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).

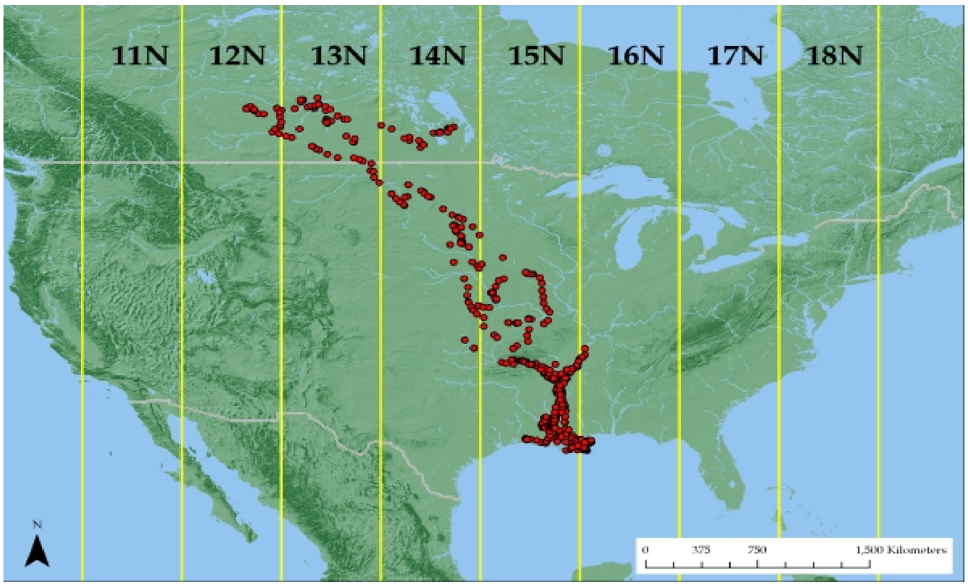


Figure 1.1: American White Pelican with locations spanning 5 UTM zones

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).

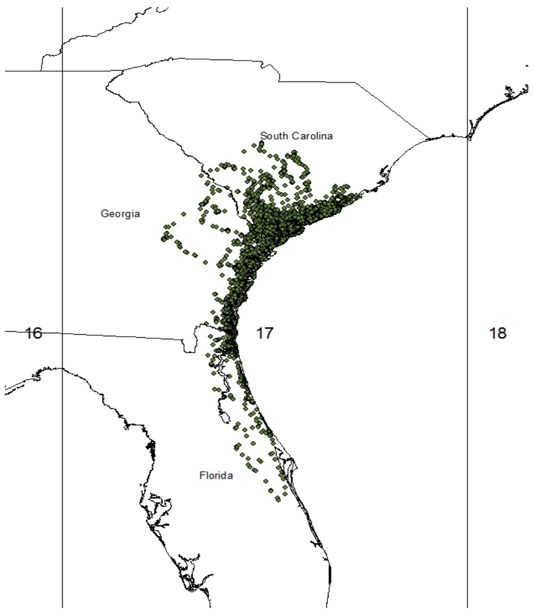


Figure 1.2: Turkey vulture locations only occurring in one UTM zone

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.

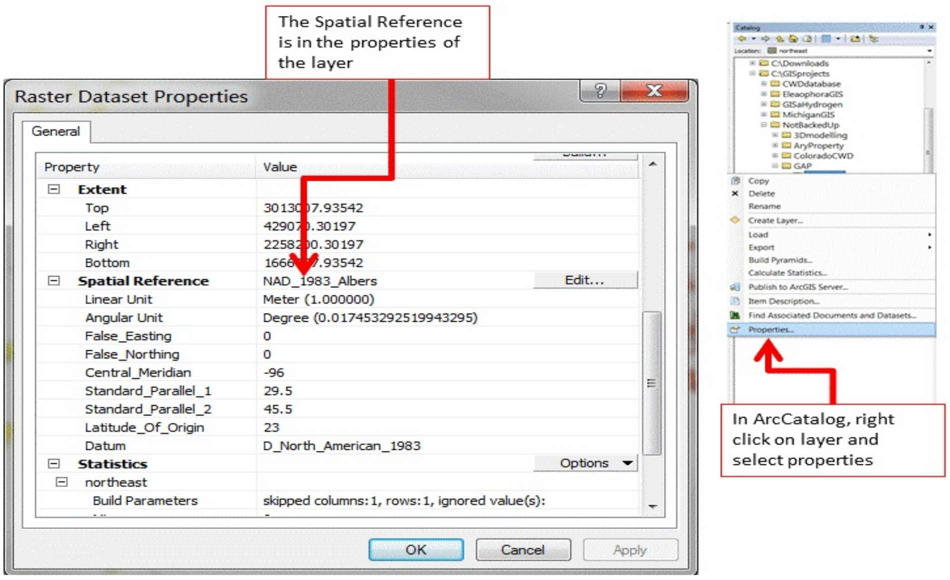
1. What coordiate system were the data collected in?

   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


The Spatial Reference is in the properties of the layer
In ArcCatalog, right click on layer and select properties
Copy
Delete
Rename
Create Layer...
Load
Export
Build Pyramids...
Calculate Statistics...
Publish to ArcGIS Server...
Item Description...
Find Associated Documents and Datasets...
Properties...
OK
Cancel
Apply

Raster Dataset Properties

General

Property	Value
Extent	
Top	3013[illegible]7.93542
Left	42907[illegible].30197
Right	2258[illegible]0.30197
Bottom	1666[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
```

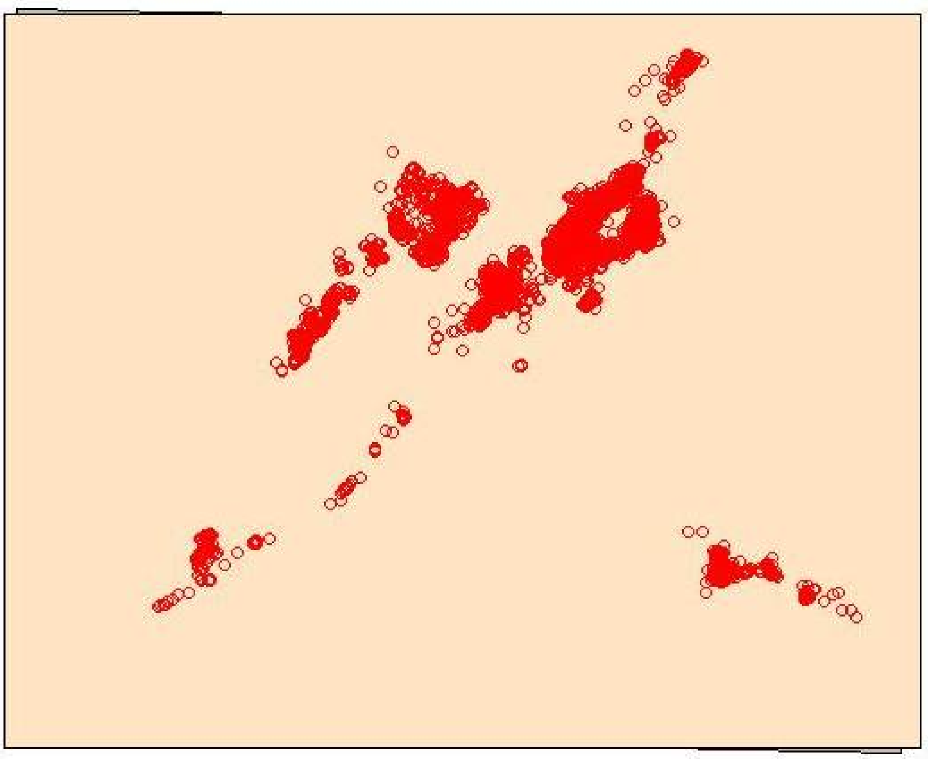


Figure 1.4: Study area projected into UTM Zone 12N with original WGS84 underneath

```
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")
```

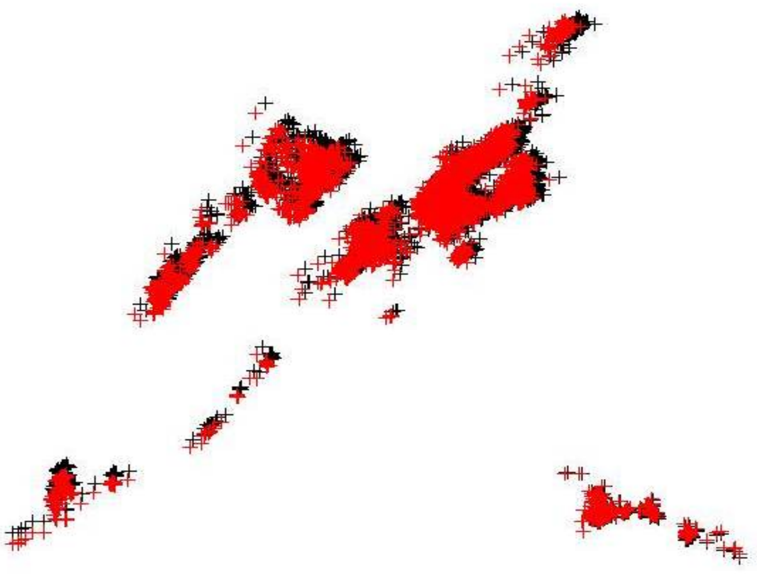


Figure 1.5: Mule deer locations projected into UTM Zone 12N (red) with original locations in WGS84 (black)

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
```

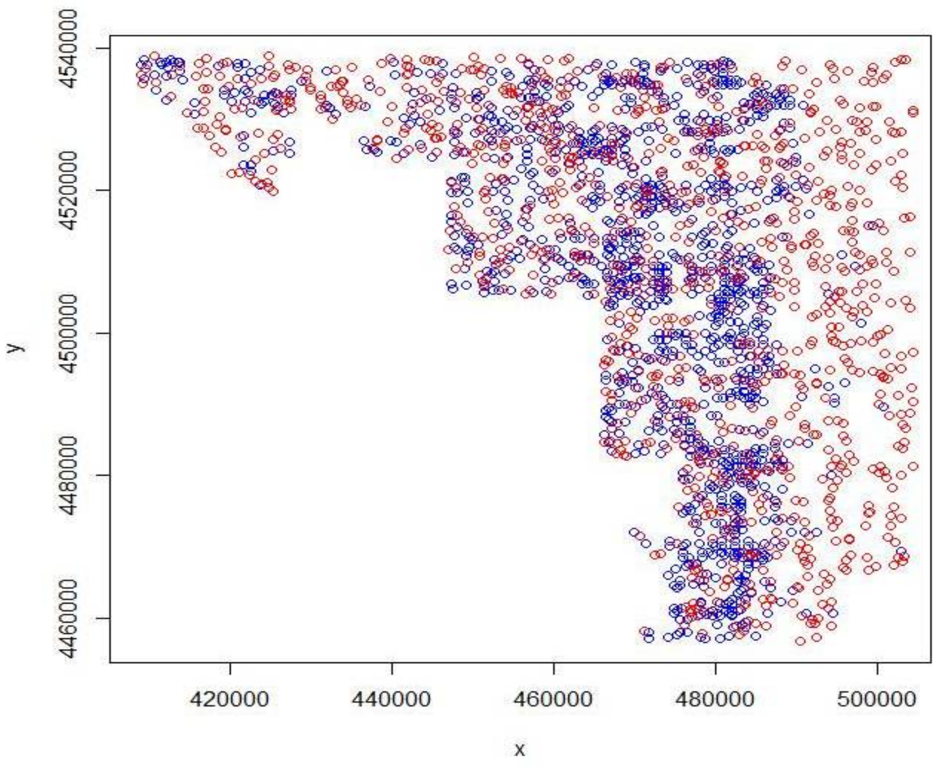


Figure 1.6: Overlay of mule deer locations with random locations generated in R

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
```

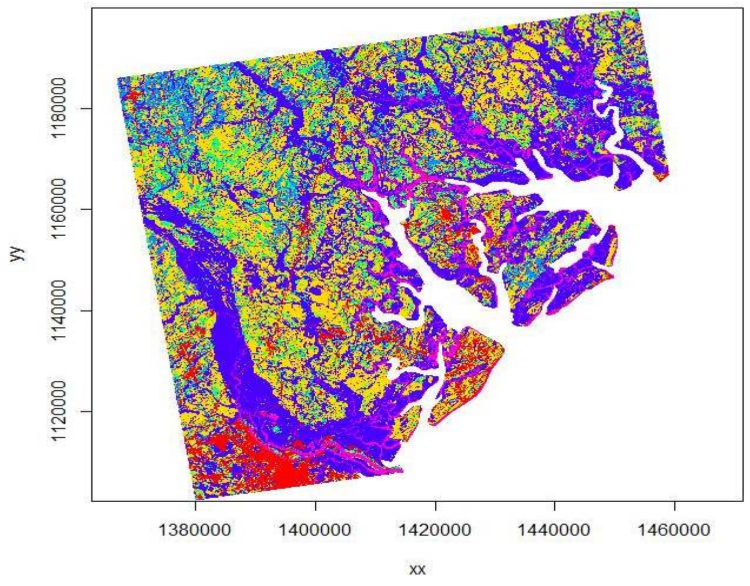


Figure 1.7: Imported raster dataset showing coastline and tributaries.

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))
```

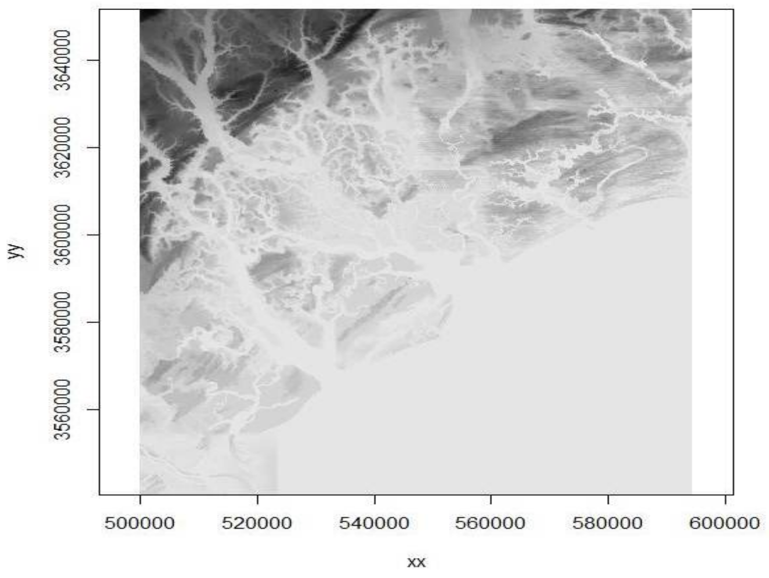


Figure 1.8: Imported Digital Elevation Model using adehabitat package showing coastline and tributaries.

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).

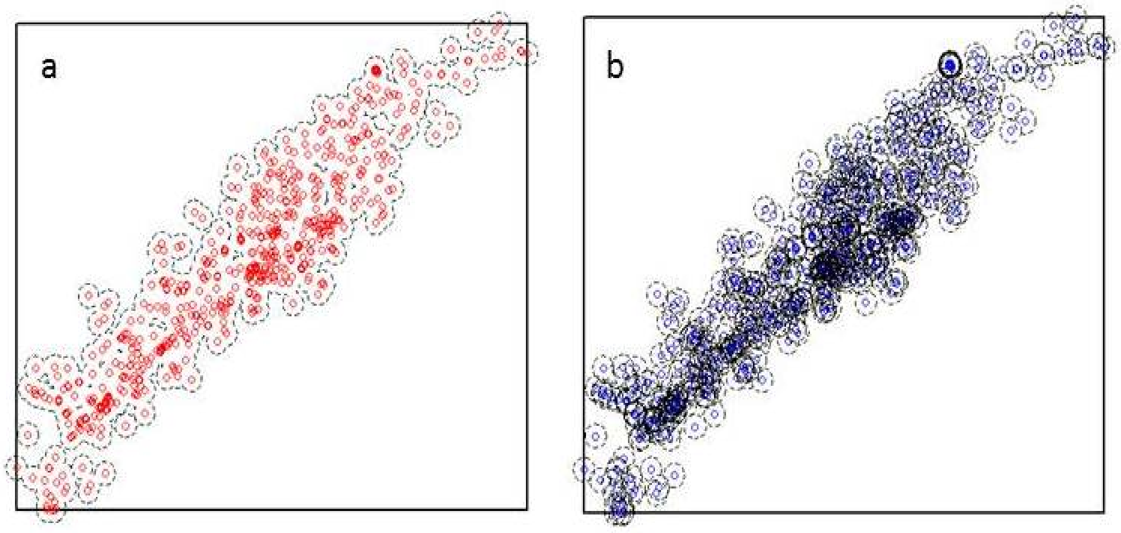


Figure 1.9: Buffers around mule deer locations using the a) byid=FALSE (default) and b) byid=TRUE for the gBuffer function using package rgeos.

```
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
```