:00:00 2012-01-02 03:00:00
#5  D15  Segment.5        2   0 2012-01-02 06:00:00 2012-01-02 09:00:00
#6  D15  Segment.6        1   0 2012-01-02 12:00:00 2012-01-02 12:00:00
#7  D15  Segment.7       64   8 2012-01-02 15:00:00 2012-01-10 12:00:00
#8  D15  Segment.8        3   1 2012-01-10 15:00:00 2012-01-10 21:00:00
#9  D15  Segment.9        2   0 2012-01-11 00:00:00 2012-01-11 03:00:00
#10 D15 Segment.10       33   4 2012-01-11 06:00:00 2012-01-15 06:00:00
#11 D15 Segment.11        5   1 2012-01-15 09:00:00 2012-01-15 21:00:00
#12 D15 Segment.12        3   0 2012-01-16 00:00:00 2012-01-16 06:00:00
#13 D15 Segment.13        2   0 2012-01-16 09:00:00 2012-01-16 12:00:00
#14 D15 Segment.14      336  46 2012-01-16 15:00:00 2012-02-27 12:00:00
#15 D15 Segment.15        4   1 2012-02-27 15:00:00 2012-02-28 00:00:00
#16 D15 Segment.16      250  35 2012-02-28 03:00:00 2012-03-30 07:00:00
#17 D15 Segment.17        1   0 2012-03-30 10:00:00 2012-03-30 10:00:00
#18 D15 Segment.18        5   2 2012-03-30 13:00:00 2012-03-31 01:00:00
#19 D15 Segment.19     1164 154 2012-03-31 04:00:00 2012-08-23 13:00:00
#20 D15 Segment.20       63   9 2012-08-23 16:00:00 2012-08-31 10:00:00

#Notice that the results show for each burst:
(1) number of relocations
(2) number of relocations removed (i.e., NA)
(3) begin and end dates
```

Now if we reduce the number of segments we get the following bursts:

```
## Segmentation of the trajectory based on these distances
lav <- lavielle(deerset1, Lmin=2, Kmax=10)
## Choose the number of segments
chooseseg(lav)
## 20 segments seem a good choice
## Show the partition
kk <- findpath(lav, 10)
kk

*********** List of class ltraj ***********

#Type of the traject: Type II (time recorded)
#Regular traject. Time lag between two locs: 10800 seconds

#Characteristics of the bursts:
#    id      burst nb.reloc NAs          date.begin            date.end
```

```
#1   D15   Segment.1       201   27  2011-10-12  04:00:00  2011-11-06  03:00:00
#2   D15   Segment.2       456   64  2011-11-06  06:00:00  2012-01-02  03:00:00
#3   D15   Segment.3         2    0  2012-01-02  06:00:00  2012-01-02  09:00:00
#4   D15   Segment.4         1    0  2012-01-02  12:00:00  2012-01-02  12:00:00
#5   D15   Segment.5       102   13  2012-01-02  15:00:00  2012-01-15  06:00:00
#6   D15   Segment.6        10    1  2012-01-15  09:00:00  2012-01-16  12:00:00
#7   D15   Segment.7       591   82  2012-01-16  15:00:00  2012-03-30  10:00:00
#8   D15   Segment.8         5    2  2012-03-30  13:00:00  2012-03-31  01:00:00
#9   D15   Segment.9      1164  154  2012-03-31  04:00:00  2012-08-23  13:00:00
#10  D15  Segment.10        63    9  2012-08-23  16:00:00  2012-08-31  10:00:00

#We can look at each segment to inspect the path traveled during the burst:
plot(kk[1])
plot(kk[2])
plot(kk[3])
plot(kk[6])
plot(kk[7])
plot(kk[9])
```

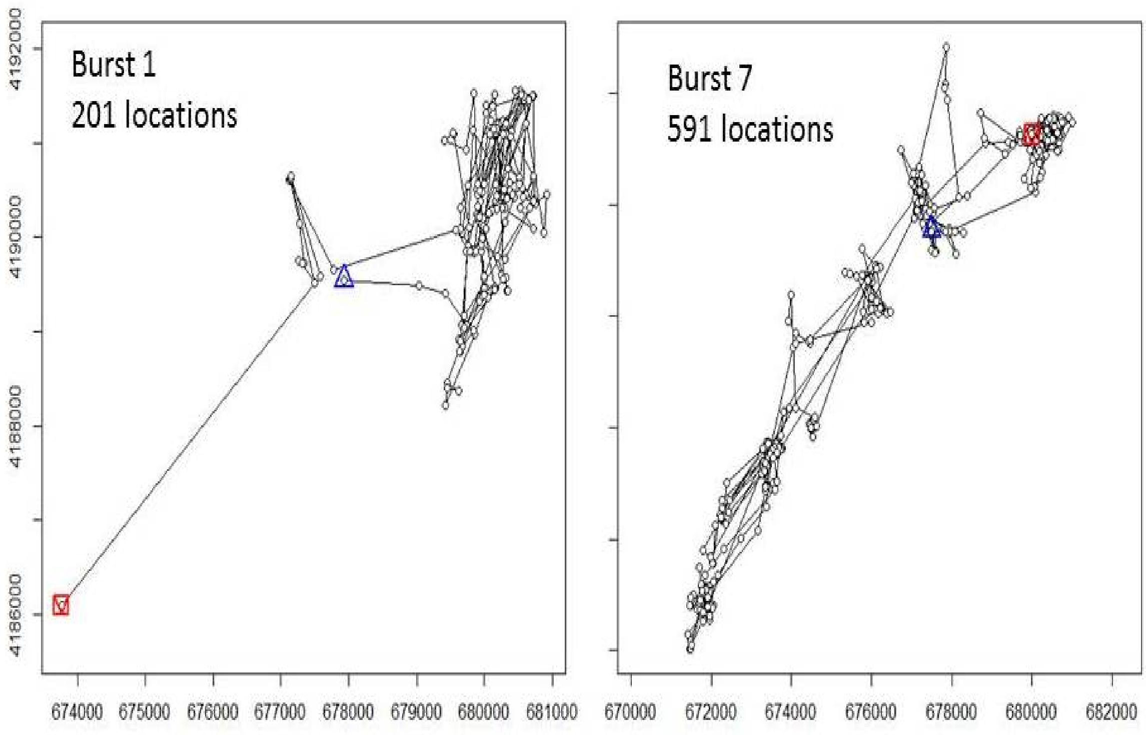


Figure 3.4: Bursts of movements for mule deer D15 after creating segments based for focal use areas.

3.6 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 ideintfy movement patters using NSD which is the straight line distance between an animals' srating 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 a 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 File - Change dir...

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

   ```
   library(trip)
   library(stringr)
   library(adehabitatHR)
   library(lattice)
   library(gmodels)
   library(spatstat)
   library(maptools)
   ```

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

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

   #Remove outlier locations, if needed, by first plotting coords below then editing
   #this line accordingly
   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)
   crs<-"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
   head(coords)
   plot(coords)

   deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys,
      proj4string = CRS(crs))
   head(deer.spdf)
   class(deer.spdf)
   proj4string(deer.spdf)
   plot(deer.spdf,col=deer.spdf$id)

   muleys$NewDate<-as.POSIXct(muleys$GPSFixTime, format="%Y.%m.%d %H:%M:%S",
       origin="1970-01-01")

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

5. 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 6 below and continue on with step 7 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)
str(muleys)
summary(muleys$YearBurst)

muleys <- subset(muleys, table(muleys$YearBurst)[muleys$YearBurst] > 100)
muleys$YearBurst <- factor(muleys$YearBurst)
```

6. Alternatively, we could assign a 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 use skip step 5 above and run step 6 instead but do not run both.

```
range(muleys$NewDate)
muleys$Year <- NULL
muleys$Year[muleys$NewDate > "2011-09-30 00:30:00" & muleys$NewDate
   < "2012-04-01 23:00:00"] <- 2011
muleys$Year[muleys$NewDate > "2012-03-31 00:30:00" & muleys$NewDate
   < "2012-10-01 23:00:00"] <- 2012
muleys$Year <- as.factor(muleys$Year)
muleys$YearBurst <- c(paste(muleys$id,muleys$Year,sep="_"))
muleys$YearBurst <- as.factor(muleys$YearBurst)
table(muleys$YearBurst)
```

7. Next we are going to rename our dataset to simply follow along with previous code unless you want to change d1 to muleys throughout

```
 d1 <- muleys

#Code separate each animal into a shapefile or text file to use as a "List"
# get input file
indata <- d1
innames <- unique(d1$YearBurst)
innames <- innames[1:12]#needs to be number of unique IDs
outnames <- innames
# begin loop to calculate home ranges
for (i in 1:length(innames)){
  data <- indata[which(indata$YearBurst==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("+proj=utm +zone=12 +datum=WGS84")
    #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.names=FALSE)
    write.table(paste(outnames[i],"txt",sep="."), sep="\t", quote=FALSE,
    row.names=FALSE, 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 otherwise it rights all animals each time
    #Note: There are 3 lines of code that are not active that can be activated
   #to export point shapefiles for all resulting animals but need to remove
   #as.POSIX column first
}
}
###############################################################################
###############################################################################
#
#
#Code below to get NSD and best movement model for each bird
#
#
###############################################################################
###############################################################################

date()

# 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 deer datasets

# Generation of results vectors
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))

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$Y/1000),(coords$X/1000))
# make ltraj: a trajectory of all the relocations
d2<-as.ltraj(coord,coords$DT,
coords$YearBurst,        #separate your data by individual.
burst=coords$YearBurst, #burst is used to creat subdivisions within an individual.
typeII=TRUE)       #typeII can be TRUE: radio-track data, or FALSE: not time
                   #recorded, such as tracks in the snow

#[1] "2007-06-05 16:00:00 EDT" "2015-03-12 14:00:00 EDT"
#you can now make your trajectory regular
#firstly create a reference start time
#refda <- strptime("00:00:00", "%H:%M:%S")   #all relocations should be altered
#to occur at 30 seconds past each minute

#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
#7200,           #stating there should be a location every 2 hours
#tol=7200,        #how many time units to search each side of expected location
#units="sec")   #specifying the time units

#you can now make your trajectory regular
#firstly create a reference start time
refda <- strptime("00:00:30", "%H:%M:%S")

##NOTE: The refda and d3 code above was not run because it results in too many
#relocations as "NA" that get removed below. Not quite sure the reason behind
# it being included? We can now make your trajectory regular
d4<-sett0(d2, 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 we 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=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(m2)        #this will print a summary of the converged model

###########################
##
##  DISPERSAL
```

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

m3 <-tryCatch(nls(nsdall ~  asym /(1+exp((xmid-zerostart)/scale)), #Equation 3 in
#Bunnefeld et al. 2011
start = c(asym=15000000,xmid=200000,scale=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(m3)        #this will print a summary of the converged model

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

m4 <- tryCatch(nls(nsdall ~ intercept, data=na.omit(datansd1),
   start = list(intercept = 0)),error=function(e)99999)
#Equation 4 in Bunnefeld et al. 2011 where c is a constant
summary(m4)        #this will print a summary of the converged model

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

m5 <- tryCatch(nls(nsdall ~ -1*zerostart,start=c(zerostart=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)        #this will print a summary of the converged model

#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])

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")#
RESULT
#write.table(RESULT,"OUT_AIC_NSD.txt",sep="\t")#Output to excel if needed

#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_5)
Nomadic <- rbind(ID,AICC_5,d_AICC_5,LL_AICC_5,wi_AICC_5)

RESULT2 <- rbind(Migratory,MixedMig,Disperser,HomeRange,Nomadic)
RESULT2
```

3.7 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 File - Change dir...
3. First we need to load the packages needed for the exercise

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

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

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

#CODE FOR AN INDIVIDUAL ANIMAL
muley15 <- subset(muleys, id=="D15")
muley15[1:10,]
muley15$id <- factor(muley15$id)
str(muley15)
summary <- table(muley15$UTM_Zone,muley15$id)
summary

#Sort data to address error in code and then look at first 10 records
#of data to confirm
muley15 <- muley15[order(muley15$GPSFixTime),]
muley15[1:10,]

######################################################
## 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))
str(muley15)

#Clean up muley15 for outliers
newmuleys <-subset(muley15, muley15$X > 599000 & muley15$X < 705000 &
   muley15$Y > 4167000 & muley15$timediff < 14401)
muley15 <- newmuleys
str(muley15)
```

5. Create a SpatialPointsDataFrame of specific columns available in the data

```
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)
str(merge)
```

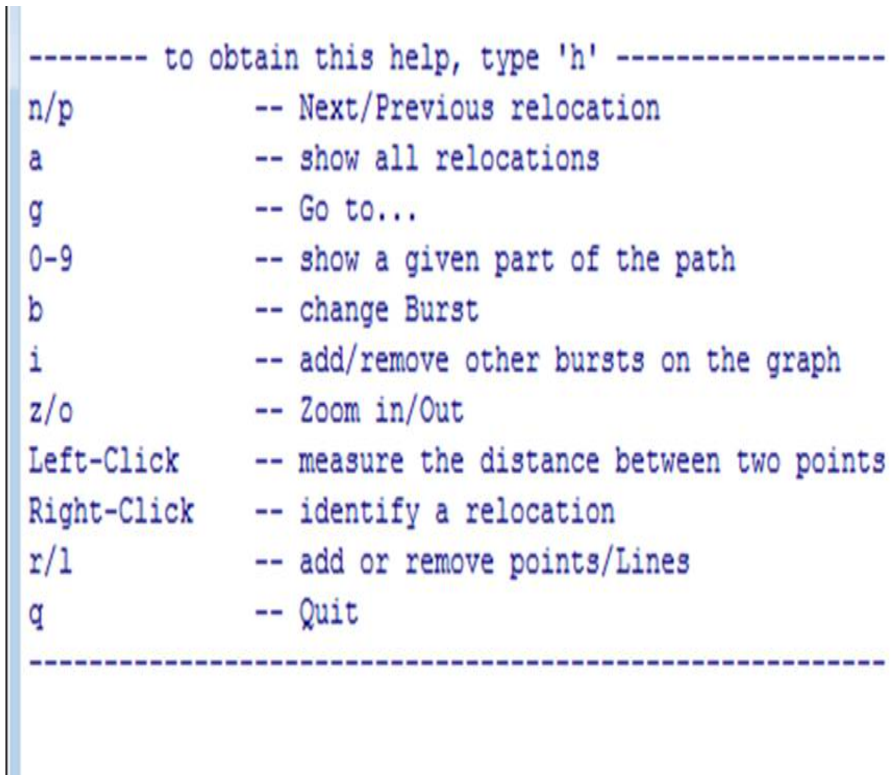
6. Now let's have a little fun with these mule deer locations and explore them as trajectories, over vegetation, or zooming in on a specific area

```
#Load vegetation raster layer textfile clipped in ArcMap
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")
proj4string(merge) <- CRS("+proj=utm +zone=12 +ellps=WGS84")
deer.albers <-spTransform(merge, CRS=Albers.crs)

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

veg <-raster("extentnlcd2.txt")
plot(veg)
points(deer.albers)
#Code below is used to just zoom in on grid using raster layer
e <- drawExtent()
#click on top left of crop box and bottom right of crop box create zoom
newclip <- crop(veg,e)
plot(newclip)
points(deer.albers, col="red")
vegspdf <- as(newclip,"SpatialPixelsDataFrame")
```

```
plot(ltr.albers, spixdf=vegspdf)

#Let's zoom in even closer
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)
```

7. Line of code below plots trajectory one location at a time so follow resulting commands with the keys on keyboard

```
trajdyn(ltr.zoom, burst = attr(ltr.zoom[[1]], "burst"),spixdf=zoomspdf)
```



```
-------- to obtain this help, type 'h' ------------------
n/p            -- Next/Previous relocation
a              -- show all relocations
g              -- Go to...
0-9            -- show a given part of the path
b              -- change Burst
i              -- add/remove other bursts on the graph
z/o            -- Zoom in/Out
Left-Click     -- measure the distance between two points
Right-Click    -- identify a relocation
r/l            -- add or remove points/Lines
q              -- Quit
---------------------------------------------------------
```

Figure 3.5: Commands to animate a trajectory using trajdyn function

Chapter 4

Home Range Estimation

Contents

Figures

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

In KDE, a kernel distribution (i.e. a three-dimensional hill or kernel) is placed on each telemetry location. The height of the hill is determined by the bandwidth of the distribution, and many distributions and methods are available (e.g. fixed versus adaptive, univariate versus bivariate bandwidth). We will focus here on "fixed kernel" but will alter the bandwidth selection. Datasets for avian and mammalian species can include as many as 10,000 locations and only the reference or default bandwidth (h_{ref}) was able to produce KDE in both Home Range Tools and adehabitat or adehabitatHR (Calenge 2007, 2011). Estimation with (h_{ref}) typically is not reliable for use on multimodal datasets because it results in over-smoothing of home ranges and multimodal distribution of locations is typical for most species (Worton 1995, Seaman et al. 1999).

1. Exercise 4.1 - Download and extract zip folder into your preferred location

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

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

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

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

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

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

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

6. Calculation of the 95 percent home range
   ```
   ver <- getverticeshr(ud, 95)
   plot(ver, add=TRUE)
   ```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

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

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

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

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

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

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

   #Now change h to href for comparison to LSCV later

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   NOTE: the h followed by a number are outputs from the $h_{plug\text{-}in}$ for the bandwidth matrix estimated for each animal. They are in there for reference but don't really need them.

7. We also need to tell R which contours to output
```
# set up levels for home range
levels <- c(50,95,99)
```

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

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

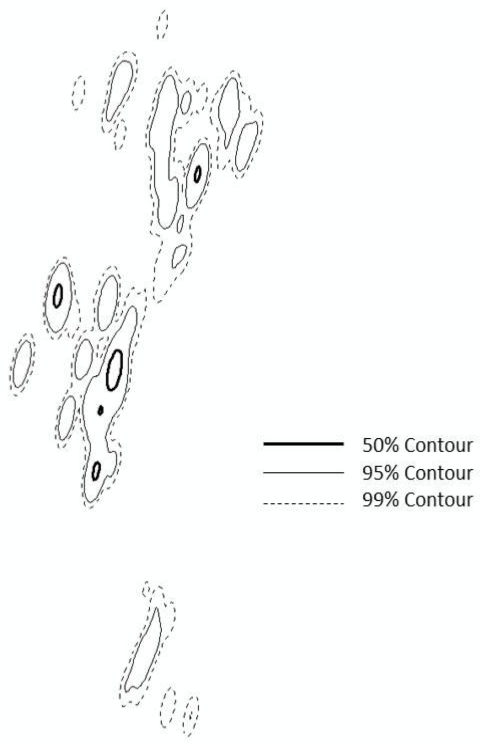


Figure 4.1: Example of KDE with h$_{plug\text{-}in}$ smoothing parameter to estimate size of home range for an American White Pelican.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

writeOGR(hpikde.50vol,dsn="output",  layer="cat143plug50",driver="ESRI Shapefile",
   overwrite=TRUE)
writeOGR(hpikde.80vol,dsn="output", layer="cat143plug80",driver="ESRI Shapefile",
   overwrite=TRUE)
writeOGR(hpikde.90vol,dsn="output", layer="cat143plug90",driver="ESRI Shapefile",
   overwrite=TRUE)
writeOGR(hpikde.95vol,dsn="output", layer="cat143plug95",driver="ESRI Shapefile",
   overwrite=TRUE)
writeOGR(hpikde.99vol,dsn="output", layer="cat143plug99",driver="ESRI Shapefile",
   overwrite=TRUE)
```

4.4 Brownian Bridge Movement Models (BBMM)

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

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

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

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

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

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

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

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

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

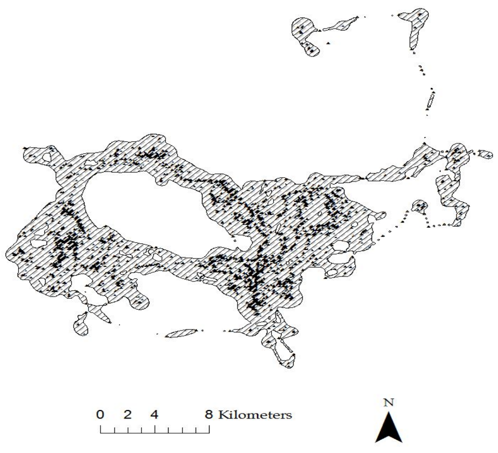


Figure 4.2: Example of 95% BBMM home range for a Florida Panther.

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

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

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

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

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

NOTE:

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

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

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

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

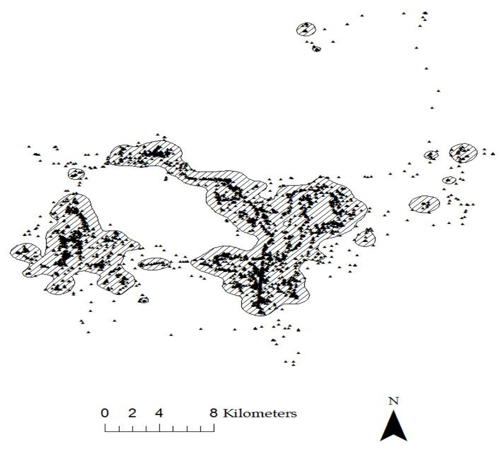


Figure 4.3: Example of 95% KDE home range with h$_{plug\text{-}in}$ for a Florida Panther.

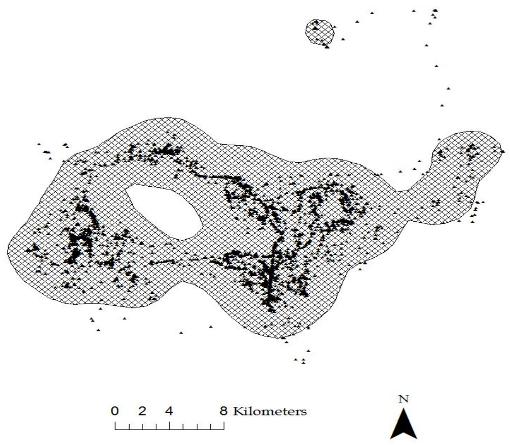


Figure 4.4: Example of 95% KDE home range with h$_{ref}$ for a Florida Panther.

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

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

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

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

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

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

   (a) Convert ASCII files to Rasters using Conversion Toolbox

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

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

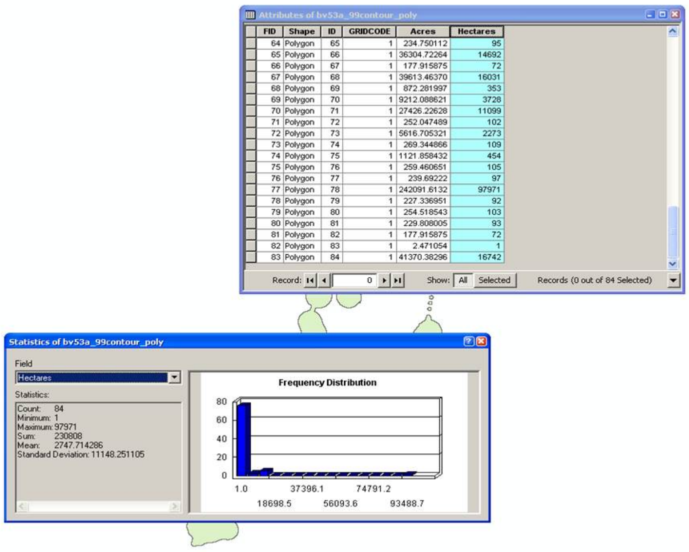
```
Toolbox>Spatial Analyst Tools>Math>Logical>Is Null
     Input raster: tv53_99contr
     Output raster: IsNull_bv53_3
```

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

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

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

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


Attributes of bv53a_99contour_poly

FID	Shape	ID	GRIDCODE	Acres	Hectares
64	Polygon	65	1	234.750112	95
65	Polygon	66	1	36304.72264	14692
66	Polygon	67	1	177.915875	72
67	Polygon	68	1	39613.46370	16031
68	Polygon	69	1	872.281997	353
69	Polygon	70	1	9212.088621	3728
70	Polygon	71	1	27426.22628	11099
71	Polygon	72	1	252.047489	102
72	Polygon	73	1	5616.705321	2273
73	Polygon	74	1	269.344866	109
74	Polygon	75	1	1121.858432	454
75	Polygon	76	1	259.460651	105
76	Polygon	77	1	239.69222	97
77	Polygon	78	1	242091.6132	97971
78	Polygon	79	1	227.336951	92
79	Polygon	80	1	254.518543	103
80	Polygon	81	1	229.808005	93
81	Polygon	82	1	177.915875	72
82	Polygon	83	1	2.471054	1
83	Polygon	84	1	41370.38296	16742

Record: 0 Show: All Selected Records (0 out of 84 Selected)

Statistics of bv53a_99contour_poly

Field: Hectares

Statistics:
Count: 84
Minimum: 1
Maximum: 97971
Sum: 230808
Mean: 2747.714286
Standard Deviation: 11148.251105

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

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

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

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

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

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

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

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

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

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

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

12. Use package rgdal to write shapefile to ArcMap

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

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

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

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

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

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

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

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

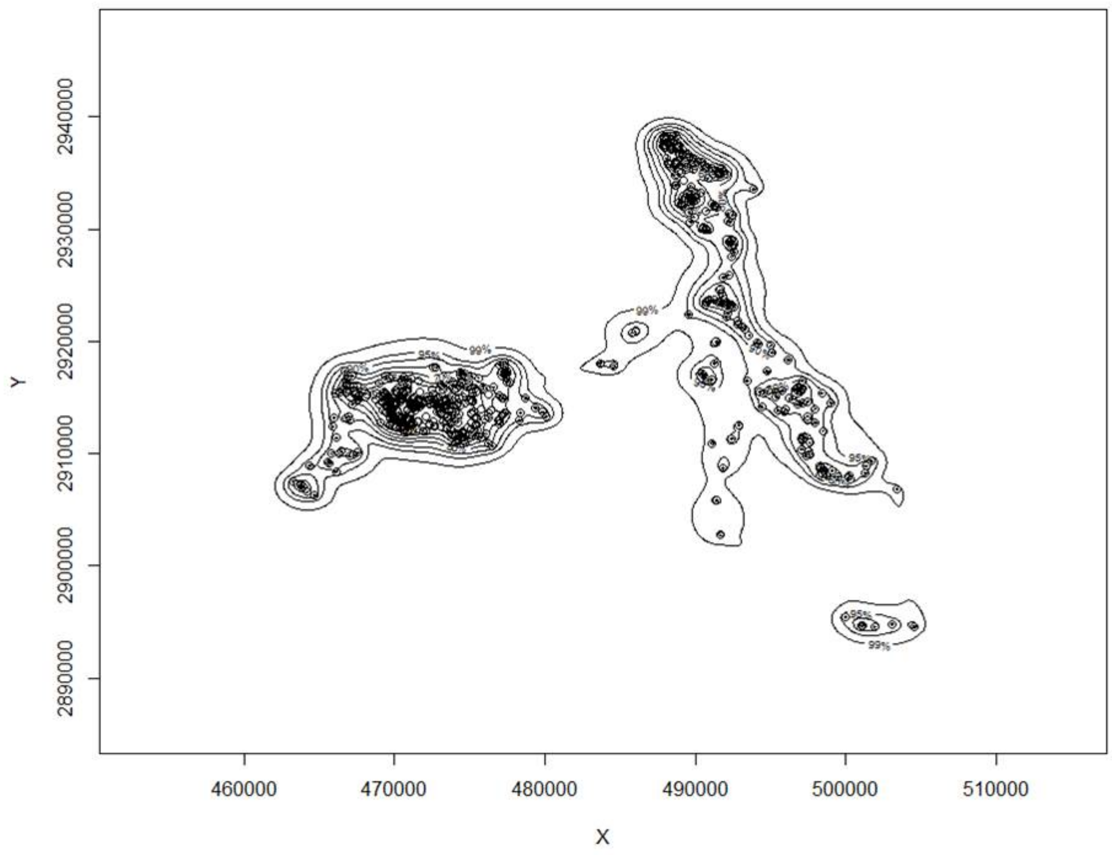


Figure 4.6: Home range of one panther using BBMM showing all contours.

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

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

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

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

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

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

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

windows()

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

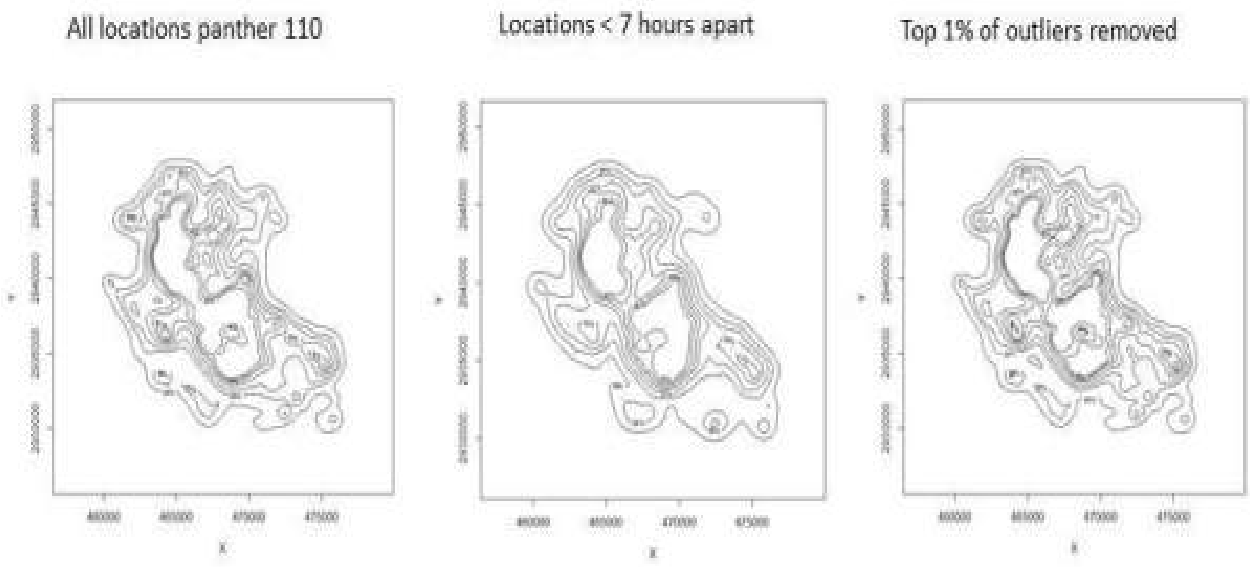


Figure 4.7: Home range using BBMM for panther 110 with various time lags incorporated.

4.5 Movement-based Kernel Density Estimation (MKDE)

If we want to take both BBMM and KDE to a higher level we can incorporate movement-based estimators of home range. Movement-based Kernel Density Estimation (MKDE) incorporates movements trajectories and habitat components of the landscape your animal occupies (Benhamou 2011, Benhamou and Cornelis 2010). This method requires a habitat layer and the adehabitatHR package requires that no duplicate entries exist for a given date so makes estimates of home range with GPS data problematic. Furthermore, after tirelessly trying this method for days using panther data with time intervals, we changed to vulture data that had dates but then had to remove duplicates. If you have worked out all of these issues, you can skip ahead to MKDE estimates with your data starting at Step 6.

1. Exercise 4.5 - Download and extract zip folder into your preferred location

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

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

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

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

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

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

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

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

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

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

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

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

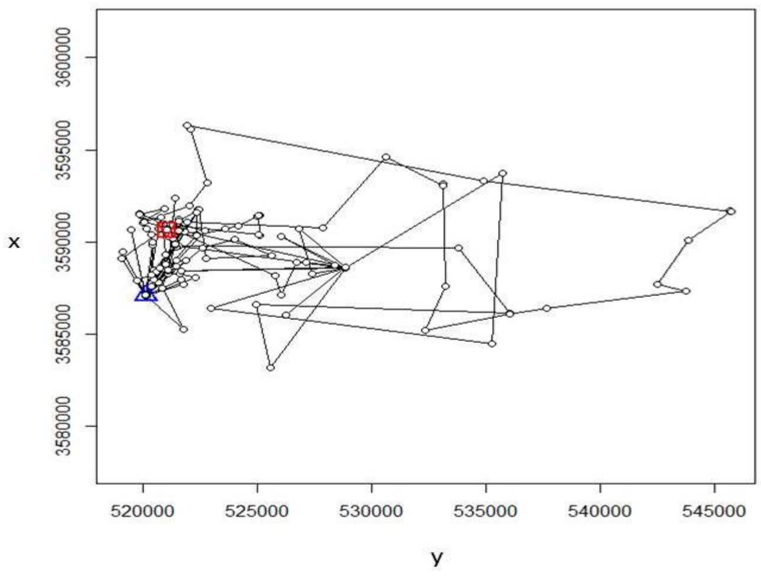


Figure 4.8: Locations of one vulture in UTM 17N.

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

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

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

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

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

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

#Cast the Dates as POSIXct dates
```

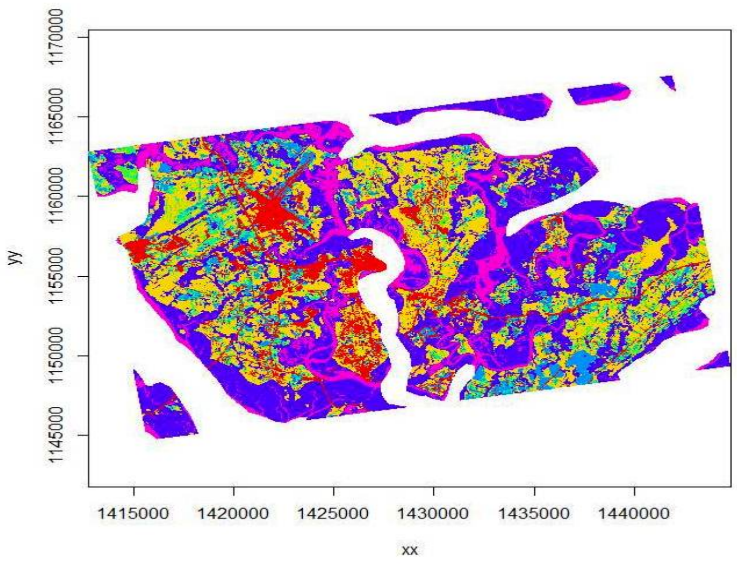


Figure 4.9: Imported habitat layer in Albers Equal Area Conic Projection.

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

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

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

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

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

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

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

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

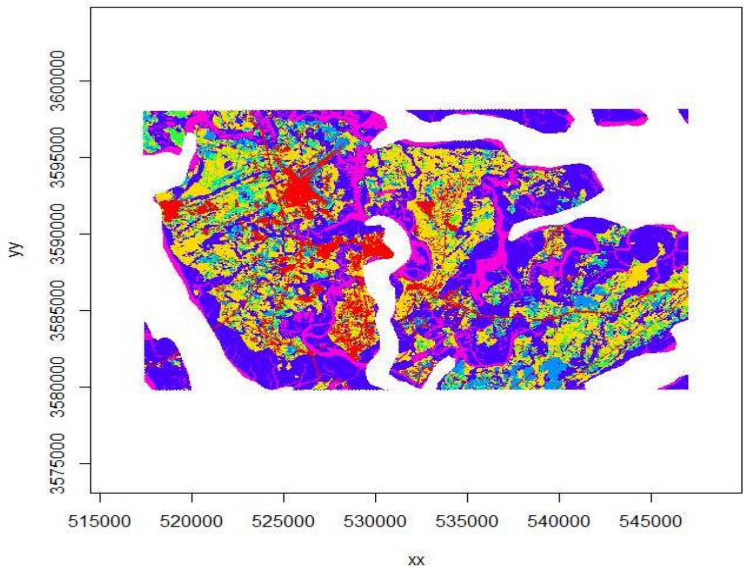


Figure 4.10: Imported habitat layer projected to UTM Zone 17N similar to vulture locations.

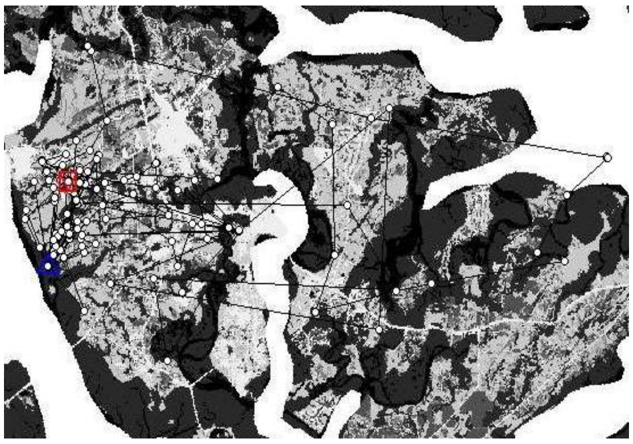


Figure 4.11: Overlay of ltraj for one bird on spixdf=habitat in UTM Zone 17N.

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


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

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

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

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

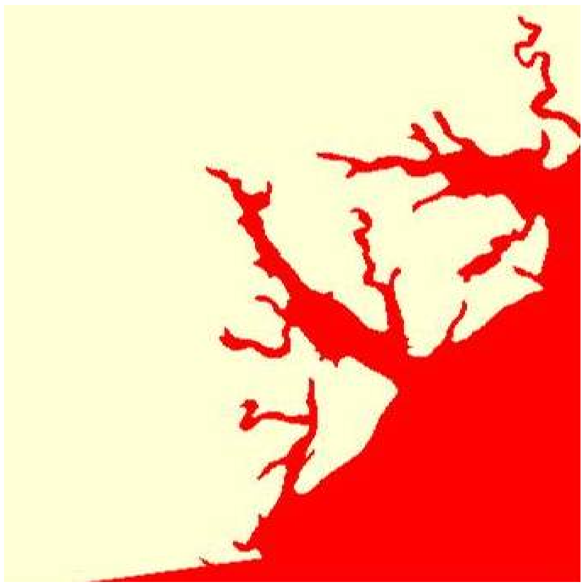


Figure 4.12: Red identifies ocean and tributaries not used by vultures.

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

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

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

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

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

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

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

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

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

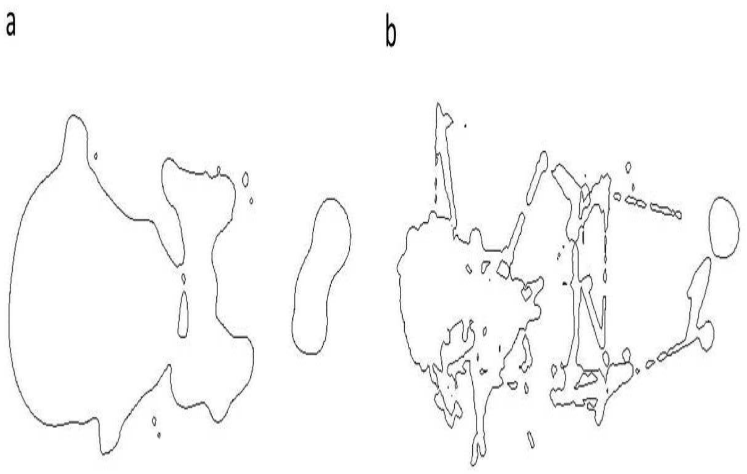


Figure 4.13: Contours of home range for 2 black vultures estimated using the Movement-based kernel density method (MKDE)

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

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

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

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

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

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

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

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

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

names(ud2) <- id(ltraj)

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

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

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

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

names(ud3) <- id(ltraj)

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

```
plot(ver2b)
```

4.6 Dynamic Brownian Bridge Movement Model (dBBMM)

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

1. Exercise 4.6 - Download and extract zip folder into your preferred location

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

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

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

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

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

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

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

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

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

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

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

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

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

8. Now create a dBBMM object

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

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

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

10. Plot the movement of the animal

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

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

4.7 Characteristic Hull Polygons (CHP)

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

1. Exercise 4.7 - Download and extract zip folder into your preferred location

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

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

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

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

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

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

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

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

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

7. Creates a Spatial Data Frame from Spatial points

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

8. Creates a spatial data frame of ID

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

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

   ```
   coordinates(idsp)<-sppt

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

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

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

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

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

    ```
    plot(res)
    ```

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

    ```
    MCHu2hrsize(res)

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

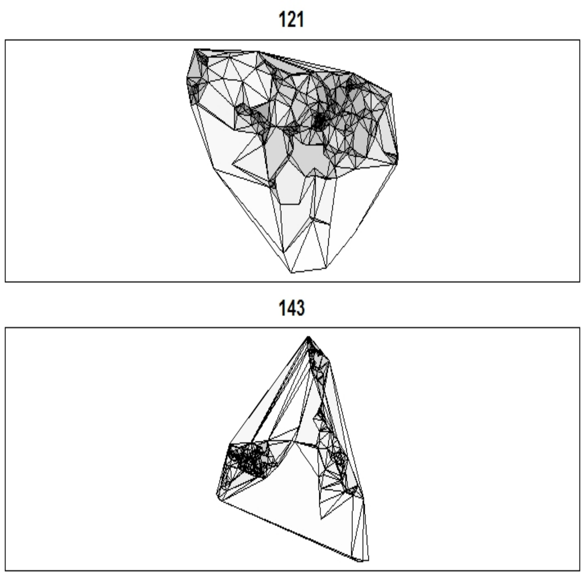


Figure 4.14: Example of CHP home range for 2 Florida panther.

```
90   2431.27046  12912.0320
100  5538.22402 103361.1336

OR
```

13. Computes the home range size for 95 percent

```
MCHu2hrsize(res, percent=95)

OR use

getverticeshr(res, percent=95)

Object of class "SpatialPolygonsDataFrame" (package sp):

Number of SpatialPolygons:  2

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

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

4.8 Local Convex Hull (LoCoH)

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

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

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

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

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

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

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

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

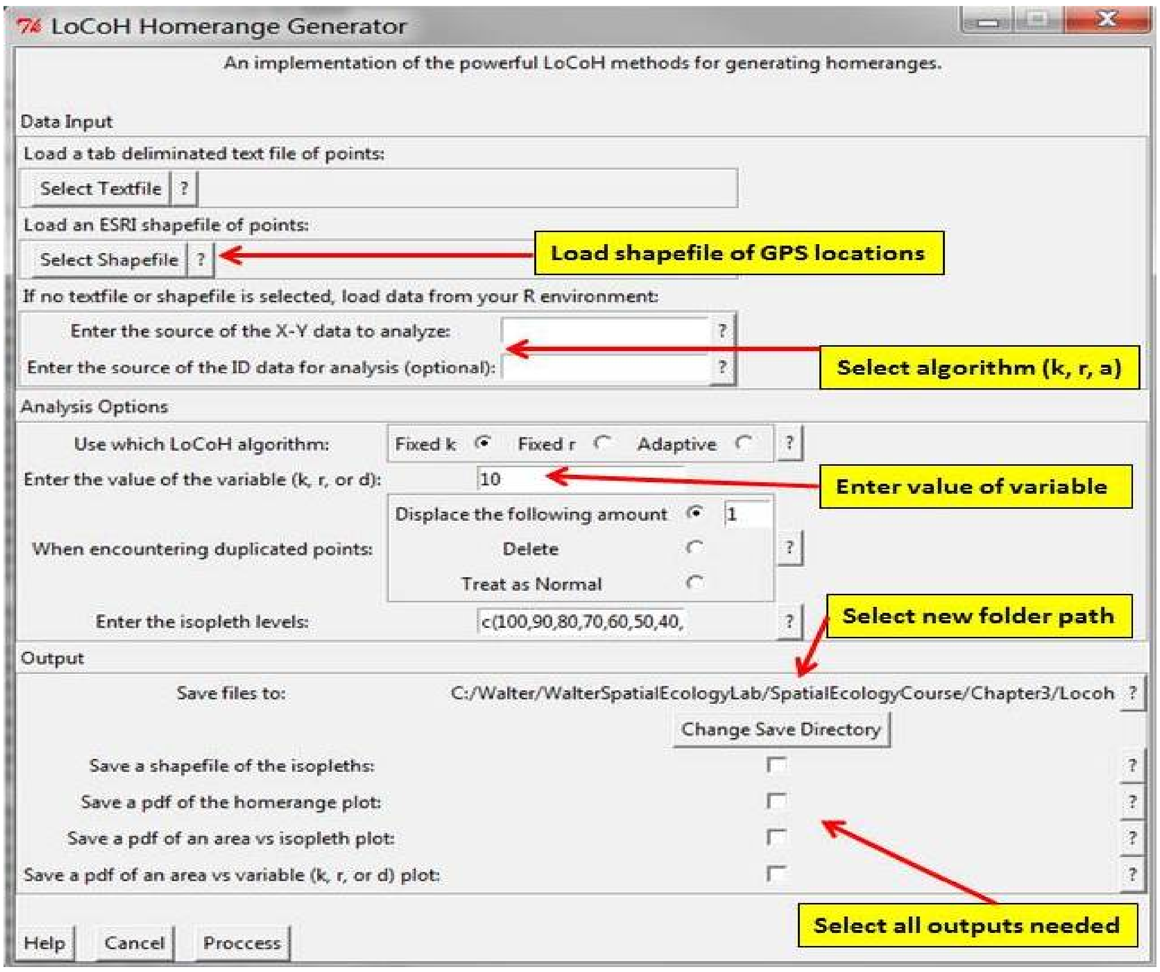
8. To access the GUI we can simply invoke LoCoH using the command

   ```
       locoh()
   ```

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

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

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


LoCoH Homerange Generator

An implementation of the powerful LoCoH methods for generating homeranges.

Data Input

Load a tab deliminated text file of points:

Load an ESRI shapefile of points:

If no textfile or shapefile is selected, load data from your R environment:

Enter the source of the X-Y data to analyze:

Enter the source of the ID data for analysis (optional):

Analysis Options

Use which LoCoH algorithm: [✓ Fixed k] Fixed r Adaptive

Enter the value of the variable (k, r, or d): 10

When encountering duplicated points: [✓ Displace the following amount] 1 / Delete / Treat as Normal

Enter the isopleth levels: c(100,90,80,70,60,50,40,

Output

Save files to: C:/Walter/WalterSpatialEcologyLab/SpatialEcologyCourse/Chapter3/Locoh

Save a shapefile of the isopleths:

Save a pdf of the homerange plot:

Save a pdf of an area vs isopleth plot:

Save a pdf of an area vs variable (k, r, or d) plot:

Figure 4.15: The LoCoH GUI.

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

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

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

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


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

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

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

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

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

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

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

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

Chapter 5

Overlap Indices

Contents

Overlap indices can be useful for determining the spatial interactions between animals using relocations of animals occupying similar areas. There are various overlap indices available and a good reference is Fieberg and Kochanny (2005). The overlap methods presented have code and more detailed descriptions in the adehabitatHR package for R (Calenge 2011). Methods of home range overlap simply require coordinate data after estimating home range. Note that example below is not the same dataset supplied for this exercise.

```
#First we can load some data and create some generic utilization distributions
#before estimating overlap:
library(adehabitatHR)

#Creates a Spatial Points Data Frame for 2 animals by ID
twocats <-read.csv("AllHRlocs.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=17N +ellps=WGS84")

#Creates a Spatial Data Frame from all locations
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

#First we need to create utilization distributions for each panther
ud <- kernelUD(idsp[,1])
```

```
OR

kernelUD(idsp[,1], h = "href", grid = 200, same4all = TRUE, hlim = c(0.1, 1.5),
     kern = c("bivnorm"), extent = 0.5)

#output of UDs for each panther
image(ud)

NOTE: kerneloverlap will just estimate overlap indices for only the locations
```

5.1 Percent overlap

HR – Proportion of animal i's home range that is overlapped by animal j's home range (Kernohan et al. 2001).

```
kerneloverlaphr(idsp[,1], grid=200, method="HR", percent=95, conditional=TRUE)

         FP048     FP094     FP110      FP113     FP121     FP128      FP130
FP048 1.0000000 0.2539683 0.3611111 0.67857143 0.2738095 0.2738095 0.56746032
FP094 0.1743869 1.0000000 0.3106267 0.23705722 0.3487738 0.3160763 0.12261580
FP110 0.1834677 0.2298387 1.0000000 0.26612903 0.2862903 0.7379032 0.07661290
FP113 0.5135135 0.2612613 0.3963964 1.00000000 0.2672673 0.3273273 0.51051051
FP121 0.1490281 0.2764579 0.3066955 0.19222462 1.0000000 0.2915767 0.11663067
FP128 0.1326923 0.2230769 0.7038462 0.20961538 0.2596154 1.0000000 0.05769231
FP130 0.7150000 0.2250000 0.1900000 0.85000000 0.2700000 0.1500000 1.00000000
```

5.2 Probability overlap

PHR – Probability of animal j being located in animal i's home range and vice versa (i.e., volume measure; Ostfeld 1986).

```
kerneloverlaphr(ud, meth="PHR", conditional=TRUE)

         FP048     FP094      FP110     FP113     FP121     FP128       FP130
FP048 0.9494305 0.1805179 0.42561390 3.1250391 1.1182583 0.3443277   0.0583
FP094 0.2669219 0.8731045 0.73504044 1.2518993 2.1795156 1.6187565   0.0262
FP110 0.4174817 0.3190549 2.67619435 1.6975608 2.8534787 4.5286474   0.0076
FP113 0.7618825 0.2652719 0.52653119 4.8046721 0.9172712 1.7769509   0.0895
FP121 0.2661436 0.3581339 0.93388969 1.2381488 7.5593950 1.7633521   0.0244
FP128 0.1645584 0.2966145 2.33213801 0.8889413 2.5113787 5.3728081   0.0062
FP130 0.6965648 0.1028496 0.07726699 3.4283998 0.4992657 0.1184470   0.0941
```

5.3 Bhattacharyya's affinity

BA – a statistical measure of affinity between 2 populations that assumes they use space independently of one another (Bhattacharyya 1943). Values range from zero (no overlap) to 1 (identical UDs).

```
kerneloverlaphr(ud, meth="BA", conditional=TRUE)

          FP048      FP094      FP110     FP113      FP121      FP128
FP048 0.9494305 0.19300494 0.32460745 1.3935548 0.44029134 0.19713762
FP094 0.1930049 0.87310451 0.42688290 0.4840179 0.76378458 0.60678828
FP110 0.3246075 0.42688290 2.67619435 0.7715258 1.39007501 2.96104840
FP113 1.3935548 0.48401785 0.77152577 4.8046721 0.93867516 1.02377927
FP121 0.4402913 0.76378458 1.39007501 0.9386752 7.55939503 1.53768461
FP128 0.1971376 0.60678828 2.96104840 1.0237793 1.53768461 5.37280812
FP130 0.1649118 0.04089019 0.02278675 0.5029844 0.09429946 0.02478678
```

5.4 Utilization distribution overlap index

UDOI – an UD overlap index similar to Hurlbert index of niche overlap that assumes they use space independently of one another (Hurlbert 1978). Values range from zero (no overlap) to 1 (uniformly distributed and have 100% overlap) but can be >1 if both UDs are nonuniformly distributed and have a high degree of overlap.

```
kerneloverlaphr(ud, meth="UDOI", conditional=TRUE)

           FP048       FP094        FP110       FP113        FP121
FP048 1.88354584 0.058535090 1.200312e-01  2.80355009   0.26373006
FP094 0.05853509 1.215285513 2.373785e-01  0.27909161   0.83782880
FP110 0.12003124 0.237378462 1.218177e+01  0.67534032   2.72820629
FP113 2.80355009 0.279091613 6.753403e-01 34.89888157   1.04793012
FP121 0.26373006 0.837828797 2.728206e+00  1.04793012 128.78219185
FP128 0.04282855 0.596978847 1.608828e+01  1.26045478   3.04242496
FP130 0.03891154 0.002097975 5.547623e-04  0.34714991   0.01114027
```

5.5 Hellinger's distance

HD – a measure of distance between 2 populations (Matusita 1973).

```
kerneloverlaphr(ud, meth="HD", conditional=TRUE)

          FP048     FP094    FP110    FP113    FP121    FP128     FP130
FP048 0.0000000 1.2858818 1.864468 1.722496 2.762445 2.606872 0.8448535
FP094 1.2858818 0.0000000 1.641808 2.605304 2.627723 2.243287 0.9504207
FP110 1.8644681 1.6418079 0.000000 3.212855 2.730465 1.458391 1.6753830
FP113 1.7224962 2.6053041 3.212855 0.000000 3.301337 3.593655 1.9730367
FP121 2.7624454 2.6277234 2.730465 3.301337 0.000000 3.207293 2.7322091
FP128 2.6068720 2.2432869 1.458391 3.593655 3.207293 0.000000 2.3475101
FP130 0.8448535 0.9504207 1.675383 1.973037 2.732209 2.347510 0.0000000
```

5.6 Volume of intersection index

Volume of intersection under the full UDs of 2 animals (Seidel 1992, Millspaugh et al. 2000). Values range from zero (no overlap) to 1 (identical UDs).

```
kerneloverlaphr(ud, meth="VI", conditional=TRUE)

           FP048       FP094       FP110       FP113       FP121
FP048 0.94973489 0.015706682 0.081581400 0.356221669 0.015685067
FP094 0.01570668 0.873256975 0.064468281 0.049201257 0.036421940
FP110 0.08158140 0.064468281 2.542692800 0.106706398 0.082470262
FP113 0.35622167 0.049201257 0.106706398 4.804125850 0.126326346
FP121 0.01568507 0.036421940 0.082470262 0.126326346 7.560716837
FP128 0.01827549 0.075271608 0.665701571 0.090174304 0.052936190
FP130 0.01682551 0.001788199 0.002577415 0.039754312 0.000000000
```

```
#Plot out to visualize overlap
plot(idsp, col="yellow")
uds <- getverticeshr(ud)
plot(uds, add=TRUE)

plot(idsp, col="yellow")
ud1 <- getverticeshr(ud[[1]])
plot(ud1, add=TRUE)
ud2 <- getverticeshr(ud[[2]])
plot(ud2, lwd=2, add=TRUE)
ud3 <- getverticeshr(ud[[3]])
plot(ud3, lwd=3, add=TRUE)
ud4 <- getverticeshr(ud[[4]])
plot(ud4, lwd=4, add=TRUE)

#An alternative way with only the locations
kerneloverlap(idsp[,1], grid=200, method="HR", percent=95, conditional=TRUE)
kerneloverlap(idsp[,1], grid=200, method="PHR", percent=95, conditional=TRUE)
kerneloverlap(idsp[,1], grid=200, method="BA", percent=95, conditional=TRUE)
kerneloverlap(idsp[,1], grid=200, method="UDOI", percent=95, conditional=TRUE)
kerneloverlap(idsp[,1], grid=200, method="HD", percent=95, conditional=TRUE)
kerneloverlap(idsp[,1], grid=200, method="VI", percent=95, conditional=TRUE)
```

Chapter 6

Three-dimensional Analyses

Contents

Figures

6.1 Three-dimensional home range

We can calculate 3-dimensional surface area in ArcMap 10.x (ArcMap; Environmental Systems Research Institute, Redlands, California) using standard 30-m United States Geological Survey DEMs and the DEM Surface Tools for ArcMap extension (Jenness 2004). We acquired all DEM data from United States Department of Agriculture, Natural Resource Conservation Service (http://datagateway.nrcs.usda.gov/) at the Geospatial Data Gateway. We can then use the surface area tool to calculate true surface area of the landscape for each grid cell using the DEM elevation from the surrounding 8 cells. The new grid cell values represented the 3-dimensional surface area for the land area contained within that cell's boundaries. We then summed all grid cell values within the animal's home-range polygon to derive a topographic home range for each individual.

Alternatively, we can create home ranges in R and look at them in 3D using the package *rasterVis* that now includes the *rgl* package. We can also view DEMs in R and create slope, aspect, or hillshade using the *rasterVis* package.

1. Exercise 6.1 - Download and extract zip folder into your preferred location

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

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

```
library(rasterVis)
library(raster)
library(sp)
library(rgdal)
library(maptools)#writeAsciiGrid
library(ks)#hpikde.ud
library(adehabitatHR)
library(adehabitatMA)
```

4. Now open the script "Raster3dScript.R" and run code directly from the script

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

#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.80)
muleys <- newmuleys
newmuleys <-subset(muleys, muleys$Long < -107)
muleys <- newmuleys

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

#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 24 hours apart in time
muleys <- subset(muleys, muleys$timelag < 18000)

#Code separate each animal into a shapefile or text file
# get input file
indata <- muleys
innames <- unique(muleys$id)
innames <- innames[1:2]
outnames <- innames
# begin loop to calculate home ranges
for (i in 1:length(innames)){
  data <- indata[which(indata$id==innames[i]),]
  if(dim(data)[1] != 0){
    # export the point data into a shp file
    data.xy = data[c("X", "Y")]
    coordinates(data.xy) <- ~X+Y
    sppt <- SpatialPointsDataFrame(coordinates(data.xy),data)
    proj4string(sppt) <- CRS("+proj=utm +zone=12 +datum=WGS84")
    #writePointsShape(sppt,fn=paste(outnames[i],sep="/"),factor2char=TRUE)
    sppt <-data[c(-8,-9)]
    write.table(sppt, paste(outnames[i],"txt",sep="."), sep="\t", quote=FALSE,
      row.names=FALSE)
    write.table(paste(outnames[i],"txt",sep="."), "In_list.txt",sep="\t",
      quote=FALSE, row.names=FALSE, col.names=FALSE, append=TRUE)
}
}
```

5. Reads in each text file for each animal using the In_list.txt file

```
List<-read.table("In_list.txt",sep="\t",header=F)
head(List) #"List" contains the filenames (e.g. "D4.txt") of the deer data
           # sets exported from code above
```

6. Begin loop to create home ranges for each deer using $h_{plug\text{-}in}$

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

coords<-read.table(as.character(List[i,]),sep="\t",header=T)
loc<-coords[,c("X", "Y")]

# Reference grid : input parameters
RESO <- 100 # grid resolution (m)
BUFF <- 5000 # grid extent (m) (buffer around location extremes)
XMIN <- RESO*(round(((min(coords$X)-BUFF)/RESO),0))
YMIN <- RESO*(round(((min(coords$Y)-BUFF)/RESO),0))
XMAX <- XMIN+RESO*(round(((max(coords$X)+BUFF-XMIN)/RESO),0))
YMAX <- YMIN+RESO*(round(((max(coords$Y)+BUFF-YMIN)/RESO),0))
NRW <- ((YMAX-YMIN)/RESO)
NCL <- ((XMAX-XMIN)/RESO)

#Generation of refgrid
refgrid<-raster(nrows=NRW, ncols=NCL, xmn=XMIN, xmx=XMAX, ymn=YMIN, ymx=YMAX)
refgrid<-as(refgrid,"SpatialPixels")

#PKDE computation
##convert the SpatialGrid class to a raster
sampRaster <- raster(refgrid)

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

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

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

##Calculate Hpi from the xy coordinates we used above, Hpi performs bivariate
## smoothing whereas hpi
#performs univariate. Bivariate is preferred.
Hpi1 <- Hpi(x = loc)

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

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

#We can take this raster and put it back into an adehabitatHR object
##Cast over to SPxDF
hpikde.px <- as(hpikde.raster,"SpatialPixelsDataFrame")

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

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

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

#Write each home range to an ascii file if needed.
#writeAsciiGrid(udvol, paste(substr(List[i,],1,7),"PKDE","asc", sep="."))
```

7. Generate 3D of each home range to plot in separate windows all within the home range loop.

```
if (require(rgl)) {
#data(loc)
r <- raster(udvol)
extent(r) <- c(0, 610, 0, 870)
drape <- cut(r, 5)
plot3D(r, drape=drape, zfac=2)
}
}
```

8. We can also run code for creating KDE using h$_{ref}$ smoothing for comparison

```
#Begin loop to generate home ranges
for(i in 1:nrow(List)) {

coords<-read.table(as.character(List[i,]),sep="\t",header=T)
head(coords)

loc<-coords[,c("X", "Y")]
coordinates(loc) = c("X", "Y")

#Coordinate system info may not be needed
proj4string(loc) = CRS("+proj=utm +zone=12 +datum=WGS84")

#Generation of a reference grid around the location data

#Reference grid : input parameters
RESO <- 100 # grid resolution (m)
BUFF <- 5000 # grid extent (m) (buffer around location extremes)
XMIN <- RESO*(round(((min(coords$X)-BUFF)/RESO),0))
YMIN <- RESO*(round(((min(coords$Y)-BUFF)/RESO),0))
XMAX <- XMIN+RESO*(round(((max(coords$X)+BUFF-XMIN)/RESO),0))
YMAX <- YMIN+RESO*(round(((max(coords$Y)+BUFF-YMIN)/RESO),0))
NRW <- ((YMAX-YMIN)/RESO)
NCL <- ((XMAX-XMIN)/RESO)

#Generation of refgrid
refgrid<-raster(nrows=NRW, ncols=NCL, xmn=XMIN, xmx=XMAX, ymn=YMIN, ymx=YMAX)
refgrid<-as(refgrid,"SpatialPixels")

#LKDE computation
ud <- kernelUD(loc, grid=refgrid, h="href")

# Volume contours computation
udvol1<-getvolumeUD(ud, standardize = FALSE)

#writeAsciiGrid(udvol, paste(substr(List[i,],1,7),"LKDE","asc", sep="."))

if (require(rgl)) {
#data(loc)
r1 <- raster(udvol1)
extent(r1) <- c(0, 610, 0, 870)
drape <- cut(r1, 5)
plot3D(r1, drape=drape, zfac=2)
}
}
```

9. We can also run code for creating BBMM computation for further comparison

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

coords<-read.table(as.character(List[i,]),sep="\t",header=T)
head(coords)

loc<-coords[,c("X", "Y")]
coordinates(loc) = c("X", "Y")

#Coordinate system info may not be needed
proj4string(loc) = CRS("+proj=utm +zone=12 +datum=WGS84")

#Generation of a reference grid around the location data
#Reference grid : input parameters
RESO <- 100 # grid resolution (m)
BUFF <- 5000 # grid extent (m) (buffer around location extremes)
XMIN <- RESO*(round(((min(coords$X)-BUFF)/RESO),0))
YMIN <- RESO*(round(((min(coords$Y)-BUFF)/RESO),0))
XMAX <- XMIN+RESO*(round(((max(coords$X)+BUFF-XMIN)/RESO),0))
YMAX <- YMIN+RESO*(round(((max(coords$Y)+BUFF-YMIN)/RESO),0))
NRW <- ((YMAX-YMIN)/RESO)
NCL <- ((XMAX-XMIN)/RESO)

#Generation of refgrid
refgrid<-raster(nrows=NRW, ncols=NCL, xmn=XMIN, xmx=XMAX, ymn=YMIN, ymx=YMAX)
##Get the center points of the mask raster with values set to 1
refgrid <- xyFromCell(refgrid, 1:ncell(refgrid))

#BBMM computation
BBMM <- brownian.bridge(x=coords$X, y=coords$Y, time.lag=coords$timelag[-1],
    area.grid=refgrid, location.error=22, max.lag=18000)

# Volume contours computation
# Create a data frame from x,y,z values
BBMM.df <- data.frame("x"=BBMM$x,"y"=BBMM$y,"z"=BBMM$probability)
##Make a raster from the x, y, z values, assign projection from above,
## match the resolution to that of the raster mask, note 100 is the cell
##resolution defined in evalPoints above
bbmm.raster <- rasterFromXYZ(BBMM.df, res=c(100,100), crs=proj4string(loc))

##Cast the data over to an adehabitatHR estUD
bbmm.px <- as(bbmm.raster, "SpatialPixelsDataFrame")
bbmm.ud <- new("estUD",bbmm.px)
bbmm.ud@vol = FALSE
bbmm.ud@h$meth = "BBMM"
##Convert the raw UD values to volume
udvol2 <- getvolumeUD(bbmm.ud, standardize=TRUE)
proj4string(udvol2) = CRS("+proj=utm +zone=12 +datum=WGS84")

if (require(rgl)) {
r2 <- raster(udvol2)
plot3D(r2,zfac=-2, xlim = XMIN, ylim = YMIN,xlab = "x", ylab = "y", zlab = "z",
   rev=TRUE)
title3d('UD with Brownian Bridge Movement Model ')
decorate3d()
}
}
```

6.2 Three-dimensional exploration of digital elevation models (DEMs)

1. Exercise 6.2 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(rasterVis)
   library(raster)
   library(rgl)
   ```

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

   ```
   dem <-raster("dcdemascii12.txt")
   plot(dem)

   #Or as a .tif file seems to look better
   ```

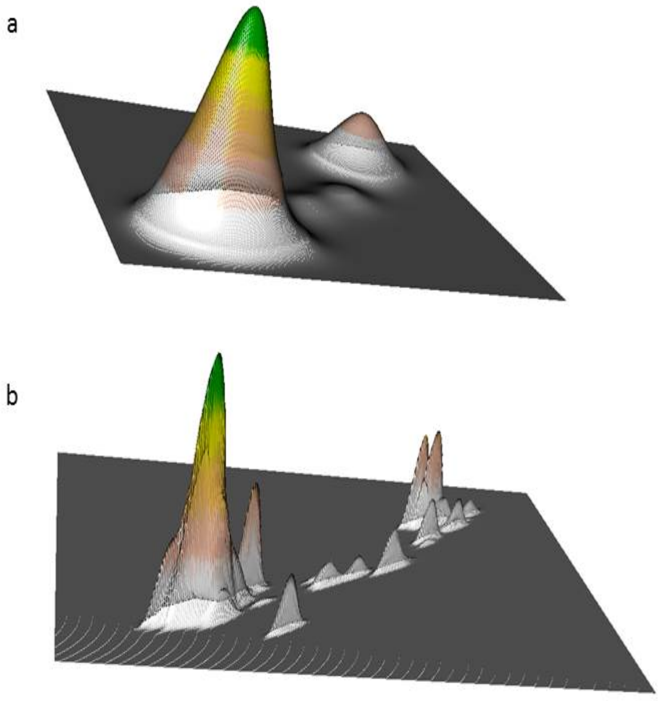


Figure 6.1: Example home range of a mule deer in 3D using KDE with a) h_{ref} and b) $h_{plug\text{-}in}$ bandwidth selection.

```
dem2 <- raster("dovecreekdem.tif")
plot(dem2)

if (require(rgl)) {
extent(dem2) <- c(0, 610, 0, 870)
drape <- cut(dem, 5)
plot3D(dem2, drape=drape, zfac=2)
}

#or simply
plot3D(dem2,drape=dem2)
```

5. We can also create a variety of covariates using DEMs (i.e., slope, aspect)

```
highGround = dem > 2000
slope = terrain(dem,opt='slope', unit='degrees')
aspect = terrain(dem,opt='aspect',unit='degrees')
hill = hillShade(slope,aspect,40,270)
plot(hill,col=grey(0:100/100),legend=FALSE)

plot(dem2,col=rainbow(25,alpha=0.35))
```

```
windows()
plot(slope,col=rainbow(25,alpha=0.35))
windows()
plot(aspect,col=rainbow(25,alpha=0.35))
```

6. We can also investigate various components of the ruggedness of terrain in a DEM with the function *terrain* in the *raster* package. TRI (Terrain Ruggedness Index) is the mean of the absolute differences between the value of a cell and the value of its 8 surrounding cells. Values of TRI are lower in flatter areas but high in both steep areas and in steep, rugged areas (Sappington et al. 2007). TPI (Topographic Position Index) is the difference between the value of a cell and the mean value of its 8 surrounding cells. Topographic position (e.g., valley bottom, mid-slope, ridge-top) can provide valuable information about the geomorphology of the region (**?**). Roughness is the difference between the maximum and the minimum value of a cell and its 8 surrounding cells. Roughness is the difference between the maximum and the minimum value of a cell and its 8 surrounding cells.

```
x <- terrain(dem2, opt=c('slope', 'aspect'), unit='degrees')
plot(x)
# TPI for different neighborhood size:
tpiw <- function(x, w=5) {
m <- matrix(1/(w^2-1), nc=w, nr=w)
m[ceiling(0.5 * length(m))] <- 0
f <- focal(x, m)
x - f
}
tpi5 <- tpiw(dem2, w=5)

tpi = terrain(dem,opt='tpi')
summary(tpi)
plot3D(tpi)
tri = terrain(dem,opt='tri')
summary(tri)
plot3D(tri)
ruf = terrain(dem,opt='roughness')
plot3D(ruf)

#Load vegetation raster layer textfile clipped in ArcMap
#crop <-raster("crop2012utm.txt")
crop <-raster("crop2012utm12.tif")
plot(crop)
plot3D(dem,drape=crop)

# reclassify into 9 groups if needed with 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,Inf,NA)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(crop, rclmat)
plot(rc)
rc
plot3D(dem2,drape=rc)
```

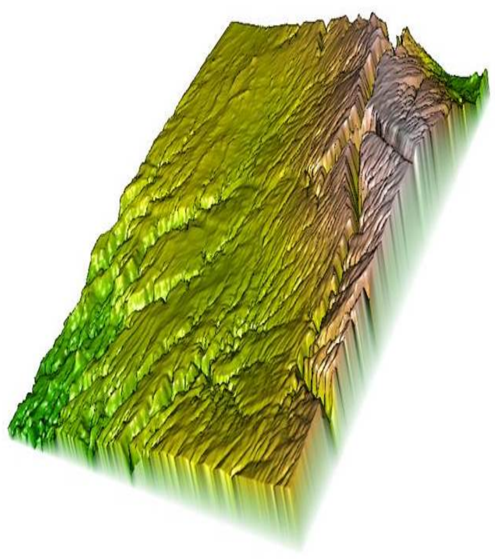


Figure 6.2: Example of a Digital Elevation Model in 3D using rasterVis package in R.

Chapter 7

Landscape Metrics

Contents

Spatial pattern analysis is a large and expanding field that permits the assessment of various landscape metrics to describe a defined region. Methods were originally introduced by Kevin McGarigal and were made available on various platforms through Fragstats (McGarigal and Marks 1995). Various landscape metrics can be estimated in Fragstats that include derivations of measures ranging from patch size and area to edge density and spatial configuration of habitat. At the current time, the only package availabe in R is the *SDMToolsl* package. Although the *SDMTools* package is limited, it provides a great deal of landscape metrics and will be the focus of the code that follows.

7.1 Landscape metrics for a single area

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_7.2- Download and extract zip folder into your preferred location

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

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

   ```
   library(SDMTools)
   library(raster)
   library(plyr)
   library(maptools)
   library(rgdal)
   ```

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

   ```
   #Load vegetation raster layer textfile clipped in ArcMap
   crops <-raster("crop2012utm12.tif")
   plot(crops)
   class(crops)
   ```

```
as.matrix(table(values(crops)))
proj4string(crops)
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)
rc
as.matrix(table(values(rc)))#Look at the resulting vegetation categories

#Now we get into Landscape Metrics with the SDTM Tool
#Calculate the Patch statistics
ps.data = PatchStat(rc)
ps.data

str(ps.data)
#data.frame':   7 obs. of  12 variables:
#$ patchID          : int  1 2 3 4 5 6 7
# $ n.cell           : int  16 7260 370104 258106 42429 1016835 726939
# $ n.core.cell      : int  0 2844 222965 107046 1241 699160 279134
# $ n.edges.perimeter: int  50 7902 206012 261672 98766 406726 662034
# $ n.edges.internal : int  14 21138 1274404 770752 70950 3660614 2245722
# $ area             : num  16 7260 370104 258106 42429 ...
# $ core.area        : num  0 2844 222965 107046 1241 ...
# $ perimeter        : num  50 7902 206012 261672 98766 ...
# $ perim.area.ratio : num  3.125 1.088 0.557 1.014 2.328 ...
# $ shape.index      : num  3.12 23.11 84.64 128.65 119.86 ...
# $ frac.dim.index   : num  1.82 1.71 1.69 1.78 1.9 ...
# $ core.area.index  : num  0 0.3917 0.6024 0.4147 0.0292 ...

#Calculate the Class statistics
cl.data = ClassStat(rc)
cl.data

str(cl.data)
#'data.frame':   7 obs. of  38 variables:

NOTE the difference in the output from function PatchStat and ClassStat.
```

7.2 Landscape 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_7.2- Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   #Clear workspace from first run
   rm(list=ls())

   library(SDMTools)
   library(raster)
   library(rgdal)
   library(maptools)
   library(sp)
   library(adehabitatHR)
   library(rgeos)
   library(plyr)
   ```

4. Now continuing along within the script "Frag_Auto.R", run code below metrics within a single area

   ```
   ### load raster file into R
   raster <- raster("county_hab")
   raster

   ### load PA shapefile into R
   HareCounties <- readOGR(dsn=".", layer="Hare_Counties")
   HareCounties
   plot(HareCounties)
   proj4string(HareCounties)
   proj4string(raster)
   image(raster)
   plot(HareCounties, add=T)

   #Let’s project Counties to habitat just to be safe
   new.crs <-CRS("+proj=lcc +lat_1=41.95 +lat_2=40.88333333333333
        +lat_0=40.16666666666666 + lon_0=-77.75 +x_0=600000 +y_0=0
        +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
   county <- spTransform(HareCounties, CRS=new.crs)
   HareCounties <- county
   proj4string(HareCounties)
   proj4string(raster)
   #Matching projections successful!
   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)
   ```

5. Now we want to export individual counties by County name (i.e., COUNTY_NAM) as individual shapefiles

   ```
   indata <- HareCounties
   innames <- unique(HareCounties@data$COUNTY_NAM)
   innames <- innames[1:2]
   outnames <- innames
   ```

```
# 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){
    writePolyShape(data,fn=paste(outnames[i],sep="/"),factor2char=TRUE)
    write.table(innames, "List.txt", eol=".shp\n", col.names=FALSE, quote=FALSE,
     row.names=FALSE)
}
}
```

6. We then need to read a list of the counties we want to use in our analysis. Multiple text files with shapefile names can be saved if you want to run separately across states or study regions.

```
#Read in a list of shapefiles files from above
Listshps<-read.table("List.txt",sep="\t",header=F)
Listshps
```

7. Now we use the *plyr* package to create a single function that runs multiple functions in a single statement.

```
shape <- function(Listshps) {
file <- as.character(Listshps[1,])
shape <-readShapeSpatial(file)
mask <- mask(raster,shape)
### Calculate the Class statistics in each county
cl.data <- ClassStat(mask)
}
```

8. The line below also uses the *plyr* package to run the newly created function (shape) over the list of shapefiles called in with the List statement (Listshps) by each ID ([.(id)]) in the Listshps

```
results <- ddply(Listshps, .(id), shape)
write.table(results, "FragCounty.txt")
```

7.3 Landscape Metrics within buffers

1. Exercise 7.3 - Download and extract zip folder into your preferred location

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

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

```
library(SDMTools)
library(raster)
library(rgeos)
library(plyr)
```

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

5. Load vegetation raster layer textfile clipped in ArcMap

```
crops <-raster("crop2012utm12.tif")
plot(crops)
```

```
class(crops)
as.matrix(table(values(crops)))
proj4string(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, 172, 7, 180, 183, 8, 189, 191, 9,192,Inf,NA)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(crops, rclmat)
plot(rc)
rc
as.matrix(table(values(rc)))
```

6. The code above looks at patch and class metrics for the entire study area or within large polygons. However, what if we wanted to compare difference in landscape metrics among all deer? The question is how do you we want to go about doing this? We could run metrics within the home range of each animal or within a buffered circle for each location. To begin this, let's return to our mule deer dataset and import the locations, cleanup, and make buffers around each location.

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

#Remove outlier locations
newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.80)
muleys <- newmuleys
newmuleys <-subset(muleys, muleys$Long < -107)
muleys <- newmuleys

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

7. Here we can use code to create a function using the using the *plyr* package that will let us select all location for each deer, create buffers around each deer, and then run landscape metrics within these merged buffers.

```
buff3rd <- function(muleys) {
  coords<-data.frame(x = muleys$X, y = muleys$Y)
  deer.spdf <- SpatialPointsDataFrame(coords=coords, data = muleys,
   proj4string = CRS("+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs
  +datum=GRS80 +towgs84=0,0,0"))
  settbuff <- gBuffer(deer.spdf, width=1000, byid=FALSE)
  buffclip <- mask(rc, settbuff)
  buff.data <- PatchStat(buffclip)
  newline <- muleys$id
  bind <-cbind(newline[1], buff.data)
}

results <- dlply(muleys, .(id), buff3rd)
results
class(results)

#Now convert results of class "List" to class "data frame"
```

```
df <- do.call(rbind.data.frame, results)
df

"Export to a table if needed"
write.table(df, "FragCombined.txt")

"Read back into R if needed
data <- read.table("FragCombined.txt", sep="", header=T)
data
```

8. The code above looks at patch and class metrics for each deer by combining all buffers into one polygon for each deer (i.e., to define 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?

```
#First we need to create buffers then re-assign a new ID for each deer and
#each buffer generated. We then can apply the function to our dataset.
coords<-data.frame(x = muleys$X, y = muleys$Y)
deer.spdf <- SpatialPointsDataFrame(coords=coords, data = muleys,
     proj4string = CRS("+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs
    +datum=GRS80 +towgs84=0,0,0"))
setbuff <- gBuffer(deer.spdf, width=1000, byid=TRUE)
setbuff
muleys$newID <- paste(muleys$id, setbuff@plotOrder, sep="_")

buff3rdA <- function(muleys) {
  bufclip <- mask(rc, setbuff)
  buf.data <- PatchStat(bufclip)
}

results2 <- ddply(muleys, .(newID), buff3rdA)
results2
```

Chapter 8

Resource Selection

Contents

Figures

Resource selection is used here in reference to a suite of methods to determine resource or habitat use by an animal that can be measured in a heirarchical fashion (i.e., First order through Fourth Order; Johnson 1980). The appropriate method to use is determined by the data that was collected during the study that is often controlled by logistics, funding, and population size of the species studied. As methods to identify animal use of the landscape (i.e., GPS technology) and measures of resource or habitats (i.e., GIS layers) have improved, the methods to analyze data has evolved as well (Cooper and Millspaugh 2001, Manly et al. 2002). Several commonly used measures of habitat use or resource selection include:

1. Compositional Analysis (Aebischer et al. 1993)
2. Mahalanobis Distance (Clark et al. 1993)
3. Selection Ratios (Manly et al. 2002)
4. Resource Selection Functions (Cooper and Millspaugh 2001)

Before we begin with resource selection, we first need to prepare the data that we are interested in incorporating into our modeling efforts. Whether these are linear measures (e.g., distance to roads) or landscape/topographic characteristics (e.g., elevation, slope), summarizing variables as *used* or *available* can be completed all in R. Some of these methods were presented in Chapter 1 so we will build on that before estimating RSF/RSPF in later sections of this chapter.

8.1 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 File - Change dir...

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

   ```
   library(adehabitatHR)
   ```

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

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

   newmuleys <-subset(muleys, muleys$Long > -110.90 & muleys$Lat > 37.8)
   muleys <- newmuleys
   newmuleys <-subset(muleys, muleys$Long < -107)
   muleys <- newmuleys
   ```

5. 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
   plot(merge)
   str(merge)
   ```

6. We are now ready to create MCPs for our new dataset "merge" by individual animal ID (Fig. 8.1).

```
## estimates the MCP
cp <- mcp(merge[,1], percent=95)#(95% is the default)
## The home-range size
as.data.frame(cp)
## Plot the home ranges
plot(cp)
plot(cp[2,])#only plot deer D8

## ... And the relocations
plot(merge, col=as.data.frame(merge)[,1], add=TRUE)
```

7. We can export the MCPs as shapefiles if needed for use in GIS using the maptools library

```
library(maptools)
writePolyShape(cp, "MCPhomerange")
```

8. We can also look at the area of each MCP as we exclude percentages of relocations (i.e., outliers). We can exclude 5% of the most extreme relocations or we can compute the home-range size 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
```

Figure 8.1: Example of 95% estimate of home range for mule deer using Minimum Convex Polygon.

8.2 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

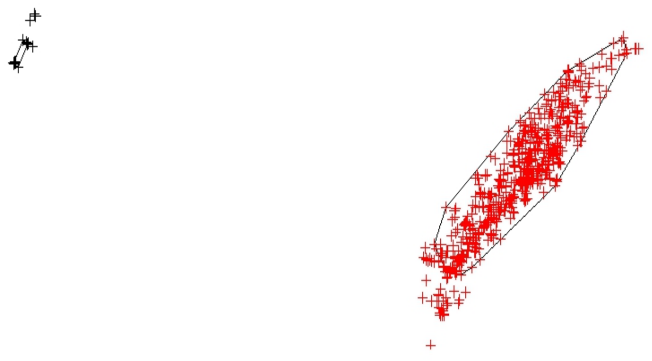


Figure 8.2: Example of 95% estimate of home range for mule deer using Minimum Convex Polygon with relocations overlayed.

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 File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(raster)
   library(sp)
   library(rgdal)
   library(lattice)
   library(rgeos)
   library(spatstat)
   ```

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

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

   #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)
   crs<-"+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80
        +towgs84=0,0,0"

   #Convert to a SpatialPointsDataFrame
   deer.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys,
   ```

```
     proj4string = CRS(crs))
deer.spdf[1:5,]
class(deer.spdf)
proj4string(deer.spdf)
```

5. Load the necessary road and rivers shapefiles already in Albers projection to match previous vegetation raster.

```
roads<-readOGR(dsn=".",layer="AlbersRoads")
rivers<-readOGR(dsn=".",layer="AlbersRivers")
plot(roads,pch=16)
points(deer.spdf, col="red")
plot(rivers,add=T, col="blue",pch=16)
```

6. We need to project the deer.spdf to Albers to match other layers

```
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")
deer.albers <-spTransform(deer.spdf, CRS=Albers.crs)
points(deer.albers, col="red")
class(deer.albers)
proj4string(deer.albers)
deer.spdf[1:5,]
deer.albers[1:5,]
```

7. Determine boundary box around mule deer locations to create a layer to clip and zoom in.

```
bbox(deer.albers)
bb1 <- cbind(x=c(-1106865,-1106865,-1145027,-1145027, -1106865),
     y=c(1695607, 1729463,1729463,1695607,1695607))
AlbersSP <- SpatialPolygons(list(Polygons(list(Polygon(bb1)),"1")),
     proj4string=CRS(proj4string(deer.albers)))
plot(AlbersSP)
```

8. Load vegetation raster layer tif that came in the Albers projection from the online source.

```
crops <-raster("crop2012utm12.tif")

#Check to see all our layers are now in Albers projection
proj4string(crops)
proj4string(deer.albers)
proj4string(AlbersSP)

plot(crops)
points(deer.albers, col="red")
```

9. Clip the raster using the bounding box (AlbersSP) created in step 5.

```
bbclip <- crop(crops, AlbersSP) #Start re-run of code here after new clip#########
cliproads <- gIntersection(roads, AlbersSP, byid=TRUE)
cliprivers <- gIntersection(rivers, AlbersSP, byid=TRUE)

#Plot all to see if it is working for us and zoomed in to mule deer locations.
```

```
plot(bbclip)
points(deer.albers, col="red")
plot(cliproads, add=T)
plot(cliprivers, col="blue",add=T)
```

8.2.1 Formatting layers for package spatstat

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

1. Let's 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)))

#The segments as a planar segment pattern:
segs.psp <- as.psp(segs, window=segs.owin)
plot(segs.psp)
points(deer.albers)
segs.psp
lengths.psp(segs.psp)

#We can cut road segments into lengths we control as well
dist <- pointsOnLines(segs.psp, eps=1000)
```

2. We need to back up a moment to handle the mule deer locations. Need to convert deer.albers from SPDF back to a dataframe because we need xy coordinates to be in Albers. NOTE: If all data is in UTM 12N or a similar projection from the beginning then no need for the next step. Then we need to make mule deer xy coordinates a planar point patter (i.e., *ppp*) for use in package *spatstat.*

```
deer2 <-as.data.frame(deer.albers)
deer2[1:5,]
newdeer <-cbind(deer2$x,deer2$y)
newdeer[1:5,]
```

```
xy.ppp <- as.ppp(newdeer,W=bdy)
plot(xy.ppp)
```

3. Also we need to back up and make the bounding polygon (AlbersSP) a class *owin* in order to proceed with functions in package *spatstat*.

```
AlbersSPDF <- as(AlbersSP, "SpatialPolygonsDataFrame")
#poly <- AlbersSPDF@polygons[[1]]@Polygons[[1]]@coords#These 2 lines won't work
#bdy.gpc <- as(poly, "gpc.poly")                      #anymore due to gpclib issues
bdy.gpc <- as(AlbersSPDF, "gpc.poly")
bdy.owin <- gpc2owin(bdy.gpc)#new code using polyCub package
bdy.owin <- as.owin(AlbersSPDF)#NOTE:If 2 lines of new code does not work use this
                               #line with Spatial Polygons Data Frame
bdy <- as.polygonal(bdy.owin)
xy.ppp <- as.ppp(newdeer,W=bdy)
plot(xy.ppp)

#Let's check to determine if mule deer locations, bounding box, and road layer
#are in the proper classes to proceed.
is.owin(bdy.owin)
#[1] TRUE
is.ppp(xy.ppp)
#[1] TRUE
is.psp(segs.psp)
#[1] TRUE
#All is TRUE so now we can move forward with the analysis
```

4. Now we can determine the distance from mule deer locations (xy.ppp) to the nearest road

```
roaddist <- nncross(xy.ppp, segs.psp)$dist
roaddist[1:5]

#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")
```

5. 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))}

#Again, the function is applied successively to each line in the list then
#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)
#[1] TRUE
#All is TRUE so now we can move forward with the analysis
```

6. Now we can determine the distance from mule deer locations (xy.ppp) to the nearest river.

```
rivdist <- nncross(xy.ppp, rivs.psp)$dist
#rivdist #activate this code to see all distances

#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")

#This code allows us to determine the nearest river to each deer location
plot(xy.ppp[290], add=TRUE, col="blue")
plot(rivs.psp[riv[290]], add=TRUE, lwd=5, col="blue")
```

8.2.2 Summarizing linear measures as covariates

1. 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(0,4000,500)
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

#Or we can place each distance into a category or Bin for each deer.
library(Rcmdr)
BinRoad <- bin.var(roaddist, bins=5, method='intervals',
     labels=c('1','2','3','4','5'))
BinRoad

BinRivers <- bin.var(rivdist, bins=5, method='intervals',
     labels=c('1','2','3','4','5'))
```

```
BinRoad #end re-run here##########

#Let's try to add these distance covariates back to the original muley dataset.
Dist <- cbind(BinRoad,BinRivers)
muleys <- cbind(muleys, Dist)#Will not work because 2 locations are out of the box
```

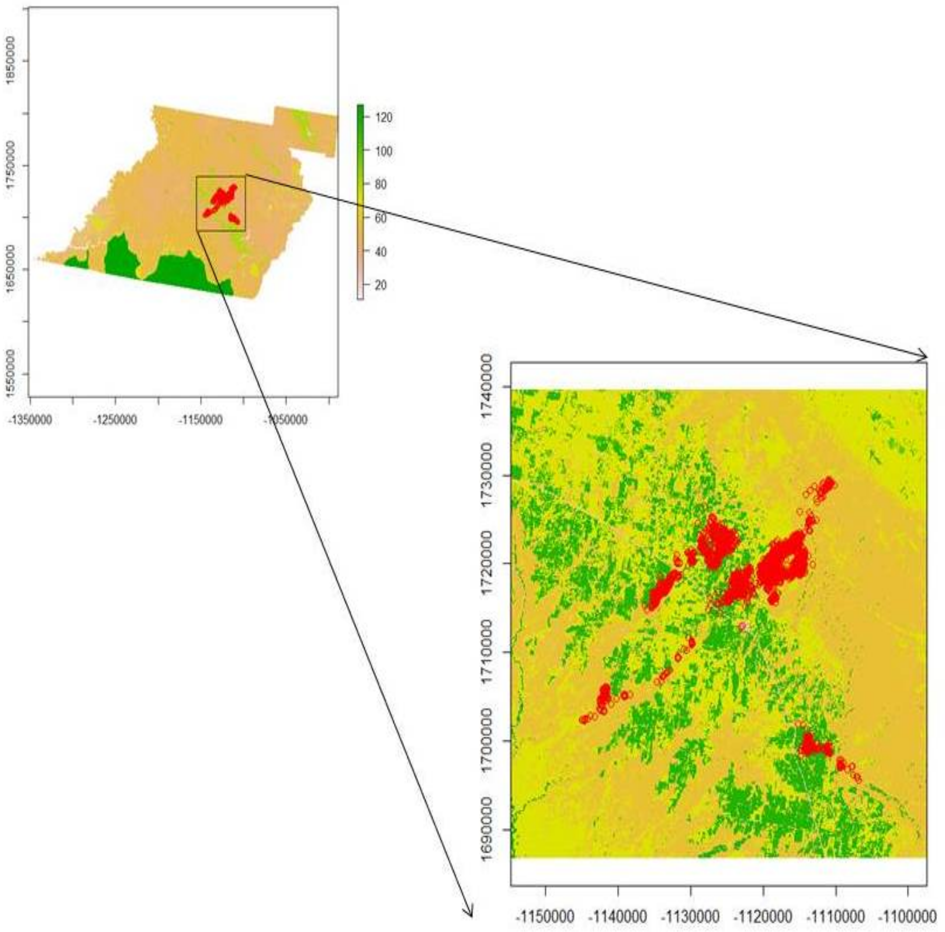


Figure 8.3: Zooming in around mule deer locations using drawExtent in the raster package in R.

8.3 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 Chapter 6 and extract those covariates for use in modeling all within the R environment.

8.3.1 Manipulating raster layers for inclusion in modeling procedures

1. Exercise 8.3 - Download and extract zip folder into your preferred location

2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(sp)
library(lattice)
library(rgdal)#readOGR
library(rgeos)#gIntersection
library(raster)#to import rasters
library(adehabitatHR)
library(maptools)#readAsciiGrid
```

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

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

#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)
utm.crs<-"+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80
     +towgs84=0,0,0"
utm.spdf <- SpatialPointsDataFrame(coords= coords, data = muleys, proj4string =
     CRS(utm.crs))

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")
deer.spdf <-spTransform(utm.spdf, CRS=Albers.crs)
```

5. We can create a bounding box around locations and clip as above or using coordinates of box.

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

#Let's buffer around the bounding box to be sure it encompasses all locations
buffSP <- gBuffer(AlbersSP,width=1000)
plot(buffSP)
points(deer.spdf,col="red")
#NOTE: Be sure that the raster layers extend beyond AlbersSP to avoid errors later.
```

6. 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.

```
crops <-raster("crop2012albers.txt")
plot(crops)
class(crops)
as.matrix(table(values(crops)))
proj4string(crops)
crops

# Reclassify crops raster from above into 9 groups as in previous exercises.
# 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,Inf,NA)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclassify(crops, rclmat)
plot(rc)
rc
as.matrix(table(values(rc)))#Confirms new number of vegetation categories

#Clip using buffSP polygon created earlier to reduce size of raster (if needed).
bbclip <- crop(rc, buffSP)

plot(bbclip)
points(deer.spdf,col="red")
plot(buffSP, add=T)

#Be sure all are in Albers projection before moving forward.
proj4string(bbclip)
proj4string(deer.spdf)
proj4string(buffSP)
```

7. We also want to look at elevation and covariates related to elevation (e.g., slope, aspect) from Section 6.2. These can be created directly in R using the *terrain* function in the *raster* package.

```
elevation <- raster("dem_albers.txt")
image(elevation, col=terrain.colors(10))
contour(elevation, add=TRUE)

#Create slope and aspect
slope = terrain(elevation,opt='slope', unit='degrees')
aspect = terrain(elevation,opt='aspect', unit='degrees')
elevation#NOTE the number of cells for all 3 layers
slope
aspect

#Clip the 3 layers within the buffSP around locations if needed
demclip <- crop(elevation, buffSP)
sloclip <- crop(slope, buffSP)
aspclip <- crop(aspect, buffSP)
```

8. Cast over all 4 layers to a Spatial Grid Data Frame to permit combining into one layer. One very important note here is that the 3 lines of code below may not work on a standard desktop PC because the memory will be maxed out.
Fortunately, we have a Super Computer as we call it that was custom built by Jeff, our

IT guy, with specs as follows:

```
######################################################################
Intel Core i7-3930K Sandy Bridge-E 3.2GHz LGA 2011 Six-Core Processor
G.SKILL Ripjaws Z Series 32GB (4 x 8GB) 240-Pin DDR3 SDRAM
Western Digital WD Black WD1002FAEX 1TB
7200 RPM 64MB Cache SATA 6.0Gb/s 3.5" Internal Hard Drive
######################################################################
nlcd <- as.data.frame(as(rc, "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 the next step of combining layers.
str(elev)
str(slo)
str(asp)

#Combine elevation, slope and aspect into one layer.
layers = cbind(nlcd,elev, asp, slo)
head(layers)
layers = layers[,-c(2,3, 5,6,8,9)]
names(layers) = c("nlcd","elevation""aspect", "slope",, "x", "y")
head(layers)

#Turn aspect into categorical variable is recommended
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)
################################################################################
#
#
# 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: muleys <- muleys[sample(nrow(muleys), 100),]
#
#
################################################################################
```

9. 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 for used and available points based on combined layer data
#set that can take several minutes.
used = grab.values(layers, muleys$X, muleys$Y)
used$x = muleys$X
used$y = muleys$Y
used$animal_id = muleys$id
used$use = 1
used[1:10,]
```

10. 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)
## Plot the home ranges and relocations.
plot(cp)
plot(deer.spdf, col=as.data.frame(deer.spdf)[,2], add=TRUE)

#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', offset=c(0,0)))
plot(cp) ; points(random[[1]], col='red', pch=3, cex=.5)
#The number in the line of code above in double brackets changes 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))

#Nice check of how many points generated per polygon
newpts
# 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))

#Plot random final on MCPs
plot(cp) ; points(random.final, col=random.final$poly_id, pch=3, cex=0.5)
random.final

#Make random.final a data frame to extract raster covariates for each
random.df = as.data.frame(random.final,coords=coords)
names(random.df) = c("ID", "x", "y")

#Grab covariates as we did for mule deer locations above
available = grab.values(layers, random.df$x, random.df$y)
available$x = random.df$x
available$y = random.df$y
available$animal_id = pt_id
available$use = 0
available[1:10,]
```

11. 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)
str(muleys)
#A quick check of the data to determine if correct number of records.
#100 locations used +
#100 locations available (2 animals X 50 random locations)
#= 100 #Confirmed in code below
str(data)
#''data.frame': 200 obs. of  9 variables:
# 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 ...
```

12. 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.

13. First create buffered circles with radius of 500 m

```
settbuff=gBuffer(deer.spdf, width=500,byid=TRUE)

#Then create random sample of points in each polygon
ranbuff <- sapply(slot(settbuff, 'polygons'), function(i) spsample(i, n=5,
     type='random', offset=c(0,0)))
plot(settbuff) ; points(ranbuff[[100]], col='red', pch=3, cex=.5)

#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'))

#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=random.final$poly_id, pch=3, cex=0.5)

buff.final[1:20,]

# make 'buff.final' a data.frame
buffer.df = as.data.frame(buff.final,coords=coords)
names(buffer.df) = c("seqIDs", "x", "y")
buffer.df[1:20,]
str(random.df)
str(buffer.df)
#N = 48115 or 9623 mule deer coordinates X 5 random locations per coordinate

# The first line below required 88 minutes on the Super Computer!!
buff_avail = grab.values(layers, 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
buff_avail[1:10,]
```

```
data2 = rbind(buff_avail, used)
str(data2)

#Before closing, we can save the "used" and available data set to use in
#selection ratio exercise if running full dataset
write.table(used, "MD_used.txt")
write.table(available, "MD_avail.txt")

#Save workspace so all analysis are available
save.image("RSF_dataprep.RData")
```

8.4 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 (Cooper and Millspaugh 2001, Erickson et al. 2001, Leban et al. 2001, Manly et al. 2002).

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

   ```
   library(adehabitatHS)
   ```

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

   ```
   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(-1,-3:-6,-8:-15)]
used <- xtabs(~used$animal_id + used$crop, used)
used <- data.frame(used[1:13, 1:5])
colnames(used) <- c("1","2","3","4","5")

rand <- subset(MDsr, MDsr$use == 0)
rand <- rand[c(-1,-3:-6,-8:-16)]
rand <- xtabs(~rand$animal_id + rand$crop, rand)
rand <- data.frame(rand[1:13, 1:5])
colnames(rand) <- c("1","2","3","4","5")

# PVT Code for VegRSF #
pvt.W <- widesIII(used,rand,avknown = FALSE, alpha = 0.1)
pvt.W
par(mfrow=c(1,2))
plot(pvt.W)

#Five categories
#1 = Sunflower,summer crops, random crops, grassland
#2 = Winter crops
#3 = Alfalfa
#4 = Forest
#5 = Shrubland

#Now run on distance to roads binned into 10 categories
MDsr <- read.csv("MD_winter12.csv",header=T)
#Delete deer that have limited data
MDsr <- subset(MDsr,MDsr$animal_id !="647582A" & MDsr$animal_id !="647584A")
MDsr$animal_id <- factor(MDsr$animal_id)

#Bin roads into 4 categories instead of 10
MDsr$NewRoad <- MDsr$BinRoad
levels(MDsr$NewRoad)<-list(class1=c("0-200","200-400"), class2=c("400-600",
   "600-800"), class3=c("800-1000","1000-12000","1200-1400"),class4=c("1400-1600",
   "1600-1800", "1800-2000"))

used <- subset(MDsr, MDsr$use == 1)
used <- used[c(-1:-6,-8:-15)]
used <- xtabs(~used$animal_id + used$NewRoad, used)
used <- data.frame(used[1:12, 1:4])

rand <- subset(MDsr, MDsr$use == 0)
rand <- rand[c(-1:-6,-8:-15)]
rand <- xtabs(~rand$animal_id + rand$NewRoad, rand)
rand <- data.frame(rand[1:12, 1:4])
colnames(rand) <- c('0-200','200-400','400-600','600-800','800-1000','1000-1200',
   '1200-1400','1400-1600','1600-1800','1800-2000')
```

```
pvt.road <- widesIII(used,rand,avknown = FALSE, alpha = 0.1)
pvt.road
par(mfrow=c(1,2))
plot(pvt.road)
```

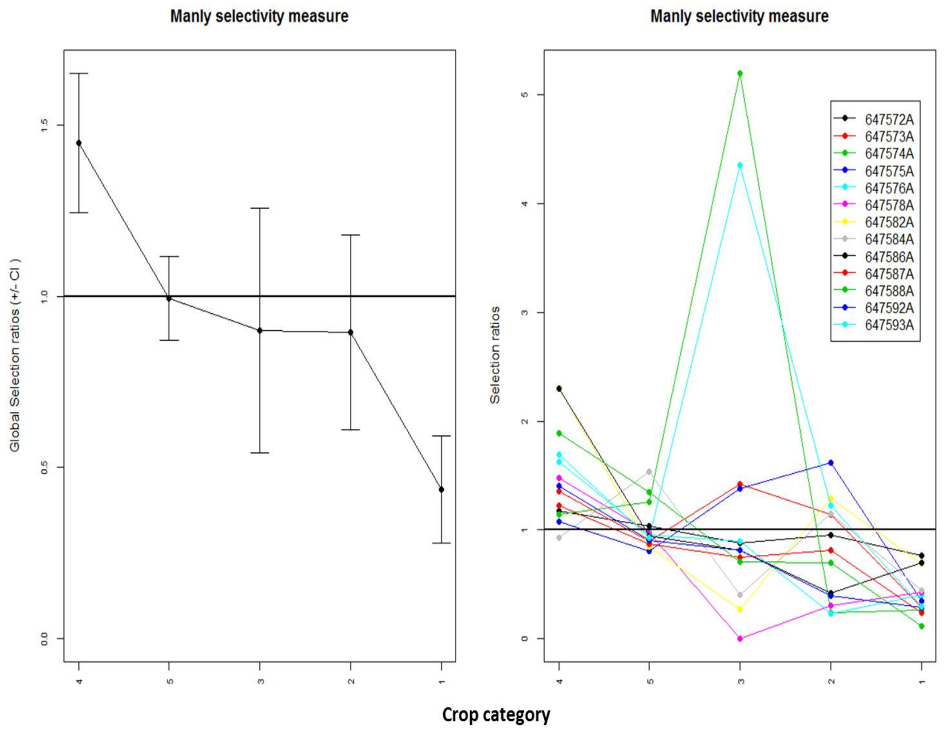


Figure 8.4: Selection ratios for mule deer for 5 habitat types (1-5) using Manly's Selectivity Measure. Habitat type is on x-axis and selectivity measure is on y-axis for a) all deer and b) each deer.

Manly's Selectivity Measure in a Design III would suggest that mule deer in southwestern Colorado are using habitat types 4 and 5 more than they are available (Figure 8.4). Use of the 5 habitat types for each mule deer identifies variability between each mule deer but the trend appears similar across all study animals. Selectivity measure is a general first step look at resource selection but does not really use all the data we have available to use in a Design III study. Although methods of resource selection can be conducted regardless of study design (i.e., I, II, III), the remaining sections of this chapter will focus on Design III because it provides the most detail for each individual animal and should be the goal of most study designs on wildlife for resource selection analysis.

8.5 Resource selection functions

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 funcitons (RSPFs) can be found in the literature (Manly et al. 2002, Johnson et al. 2006, Millspaugh 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 variious models for RSF analysis in R after determining study design, data collection protocol, and methodology to best achieve your research objectives.

8.5.1 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 functions in the *lme4* package is often used for estimating population-level models only. Alternatively, we can assess intra-population variation using the *lmer* function. Assessing intra-population variation in 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 8.4 - Download and extract zip folder into your preferred location

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

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

```
library(lme4)
library(AICcmodavg)
library(adehabitatHR)
```

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

```
data_1 <- read.csv("MD_winter12.csv",header=T)
str(data_1)
```

5. We may need to identify some numerical data as factors or standardize some data prior to implementing resource selection analysis.

```
data_1$crop=as.factor(data_1$crop)
data_1[,2:3]=scale(data_1[,2:3],scale=TRUE)#standardize data to mean of zero
str(data_1)
```

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 sunflowere 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"),nAGQ = 0)#Distance to cover only model
fit3 = glmer(use ~ d_roads+(1|animal_id), data=data_1, family=binomial(link=
   "logit"),    nAGQ = 0)#Distance to roads only model
fit4 = glmer(use ~ d_cover+d_roads+(1|animal_id), data=data_1,
    family=binomial(link="logit"),nAGQ = 0)#Distance to cover and roads model
fit5 = glmer(use ~ 1|animal_id, data=data_1, family=binomial(link="logit"),
   nAGQ = 0)#Intercept model
```

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
fit2
fit3
fit4
fit5

AIC(fit1,fit2,fit3,fit4,fit5)

mynames <- paste("fit", as.character(1:5), sep = "")
myaicc <- aictab(list(fit1,fit2,fit3,fit4,fit5), modnames = mynames)
print(myaicc, LL = FALSE)
```

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.

```
#Get confidence intervals from the top model to interpret results
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)
```

9. 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=",")
str(layer1)
names(layer1) = c("crop", "d_cover", "d_roads","x", "y")
str(layer1)
head(layer1)
#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)
}

str(layer1)
layer1$prediction.class = F.prediction.classes(layer1$predictions, 6)
   #attr should be column number for 'predictions'
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)

############################################################################
#
# We are in the process of adding negative binomial, discrete choice, and
#  synoptic BBMM for resource selection analysis so check back for updates!
#
############################################################################
```

Chapter 9

Spatial Epidemiology in WinBUGS

Contents

Figures

WinBUGS is specialized program that can incorporate spatial variability in a variety of modeling procedures so the general framework of running a model will be described. While there are numerous concepts for spatial models and alternate ways to get models into WinBUGS (e.g., R2WinBUGS), we will go over the basics of running heirarchical Bayesian models in WinBUGS. Although we will not go over the concepts in detail, this short tutorial should enable a novice to load models and data into the WinBUGS environment. All pertinent data can be prepared in any platform but needs to be presented to WinBUGS in the proper format. If not R then Notepad works well for this as the data needs to be presented as comma-separated values to load the data. WinBUGS requires that each section be highlighted or called in order to perform the components. The code to follow will assist in setting up data for use in WinBUGS but an entire book would be needed to explain Bayesian Hierachical Models so we will not cover the theory here. For those interested, we would recommend attending a workshop and reading several books of varying levels of complexity such as Hierarchical Modeling and Analysis for Spatial Data (Bannerjee et al. 2004), Bayesian Disease Mapping (Lawson 2009), and Applied Spatial Data Analysis with R (Bivand et al. 2008).

Sections 9.1 to 9.3 will detail the code necessary to manipulate all necessary location and GIS data within R. These sections will enable the user to load in covariate data, extract data from within a sampling gird, and prepare data to be used in WinBUGS or using R2WinBUGS. Sections 9.4 to 9.11 will detail the process of entering the appropriate data directly into WinBUGS provided the adjacency matrix and data is formatted properly.

9.1 Data preparation in R

1. Exercise 9.1 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

   ```
   library(sp)
   library(lattice)
   library(rgdal)#readOGR
   library(rgeos)#gIntersection
   library(raster)#to use "raster" function
   library(adehabitatHR)
   library(maptools)#readAsciiGrid
   library(zoo)
   ```

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

   ```
   #Load and clean up the location of samples collected during disease surveillance
   #for moose
   snowy <-read.csv("SnowySamples.csv", header=T)
   str(snowy)

   #Clean up by deleting extraneous columns if needed
   snowy <- snowy[c(-20:-38, -40:-64)]
   snowy$Status <- snowy$E_schneide

   #Make a spatial data frame of locations after removing outliers
   coords<-data.frame(x = snowy$X_Coordina, y = snowy$Y_Coordina)
   utm.crs<-"+proj=utm +zone=13 +datum=NAD83 +units=m +no_defs +ellps=GRS80
       +towgs84=0,0,0"
   utm.spdf <- SpatialPointsDataFrame(coords= coords, data = snowy,
       proj4string = CRS(utm.crs))
   ```

5. We now need to load some raster layers of covariates that may be related to disease occurrence

   ```
   #Load DEM raster layer
   dem <-raster("snowydem")
   image(dem)
   class(dem)
   proj4string(dem)

   #Now transform projections all to match DEM (i.e., Albers)
   Albers.crs <-CRS("+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96
        +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
   snowy.spdf <-spTransform(utm.spdf, CRS=Albers.crs)
   ```

6. Now we can create a sampling grid that overlaps our disease locations by getting boundary box information from our locations. We added 3 rows of cells (3610 x 3 = 10830) around our outer most samples to encompass all disease samples and neighboring cells until we can figure out how to expand grid polygons in a simpler way. Alternatively, simply use the coordinates from the boundary box (bbox code) of your locations to create your sampling grid.

```
sublette.df <- as.data.frame(sublette.spdf)
str(sublette.df)
minx <- (min(sublette.df$x)-10830)
maxx <- (max(sublette.df$x)+10830)
miny <- (min(sublette.df$y)-10830)
maxy <- (max(sublette.df$y)+10830)

## create vectors of the x and y points
x <- seq(from = minx, to = maxx, by = 3610)
y <- seq(from = miny, to = maxy, by = 3610)

#Alternate bbox code for spatial points
#       min       max
#x -854784.4 -724665.0
#y  156859.0  247343.2

## create vectors of the x and y points
#x <- seq(from = -854784.4, to = -724665.0, by = 3610)
#y <- seq(from = 156859.0, to = 247343.2, by = 3610)

## create a grid of all pairs of coordinates (as a data.frame)
xy <- expand.grid(x = x, y = y)
class(xy)
str(xy)

#Identifiy projection before creating Spatial Points Data Frame
Albers.crs2 <-"+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96
    +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"

#NOTE: Albers.crs2 is needed because SPDF needs different projection command
#than spTransform above

grid.pts<-SpatialPointsDataFrame(coords= xy, data=xy,
     proj4string = CRS(Albers.crs2))
proj4string(grid.pts)
plot(grid.pts)
gridded(grid.pts)
class(grid.pts)

#Need to define points as a grid to convert to Spatial Polygons below
gridded(grid.pts) <- TRUE
gridded(grid.pts)
str(grid.pts)
plot(grid.pts)
```

```
#Convert grid points to Spatial Polygons in essence converting to a shapefile
gridsp <- as(grid.pts, "SpatialPolygons")
str(gridsp)
plot(gridsp)
class(gridsp)
summary(gridsp)
```

7. Now convert gridpts to Spatial Polygons Data Frame for added flexibility in manipulating layer

```
grid <- SpatialPolygonsDataFrame(gridsp, data=data.frame(id=row.names(gridsp),
     row.names=row.names(gridsp)))
class(grid)
plot(grid)
names.grd<-sapply(grid@polygons, function(x) slot(x,"ID"))
text(coordinates(grid), labels=sapply(slot(grid, "polygons"),
     function(i) slot(i, "ID")), cex=0.3)

#Let's check to see if all grid cells are the same size?
summary(grid)
getSlots(class(grid))
class(slot(grid, "polygons")[[1]])
getSlots(class(slot(grid, "polygons")[[1]]))

#Check area of each cell in the sampling grid in square meters
sapply(slot(grid, "polygons"), function(x) slot(x,"area"))
#[1] 13032100

#Grid cell size converted strto square kilometers
13032100/1000000
#[1] 13.0321 is grid cell size in square kilometers
```

9.2 Raster manipulation in R

8. Continue with same script loaded after finishing Exercise 9.1

9. We loaded moose sample locations and created our sampling grid above so now it is time to work with covariate data that was provided in various forms. We imported DEM in the last exercise because we needed to determine the projection information early on to prepare our grid. It is easier to project moose data to fit a raster projection than vice versa so now let's continue adding additional covariates from raster data.

```
#The layer below is a mule deer HSI raster layer without disturbance from based
#on data from Sawyer et al. 2009 incorporated into a layer because mule deer are
#considered host for the parasite we are investigating
nodis <-raster("snowynodis")
nodis
plot(nodis)
summary(nodis)

#Need to remove NoData from mule deer HSI layer
nodis[is.na(nodis[])] <- 0
```

10. Using functions from the *raster* package, we can calculate slope and aspect from DEM layer imported above

```
slope = terrain(dem,opt='slope', unit='degrees')
aspect = terrain(dem,opt='aspect', unit='degrees')
dem #Now let's see metadata for each layer
slope
aspect

plot(dem)
plot(grid, add=T)
```

11. We also want to look at Land Cover data for this region and reclassify it into fewer categories for comparison and manipulation

```
nlcdall <- raster("nlcd_snowy")
nlcdall #Look at raster values for the habitat layer
#Values range from 11 to 95

#Or plot to visualize categories in legend
plot(nlcdall)

#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(nlcdall, rclmat)
plot(rc) #Now only 7 categories
rc #Now only 7 categories
class(rc)

#Check to be sure all raster have same extent for Stack creation
compareRaster(dem,slope,aspect,nodis,rc)
#[1] TRUE
```

12. Minimize the size of the data for demonstartion purposes

```
###########################################################################
###########################################################################
##NOTE: Code in this box was simply for demonstration purposes to reduce overall
## time for processing during class. Skip this section of code if using your
## own data and your computer has the appropriate processing capabilities.

#First we will clip out the raster layers by zooming into only a few locations
plot(rc)
plot(grid, add=T)
points(snowy.spdf)
#Code below is used to just zoom in on grid using raster layer
e <- drawExtent()
#click on top left of crop box and bottom right of crop box create zoom
newclip <- crop(rc,e)
plot(newclip)
plot(grid, add=T)
points(snowy.spdf, col="red")
```

```
#Clip locations within extent of raster
samp_clip <- crop(snowy.spdf,newclip)
plot(newclip)
plot(samp_clip, add=T)
grid_clip <- crop(grid, newclip)
plot(grid_clip, add=T)
slope2 <- crop(slope,newclip)
aspect2 <- crop(aspect,newclip)
dem2 <- crop(dem,newclip)
HSI <- crop(nodis, newclip)

#Check to be sure all rasters have same extent for Stack creation
compareRaster(dem2,slope2,aspect2,HSI,newclip)
#[1] TRUE

grid <- grid_clip #rename clipped grid as grid to match code below
rc <- newclip #rename clipped Land Cover as "rc" to match code below
snowy.spdf <- samp_clip

#Create a Stack of all Raster layers
#This will take a long long time if rasters have a large extent
r <- stack(list(dem=dem2, slope=slope2, aspect=aspect2, "mule deer HSI"=HSI,
     nlcd=newclip))

#END Demonstration code
#######################################################################
#######################################################################
```

13. Now we want to combine all raster layers into a multi-layered raster called a "stack" below before proceeding if using your own data. Otherwise skip Line 13 if using demonstration code above and continue on with Line 14

```
#Create a Stack of all Rasters
#This will take a long long time if rasters have a large extent
r <- stack(list(dem=dem, slope=slope, aspect=aspect, "HSI"=nodis, nlcd=rc))

nlayers(r) #Show how many layers are in the "r" stack
plot(r) #Visualize "r"
names(r) #Names of each raster in the "r" stack
```

9.3 Combine data formats in R

14. Continue with same script loaded after finishing Exercise 9.1 and 9.2
15. Here we need to get a mean for each covariate for each 13 km^2 cell of our sampling grid

```
#Extracts all rasters within each sampling grid cell
ext <- extract(r, grid, weights=TRUE, fun = mean)

head(ext) #view the results
```

```
#      dem      slope     aspect   mule.deer.HSI   nlcd
#[1,] 2730.461  9.801106 227.6434    0.28312888  3.202686
#[2,] 2661.003 11.862190 120.0121    0.05178489  3.195367
#[3,] 2443.063  1.644629 136.0088    0.01859540  3.989930
#[4,] 2450.666  8.212403 199.1163    0.23681863  3.837849
#[5,] 2570.362  8.625199 212.5139    0.26878506  3.333714
#[6,] 2685.348  5.395405 205.6125    0.22692773  3.086528

############################################################################
#NOTE above that for each grid cell in the sampling grid layer (i.e., grid),
# the "extract" function resulted in means for dem, slope, aspect, and HSI
#for all sampling grid cells but "nlcd" (last column of data)  resulted in mean
# cover categories so need to run nlcd separate with more appropriate code below.
############################################################################
```

16. Mean land cover category is not appropriate here so we need to handle Land Cover layer (i.e., rc) separately

```
#Code below extracts nlcd by land cover category and determines how many
#cells of each type were in each sampling grid.
ovR = extract(rc,grid, byid=TRUE)
head(ovR)

#Summarize results by sampling grid ID
#Land Cover category and number of cells (30x30m raster cells)
tab <- lapply(ovR,table)
tab[[1]]
# 3
#18

tab[[48]]
#   3    4    5    7
#6618   80   95  167

############################################
#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 <- 1:7

##### Loop through and append missing land use categories to each grid cell
ovRnew <- list()
for(i in 1:length(grid)[1] ){
# Land use cats in a given cell
temp1 <- unique(ovR[[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)
ovRnew[[i]] <- c(ovR[[i]], miss)
}

# New summary of land use in a grid cell
tab2 <- lapply(ovRnew,table)
tab2[[1]]
tab[[1]]

# Proportions of all land cover types per grid cell
prop <- list()
for(i in 1:length(grid)[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
#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)
matrix

#Now convert the matrix to a data frame so it is easier to manipulate
dfland <- as.data.frame(matrix)

#Assigning column names to land cover
colnames(dfland) <- c("water","developed","forest","shrub","grass","crop","wetland")
dfland[1:5,]
```

17. Now that we have Land Cover in a similar format as the DEM-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.

```
elecov <- as.data.frame(ext)
head(elecov)
```

18. To bring all of these datasets together for Bayesian hierarchical modeling, we need to assign sampling grid identification numbers to each moose based on where it was harvested. We also need to assign sampling grid cell identification for the covariate summaries so we can join all into one master data set

```
#Plot and look at sampling grid ID
plot(grid)
text(coordinates(grid), labels=sapply(slot(grid, "polygons"), function(i)
     slot(i, "ID")), cex=0.8)

#Now assign actual sampling grid IDs to the covariate summaries in "elecov"
elecov$id <- paste(grid@data$id)#, function(x) slot(x,"ID"))
demdata <- elecov[-c(5)]#rename to cleanup by removing incorrect nlcd column
head(demdata)

#We can now combine dem data with Land Cover data to get our master dataset for
```

```
#each sampling grid cell
data <- cbind(demdata, dfland) # rbind list elements
head(data)
data$GRID <- data$id #This is mainly just to a check on matching grid IDs for later
data[1:10,]
```

19. We also need to identify the sampling grid cell that each moose was located and match the moose up with its respective covariate data based on grid cell ID

```
#First let's assign the grid cell ID to each moose sampled
snowy2 <- over(snowy.spdf,grid)
snowy2[1:5,]
#Now add column to moose demographic data identifying sampling grid cell ID
new <- cbind(snowy.spdf@data,snowy2)
new[1:5,]

#Now we need to "join" the appropriate covariate data to each moose sampled based on
#the sampling grid cell it occurred in (e.g., g953)
data[1:5,]
data <- data[-c(13)]#remove duplicate id column or program throws an error
mydata <- merge(new, data, by=c("id"))
mydata[1:10,]

#Save data to use in exercise later
write.table(mydata,"SnowyData.txt", sep="\t")

#Save workspace so all data is available
save.image("Epi_dataprep.RData")
```

9.4 Data preparation of NDVI covariate

Normalized Difference Vegetation Index (NDVI) is a satellite-derived global vegetation indicator based on vegetation reflectance that proivdes information on vegetation productivity and phenology (Hamel et al. 2009). NDVI data comes in 15-day composite images, values range from 0.0 to 1.0, and data manipulation is required to identify the parameter of interest (e.g., peak timing of high quality vegetation). For our purpose, we will determine maximum increase between successive NDVI sampling periods and the sum of bimonthly values for each month they area available (i.e., May, July, September) similar to previous research (Hamel et al. 2009)

1. Exercise 9.4 - Download and extract zip folder into your preferred location

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

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

```
library(sp)
library(lattice)
library(rgdal)#readOGR
library(rgeos)#gIntersection
library(raster)#to use "raster" function
library(adehabitatHR)
library(maptools)#readAsciiGrid
library(zoo)
```

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

5. We will start by importing individual rasters of our study site for each time period NDVI was available. We will then combine all layers into a raster "stack" for ease of manipulation.

```
ndviMay09 <- raster("snowyMay09.tif")
ndviMay25 <- raster("snowyMay25.tif")
ndviJune10 <- raster("snowyJun10.tif")
ndviJuly12 <- raster("snowyJuly12.tif")
ndviJuly28 <- raster("snowyJuly28.tif")
ndviAug29 <- raster("snowyAug29.tif")
ndviSep14 <- raster("snowySep14.tif")
ndviSep30 <- raster("snowySep30.tif")
proj4string(ndviMay09)

#Create a Stack of all Rasters
r <- stack(list(ndviMay09=ndviMay09, ndviMay25=ndviMay25, ndviJune10=ndviJune10,
    ndviJuly12=ndviJuly12,ndviJuly28=ndviJuly28,ndviAug29=ndviAug29,
    ndviSep14=ndviSep14, ndviSep30=ndviSep30))
nlayers(r)
plot(r)
names(r)
```

6. Then we can get a mean for each NDVI layer for each 13 km2 cell of our sampling grid by extracting all rasters within each sampling grid cell. We will start first by creating our own functions using the raster package.

```
#We will start by creating a function to determine
#maximum increase between successive NDVI periods
#NOTE: x[1] refers to the first layer in your raster stack

maxmay <- function(x){x[1]-x[2]}
maxjune <- function(x){x[2]-x[3]}
maxjuly1 <- function(x){x[3]-x[4]}
maxjuly2 <- function(x){x[4]-x[5]}
maxaug <- function(x){x[5]-x[6]}
maxsept1 <- function(x){x[6]-x[7]}
maxsept2 <- function(x){x[7]-x[8]}


#Now perform the function on each raster in the stack
may <- calc(r,maxmay)
june <- calc(r,maxjune)
july1 <- calc(r,maxjuly1)
july2 <- calc(r,maxjuly2)
aug <- calc(r,maxaug)
sept1 <- calc(r,maxsept1)
sept2 <- calc(r,maxsept2)
```

7. We can plot out each layer in 4 x 3 dimensions if we want to look over what the function created and determine values that resulted from performing the functions on each raster

group.

```
#Set up the figure layout
par(mfcol=c(3,3),mar=c(2,3.5,2,2),oma=c(3,3,3,3)) #Bottom,Left,Top,Right.

plot(may)
# Create a title with a red, bold, italic font
title(main="May", col.main="black", font.main=4)

plot(june)
# Create a title with a red, bold, italic font
title(main="June", col.main="black", font.main=4)

plot(july1)
# Create a title with a red, bold, italic font
title(main="July 1", col.main="black", font.main=4)

plot(july2)
# Create a title with a red, bold, italic font
title(main="July 2", col.main="black", font.main=4)

plot(aug)
# Create a title with a red, bold, italic font
title(main="Aug", col.main="black", font.main=4)

plot(sept1)
# Create a title with a red, bold, italic font
title(main="Sept 1", col.main="black", font.main=4)

plot(sept2)
# Create a title with a red, bold, italic font
title(main="Sept 2", col.main="black", font.main=4)
```

8. Now we can use the raster package to create functions to determine sum of the bimonthly values for May, July, and September

```
#Start by creating a function to sum the bimonthly values for May, July,
#and September
summay <- function(x){x[1]+x[2]}
sumjuly <- function(x){x[4]+x[5]}
sumsept <- function(x){x[7]+x[8]}

#Now perform the function on each month that we have 2 layers of NDVI
maysum <- calc(r,summay)
julysum <- calc(r,sumjuly)
septsum <- calc(r,sumsept)
```

9. Plot out each layer to look over what the function created and determine values that resulted from performing the functions for each month.

```
windows()#opens a new graphic window if needed
par(mfcol=c(2,2))

plot(maysum)
```

```
# Create a title with a red, bold, italic font
title(main="May", col.main="black", font.main=4)
plot(julysum)

# Create a title with a red, bold, italic font
title(main="July", col.main="black", font.main=4)

plot(septsum)
# Create a title with a red, bold, italic font
title(main="Sept", col.main="black", font.main=4)
```

10. Now we need to get a mean for each covariate for each 13 km^2 cell of our sampling grid similar to our DEM layer means

```
#Means for maximum increase
extmay <- extract(may, grid, weights=TRUE, fun = mean)
extjune <- extract(june, grid, weights=TRUE, fun = mean)
extjuly1 <- extract(july1, grid, weights=TRUE, fun = mean)
extjuly2 <- extract(july2, grid, weights=TRUE, fun = mean)
extaug <- extract(aug, grid, weights=TRUE, fun = mean)
extsept1 <- extract(sept1, grid, weights=TRUE, fun = mean)
extsept2 <- extract(sept2, grid, weights=TRUE, fun = mean)

#Means for bimontly sums
extmaysum <- extract(maysum, grid, weights=TRUE, fun = mean)
extjulysum <- extract(julysum, grid, weights=TRUE, fun = mean)
extseptsum <- extract(septsum, grid, weights=TRUE, fun = mean)
```

11. Now convert each matrix to a data frame so it is easier to manipulate then combine into a single dataset

```
mayNDVImax <- as.data.frame(extmay)
juneNDVImax <- as.data.frame(extjune)
jul1NDVImax <- as.data.frame(extjuly1)
jul2NDVImax <- as.data.frame(extjuly2)
augNDVImax <- as.data.frame(extaug)
sep1NDVImax <- as.data.frame(extsept1)
sep2NDVImax <- as.data.frame(extsept2)

#Means for bimontly sums
mayNDVIsum <- as.data.frame(extmaysum)
julyNDVImax <- as.data.frame(extjulysum)
sepNDVImax <- as.data.frame(extseptsum)

#Bind all NDVI layers created above
Snowy_NDVI <- cbind(maySnowymax,juneSnowymax,jul1Snowymax,jul2Snowymax,
   augSnowymax,sep1Snowymax,sep2Snowymax,maySnowysum,julySnowysum,sepSnowysum)
colnames(Snowy_NDVI) <- c("maySnowymax","juneSnowymax","jul1Snowymax",
   "jul2Snowymax","augSnowymax","sep1Snowymax","sep2Snowymax","maySnowysum",
   "julySnowysum","sepSnowysum")
head(Snowy_NDVI)

#Now assign actual sampling grid IDs to the covariate summaries in "Snowyl_NDVI"
```

```
Snowy_NDVI$id <- paste(snowygrid@data$id)
head(Snowy_NDVI)

write.table(Snowy_NDVI,"SnowyNDVI.txt", sep="\t")

#Save workspace so all data is available
save.image("Snowy_NDVI.RData")
```

12. Now let's combine Habitat and DEM data from previous exercise to NDVI data just created.

```
#First let's import the previous NLCD and DEM data we created
snowy2 <- read.csv("SnowyData.txt", sep="\t")
snowy2[1:5,]

#Now we need to "join" the appropriate covariate data to each moose sampled
#based on the sampling grid cell it occurred in (e.g., g953)
Snowy_NDVI[1:5,]
SnowyFinal <- merge(snowy2, Snowy_NDVI, by=c("id"))
SnowyFinal[1:10,]

write.table(SnowyFinal,"SnowyFinal.txt", sep="\t")
#Copy file into R2WinBUGS exercise 9.5
```

9.5 Data preparation for R2WinBUGS

1. Exercise 9.5 - Download and extract zip folder into your preferred location
2. Set working directory to the extracted folder in R under File - Change dir...
3. First we need to load the packages needed for the exercise

```
library(R2WinBUGS)
library(sp)
library(spdep)
library(rgdal)
```

4. Now open the script "SnowyR2winbugs.R" and run code directly from the script
5. Import dataset created previously

```
df <- read.table("SnowyFinal.txt", sep="\t")
head(df)
str(df)
df$IDvar <- substr(df$id, 2,5)
df$IDvar <- as.integer(df$IDvar)

#Clean up and remove missing data for disease status, sex, and age
summary(df$Status)
df <- subset(df, df$Status !="Unknown")
df$Status <- factor(df$Status)
summary(df$Status)

#Recode Sex classes and remove NAs
```

```
summary(df$Sex)
df <- subset(df, df$Sex !="")
df$Sex <- factor(df$Sex)
df$Sex2 <- as.character(df$Sex)
df$Sex2[df$Sex2 == "Male"] <- "1"
df$Sex2[df$Sex2 == "Female"] <- "0"
df$Sex2

#Recode Age classes and remove NAs
summary(df$Age)
df <- subset(df, df$Age !="")
df$Age <- factor(df$Age)
table(df$Age)
#Age classes
# 2-5       3      6+    Adult     Calf Yearling
# 188       1      79        5       36       30

#Combine ages classes
df$NewAge <- df$Age
levels(df$NewAge)<-list(Yearling=c("Calf","Yearling"),Adult=c("2-5","3","6+",
     "Adult"))
summary(df$NewAge)

#Now convert age to numeric with Yearling=0 (baseline) and Adult=1
df$Age2 <- df$NewAge
df$Age2 <- as.character(df$Age2)
df$Age2[df$Age2 == "Adult"] <- "1"
df$Age2[df$Age2 == "Yearling"] <- "0"
df$Age2

df$KillYear <- as.factor(df$KillYear)

summary(df$KillYear)
#2009 2011 2012
  23   15    1
```

6. Now we are going to re-create our spatial grid used in the previous code. Alternatively, we could export the grid as a shapefile and import it here but this is preferable so sampling grid cell IDs match up

```
#Make a spatial data frame of locations after removing outliers
coords<-data.frame(x = df$X_Coordina, y = df$Y_Coordina)
utm.crs<-"+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80
     +towgs84=0,0,0"
utm.spdf <- SpatialPointsDataFrame(coords= coords, data = df, proj4string =
    CRS(utm.crs))

#Now transform projections all to match DEM (i.e., Albers)
Albers.crs <-CRS("+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96 +x_0=0
    +y_0=0 +ellps=GRS80
    +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
df.spdf <-spTransform(utm.spdf, CRS=Albers.crs)
```

```
#NOTE: We added 3 cells around outer most samples to encompass all disease
#samples until can figure out how to expand grid polygons
snowy.df <- as.data.frame(df.spdf)
str(snowy.df)
minx <- (min(snowy.df$x)-10830)
maxx <- (max(snowy.df$x)+10830)
miny <- (min(snowy.df$y)-10830)
maxy <- (max(snowy.df$y)+10830)

## create vectors of the x and y points
x <- seq(from = minx, to = maxx, by = 3610)
y <- seq(from = miny, to = maxy, by = 3610)

## create a grid of all pairs of coordinates (as a data.frame)
xy <- expand.grid(x = x, y = y)

#Identifiy projection before creating Spatial Points Data Frame
Albers.crs2 <-"+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-96 +x_0=0
    +y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
#NOTE: Albers.crs2 is needed because SPDF needs different projection command
#than spTransform above
grid.pts<-SpatialPointsDataFrame(coords= xy, data=xy, proj4string =
    CRS(Albers.crs2))
#Need to define points as a grid to convert to Spatial Polygons below
gridded(grid.pts) <- TRUE
#Convert grid points to Spatial Polygons in essence converting to a shapefile
gridsp <- as(grid.pts, "SpatialPolygons")

#Now convert gridpts to Spatial Polygons Data Frame for added flexibility in
#manipulating layer
grid <- SpatialPolygonsDataFrame(gridsp, data=data.frame(id=row.names(gridsp),
      row.names=row.names(gridsp)))
class(grid)
plot(grid)
```

7. Compute adjacency matrix and adj, N, sumNumNeigh required by car.normal function

```
shape_nb <- poly2nb(grid)
NumCells= length(shape_nb)
num=sapply(shape_nb,length)
adj=unlist(shape_nb)
sumNumNeigh=length(unlist(shape_nb))
```

8. Run correlation on covariates to prevent modeling similar covariates

```
rcorr.adjust(df[,c("Status","water","developed","wetland","forest","shrub",
    "grass","crop","HSI","dem","slope","aspect","ndviMay09","ndviMay25",
    "ndviJune10","ndviJuly12", "ndviJuly28","ndviAug29","ndviSep14",
    "ndviSep30"")], type="pearson")
```

9. Prepare values for model inputs

```
Result<-df$Status2
```

```
Grid_ID<-df$IDvar
Sex<-df$Sex2
Age<-df$Age2
Dem <- df$dem
Slope<-df$slope
Aspect <- df$aspect
HSI <- df$HSI
Wat<-df$water
Dev<-df$developed
For<-df$forest
Shru<-df$shrub
Gras<-df$grass
Crop<-df$crop
Wet<-df$wetland
Maymax <- df$mayBigmax
Junmax <- df$juneBigmax
Jul1max <- df$jul1Bigmax
Jul2max <- df$jul2Bigmax
Augmax <- df$augBigmax
Sep1max <- df$sep1Bigmax
Sep2max <- df$sep2Bigmax
Maysum <- df$mayBigsum
Julsum <- df$julyBigsum
Sepsum <- df$sepBigsum
```

10. Define the model in BUGS language

```
sink("sublettepriors.bug")
cat("

model
{

#Priors for CAR model spatial random effects:

b.car[1:NumCells] ~ car.normal(adj[], weights[], num[], tau.car)
for (k in 1:sumNumNeigh)

{
      weights[k] <- 1
}

for (j in 1:NumCells)
{
epsi[j] ~ dnorm(0,tau.epsi)
}

#Other priors
alpha ~ dflat()
beta1 ~ dnorm(0,1.0E-5)
beta2 ~ dnorm(0,1.0E-5)
beta3 ~ dnorm(0,1.0E-5)
beta4 ~ dnorm(0,1.0E-5)
beta5 ~ dnorm(0,1.0E-5)
beta6 ~ dnorm(0,1.0E-5)
beta7 ~ dnorm(0,1.0E-5)
beta8 ~ dnorm(0,1.0E-5)
beta9 ~ dnorm(0,1.0E-5)
tau.car ~ dgamma(1.0,1.0)
tau.epsi ~ dgamma(17.0393,4.1279)

sd.car<-sd(b.car[])
sd.epsi<-sd(epsi[])
lambda <- sd.car/(sd.car+sd.epsi)

for (i in 1 : 339)
{
Result[i] ~ dbern(pi[i])
logit(pi[i]) <- mu[i]
mu[i] <- alpha + beta1*Sex[i] + beta2*Age[i] + beta3*Dem[i] + beta4*Slope[i]
     + beta5*Aspect[i] + beta6*HSI[i] + beta7*Dev[i] + beta8*For[i]
     + beta9*Gras[i] + b.car[Grid_ID[i]] + epsi[Grid_ID[i]]
}
} # end model
", fill=TRUE)
sink()

# Bundle data
bugs.data <- list(Result=Result, Grid_ID=Grid_ID, NumCells=NumCells,
     sumNumNeigh=sumNumNeigh, num=num, adj=adj, Sex=Sex, Age=Age,
     Dem=Dem, Slope=Slope, Aspect=Aspect, HSI=HSI, Dev=Dev, For=For, Gras=Gras)
```

11. Load initial values

```
inits1<- list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0,
     beta6 = 0, beta7 = 0, beta8 = 0, beta9 = 0)
inits2<- list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0,
     beta6 = 0, beta7 = 0, beta8 = 0, beta9 = 0)
inits3<- list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0,
     beta6 = 0, beta7 = 0, beta8 = 0, beta9 = 0)

inits<- list(inits1, inits2, inits3)

# Paramters to estimate and keep track of
parameters <- c("alpha","beta1","beta2", "beta3", "beta4", "beta5", "beta6",
      "beta7", "beta8", "beta9", "lambda")

# MCMC settings
niter <- 150000
nthin <- 20
nburn <- 10000
nchains <- 3
```

12. Locate WinBUGS by setting path below specifically for the computer used.

```
bugs.dir<-"C:\\Program Files\\WinBUGS14"

# Do the MCMC stuff from R
out <- bugs(data = bugs.data, inits = inits, parameters.to.save = parameters,
model.file = "sublettepriors.bug", n.chains = nchains, n.thin=nthin, n.iter=niter,
n.burnin=nburn, debug=TRUE, bugs.directory=bugs.dir)


print(out, 3)
```

9.6 Data preparation within ArcMap

This section does not have an exercise becasuse we are unable to provide this data due to an agreement with collaborators on this project. This is just a small tutorial on how to prepare data within ArcMap and some considerations for spatial epidemiology. A previous study on bovine tuberculosis (TB) in the northern lower peninsula of Michigan used multivariate conditional logistic regression in a case-control design (Kaneene et al. 2002). Kaneene et al. (2002) used 18 covariates and deer TB prevalence summarized in 3 x 3 blocks (∼23 km2) in an area surrounding farms that resulted in P-values and Odds Ratios of risk of disease. Note that this design does not borrow from strength or knowledge of data from adjacent areas.

An alternative way is to link the disease status (positive or negative) of each farm sampled to some landscape-level predictors. This is a multi-step process that can be done in WinBUGS with data prepared in R or ArcMap depending on your level of experience or comfort with either program. There are 3 major considerations to approaching spatial epidemiology that was used in a study on bovine tuberculosis on cattle farms prepared in ArcMap that is the basis for this section (Walter et al. 2014):

1. Spatial Resolution

   First we overlayed a 5 x 5 km grid having a resolution of 25 km^2, which is approximately equal to a quarter township in size. We selected quarter townships as the proper resolution given that township would likely be too coarse a scale and section would be too fine a resolution for model convergence based on previous research with Bayesian hierarchical models (Farnsworth et al. 2006, Walter et al. 2011). This would result in a total of 368 cells covering the Modified Accredited Zone (MAZ; 5 counties) in Michigan and we can then assign the value associated with each landscape-level predictor variable to a farm in our study based on the grid cell that an individual farm was sampled from; thus, all farms sampled from within a particular grid cell were assigned the same value for each landscape-level predictor.

2. Covariates

   It is very important that covariates are based on some *a priori* knowledge of factors contributing to an increase in risk for infection of disease. To simply data dredge and hope some covariates are contained within the top model(s) is wrong and a study should not be designed this way. Researchers designing a study on spatial epidemiology should consider the demographic variables of the host and/or reservoir and well as any environmental or landscape variables that may influence host/reservoir distribution in the landscape. To simply include elevation, slope, and aspect because previous researchers included them is simply incorrect and should be avoided.

3. Distribution of data

The spatial extent of the data across the study area of interest is also of importance due to limitations in computer processors. If the spatial resolution is small and the extent is large and results in >2000 cells across your study regoin, it may take weeks to run models or models may not run at all. The spatial extent of the data that would be suitable to achieve objectives of the study should be determined prior to initiating studies using Bayesian hierarchical modeling in WinBUGS.

9.6.1 Adjacency matrices with weights = 1

Spatial resolution can be handled and incorporated into modeling efforts using Intrinsic Gaussian Conditional Autoregressive Models (ICAR) in WinBUGS using:

```
car.normal(adj[], weights[], num[], tau)
```

where:

Adjacency - a vector listing the ID numbers of the adjacent areas for each area (this can be generated using the Adjacency Tool in R or ArcMap)

Number - A vector of length N (the total number of areas) giving the number of neighbors for each area

Weights - A vector the same length as adj[] giving unnormalized weights associated with each pair of areas

Thus, the random effect of the jth grid cell is conditional on the values of its (usually = 8) neighboring cells. Adjacency matrices were created with the Adjacency for WinBUGS Tool that provides a matrix relating one areal unit to a collection of neighboring areal units in text files for use in WinBUGS (Fig. 7.1). In ArcMap, an adjacency matrix can be created by installing a Toolbox created by the USGS that will result in 3 separate textfiles. Results of these textfiles can be used within your program to run models in WinBUGS.

1. Install Adjacency for WinBUGS Tool and follow program page for setup.
2. Create the adjacency matrix in the GUI that will result in 4 text files although we will only need to use first 2 in our models:
   (a) *Adj.txt* identifies each cell by unique ID that is adjacent to cell 1, cell 2, cell 3, etc., in sequential order (NOTE: Cell ID is not in file, only IDs of adjacent cells)

   ```
   2,3,4,40,
   1,3,4,5,8,39,40,44,
   1,2,4,5,8,
   1,2,3,
   2,3,6,7,8,39,43,44,
   5,7,8,9,12,43,44,48,
   5,6,8,9,12,
   ```

   (b) *Num.txt* identifies the number of neighbors for each cell in Adj.txt

   ```
   4
   8
   5
   3
   8
   8
   ```

5

(c) *Raw.txt* is similar to Adj.txt with the exception that the first number refers to the cell ID that the neighboring cells are adjacent to.

(d) *SumNumNeigh.txt* shows the overall numbers of neighbors that will be manually entered into WinBUGS code.

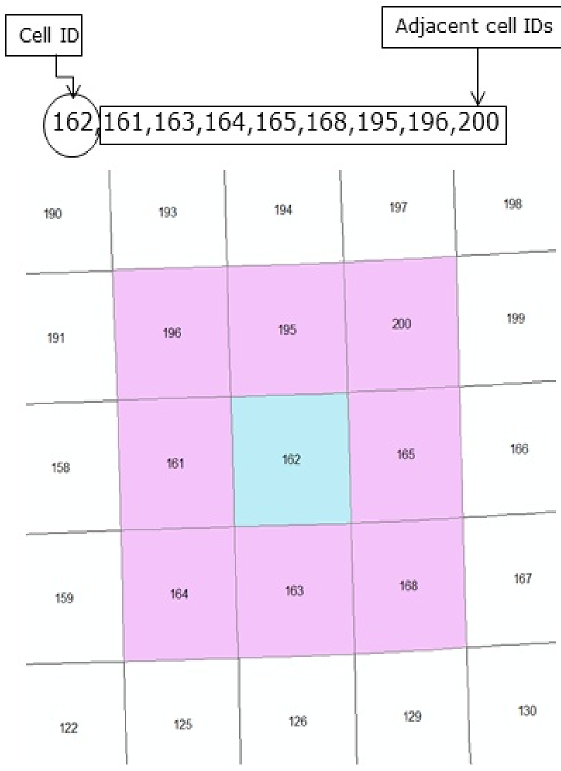


Figure 9.1: Adjacency matrix created in ArcMap using the Adjacency Toolbox.

9.6.2 Adjacency weights other than 1

If we don't want the adjacent 8 cells having equal weight, we can have weights based on neighbours that share common boundaries (Rook) or that share common boundaries and vertices (Queen). There are also distance-based matrices that can incorporate proximity, population densities, or covariates such as age or sex (Earnest et al. 2007) . Spatial resolution other than equal weight in the surrounding 8 cells can be handled and incorporated into modeling efforts using Conditional Autoregressive Models (CAR) in WinBUGS and can be created using the GeoDa program.

Earnest et al. (2007) identifies terms to describe several adjacency matrices that were based on neighborhood or distances that included:

1. *Queen* - neighborhood-based that refers to neighbors that share common boundaries and vertices (n=8 neighbors)
2. *Rook* - neighborhood-based that refers to neighbors that share common boundaries only (n=4 neighbors)
3. *Weights* - distance-based that refers to neighbors at various distances away are less influential

4. *Gravity* - distance-based that refers to neighbors that are more populated have greater influence

5. *Entropy* - distance-based that refers to neighbors that are closer provide more weight than those farther away

6. *Density* - distance-based similar to Gravity except refers to neighbors that have greater density and not just population size so takes into account area

7. *Covariate* - distance-based that identifies *a priori* knowledge of a variable as influential in determining a regions or cells disease rate

9.6.3 Covariates

We can extract covariates within each grid cell for any variable we have *a priori* knowledge that it may influence potential for transmission of TB. For example, the Michigan Department of Agriculture and Rural Development (MDA) provided georeferenced data and herd size (i.e., number of cattle per farm) for all farms in the 5 county area of the Modified Accredited Zone (MAZ) that encompassed about 8,074 km^2 of white-tailed deer habitat. We could have included a herd size effect in all models because these effects have been shown to influence *Mycobacterium bovis* (the bacteria responsible for TB) presence on farms or infection probability for farms in Europe (O'Reilly and Daborn 1995, Hutchings and Harris 1997, Phillips et al. 2003).

The main components of initiating a WinBUGS model section include:

1. Check Model

2. Load Data

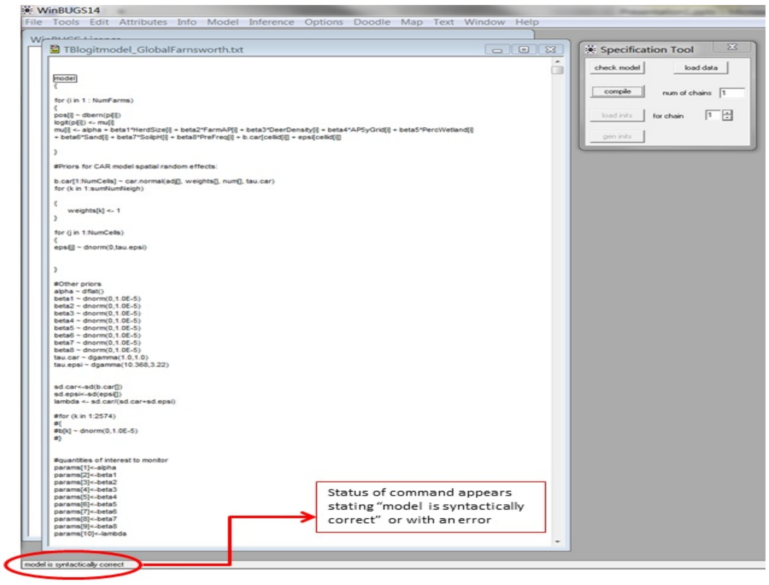
3. Compiling chains

4. Load initial values

9.7 Check model

The Check Model component determines if the model structure is presented properly for the program to run. If the model structure is appropriate, the bottom left corner of the screen will read *model is syntactically correct* (Fig. 9.2).

```
model
{
for (i in 1 : NumFarms)
{
pos[i] ~ dbern(pi[i])
logit(pi[i]) <- mu[i]
mu[i] <- alpha + beta1*DeerDensity[i] + beta2*AP5yGrid[i] + beta3*PercWetland[i]
+ beta4*Sand[i] + beta5*SoilpH[i] + beta6*PreFreq[i] + b.car[cellid[i]] + epsi[cellid[i]]
}
```


WinBUGS14
File Tools Edit Attributes Info Model Inference Options Doodle Map Text Window Help
TBlogitmodel_GlobalFarnsworth.txt
Specification Tool
check model
load data
compile
num of chains 1
load inits
for chain 1
gen inits
Status of command appears stating "model is syntactically correct" or with an error
model is syntactically correct

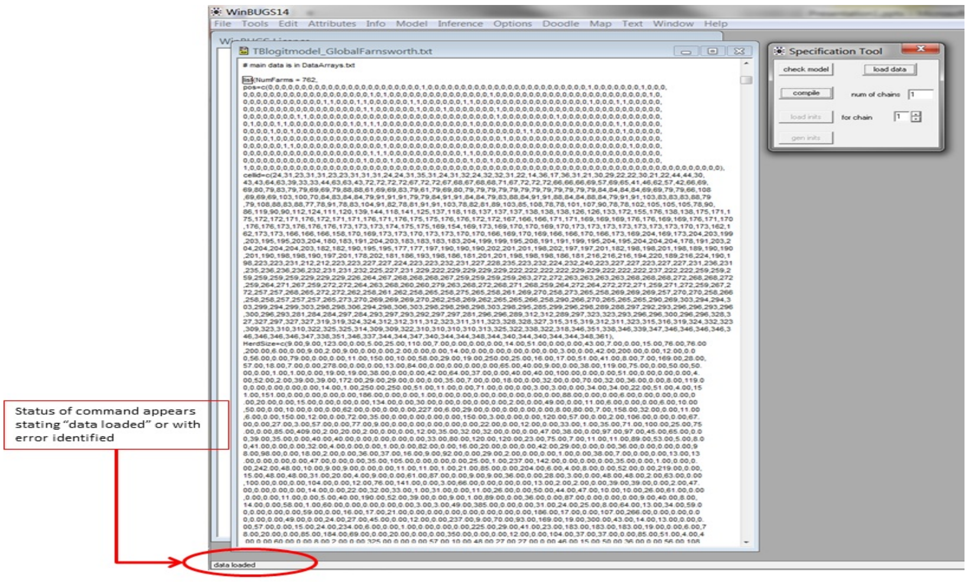
```
model
{

for (i in 1 : NumFarms)
{
pos[i] ~ dbern(pi[i])
logit(pi[i]) <- mu[i]
mu[i] <- alpha + beta1*HerdSize[i] + beta2*FarmAP[i] + beta3*DeerDensity[i] + beta4*AP5yGrid[i] + beta5*PercWetland[i]
+ beta6*Sand[i] + beta7*SoilpH[i] + beta8*PreFreq[i] + b.car[cellid[i]] + epsi[cellid[i]]

}

#Priors for CAR model spatial random effects:

b.car[1:NumCells] ~ car.normal(adj[], weights[], num[], tau.car)
for (k in 1:sumNumNeigh)

{
     weights[k] <- 1
}

for (j in 1:NumCells)
{
epsi[j] ~ dnorm(0,tau.epsi)


}

#Other priors
alpha ~ dflat()
beta1 ~ dnorm(0,1.0E-5)
beta2 ~ dnorm(0,1.0E-5)
beta3 ~ dnorm(0,1.0E-5)
beta4 ~ dnorm(0,1.0E-5)
beta5 ~ dnorm(0,1.0E-5)
beta6 ~ dnorm(0,1.0E-5)
beta7 ~ dnorm(0,1.0E-5)
beta8 ~ dnorm(0,1.0E-5)
tau.car ~ dgamma(1.0,1.0)
tau.epsi ~ dgamma(10.368,3.22)


sd.car<-sd(b.car[])
sd.epsi<-sd(epsi[])
lambda <- sd.car/(sd.car+sd.epsi)

#for (k in 1:2574)
#{
#b[k] ~ dnorm(0,1.0E-5)
#}


#quantities of interest to monitor
params[1]<-alpha
params[2]<-beta1
params[3]<-beta2
params[4]<-beta3
params[5]<-beta4
params[6]<-beta5
params[7]<-beta6
params[8]<-beta7
params[9]<-beta8
params[10]<-lambda
```

Figure 9.2: Compiling the model structure in WinBUGS.

9.8 Load data

The data needs to be separated by column and covariates reflect data for each positive and negative animal sampled. Example code is just an abbreviated version of data but each piece is separated by a comma after parenthesis for each variable. Note that the cell id is also necessary to include in the Load Data section. If the data is loaded successfully, the bottom left corner of the screen will read *data loaded* (Fig. 9.3).

```
list(NumFarms = 762, pos=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
cellid=c(24,31,23,31,31,23,23,31,31,31,24,24,31,35,31,24,31),
AP5yGrid=c(0,0,4.545454545,0,0,4.545454545,4.545454545,0)
```

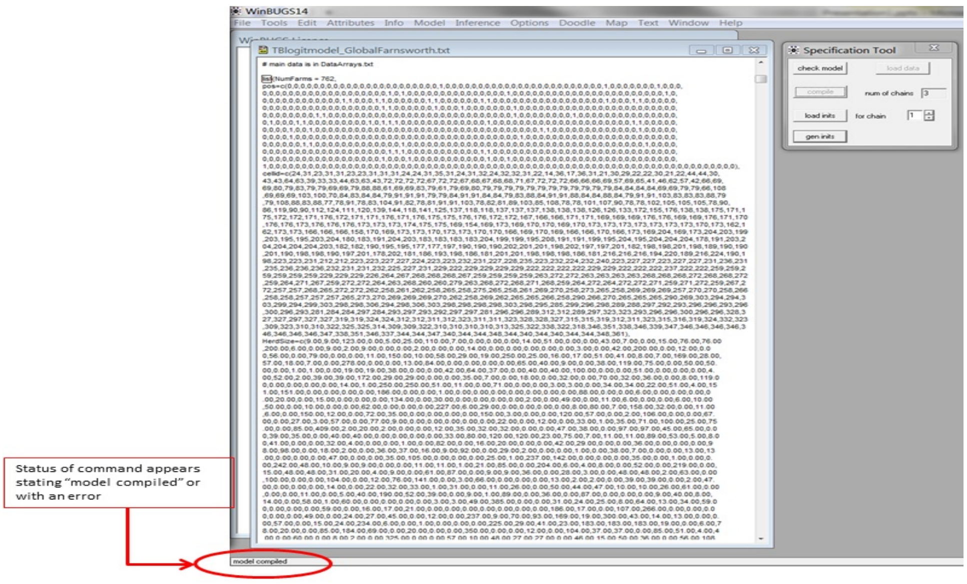

WinBUGS14
File Tools Edit Attributes Info Model Inference Options Doodle Map Text Window Help
TBlogitmodel_GlobalFarnsworth.txt
Specification Tool
check model
load data
compile
num of chains 1
load inits
for chain 1
gen inits
Status of command appears stating "data loaded" or with error identified
data loaded

```
# main data is in DataArrays.txt

list(NumFarms = 762,
pos=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,
0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
cellid=c(24,31,23,31,31,23,23,31,31,31,24,24,31,35,31,24,31,32,24,32,32,31,22,14,36,17,36,31,21,30,29,22,22,30,21,22,44,44,30,
43,43,64,63,39,33,33,44,63,63,43,72,72,72,72,67,72,72,67,68,67,68,68,71,67,72,72,72,66,66,66,69,57,69,65,41,46,62,57,42,66,69,
69,80,79,83,79,79,69,69,79,88,88,61,69,69,83,79,61,79,69,80,79,79,79,79,79,79,79,79,79,79,79,79,84,84,84,84,69,69,79,79,66,108
,69,69,69,103,100,70,84,83,84,84,79,91,91,91,79,79,84,91,91,84,84,79,83,88,84,91,91,88,84,84,88,84,79,91,91,103,83,83,83,88,79
,79,108,88,83,88,77,78,91,78,83,104,91,82,78,81,91,91,103,78,82,81,89,103,85,108,78,78,101,107,90,78,78,102,105,105,105,78,90,
86,119,90,90,112,124,111,120,139,144,118,141,125,137,118,118,137,137,137,138,138,138,126,126,133,172,155,176,138,138,175,171,1
75,172,172,171,176,172,171,171,176,171,176,175,175,176,176,172,172,167,166,166,171,171,169,169,169,176,176,169,169,176,171,170
,176,176,173,176,176,176,173,173,173,174,175,175,169,154,169,173,169,170,170,169,170,173,173,173,173,173,173,173,170,173,162,1
62,173,173,166,166,166,158,170,169,173,173,170,173,173,170,170,166,169,170,169,166,166,170,166,173,169,204,169,173,204,203,199
,203,195,195,203,204,180,183,191,204,203,183,183,183,183,204,199,199,195,208,191,191,199,195,204,195,204,204,204,178,191,203,2
04,204,204,204,203,182,182,190,195,195,177,177,197,190,190,190,202,201,201,198,202,197,197,201,182,198,198,201,198,189,190,190
,201,190,198,198,190,197,201,178,202,181,186,193,198,186,181,201,201,198,198,198,186,181,216,216,216,194,220,189,216,224,190,1
98,223,223,231,212,212,223,223,227,227,224,223,223,232,231,227,228,235,223,232,224,232,240,223,227,227,223,227,227,231,236,231
,235,236,236,236,232,231,231,232,225,227,231,229,222,229,229,229,229,222,222,222,222,229,229,222,222,222,237,222,222,259,259,2
59,259,259,259,229,229,229,226,264,267,268,268,268,267,259,259,259,259,263,272,272,263,263,263,263,268,268,268,272,268,268,272
,259,264,271,267,259,272,272,264,263,268,260,260,279,263,268,272,268,271,268,259,264,272,264,272,272,271,259,271,272,259,267,2
72,257,257,268,265,272,272,262,258,261,262,258,265,258,275,265,258,261,269,270,258,273,265,258,269,269,269,257,270,270,258,266
,258,258,257,257,257,265,273,270,269,269,269,270,262,258,269,262,265,265,266,258,290,266,270,265,265,265,290,269,303,294,294,3
03,299,294,299,303,298,298,306,294,298,306,303,298,298,298,298,303,298,295,285,299,296,298,289,288,297,292,293,296,296,293,296
,300,296,293,281,284,284,297,284,293,297,293,292,297,297,281,296,296,289,312,312,289,297,323,323,293,296,296,300,296,296,328,3
27,327,297,327,327,319,319,324,324,312,312,311,312,323,311,311,323,328,328,327,315,315,319,312,311,323,315,316,319,324,332,323
,309,323,310,310,322,325,325,314,309,309,322,310,310,310,310,313,325,322,338,322,318,346,351,338,346,339,347,346,346,346,346,3
46,346,346,346,347,338,351,346,337,344,344,347,340,344,344,348,344,340,344,340,344,344,348,361),
HerdSize=c(9.00,9.00,123.00,0.00,5.00,25.00,110.00,7.00,0.00,0.00,0.00,14.00,51.00,0.00,0.00,43.00,7.00,0.00,15.00,76.00,76.00
,200.00,6.00,0.00,9.00,2.00,9.00,0.00,0.00,2.00,0.00,0.00,14.00,0.00,0.00,0.00,0.00,0.00,3.00,0.00,42.00,200.00,0.00,12.00,0.0
0,56.00,0.00,79.00,0.00,0.00,11.00,150.00,10.00,58.00,29.00,19.00,250.00,25.00,16.00,17.00,51.00,41.00,8.00,7.00,169.00,28.00,
57.00,18.00,7.00,0.00,278.00,0.00,0.00,13.00,84.00,0.00,0.00,0.00,0.00,65.00,40.00,9.00,0.00,38.00,119.00,75.00,0.00,50.00,50.
00,0.00,1.00,1.00,0.00,19.00,19.00,38.00,0.00,0.00,42.00,64.00,37.00,0.00,40.00,40.00,100.00,0.00,0.00,51.00,0.00,0.00,0.00,4.
00,52.00,2.00,39.00,39.00,172.00,29.00,29.00,0.00,0.00,35.00,7.00,0.00,18.00,0.00,32.00,0.00,70.00,32.00,36.00,0.00,8.00,119.0
0,0.00,0.00,0.00,0.00,14.00,1.00,250.00,250.00,51.00,11.00,0.00,71.00,0.00,0.00,3.00,3.00,0.00,34.00,34.00,22.00,51.00,4.00,15
1.00,151.00,0.00,0.00,0.00,0.00,186.00,0.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,88.00,0.00,0.00,6.00,0.00,0.00,0.00,0
.00,20.00,0.00,15.00,0.00,0.00,0.00,134.00,0.00,30.00,0.00,0.00,0.00,0.00,2.00,0.00,49.00,0.00,11.00,6.00,0.00,0.00,6.00,10.00
,50.00,0.00,10.00,0.00,0.00,62.00,0.00,0.00,0.00,227.00,6.00,29.00,0.00,0.00,0.00,0.00,8.00,80.00,7.00,158.00,32.00,0.00,11.00
,6.00,0.00,150.00,12.00,0.00,72.00,35.00,0.00,0.00,0.00,0.00,150.00,3.00,0.00,0.00,120.00,57.00,0.00,2.00,106.00,0.00,0.00,67.
00,0.00,27.00,3.00,57.00,0.00,77.00,9.00,0.00,0.00,0.00,0.00,0.00,22.00,0.00,12.00,0.00,33.00,1.00,35.00,71.00,100.00,25.00,75
.00,0.00,85.00,409.00,2.00,20.00,2.00,0.00,0.00,12.00,35.00,32.00,32.00,0.00,0.00,47.00,38.00,0.00,97.00,97.00,45.00,65.00,0.0
0,39.00,35.00,0.00,40.00,40.00,0.00,0.00,0.00,0.00,33.00,80.00,120.00,120.00,23.00,75.00,7.00,11.00,11.00,89.00,53.00,5.00,8.0
0,41.00,0.00,0.00,32.00,4.00,0.00,0.00,1.00,0.00,82.00,0.00,16.00,20.00,0.00,0.00,42.00,29.00,0.00,0.00,36.00,0.00,0.00,0.00,9
8.00,98.00,0.00,18.00,2.00,0.00,36.00,37.00,16.00,9.00,92.00,0.00,29.00,2.00,0.00,0.00,1.00,0.00,38.00,7.00,0.00,0.00,13.00,13
.00,0.00,0.00,0.00,47.00,0.00,0.00,35.00,105.00,0.00,0.00,0.00,25.00,1.00,237.00,142.00,0.00,0.00,0.00,35.00,0.00,1.00,0.00,0.
00,242.00,48.00,10.00,9.00,9.00,0.00,0.00,11.00,11.00,1.00,21.00,85.00,0.00,204.00,6.00,4.00,8.00,0.00,52.00,0.00,219.00,0.00,
15.00,48.00,48.00,31.00,20.00,4.00,9.00,0.00,61.00,87.00,0.00,9.00,9.00,36.00,0.00,28.00,3.00,0.00,48.00,48.00,2.00,63.00,0.00
,100.00,0.00,0.00,104.00,0.00,12.00,76.00,141.00,0.00,3.00,66.00,0.00,0.00,0.00,13.00,2.00,2.00,0.00,39.00,39.00,0.00,2.00,47.
00,0.00,0.00,0.00,14.00,0.00,22.00,32.00,33.00,1.00,31.00,0.00,11.00,26.00,0.00,50.00,44.00,47.00,10.00,10.00,26.00,61.00,0.00
,0.00,0.00,11.00,0.00,5.00,40.00,190.00,52.00,39.00,0.00,9.00,1.00,89.00,0.00,36.00,0.00,87.00,0.00,0.00,0.00,9.00,40.00,8.00,
14.00,0.00,58.00,1.00,60.00,0.00,0.00,0.00,0.00,3.00,3.00,49.00,385.00,0.00,0.00,31.00,24.00,25.00,8.00,64.00,13.00,34.00,59.0
0,0.00,0.00,0.00,59.00,0.00,16.00,17.00,21.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,186.00,17.00,0.00,107.00,266.00,0.00,0.00,0.0
0,0.00,0.00,49.00,0.00,24.00,27.00,45.00,0.00,12.00,0.00,237.00,9.00,70.00,93.00,169.00,19.00,300.00,43.00,14.00,13.00,0.00,0.
00,57.00,0.00,15.00,24.00,234.00,6.00,0.00,1.00,0.00,0.00,0.00,225.00,29.00,41.00,23.00,183.00,183.00,183.00,19.00,0.00,6.00,7
8.00,20.00,0.00,85.00,184.00,69.00,0.00,20.00,0.00,0.00,350.00,0.00,0.00,12.00,0.00,104.00,37.00,37.00,0.00,85.00,51.00,4.00,4
.00,0.00,60.00,0.00,8.00,2.00,0.00,325.00,0.00,0.00,57.00,10.00,48.00,27.00,27.00,0.00,46.00,15.00,50.00,36.00,0.00,56.00,108
```

Figure 9.3: Loading the data into WinBUGS.

9.9 Compiling chains

Here you need to load the number of chains you plan to run before selecting the compile button. The number of chains will determine how many chains need to be loaded in the subsequent step. If this step is appropriate, the bottom left corner of the screen will read *model compiled* (Fig. 9.4).



WinBUGS14
File Tools Edit Attributes Info Model Inference Options Doodle Map Text Window Help
TBlogitmodel_GlobalFarnsworth.txt
Specification Tool
check model
load data
compile
num of chains 3
load inits
for chain 1
gen inits
Status of command appears stating "model compiled" or with an error
model compiled

```
# main data is in DataArrays.txt

list(NumFarms = 762,
pos=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,
0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
cellid=c(24,31,23,31,31,23,23,31,31,31,24,24,31,35,31,24,31,32,24,32,32,31,22,14,36,17,36,31,21,30,29,22,22,30,21,22,44,44,30,
43,43,64,63,39,33,33,44,63,63,43,72,72,72,72,67,72,72,67,68,67,68,68,71,67,72,72,72,66,66,66,69,57,69,65,41,46,62,57,42,66,69,
69,80,79,83,79,79,69,69,79,88,88,61,69,69,83,79,61,79,69,80,79,79,79,79,79,79,79,79,79,79,79,79,84,84,84,84,69,69,79,79,66,108
,69,69,69,103,100,70,84,83,84,84,79,91,91,91,79,79,84,91,91,84,84,79,83,88,84,91,91,88,84,84,88,84,79,91,91,103,83,83,83,88,79
,79,108,88,83,88,77,78,91,78,83,104,91,82,78,81,91,91,103,78,82,81,89,103,85,108,78,78,101,107,90,78,78,102,105,105,105,78,90,
86,119,90,90,112,124,111,120,139,144,118,141,125,137,118,118,137,137,137,138,138,138,126,126,133,172,155,176,138,138,175,171,1
75,172,172,171,176,172,171,171,176,171,176,175,175,176,176,172,172,167,166,166,171,171,169,169,169,176,176,169,169,176,171,170
,176,176,173,176,176,176,173,173,173,174,175,175,169,154,169,173,169,170,170,169,170,173,173,173,173,173,173,173,170,173,162,1
62,173,173,166,166,166,158,170,169,173,173,170,173,173,170,170,166,169,170,169,166,166,170,166,173,169,204,169,173,204,203,199
,203,195,195,203,204,180,183,191,204,203,183,183,183,183,204,199,199,195,208,191,191,199,195,204,195,204,204,204,178,191,203,2
04,204,204,204,203,182,182,190,195,195,177,177,197,190,190,190,202,201,201,198,202,197,197,201,182,198,198,201,198,189,190,190
,201,190,198,198,190,197,201,178,202,181,186,193,198,186,181,201,201,198,198,198,186,181,216,216,216,194,220,189,216,224,190,1
98,223,223,231,212,212,223,223,227,227,224,223,223,232,231,227,228,235,223,232,224,232,240,223,227,227,223,227,227,231,236,231
,235,236,236,236,232,231,231,232,225,227,231,229,222,229,229,229,229,222,222,222,222,229,229,222,222,222,237,222,222,259,259,2
59,259,259,259,229,229,229,226,264,267,268,268,268,267,259,259,259,259,263,272,272,263,263,263,263,268,268,268,272,268,268,272
,259,264,271,267,259,272,272,264,263,268,260,260,279,263,268,272,268,271,268,259,264,272,264,272,272,271,259,271,272,259,267,2
72,257,257,268,265,272,272,262,258,261,262,258,265,258,275,265,258,261,269,270,258,273,265,258,269,269,269,257,270,270,258,266
,258,258,257,257,257,265,273,270,269,269,269,270,262,258,269,262,265,265,266,258,290,266,270,265,265,265,290,269,303,294,294,3
[illegible]
HerdSize=c(
[illegible]
```

Figure 9.4: Compiling the number of chains in WinBUGS.

9.10 Load initial values

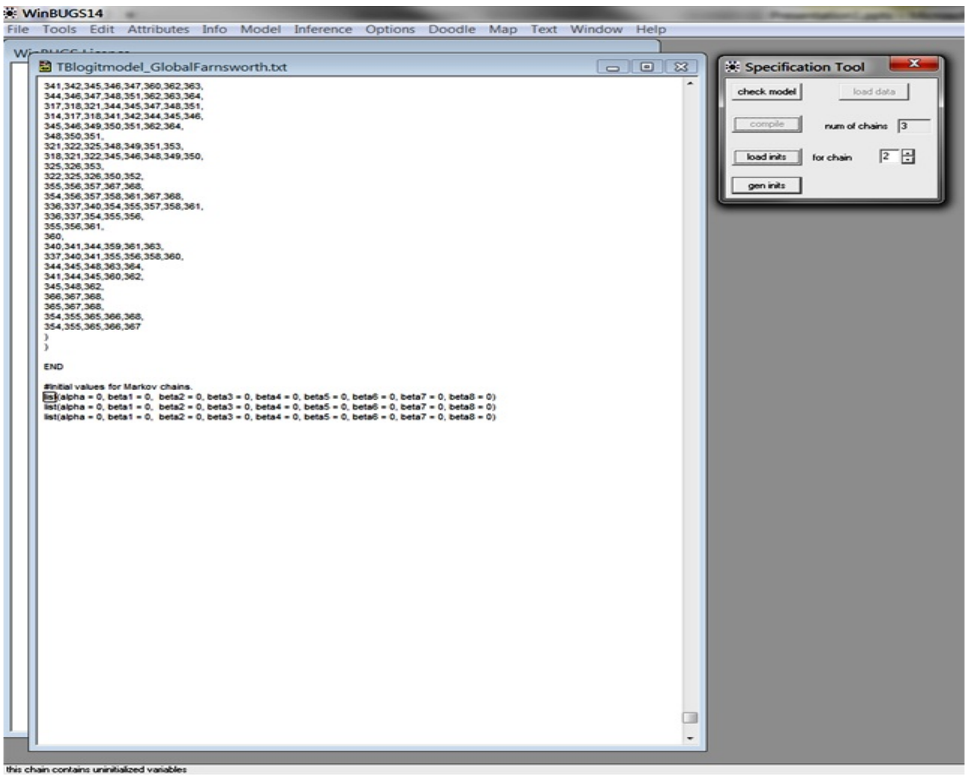
To load initial values after compiling the 3 chains, you need to select drag a box over the term *list* then seelct *load inits*. After doing this for each chain, you may have to select *gen inits* until the bottom left corner of the screen reads *initial values generated, model initialized* (Fig. 9.5).

```
#Initial values for Markov chains.
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0,
     beta7 = 0, beta8 = 0)
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0,
      beta7 = 0, beta8 = 0)
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0,
      beta7 = 0, beta8 = 0)
```

9.11 Sample monitor tool

The *Sample Monitor Tool* allows WinBUGS to store every value it simulates for that parameter. This will enable us to view trace plots of the samples to check convergence and to obtain posterior quantiles for a parameter (Fig. 9.6). Parameters were set in the model statement below:

```
#quantities of interest to monitor
```


WinBUGS14
File Tools Edit Attributes Info Model Inference Options Doodle Map Text Window Help
TBlogitmodel_GlobalFarnsworth.txt
Specification Tool
check model
load data
compile
num of chains 3
load inits
for chain 2
gen inits

```
341,342,345,346,347,360,362,363,
344,346,347,348,351,362,363,364,
317,318,321,344,345,347,348,351,
314,317,318,341,342,344,345,346,
345,346,349,350,351,362,364,
348,350,351,
321,322,325,348,349,351,353,
318,321,322,345,346,348,349,350,
325,326,353,
322,325,326,350,352,
355,356,357,367,368,
354,356,357,358,361,367,368,
336,337,340,354,355,357,358,361,
336,337,354,355,356,
355,356,361,
360,
340,341,344,359,361,363,
337,340,341,355,356,358,360,
344,345,348,363,364,
341,344,345,360,362,
345,348,362,
366,367,368,
365,367,368,
354,355,365,366,368,
354,355,365,366,367
}
}

END

#initial values for Markov chains.
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0, beta7 = 0, beta8 = 0)
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0, beta7 = 0, beta8 = 0)
list(alpha = 0, beta1 = 0,  beta2 = 0, beta3 = 0, beta4 = 0, beta5 = 0, beta6 = 0, beta7 = 0, beta8 = 0)
```

```
this chain contains uninitialized variables
```

Figure 9.5: Loading the initial values for each chain in WinBUGS.

```
params[1]<-alpha
params[2]<-beta1
params[3]<-beta2
params[4]<-beta3
params[5]<-beta4
params[6]<-beta5
params[10]<-lambda
```

9.12 Update tool

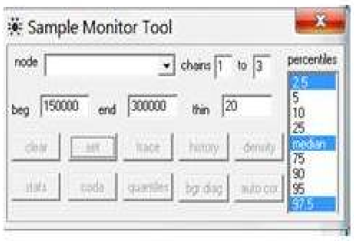
The *Update Tool* under Model in WinBUGS is used to set the number of iterations, refresh to keep you up to date at what iteration the program is on, and the number to thin . This will enable us to view trace plots of the samples to check convergence and to obtain posterior quantiles for a parameter (Fig. 9.6).

9.13 Further considerations

1. *Prior Distributions* - Prior distributions (e.g., non-informative N (0, 100,000)) for each of the parameters, and over the entire real line for μ (e.g., an improper (flat) prior). Prior distributions for the random effect describing region-wide heterogeneity and to describe the spatial structure (e.g., intrinsic Gaussian conditional autoregressive (ICAR) prior with a sum to zero constraint) should also be determined. Because of the marginal specification for region-wide heterogeniety and spatial structure, prior distributions for

Sample Monitor Tool

- -Alpha
- -Tau.car
- -Tau.epsi
- -All beta
- -Deviance

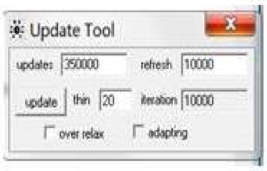

Sample Monitor Tool

node chains 1 to 3 percentiles
beg 150000 end 300000 thin 20

percentiles: 2.5, 5, 10, 25, median, 75, 90, 95, 97.5 (selected: 2.5, median, 97.5)

Update Tool

- -Number of iterations
- -Refresh options
- -Thin


Update Tool

updates 350000 refresh 10000
thin 20 iteration 10000
over relax
adapting

Figure 9.6: Setting the sample monitor tool and initiating program to run in WinBUGS.

the precisions using simulations in WinBUGS should be determined using the *psi* metric (Eberly and Carlin 2000).

2. *Model Selection* - candidate models can consist of different structures with strictly additive effects, environmental predictors can be grouped, such that they can all be entered or removed from the models together. Also, to account for the spatial structure of the data, random effects parameters can be included in some models to represent region-wide heterogeneity and local clustering. Therefore, all models can consist of all possible combinations of the grouped variables, other covariates, and random effects. Deviance information criterion (DIC) can then be used to evaluate this candidate set of models with DIC weights allowing for an intuitive comparison of the evidence in the data for each candidate model. The weights are considered a measure of the strength of evidence in the data for ith model being the "best" model of those within the candidate set, and therefore provide a measure of model selection uncertainty (Burnham and Anderson 2002, Spiegelhalter et al. 2002).

3. *Goodness-of-Fit* - to examine the goodness-of-fit of the top model from candidate sets, a numerical posterior predictive check can be conducted (Gelman et al. 2004). We can use the total number of positive subjects (farm's in our case) conditioned on the observed covariate values in our sample as our test statistic. Generating the posterior distribution of this statistic using parameter estimates from the marginal posterior distribution contained in MCMC chains. Thus given each farm's covariate values, we generated estimates of individual infection probabilities for every location for each of the 250,000 iterations of our MCMC chains, and created a Bernoulli random variable using this probability of M. bovis infection. We then summed these random variables across all farms to create our test statistic. The posterior distribution of the test statistic was created from the values of these test statistics across all iterations of our MCMC chains. Finally, we can calculate the posterior predictive P-value as the probability of having

fewer M. bovis-positive farms then the total number of infected farms observed in the sample based on this posterior distribution of the test statistic.

4. *Convergence and prior sensitivity* - examination of correlation and trace plots, as well as estimates of the corrected scale reduction factor for each parameter and multivariate potential scale reduction factors can provide evidence that chains for each model had converged. Additionally, the posterior distributions can be assessed to determine if they are overly sensitive to prior specification.

Chapter 10

Miscellaneous Code

Contents

10.1 Remove or search for duplicated GPS locations in a data frame

```
#Removes duplicate entries
newgps <- newgps[!duplicated(newgps$DT),]

#Use code to look for NAs which are very bad and can cause code failure
#merge$DT

#Check for duplicates on a variety of data types

duplicated(HexPols2@polygons)#No duplicates here!
duplicated(deer.spdf@data)#No duplicates here!
duplicated(o2)
```

10.2 Need to convert back to a matrix to be able to export the data or manipulate the data

```
#Convert matrix from data.frame to export into csv file
mean <- as.matrix(summary$table)
#Write.table gives csv output of Summary.  Be sure to specify the directory and
#the output files will be stored there

write.table(mean, file = "Distance.csv", sep =",", row.names = TRUE, col.names = TRUE,
     qmethod ="double")
```

10.3 Remove quotations marks around values in results table or printout

```
m1 <-noquote(m1)
```

10.4 Bin numeric variables into categories

```
library(Rcmdr)
BinAlt <- bin.var(pelican$Altitude, bins=10, method='intervals',
     labels=c('1','2','3','4','5','6','7','8','9','10'))
```

10.5 Recode variables in Rcmdr

```
NA = "NA"
0 = "0"
1:200 = "1-200"
201:400 = "201-400"
401:600 = "401-600"
601:800 = "601-800"
801:1000 = "801-1000"
1001:1200 = "1001-1200"
1201:1400 = "1201-1400"
1401:1600 = "1401-1600"
1601:1800 = "1601-1800"
1801:2040 = "1801-2040"
```

10.6 Jitter UTM coordinates before making SpatialPointsDataFrame

```
#Jitter x coordinate before making dataframe
muleys$Xj <- jitter(muleys$X, factor=50, amount=NULL)
muleys$Yj <- jitter(muleys$Y, factor=50, amount=NULL)
coords2<-data.frame(x = muleys$Xj, y = muleys$Yj)
crs<-"+proj=utm +zone=12 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
jitter.spdf <- SpatialPointsDataFrame(coords= coords2, data = muleys,
      proj4string = CRS(crs))
proj4string(jitter.spdf)
points(jitter.spdf, col="red")
```

10.7 Remove extraneous locations or remove all data for a single animal by animal ID

```
#Import original dataset
muleys <-read.csv("DCmuleysedited.csv", header=T)
str(muleys)
#Remove outlier locations and Mule deer D12 with too few locations
muleys <-subset(muleys, muleys$X > 599000 & muleys$X < 705000 & muleys$Y > 4167000
      & muleys$id != "D12")
muleys$id <- factor(muleys$id)#This step must be done to completely remove D12
summary(muleys$id)
```

10.8 Generate sequential numbers as ID's for each location then add back to original dataset

```
seqIDs <-c(1:nrow(muleys))
muleys <- cbind(muleys,seqIDs)
```

10.9 Rename data by deleting a portion of the string

```
#Remove text using substring function
# (i.e., "2004_Resident_1315_Adult" to "Resident_1315_Adult")
awp$code <- substr(awp$res_mig1, 6, 24)

#Concatenate above with "Season" column to make unique categories
awp$season_code <- paste(awp$Season, awp$code, sep="_")

#Results in "Summer_Resident_1315_Adult"
```

10.10 Grab text included in strings of varying lengths

```
#This grabs the last character of a string that has different lengths.
#For example, a vector of animal ID along with sex
#Code courtesy of D. Diefenbach, PA Coop Unit

#First read in a table with string as a factor
locweather <-read.table(file="locweather.txt",header=T, stringsAsFactors=FALSE)
```

```
#Create new variable by pasting last digit of variable regardless of variable length
#(i.e., number of characters)
sex <- as.factor(substr(locweather$indlocalident,nchar(locweather$indlocalident),
nchar(locweather$indlocalident)))

#This will change a vector from this:
#3455F
#21M
#44890F

#Returning a vector 'sex'
#F
#M
#F
```

10.11 Rename levels of factor

```
#Original age classes
# 2-5        3       6+    Adult     Calf Yearling
# 188        1       79        5       36       30

#Recode variables within a column that are factors such as combining age classes
#Combine ages classes
df$NewAge <- df$Age
levels(df$NewAge)<-list(Yearling=c("Calf","Yearling"),Adult=c("2-5","3","6+","Adult"))

summary(df$NewAge)
#Yearling Adult
#  66     273
```

10.12 Recode numeric values as factors into categories

```
#Originally we have multiple numbers of GPS locations that we
#want to recode into categories 1-4 or some other descriptor
#We will give NBLOCS a new name (LocsCat) in our dataframe, make it a
#factor (as.factor) then include breaks that we want to represent each
#category. For example, 0:100 represents category 1 with locations from
#0 to 100 m

merge$LocsCat <-  as.factor(recode(merge$NBLOCS, "0:100='1';101:500='2';
     501:1000='3'; 1001:10000='4'"))
```

10.13 Add leading zeros to single digit numbers to match datasets

```
#Sometimes we may want to "join" data and one may have single digits
#and the other may have leading zeros. We can use the "stringr" package
#to add leading zeros to only the numbers that occur in single digits
```

```
 Label WMU COUNTY
14_D320 4D      7     #note COUNTY is an integer here

library(stingr)
none$COUNTY <- str_pad(none$COUNTY,2, pad = "0")

#Now the resulting data
 Label WMU COUNTY
14_D320 4D     07
```

10.14 Force a DBRB class output to a data frame

```
#Allows us to write diffusion coefficients from movement-based
#home range output to a more useable form
dafr <- do.call(rbind.data.frame, vv)
write.table(dafr, "DiffCoeff2.txt", sep="\t", append=TRUE, col.names=F)
```

10.15 Subset GPS locations by a date range

```
#First make original date field (GPS.Fix.Time) a Date
muleys$Date <- as.Date(muleys$GPS.Fix.Time, "%Y.%m.%d")
#NOTE: The date in GPS.Fix.Time is formatted 2011.12.31 so formats
#must match the date format in line of code above.

locs2012 <- subset(muleys, Date > "2011-12-31" & Date < "2012-12-31")
```

10.16 Drivers for rdgal input/ouput but run command for complete list if needed

```
>getGDALDriverNames()
             name                                         long_name create  copy
          AAIGrid                               Arc/Info ASCII Grid  FALSE  TRUE
              AIG                              Arc/Info Binary Grid  FALSE FALSE
             DOQ1                             USGS DOQ (Old Style)  FALSE FALSE
             DOQ2                             USGS DOQ (New Style)  FALSE FALSE
          E00GRID                        Arc/Info Export E00 GRID  FALSE FALSE
              HFA                       Erdas Imagine Images (.img)   TRUE  TRUE
             NITF              National Imagery Transmission Format   TRUE  TRUE
          USGSDEM                 USGS Optional ASCII DEM (and CDED)  FALSE  TRUE
```

Literature Cited

Aebischer, N. J., P. A. Robertson, and R. E. Kenward (1993). Compositional analysis of habitat use from animal radio-tracking data. *Ecology 74*, 1313–1325.

Amstrup, S. C., T. L. McDonald, and G. M. Durner (2004). Using satellite radiotelemetry data to delineate and manage wildlife populations. *Wildlife Society Bulletin 32*(3), 661–679. Wildl Soc Bull.

Bannerjee, S., B. P. Carlin, and A. E. Gelfand (2004). *Hierarchical modeling and analysis for spatial data.* New York: Chapman and Hall/CRC. New York USA.

Benhamou, S. (2011). Dynamic approach to space and habitat use based on biased random bridges. *PLoS ONE 6*(1), e14592.

Benhamou, S. and D. Cornelis (2010). Incorporating movement behavior and barriers to improve kernel home range space use estimates. *Journal of Wildlife Management 74*(6), 1353–1360. J Wildl Manage.

Bhattacharyya, A. (1943). On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society 35*, 99–109.

Bivand, R. S., E. J. Pebesma, and V. GÂćmez-Rubio (2008). *Applied Spatial Data Analysis with R.* New York: Springer. New York USA.

Bunnefeld, N., L. BâĂİrger, B. Van Moorter, C. M. Rolandsen, H. Dettki, E. J. Solberg, and G. Ericsson (2011). A model-driven approach to quantify migration patterns: individual, regional and yearly differences. *Journal of Animal Ecology 80*(2), 466–476. J Anim Ecol.

Burnham, K. P. and D. R. Anderson (2002). *Model selection and multimodel inference: a practical information-theoretic approach*, Volume 2nd. New York: Springer-Verlag. New York USA.

Calenge, C. (2007). Exploring habitat selection by wildlife with adehabitat. *Journal of Statistical Software 22*(6), 1–18.

Calenge, C. (2011). Package adehabitathr.

Calenge, C. (2012). Packag adehabitatlt.

Clark, J. D., J. E. Dunn, and K. G. Smith (1993). A multivariate model of female black bear habitat use for a geographic information system. *Journal of Wildlife Management 57*(3), 519–526. J Wildl Manage.

Cooper, A. B. and J. J. Millspaugh (2001). *Accounting for variation in resource availability and animal behavior in resource selection studies*, pp. 243 –273. San Diego: Academic Press. California USA.

Downs, J. A. and M. W. Horner (2009). A characteristic-hull based method for home range estimation. *Transactions in GIS 13*(5-6), 527–537.

Duong, T. (2007). ks: kernel density estimation and kernel discriminant analysis for multivariate data in r. *Journal of Statistical Software 21*(7), 1–16.

Duong, T. and M. L. Hazelton (2003). Plug-in bandwidth matrices for bivariate kernel density estimation. *Nonparametric Statistics 15*(1), 17–30.

Earnest, A., G. Morgan, K. Mengersen, L. Ryan, R. Summerhayes, and J. Beard (2007). Evaluating the effect of neighbourhood weight matrices on smoothing properties of conditional autoregressive (car) models. *International Journal of Health Geographics 6*(54).

Eberly, L. E. and B. P. Carlin (2000). Identifiability and convergence issues for markov chain monte carlo fitting of spatial models. *Statistics in Medicine 19*, 2279–2294.

Erickson, W. P., T. L. McDonald, K. G. Gerow, S. Howlin, and J. W. Kern (2001). *Statistical issues in resource selection studies using radio-marked animals*, pp. 209–242. San Diego: Academic Press. California USA.

Farnsworth, M. L., J. A. Hoeting, N. T. Hobbs, and M. W. Miller (2006). Linking chronic wasting disease to mule deer movement scales: a hierarchical bayesian approach. *Ecological Applications 16*(3), 1026–1036. Ecol Appl.

Fauchald, P. and T. Tverra (2003). Using first-passage time in the analysis of area-restricted search and habitat selection. *Ecology 84*(2), 282–288.

Fieberg, J. and C. O. Kochanny (2005). Quantifying home-range overlap: the importance of the utilization distribution. *Journal of Wildlife Management 69*(4), 1346–1359. J Wildl Manage.

Gelman, A., J. B. Carlin, H. S. Stern, and D. B. Rubin (2004). *Bayesian data analysis.* New York: Chapman and Hall/CRC. New York USA.

Getz, W. M., S. Fortmann-Roe, P. C. Cross, A. J. Lyons, S. J. Ryan, and C. C. Wilmers (2007). Locoh: nonparametric kernel methods for constructing home ranges and utilization distributions. *PLoS ONE 2*(2), e207.

Getz, W. M. and C. C. Wilmers (2004). A local nearest-neighbor convex-hull construction of home ranges and utilization distributions. *Ecography 27*(4), 489–505.

Gillies, C. S., M. Hebblewhite, S. E. Nielsen, M. A. Krawchuk, C. L. Aldridge, J. L. Frair, D. J. Saher, C. E. Stevens, and C. L. Jerde (2006). Application of random effects to the study of resource selection by animals. *Journal of Animal Ecology 75*, 887–898. J Anim Ecol.

Girard, I., J. Ouellet, R. Courtois, C. Dussault, and L. Breton (2002). Effects of sampling effort based on gps telemetry on home-range size estimations. *Journal of Wildlife Management 66*(4), 1290–1300. J Wildl Manage.

Gitzen, R. A., J. J. Millspaugh, and B. J. Kernohan (2006). Bandwidth selection for fixed-kernel analysis of animal utilization distributions. *Journal of Wildlife Management 70*(5), 1334–1344. J Wildl Manage.

Hamel, S., M. Garel, M. Festa-Bianchet, J. M. Gaillard, and S. D. Cote (2009). Spring normalized difference vegetation index (ndvi) predicts annual variation in timing of peak faecal crude protein in mountain ungulates. *Journal of Applied Ecology 46*, 582–589. J Appl Ecol.

Horne, J. S., E. O. Garton, S. M. Krone, and J. S. Lewis (2007). Analyzing animal movements using brownian bridges. *Ecology 88*(9), 2354–2363.

Hurlbert, S. H. (1978). The measurement of niche overlap and some relatives. *Ecology*, 67–77.

Hutchings, M. R. and S. Harris (1997). Effects of farm management practices on cattle grazing behaviour and the potential for transmission of bovine tuberculosis from badgers to cattle. *The Veterinary Journal 153*(2), 149–162.

Jenness, J. S. (2004). Calculating landscape surface area from digital elevation models. *Wildlife Society Bulletin 32*(3), 829–839. Wildl Soc Bull.

Johnson, A. R., B. T. Milne, and J. A. Wiens (1992). Diffusion in fractal landscapes: simulations and experimental studies of tenebrionid beetle movements. *Ecology*, 1968–1983.

Johnson, C. J., S. E. Nielsen, E. H. Merrill, T. L. McDonald, and M. S. Boyce (2006). Resource selection functions based on use-availability data: theoretical motivation and evaluation methods. *Journal of Wildlife Management 70*(2), 347–357. J Wildl Manage.

Johnson, D. H. (1980). The comparison of usage and availability measurements for evaluating resource preference. *Ecology 61*, 65–71.

Kaneene, J. B., C. S. Bruning-Fann, L. M. Granger, R. Miller, and B. A. Porter-Spalding (2002). Environmental and farm management factors associated with tuberculosis on cattle farms in northeastern michigan. *Journal of the American Veterinary Medical Association 221*(6), 837–842. Mgt practices to exclude deer protective, deer access to feed a risk factor".

Kernohan, B. J., R. A. Gitzen, and J. J. Millspaugh (2001). *Analysis of animal space use and movements*, pp. 125–166. San Diego: Academic Press. California USA.

Kranstauber, B., R. Kays, S. D. LaPoint, M. Wikelski, and K. Safi (2012). A dynamic brownian bridge movement model to estimate utilization distributions for heterogeneous animal movement. *Journal of Animal Ecology 81*(4), 738–746. J Anim Ecol.

Lawson, A. B. (2009). *Bayesian disease mapping: hierarchical modeling in spatial epidemiology.* Boca Raton: Chapman and Hall/CRC. Florida USA.

Leban, F. A., M. J. Wisdom, E. O. Garton, B. K. Johnson, and J. G. Kie (2001). *Effect of sample size on the performance of resource selection analysis*, pp. 291 –307. San Diego, California: Academic Press. USA.

Loader, C. R. (1999). Bandwidth selection: classical or plug-in? *The Annals of Statistics 27*(2), 415–438.

Manly, B. F. J., L. L. McDonald, and D. L. Thomas (2002). *Resource selection by animals: statistical design and analysis for field studies*, Volume 2nd. Dordrecht: Kluwer Academic Publishers. The Netherlands.

Matusita, K. (1973). Discrimination and the affinity of distributions. *Discriminant Analysis and Applications*, 213–223.

McGarigal, K. and B. J. Marks (1995). Fragstats: spatial pattern analysis program for quantifying landscape structure. IN FILE Oregon USA.

Millspaugh, J. J., G. C. Brundige, R. A. Gitzen, and K. J. Raedeke (2000). Elk and hunter space-use sharing in south dakota. *Journal of Wildlife Management 64*(4), 994–1003. J Wildl Manage Maryland USA.

Millspaugh, J. J., R. M. Nielson, L. McDonald, J. M. Marzluff, R. A. Gitzen, C. D. Rittenhouse, M. W. Hubbard, and S. L. Sheriff (2006). Analysis of resource selection using utilization distributions. *Journal of Wildlife Management 70*(2), 384–395. J Wildl Manage.

Mohr, C. O. (1947). Table of equivalent populations of north american small mammals. *American Midland Naturalist 37*, 223–449. Am Midl Nat.

O'Reilly, L. M. and C. J. Daborn (1995). The epidemiology of mycobacterium bovis infections in animals and man: a review. *Tubercle and Lung Disease 76 Supplement 1*, 1–46. overview of factors that contributed to MI outbreak and eradication efforts. Human behavior a large part of program success: cooperation with wildlife and cattle management efforts.

Ostfeld, R. S. (1986). Territoriality and mating system of california voles. *Journal of Animal Ecology 55*(2), 691–706. J Anim Ecol.

Papworth, S. K., N. Bunnefeld, K. Slocombe, and E. J. Milner-Gulland (2012). Movement ecology of human resource users: using net squared displacement, biased random bridges and resource utilization functions to quantify hunter and gatherer behaviour. *Methods in Ecology and Evolution 3*(3), 584–594.

Pellerin, M., S. Said, and J. M. Gaillard (2008). Roe deer capreolus capreolus home-range sizes estimated from vhf and gps data. *Wildlife Biology 14*(1), 101–110. Wildl Biol.

Phillips, C. J. C., C. R. W. Foster, P. A. Morris, and R. Teverson (2003). The transmission of mycobacterium bovis infection to cattle. *Research in Veterinary Science 74*, 1–15. Comprehensive pathway review with focus on routes to pulmonary infection. UK focus– no attention paid to shared feeds, but other environmental sources considered.

Rees, E. E., E. H. Merrill, T. K. Bollinger, Y. T. Hwang, M. J. Pybus, and D. W. Coltman (2011). Targeting the detection of chronic wasting disease using the hunter harvest during early phases of an outbreak in saskatchewan, canada. *Preventive Veterinary Medicine 104*, 149–159. Prev Vet Med.

Rodgers, A. R. and J. G. Kie (2010). Hrt: Home range tools for arcgis r. Thunder Bay, Ontario, Canada http://flash.lakeheadu.ca/ arodgers/hre/.

Sappington, J. M., K. M. Longshore, and D. B. Thompson (2007). Quantifying landscape ruggedness for animal habitat analysis: a case study using bighorn sheep in the mojave desert. *Journal of Wildlife Management 71*(5), 1419–1426. J Wildl Manage.

Seaman, D. E., J. J. Millspaugh, B. J. Kernohan, G. C. Brundige, K. J. Raedeke, and R. A. Gitzen (1999). Effects of sample size on kernel home range estimates. *Journal of Wildlife Management 63*(2), 739–747. J Wildl Manage.

Seidel, K. D. (1992). *Statistical properties and applications of a new measure of joint space use for wildlife.* Thesis. Seattle, Washington USA.

Spiegelhalter, D. J., N. G. Best, B. P. Carlin, and A. van der Linde (2002). Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society.Series B (Statistical Methodology) 64*(4), 583–639.

Walter, W. D., J. W. Fischer, S. Baruch-Mordo, and K. C. VerCauteren (2011). *What is the proper method to delineate home range of an animal using today's advanced GPS telemetry systems: the initial step*, pp. 249–268. InTech - Open Access Publisher. Croatia.

Walter, W. D., R. Smith, M. Vanderklok, and K. C. VerCauteren (2014). Linking bovine tuberculosis on cattle farms to white-tailed deer and environmental variables using bayesian hierarchical analysis. *PLoS ONE 9*(3), e90925.

Walter, W. D., D. P. Walsh, M. L. Farnsworth, D. L. Winkelman, and M. W. Miller (2011). Soil clay content underlies prion infection odds. *Nature Communications 2*(200), 1–6.

Worton, B. J. (1995). Using monte carlo simulation to evaluate kernel-based home range estimators. *Journal of Wildlife Management 59*(4), 794–800. J Wildl Manage.