ANALYZING WILDLIFE TELEMETRY DATA IN R

By John Leonard

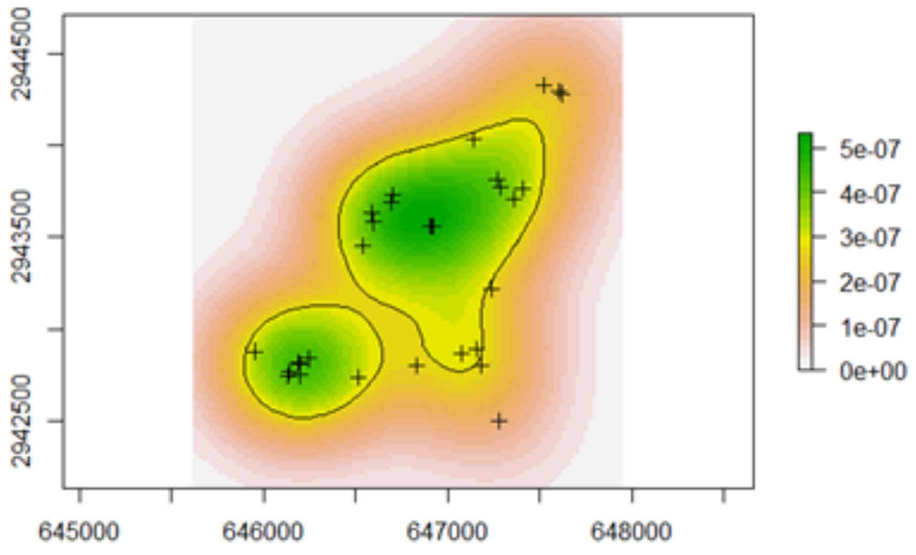


TABLE OF CONTENTS

INTRODUCTION

R has become the dominant platform for data manipulation and analysis in wildlife science, yet it has a reputation among wildlife scientists as being counterintuitive and difficult to use. There are a number of introductory R manuals available, but strangely, very few people seem to be able to learn to code in R by reading these manuals. Resources available for learning R tend to fall into two categories. The first category consists of basic introductory texts with names like "Introduction to R" and "R for Dummies". These books attempt to give beginners an overview of all the functionalities of R, yet they are often overly broad, tending to inundate the user with unnecessary information and giving little in the way of immediately useful techniques. After reading one of these texts few people are able to immediately start using R to analyze their own data.

At the other end of the spectrum are vignettes developed for specific packages relevant to wildlife scientists, such as "adehabitatHR". In my opinion, working through these vignettes, complete with sample data, is a better use of time than reading the "Introduction to R" books. However, these vignettes tend to have some major shortcomings. For one, the sample data is almost always formatted in some structure that makes it work seamlessly with the package's functions, and little information is typically provided about how to put one's raw data in the correct format. Additionally, the package vignettes typically assume that the user wants to conduct all analyses in R, rather than use a GIS program such as ArcGIS to view shapefiles and produce the final maps. In my experience, most people would like to learn to use R to automate tedious and time consuming tasks, but still plan to use ArcGIS for making visually-appealing representations of their data.

This manual falls somewhere in-between the introductory texts and package vignettes. It describes a workflow for manipulating and processing large amounts of animal location data obtained from GPS or VHF radio collars. It uses existing packages to perform operations such as home range estimation, but it allows these functions to be applied to a large number of files quickly and efficiently.

As such, the code provided in this manual does not constitute a package, but rather an integrated workflow for using existing packages. My goal was to write a manual that allows a researcher with no previous experience coding in R to perform all functions described in this manual on his or her datasets. I therefore tried to leave nothing out of this manual, describing the entire workflow from modifying the original telemetry files to exporting shapefiles. For someone comfortable with R this will probably be overkill, but for a first time user I believe this level of detail is essential.

In my experience, the best way to learn to use R is to immediately start analyzing your own data. If you know a few basic functions that are actually useful and produce meaningful results you will be well on your way to making R work for you. With that in mind, this manual describes a relatively limited number of operations and does not go into great detail explaining how R works.

My intention is for this manual to be a stand-alone document, meaning that links to additional files are not necessary. Instead of providing links to files containing the R code, all code described in this document can be found in Appendix A. Some very simple sample data files are provided as plain text in Appendix B. This is a rather unconventional approach to providing example data and R code, but it has the advantage of providing the user with everything he or she needs to analyze wildlife telemetry data in a single document. My recommendation is to first use the sample data provided in Appendix B to work through the entire workflow described in this manual, and then to repeat the process using one's own data.

USING RSTUDIO

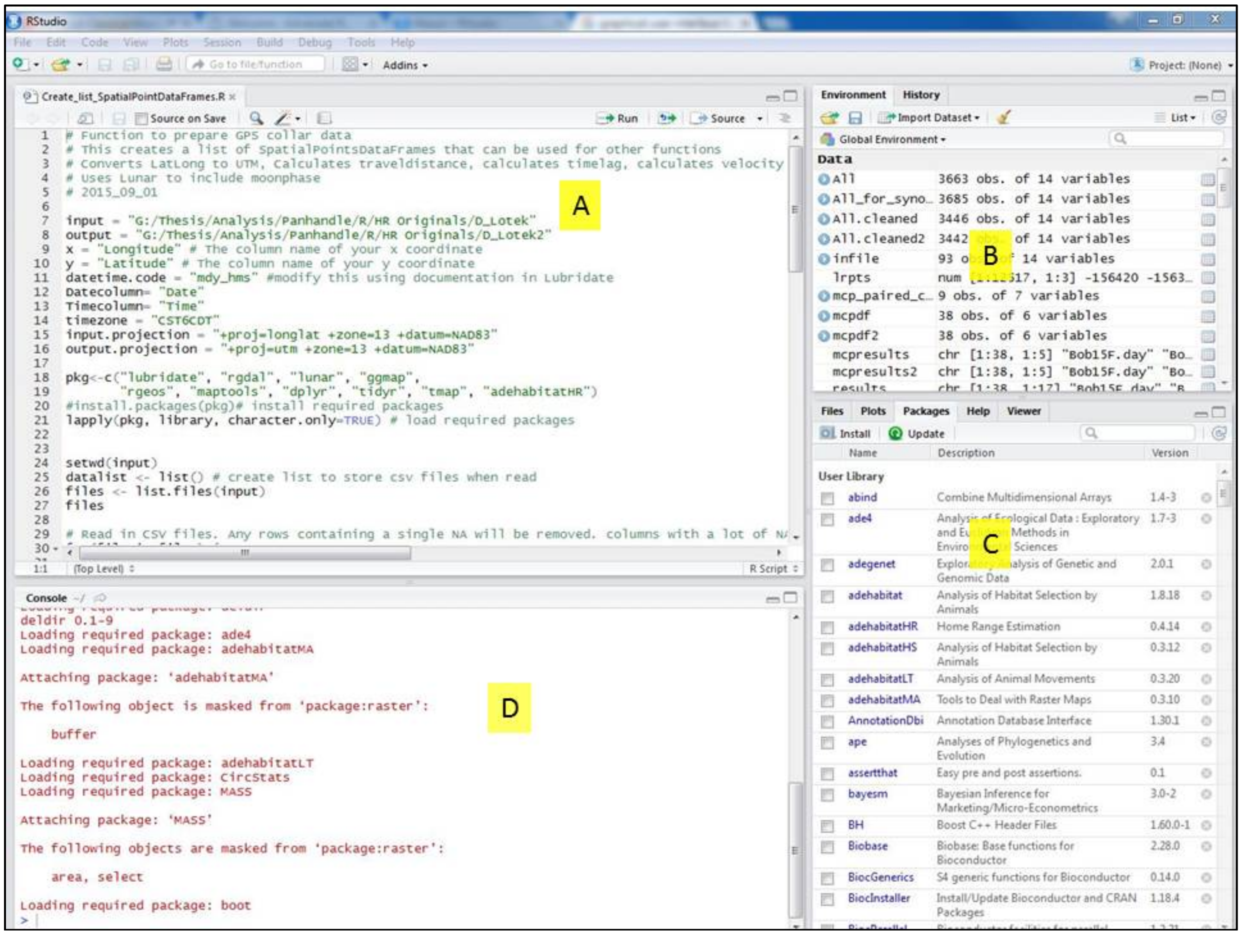
Most introductory R books strongly suggest using RStudio, but I consider its use mandatory. It is counterproductive to try to learn R just using the base R GUI. RStudio provides tools for editing commands before executing them in R, lets the user view plots without having to save them, and displays all objects that are present in the working environment. RStudio provides an easy way to access and use R, so it only works if you have R installed. See Figure 1 for a simple key to the different parts of the RStudio interface.

INSTALLING R AND RSTUDIO

To install R, go to https://www.r-project.org and follow the instructions for installing the latest version of R. To install RStudio go to https://www.rstudio.com/products/rstudio/download/ and follow the instructions to install the latest version of RStudio. These steps are pretty self-explanatory, and I have never run into any problems with this. R is continuously updated, and it is probably a good idea to always have the most up-to-date version of R installed. If you already have an earlier version of R installed and would like to update it, the easiest way to do this is to copy and paste the following code directly into the R GUI (not RStudio).

```
# installing/loading the package:
if(!require(installr)) {
install.packages("installr"); require(installr)} #load / install+load
installr

# using the package:
updateR() # this will start the updating process of your R installation.  It
will check for newer versions, and if one is available, will guide you
through the decisions you'd need to make.
```



```r
# Function to prepare GPS collar data
# This creates a list of SpatialPointsDataFrames that can be used for other functions
# Converts LatLong to UTM, Calculates traveldistance, calculates timelag, calculates velocity
# Uses Lunar to include moonphase
# 2015_09_01

input = "G:/Thesis/Analysis/Panhandle/R/HR Originals/D_Lotek"
output = "G:/Thesis/Analysis/Panhandle/R/HR Originals/D_Lotek2"
x = "Longitude" # The column name of your x coordinate
y = "Latitude" # The column name of your y coordinate
datetime.code = "mdy_hms" #modify this using documentation in Lubridate
Datecolumn= "Date"
Timecolumn= "Time"
timezone = "CST6CDT"
input.projection = "+proj=longlat +zone=13 +datum=NAD83"
output.projection = "+proj=utm +zone=13 +datum=NAD83"

pkg<-c("lubridate", "rgdal", "lunar", "ggmap",
       "rgeos", "maptools", "dplyr", "tidyr", "tmap", "adehabitatHR")
#install.packages(pkg)# install required packages
lapply(pkg, library, character.only=TRUE) # load required packages


setwd(input)
datalist <- list() # create list to store csv files when read
files <- list.files(input)
files

# Read in CSV files. Any rows containing a single NA will be removed. columns with a lot of NA
```

```
Data
All               3663 obs. of 14 variables
All_for_syno…     3685 obs. of 14 variables
All.cleaned       3446 obs. of 14 variables
All.cleaned2      3442 obs. of 14 variables
infile            93 obs. of 14 variables
lrpts             num [1:12517, 1:3] -156420 -1563…
mcp_paired_c…     9 obs. of 7 variables
mcpdf             38 obs. of 6 variables
mcpdf2            38 obs. of 6 variables
mcpresults        chr [1:38, 1:5] "Bob15F.day" "Bo…
mcpresults2       chr [1:38, 1:5] "Bob15F.day" "Bo…
results           chr [1:38, 1:17] "Bob15F.day" "B…
```

Name	Description	Version
User Library		
abind	Combine Multidimensional Arrays	1.4-3
ade4	Analysis of Ecological Data : Exploratory and Euclidean Methods in Environmental Sciences	1.7-3
adegenet	Exploratory Analysis of Genetic and Genomic Data	2.0.1
adehabitat	Analysis of Habitat Selection by Animals	1.8.18
adehabitatHR	Home Range Estimation	0.4.14
adehabitatHS	Analysis of Habitat Selection by Animals	0.3.12
adehabitatLT	Analysis of Animal Movements	0.3.20
adehabitatMA	Tools to Deal with Raster Maps	0.3.10
AnnotationDbi	Annotation Database Interface	1.30.1
ape	Analyses of Phylogenetics and Evolution	3.4
assertthat	Easy pre and post assertions.	0.1
bayesm	Bayesian Inference for Marketing/Micro-Econometrics	3.0-2
BH	Boost C++ Header Files	1.60.0-1
Biobase	Biobase: Base functions for Bioconductor	2.28.0
BiocGenerics	S4 generic functions for Bioconductor	0.14.0
BiocInstaller	Install/Update Bioconductor and CRAN Packages	1.18.4

```
deldir 0.1-9
Loading required package: ade4
Loading required package: adehabitatMA

Attaching package: 'adehabitatMA'

The following object is masked from 'package:raster':

    buffer

Loading required package: adehabitatLT
Loading required package: CircStats
Loading required package: MASS

Attaching package: 'MASS'

The following objects are masked from 'package:raster':

    area, select

Loading required package: boot
>
```

Figure 1: Screen capture of RStudio interface with some data loaded.

A) Source browser notebook- You can open and edit R scripts in this window. Nothing happens if you click "enter" on a line in this window, though. You have to either use "ctrl+enter" or click "Run" to actually send a command to the Console where the command is carried out. You can also view data frames, matrices and other objects in this window.

B) Environment tab- This shows all the objects present in your working environment. Clicking on the "History" tab will show a history of everything you typed into the console. For virtually any project, you will be creating a large number of objects. Having an environment tab that shows you all the objects that have been created is extremely useful.

C) Packages tab- This tab shows all the packages that have been installed. To load a particular package, click on the open box by the package name. The other tabs in this window are "**Files**", "**Plots**", "**Help**" and "**Viewer**". The "**Files**" browser shows the files and subdirectories of a given directory. The "**Plots**" tab will let you scroll through any plots or images you created. The "**Help**" tab will show you the documentation for most functions or packages you will be using. The **"Viewer"** tab is used to view local web content.

D) The R Console- This is where things actually happen. This is where you will run scripts, execute functions, and get error messages. When you decide to run some code that you typed up in tab "A" it will be sent down to tab "D"- the R console. If you use the base R GUI this R Console tab is pretty much all you get.

INSTALLING AND LOADING REQUIRED PACKAGES

Packages are collections of functions and compiled R code that build upon and expand the base functions of R. In order to use a package you must first install the package and then load it into your R working environment. The following packages are required to work through this manual: “sp”, “lubridate”, “rgdal”, “maptools”, “adehabitatHR”, “rgdal”, “raster”, “BBMM”, “lunar”, “oce”, and “rhr”. The first code block provided in Appendix A should install all packages to a local library and load them into the R working environment. Usually, everything will work if you just highlight the entire code block and run it in RStudio. You can check which R packages have been installed by clicking on the “Packages” tab in RStudio. This will open up a list of available packages with open boxes located to the left of each box. Clicking on the box to fill it in with a checkmark will load that package into the working environment. If some of the required packages fail to load you can manually load them by clicking on the open boxes. If one or more packages did not install you can manually install it through RStudio by clicking on Tools>Install Packages, and typing in the name of the package. This will work for all packages except “rhr” which must be installed using the code provided in the first code block of Appendix A. I recommend making sure that all required packages are installed and loaded before trying to use any of the other code blocks. In each code block, I included a few lines to make sure the required packages for that code block were installed, but it is still probably best to ensure all packages are installed and loaded at the very beginning.

LOADING R CODE INTO RSTUDIO

The workflow I created for analyzing and manipulating GPS collar data is saved as 10 different text files which can be thought of as “code blocks”. Even though this workflow is intended to move seamlessly through these code blocks, I decided to keep these separate as I believe this keeps things neat and easier to use in RStudio. All 10 code blocks are saved in Appendix A of this manual.

To load these code blocks into RStudio first go to File>New File>R Script. This will open up a blank page in the Source browser notebook of RStudio. Next, highlight one of the code blocks to copy and paste it into this blank page. Repeat the process for all code blocks, giving each code block its own tab. You can then easily move from one code block to another by selecting the correct tab.

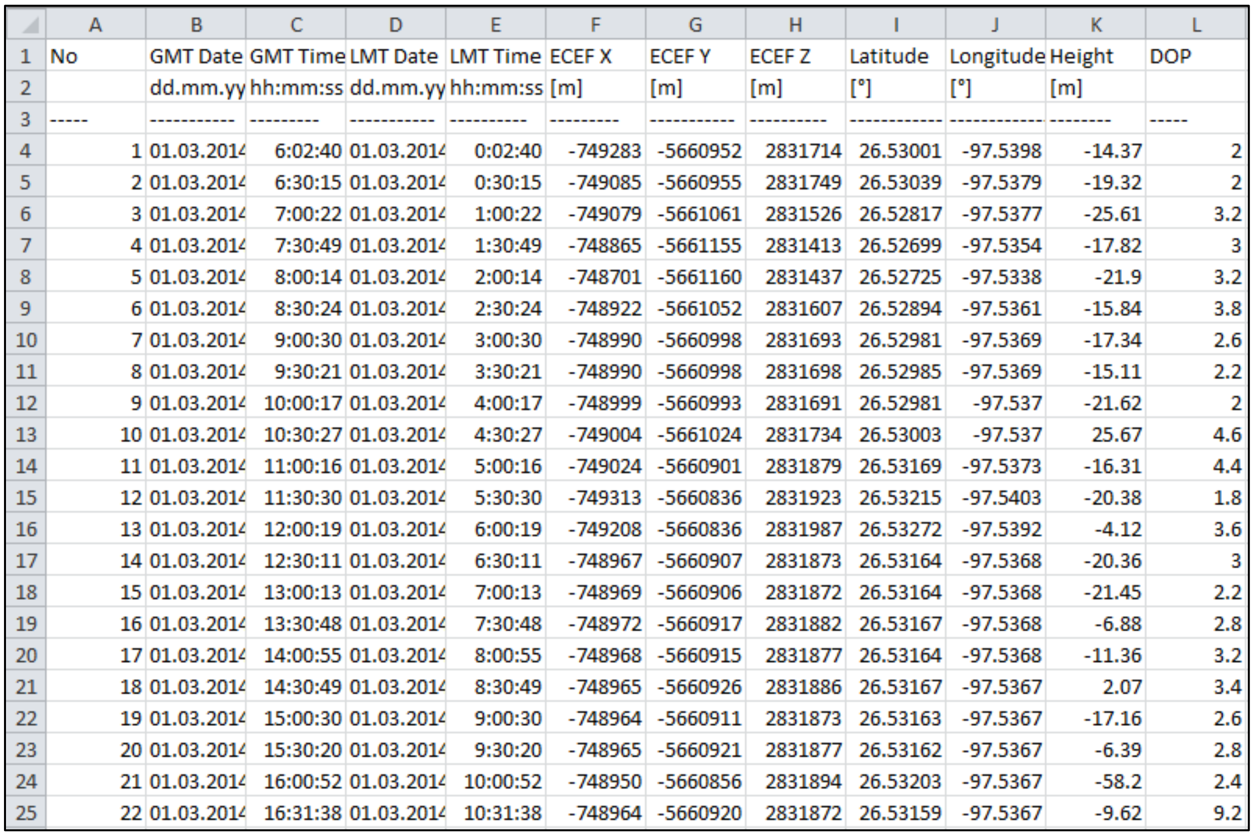
To run the code in any one of these code blocks, simply highlight the entire page (after correctly modifying any user defined input) and click "Run" at the top of the Source browser notebook. This will send the code to the R Console and execute the functions. As mentioned in the introduction, there are no additional files that need to be downloaded to use this manual. All functions described in this document are provided as blocks of plain text in Appendix A.

FORMATTING INPUT LOCATION DATA

The R code described in this manual is intended to save time by automating many of the operations people perform on animal location files. For this to work, all files must be saved as comma delimited csv files and placed in the same file folder. The projection and datum used should be identical for all files, and the column headings specifying x and y coordinates, and any date and time fields, should be the same. Any weird formatting, such as skipped rows or single headings covering multiple columns, should be removed. Column headings should not contain spaces and should not begin with a number. An easy way to format input files is to open each one in Excel, delete any extraneous columns or rows, modify column headings as needed, and resave the file as a csv file (Fig. 2).

You can use your own animal telemetry data as the input for these functions, or you can use the sample data in Appendix B. Appendix B contains 6 simplified comma-delimited files with animal GPS data. The sample files named "GPS_A", "GPS_B", and "GPS_C" contain GPS locations collected far enough apart temporally to be considered independent. These files should be used for any functions involving kernel or minimum convex polygon (MCP) home range estimation. The sample files named

"GPS_D" and "GPS_E" contain location points collected at 30-minute intervals and therefore are highly auto-correlated. These should be used for any functions involving Brownian bridges. To use the sample data, highlight it and paste it into a text editor such as Notepad. Then save it with a ".csv" extension (e.g., "GPS_A.csv"). Place all of these csv files into a file folder containing no other files.



	A	B	C	D	E	F	G	H	I	J	K	L
1	No	GMT Date	GMT Time	LMT Date	LMT Time	ECEF X	ECEF Y	ECEF Z	Latitude	Longitude	Height	DOP
2		dd.mm.yy	hh:mm:ss	dd.mm.yy	hh:mm:ss	[m]	[m]	[m]	[°]	[°]	[m]	
3	-----	-----------	---------	-----------	----------	---------	-----------	----------	------------	------------	--------	-----
4	1	01.03.2014	6:02:40	01.03.2014	0:02:40	-749283	-5660952	2831714	26.53001	-97.5398	-14.37	2
5	2	01.03.2014	6:30:15	01.03.2014	0:30:15	-749085	-5660955	2831749	26.53039	-97.5379	-19.32	2
6	3	01.03.2014	7:00:22	01.03.2014	1:00:22	-749079	-5661061	2831526	26.52817	-97.5377	-25.61	3.2
7	4	01.03.2014	7:30:49	01.03.2014	1:30:49	-748865	-5661155	2831413	26.52699	-97.5354	-17.82	3
8	5	01.03.2014	8:00:14	01.03.2014	2:00:14	-748701	-5661160	2831437	26.52725	-97.5338	-21.9	3.2
9	6	01.03.2014	8:30:24	01.03.2014	2:30:24	-748922	-5661052	2831607	26.52894	-97.5361	-15.84	3.8
10	7	01.03.2014	9:00:30	01.03.2014	3:00:30	-748990	-5660998	2831693	26.52981	-97.5369	-17.34	2.6
11	8	01.03.2014	9:30:21	01.03.2014	3:30:21	-748990	-5660998	2831698	26.52985	-97.5369	-15.11	2.2
12	9	01.03.2014	10:00:17	01.03.2014	4:00:17	-748999	-5660993	2831691	26.52981	-97.537	-21.62	2
13	10	01.03.2014	10:30:27	01.03.2014	4:30:27	-749004	-5661024	2831734	26.53003	-97.537	25.67	4.6
14	11	01.03.2014	11:00:16	01.03.2014	5:00:16	-749024	-5660901	2831879	26.53169	-97.5373	-16.31	4.4
15	12	01.03.2014	11:30:30	01.03.2014	5:30:30	-749313	-5660836	2831923	26.53215	-97.5403	-20.38	1.8
16	13	01.03.2014	12:00:19	01.03.2014	6:00:19	-749208	-5660836	2831987	26.53272	-97.5392	-4.12	3.6
17	14	01.03.2014	12:30:11	01.03.2014	6:30:11	-748967	-5660907	2831873	26.53164	-97.5368	-20.36	3
18	15	01.03.2014	13:00:13	01.03.2014	7:00:13	-748969	-5660906	2831872	26.53164	-97.5368	-21.45	2.2
19	16	01.03.2014	13:30:48	01.03.2014	7:30:48	-748972	-5660917	2831882	26.53167	-97.5368	-6.88	2.8
20	17	01.03.2014	14:00:55	01.03.2014	8:00:55	-748968	-5660915	2831877	26.53164	-97.5368	-11.36	3.2
21	18	01.03.2014	14:30:49	01.03.2014	8:30:49	-748965	-5660926	2831886	26.53167	-97.5367	2.07	3.4
22	19	01.03.2014	15:00:30	01.03.2014	9:00:30	-748964	-5660911	2831873	26.53163	-97.5367	-17.16	2.6
23	20	01.03.2014	15:30:20	01.03.2014	9:30:20	-748965	-5660921	2831877	26.53162	-97.5367	-6.39	2.8
24	21	01.03.2014	16:00:52	01.03.2014	10:00:52	-748950	-5660856	2831894	26.53203	-97.5367	-58.2	2.4
25	22	01.03.2014	16:31:38	01.03.2014	10:31:38	-748964	-5660920	2831872	26.53159	-97.5367	-9.62	9.2

Figure 2: Example of raw GPS collar data that needs to be modified. Rows 2 and 3 should be deleted. Any column headings with a space in them (e.g. "GMT Date") should be changed to something with no space (e.g., "GMT.Date"). The entire file then needs to be saved as a comma-delimited csv file.

CREATING A LIST OF SPATIAL POINTS DATA FRAMES

A Spatial Points Data Frame is a special class of R object that can be created using the package "sp". At a minimum, a Spatial Points Data Frame will contain defined x and y coordinates. The Spatial Points Data Frame does not necessarily need to contain projection information, but all subsequent analyses will be easier if we do specify projection information. Spatial Points Data Frames differ from

“Spatial Points” objects in that the former allows the inclusion of any number of non-spatial attributes. This is convenient if we want to later subset location data based on attributes such as HDOP, number of satellites, or time of day.

A list is the most general type of data structure in R, and it basically consists of any number of data objects, of similar or different classes, grouped together in a specific order. An easy way to automate functions in R is to tell R to apply a specific set of commands to each object in a list. I like to think of this as creating an assembly line with your data, and then training a machine to perform specific operations on each object that is passed down the assembly line.

Once the input GPS files are formatted correctly, load the code block titled “Create List of Spatial Points Data Frames” into the source browser notebook in RStudio. The first several lines of this R file will appear as below:

```
# Create List of Spatial Points Data Frames
# Animal locations must be saved as separate csv files in a single folder
# Column headings for x and y coordinates and Date and Time fields must be
identical
# Created by John Leonard 08_23_2016

# User defined input. Modify lines below.
##############################################################################
input = "J:/R/R for GPS collar data/Test Collar Data"
x = "Longitude" # The column name of your x coordinate
y = "Latitude" # The column name of your y coordinate
datetime.code = "dmy_HMS" #modify this using documentation in Lubridate
Datecolumn= "GMT.Date"
Timecolumn= "GMT.Time"
timezone = "UTC"
input.projection = "+init=epsg:4326"
##############################################################################
```

The code falling between the lines of hashtags (#) is considered “user defined input” and will need to be modified. Anything within quotation marks can be changed to reflect the file folder location, column headings, and format of your data. Any text appearing after the hashtags is only for explanation purposes and will not actually be run. The meanings of the various fields are provided below:

“input” refers to the complete file pathway where your GPS locations files are stored. It is important to note that forward slashes are used instead of back slashes to separate file folders. Also, if you are storing your data on an external hard-drive and using different computers each time, the full file pathway may be different each time. For example, depending on which computer and USB port I use, the file pathway for my data will usually either start with a “J” or an “F”. Make sure that you have permission to modify the file folder specified. For example, you might need administrator privileges to write files directly to the C drive of a University computer.

“x” is the column name for the x coordinate. Spell this out exactly as it appears in your GPS data files.

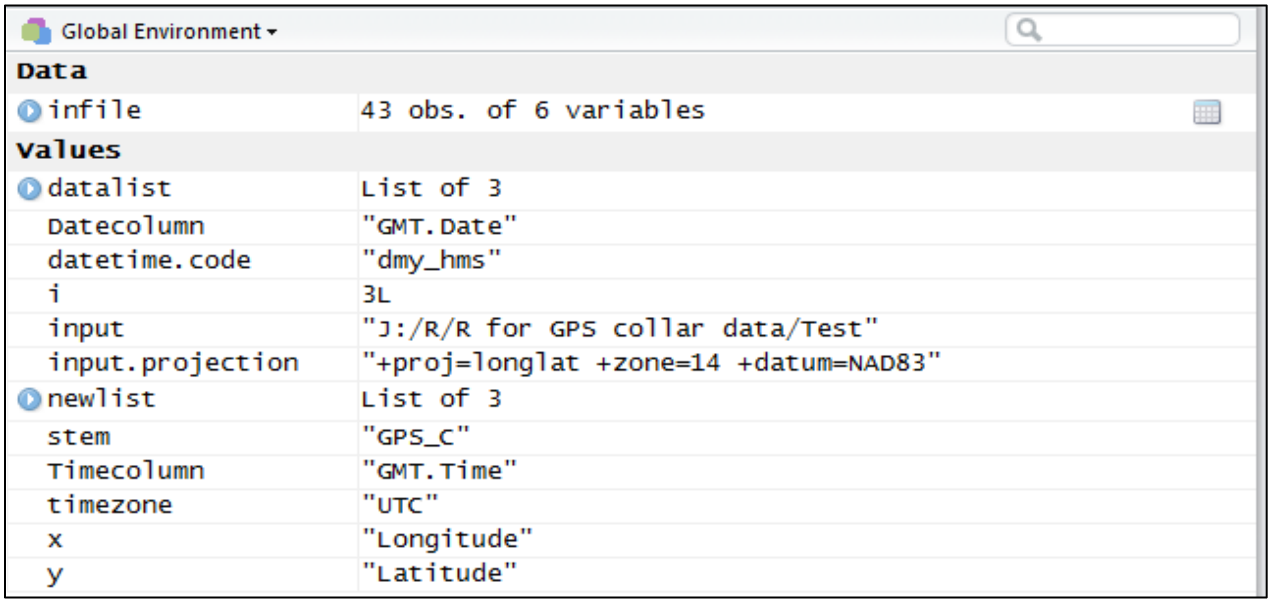
“y” is the column name for the y coordinate. Spell this out exactly as it appears in your GPS data files.

“datetime.code” is a code that specifies the order in which day, month, year, hour, minute, and second will appear in your combined “Date_Time” column. “dmy_HMS” means: day, month, year, (space), hour, minute, second. See documentation for the package “lubridate” for more details.

“timezone” – to fill out this field you need to know the correct time zone code for your data. “UTC” is Coordinated Universal Time which is the time zone formerly known as Greenwich Mean Time (GMT). See “lubridate” documentation for more details.

“input.projection”- this is the most difficult field to fill out correctly. It specifies the coordinate reference system of your x and y fields. The code displayed above specifies Latitude/Longitude with decimal degrees.

The next step is to highlight the entire page of code and click Run. This may take a few minutes depending on how many different GPS data files you have. If all goes well, the Environment tab (usually to the far right) in RStudio will be populated with several new objects (Fig. 3).

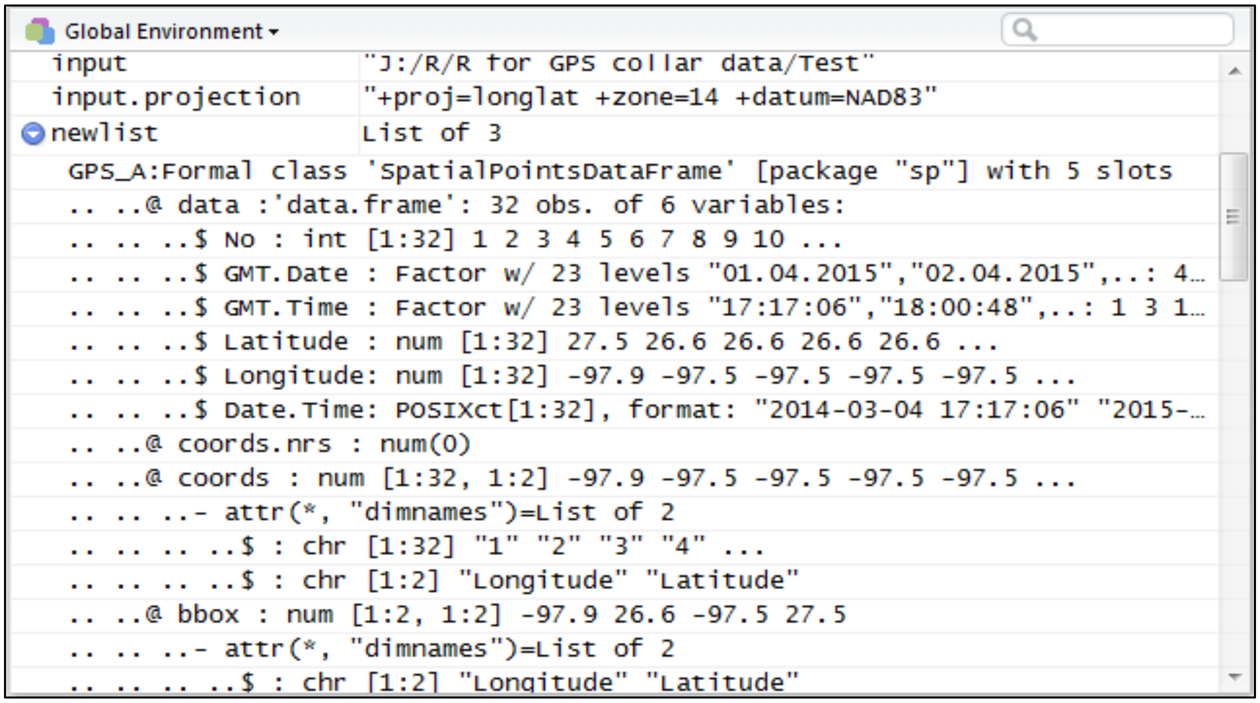


```
Data
infile                43 obs. of 6 variables
Values
datalist              List of 3
Datecolumn            "GMT.Date"
datetime.code         "dmy_hms"
i                     3L
input                 "J:/R/R for GPS collar data/Test"
input.projection      "+proj=longlat +zone=14 +datum=NAD83"
newlist               List of 3
stem                  "GPS_C"
Timecolumn            "GMT.Time"
timezone              "UTC"
x                     "Longitude"
y                     "Latitude"
```

Figure 3: Environment tab in R-studio after importing GPS locations files and creating list of Spatial Points Data Frames.

The one object we care about is “newlist” which is our list of Spatial Points Data Frames. We can click on the blue arrow to the left of “newlist” in our Environment tab to see what is inside this object (Fig. 4). For my particular dataset, the newlist object contains 3 Spatial Points Data Frames, each corresponding to one of 3 individuals. All attributes originally present in my data are listed under the “@data” slot. It is worth spending a bit of time looking at the contents of newlist in R-studio. Notice that some of the attributes are coded as Factor classes, whereas others are coded as integer or numerical. The Date.Time field that we created is class POSIXct, which pertains specifically to date/time fields.

Scrolling down under the Environment tab reveals several other slots, including “@coords”, which contains our coordinate information, and “@proj4string”, which contains projection information.



```
  input                  "J:/R/R for GPS collar data/Test"
  input.projection       "+proj=longlat +zone=14 +datum=NAD83"
  newlist                List of 3
   GPS_A:Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
   .. ..@ data :'data.frame': 32 obs. of 6 variables:
   .. .. ..$ No : int [1:32] 1 2 3 4 5 6 7 8 9 10 ...
   .. .. ..$ GMT.Date : Factor w/ 23 levels "01.04.2015","02.04.2015",..: 4…
   .. .. ..$ GMT.Time : Factor w/ 23 levels "17:17:06","18:00:48",..: 1 3 1…
   .. .. ..$ Latitude : num [1:32] 27.5 26.6 26.6 26.6 26.6 ...
   .. .. ..$ Longitude: num [1:32] -97.9 -97.5 -97.5 -97.5 -97.5 ...
   .. .. ..$ Date.Time: POSIXct[1:32], format: "2014-03-04 17:17:06" "2015-…
   .. ..@ coords.nrs : num(0)
   .. ..@ coords : num [1:32, 1:2] -97.9 -97.5 -97.5 -97.5 -97.5 ...
   .. .. ..- attr(*, "dimnames")=List of 2
   .. .. .. ..$ : chr [1:32] "1" "2" "3" "4" ...
   .. .. .. ..$ : chr [1:2] "Longitude" "Latitude"
   .. ..@ bbox : num [1:2, 1:2] -97.9 26.6 -97.5 27.5
   .. .. ..- attr(*, "dimnames")=List of 2
   .. .. .. ..$ : chr [1:2] "Longitude" "Latitude"
```

Figure 4: Viewing structure of newlist in the RStudio Environment tab.

PLOT POINTS TO REMOVE OUTLIERS

We are now ready to plot points and remove any outliers that may mess up our analysis. Before deploying GPS collars, it is common for researchers to turn the GPS collar on to acquire a few fixes. This usually results in a GPS locations file that contains some erroneous points. These are usually pretty easy to spot as they are often located far from the animal's actual home range. You can always remove these points in Excel while you are formatting your GPS locations files, or you can do it later in R. Even if you do remove erroneous points early on in Excel, it is still a good idea to plot points to make sure everything looks correct before trying to generate home ranges.

Load the code block named "Plot Points to Remove Outliers" in the RStudio Source browser notebook tab. This code uses the "newlist" object you created earlier as input. To plot the points you just need to run the following 3 lines of code:

```
for (i in 1:length(newlist)){
  plot(newlist[[i]]@coords, main=names(newlist[i]))
}
```

This will produce a unique plot of points for each GPS locations file. The file name will be displayed as the plot title and the x and y coordinates will be displayed on the x and y axes (Fig. 5).

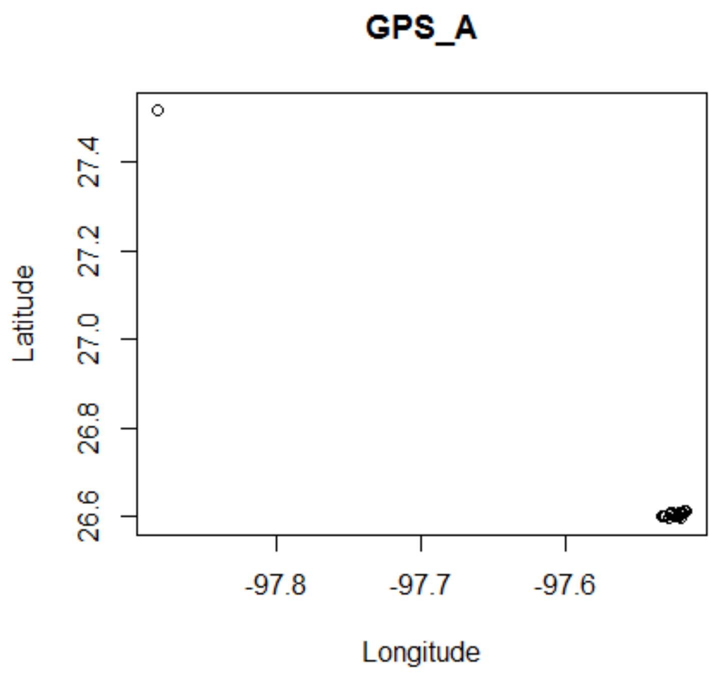


Figure 5: Example plot created using code block entitled “Plot points to remove outliers”.

From the example displayed in Fig. 5 you can see at least 1 point that probably shouldn’t be there. The point located at the far upper left of the plot probably resulted from testing the GPS collar at home. You could go into Excel and try to individually remove these points, or you could automate this process using R.

If you are carrying out any functions that have the potential to permanently alter your data it is a good practice to first make a copy of the data. We will create a copy of newlist called “newlist2”. We will run some functions intended to remove outliers from newlist2 and then plot the results to make sure it worked. Note that all code below line 8 in “Plot points to remove outliers” needs to be modified for your particular dataset.

There are a lot of ways to modify individual Spatial Points Data Frames within newlist, but the example included in the R file “Plot points to remove outliers” uses the range of Latitude and Longitude values to accomplish this. For the sample data, GPS_A (Fig. 5), we can see that all but one of the points are located in the bottom right corner of the graph. We can subset this file to include only points with Latitude (y coordinate) below 27. This will effectively remove the outlier point in the upper left corner of the graph. For GPS_B (not shown) we instead set a minimum value for Longitude (x coordinate).

```
# First make duplicate of newlist called newlist2 just to be safe

newlist2<-newlist

# Modify code below to select maximum or minimum x or y values
# Code below makes "27" the maximum latitude value for GPS_A

newlist2$GPS_A<-subset(newlist2$GPS_A, newlist2$GPS_A@coords[,y]<27)
```

The above code can be modified to fit your dataset. The fields after the “$” (e.g. `GPS_A`, `GPS_B`) should be changed to the names of whichever locations files you want to work with. If you want to subset points based on Latitude you should still use “`@coords[,y]`”, but specify whichever minimum or maximum value is appropriate for your data. After you think you have removed the outliers from each Spatial Points Data Frame in your newlist object, plot the points again (Fig. 6).

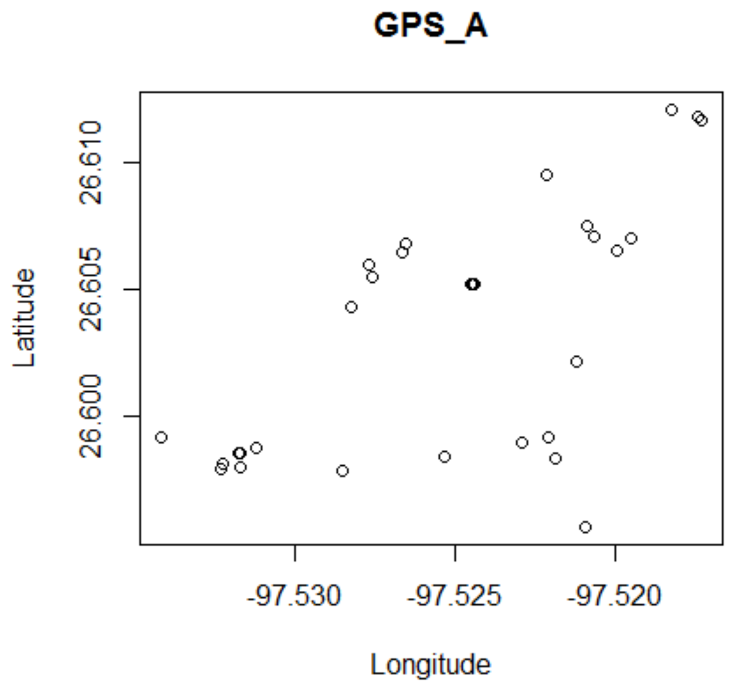


Figure 6: Plot of points for GPS_A after outlier point has been removed.

If everything looks good, you can convert newlist2 back into newlist with the following code:

```
newlist<-newlist2
```

You should be careful with the above step, as you cannot undo it. If you want to modify the code in "Plot points to remove outliers" to perform more complex subsetting operations, such as removing locations with high HDOP, you may want to skip the above step and instead modify all following code blocks by replacing "newlist" with "newlist2".

REPROJECTING POINTS

The code to generate MCP and kernel home range estimates will work fine regardless of the coordinate system used for your input points. Shapefiles will be generated with the appropriate projection and they will therefore appear in the correct locations when you open them in ArcGIS. However, these code blocks also automatically calculate home range area (ha.) and output this information to a matrix. In order to do this correctly, the units of the x and y coordinates must be in meters. The easiest way to accomplish this is to change the coordinate system of the data to UTM using the code block “Re-project Points to UTM”.

After executing this code, “newlist” will contain Spatial Points Data Frames with x and y coordinates transformed to the desired projection and datum (e.g., UTM NAD 83). If your locations files are already given as UTM coordinates you can skip this step entirely. As long as you specify the correct projection in the “Create List of Spatial Points Data Frames” step they will be projected correctly.

EXPORTING POINTS AS SHAPEFILES

Importing text files into ArcGIS manually and then converting these into shapefiles isn’t particularly difficult, but it is time consuming. If you have a lot of telemetry files you can automate the process by having R loop through each Spatial Points Data Frame in “newlist” and write a shapefile for each one. To do this, load the code block titled “Export Points” into the RStudio Source notebook browser and simply highlight and run all code. The shapefiles will be written to your working directory and a “.prj” file specifying projection information will be given. The projection will be whatever you re-projected your points to. If some of your columns are formatted as unusual classes (e.g., POSIXct) they might not import into ArcGIS correctly. To prevent this from being a problem, convert any columns with questionable classes to character fields. This has already been done for the POSIXct field called “Date.Time” with the following code:

```
infile$Date.Time<-as.character(infile$Date.Time)
```

When you open the new points shapefiles in ArcGIS they will be projected in whatever coordinate system you selected in the previous step. However, the default names given to the x and y coordinates will not be correct. If your original points were in Latitude/Longitude and you changed them to UTM, the points will show up in the correct location in ArcGIS, but the attribute table will read Longitude_1, and Latitude_1 for UTM x and y coordinates respectively.

MAKING MINIMUM CONVEX POLYGON HOME RANGES

If all outlier points have been removed from each of the Spatial Points Data Frames inside "newlist", we are ready to generate home range estimates. Minimum convex polygons (MCPs) are among the simplest home range estimators so we will start with those. Load the R file called "Create MCP" into the Source browser notebook. The only user defined input for this one is MCP percent which is specified in line 2. Once this has been specified, highlight the entire page of R code and click "Run". This should loop through all sets of points in your "newlist" file and produce a unique MCP plot for each individual (Fig. 7). This code uses the R package "adehabitatHR" to generate MCPs using the function "mcp". This function is also present in the older R package "adehabitat". If both "adehabitatHR" and "adehabitat" are loaded into R this may lead to an error message. Make sure "adehabitat" is not loaded before running this code.

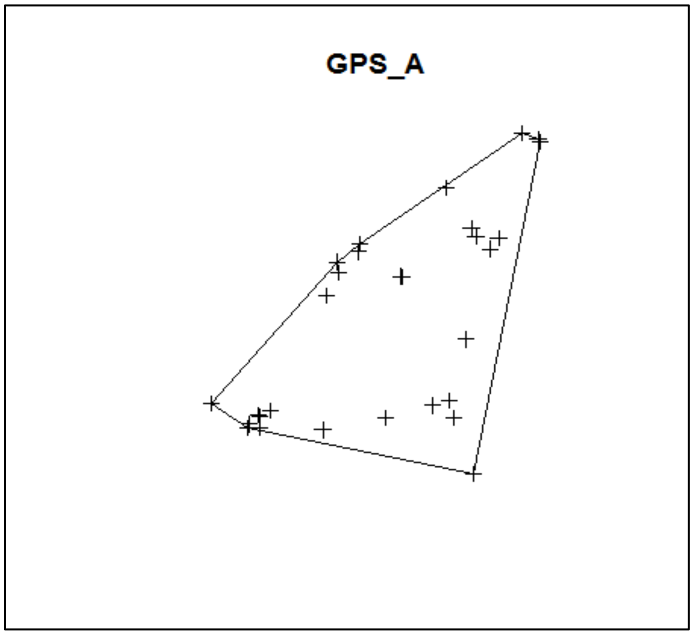
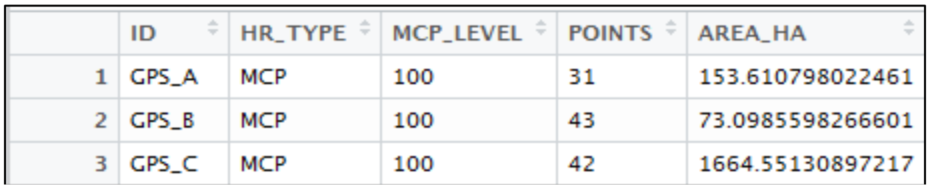


Figure 7: Minimum Convex Polygon (100% MCP) generated for GPS_A.

In addition to producing images of MCP home ranges on top of location points, this code calculates summary statistics for each individual. This information is stored in a matrix called "mcpresults". This matrix will appear as a newly created object in the Environment tab. Click on it to view it (Fig. 8). If the values under "AREA_HA" seem strange this is probably because some coordinate system other than UTM has been specified. You can still create home ranges with other coordinate systems, but the area calculations will be incorrect.



	ID	HR_TYPE	MCP_LEVEL	POINTS	AREA_HA
1	GPS_A	MCP	100	31	153.610798022461
2	GPS_B	MCP	100	43	73.0985598266601
3	GPS_C	MCP	100	42	1664.55130897217

Figure 8: Summary statistics for 100% MCP home ranges created for 3 individuals.

If you change the MCP percentage and re-run the function, it will draw new home range boundaries and calculate new area values for each individual, however, the file "mcpresults" will be re-written with the new data. To save the results file as a csv file use the "write.csv" function as below:

```
write.csv(mcpresults, file="mcpresults.csv")
```

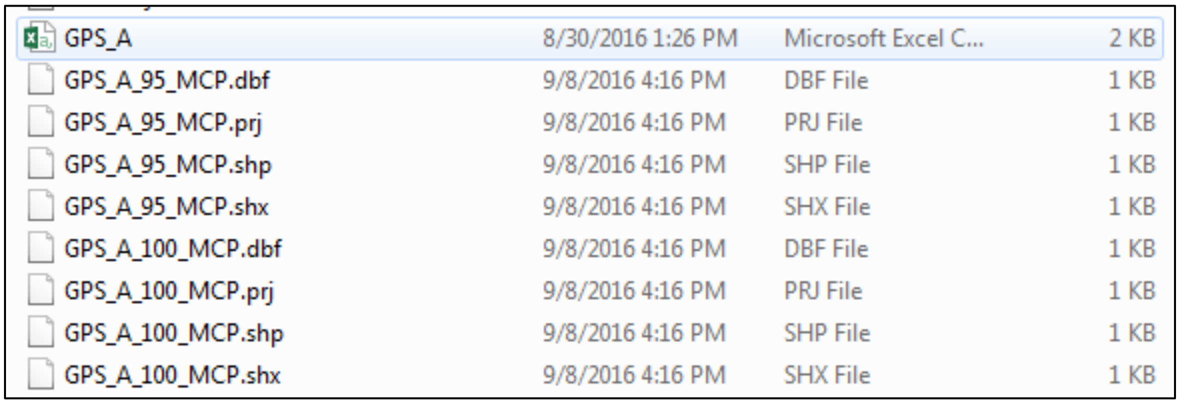
This will cause a new csv file called "mcpresults.csv" to appear in whatever file folder was specified as your working directory. This will be the file pathway you entered earlier on when we were creating a list of Spatial Points Data Frames. To verify your working directory, use the following command:

```
getwd()
```

This will print the full file pathway of your working directory. In my case, it produces the following:

```
getwd()
[1] "F:/R/R for GPS collar data/Test Collar Data"
```

If you navigate to this file folder you will also find the MCP shapefiles that were created. Each MCP shapefile will actually consist of 4 separate files with the following extensions: ".dbf", ".shp", ".shx", and ".prj". The ".prj" file specifies the projection and is necessary for the shapefiles you created to appear in the correct locations when you import them into ArcGIS. The names of the various files include both the original GPS file name and the MCP percent level. If you were to re-run the code using a different MCP percent level the new shapefiles would have different names, allowing you to differentiate between, say, 100% MCP shapefiles and 95% MCP shapefiles.



GPS_A	8/30/2016 1:26 PM	Microsoft Excel C...	2 KB
GPS_A_95_MCP.dbf	9/8/2016 4:16 PM	DBF File	1 KB
GPS_A_95_MCP.prj	9/8/2016 4:16 PM	PRJ File	1 KB
GPS_A_95_MCP.shp	9/8/2016 4:16 PM	SHP File	1 KB
GPS_A_95_MCP.shx	9/8/2016 4:16 PM	SHX File	1 KB
GPS_A_100_MCP.dbf	9/8/2016 4:16 PM	DBF File	1 KB
GPS_A_100_MCP.prj	9/8/2016 4:16 PM	PRJ File	1 KB
GPS_A_100_MCP.shp	9/8/2016 4:16 PM	SHP File	1 KB
GPS_A_100_MCP.shx	9/8/2016 4:16 PM	SHX File	1 KB

Figure 9: Example of MCP shapefiles written to the working directory. The file "GPS_A.csv" is the original file of GPS location points. The other 4 files make up the 100% and 95% MCP shapefiles for this individual. The "100_MCP" appearing to the right of the original file name indicates that these are 100% MCP shapefiles. All files starting with "GPS_A_95" are 95% MCP shapefiles.

MAKING KERNEL HOME RANGES IN ADEHABITATHR

This code operates in a similar way as the code to generate MCPs. It also loops through all individuals in "newlist", and uses adehabitatHR to generate fixed kernel density home range estimates (KDE), however, there are more input parameters and you are more likely to get error messages with this particular code block. The best way to avoid error messages is to make absolutely certain all extreme outlier points have been removed. You can still generate MCPs using location data with extreme outliers, but kernel home range generation will likely fail on such datasets. To create KDE polygons for each file in “newlist” load the code block “Create KDE adehabitatHR” into the Source browser notebook in RStudio. User defined input for this set of code consists of 3 lines:

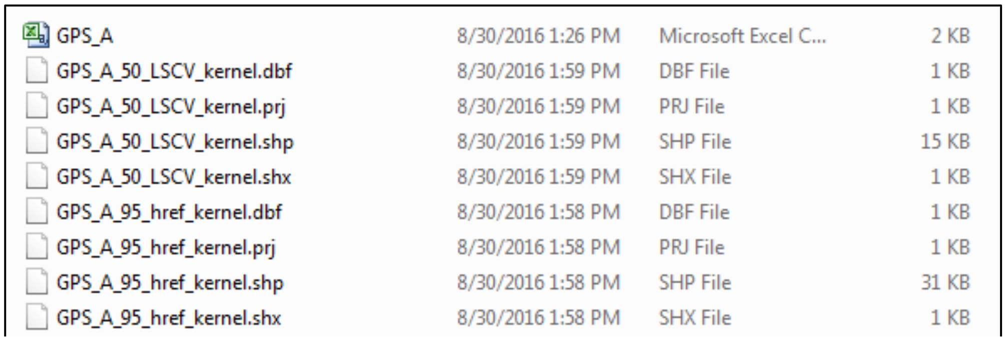
```
# User defined input. Modify lines below.
###########################################################################
per =95 # percent utilization distribution estimated
h="href" # bandwidth setting ("href", "LSCV", or user specified number)
grid=1000
###########################################################################
```

To specify the percent density contour at which you would like to draw the home range boundary (i.e., 50% KDE, 95% KDE, etc.) modify the number after “per”. The smoothing parameter (h) can be selected using either the reference bandwidth (href) method or least squares cross validation (LSCV). Alternatively, you may enter a number for “h” if you want to use the same smoothing parameter for all individuals. “Grid” is a tricky field as it specifies the extent of the picture you are drawing. If the value for grid is too small it will result in the following error message:

```
Error in getverticeshr.estUD(ud, per) :
The grid is too small to allow the estimation of home-range.
You should rerun kernelUD with a larger extent parameter
```

If this happens, you should re-run the code with a larger grid size (e.g., 10,000), although this will slow down calculations. After you run the code, the file folder selected as your working directory will be populated with a KDE shapefile for each individual. As with the MCP shapefiles, each of these KDE shapefiles actually consists of 4 separate files, with the following extensions: “.dbf”, “.prj”, “.shp”, and “.shx”.

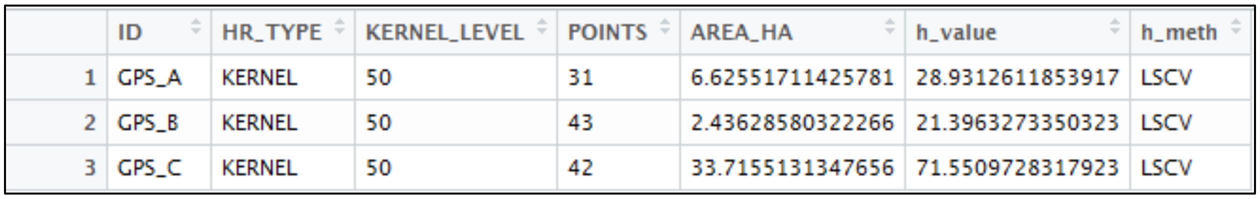
The shapefile names will be a bit more complicated as they will specify the percent density contour for your home range polygon and the bandwidth selection method. For an example of the shapefiles produced by this code see Fig. 10.



GPS_A	8/30/2016 1:26 PM	Microsoft Excel C...	2 KB
GPS_A_50_LSCV_kernel.dbf	8/30/2016 1:59 PM	DBF File	1 KB
GPS_A_50_LSCV_kernel.prj	8/30/2016 1:59 PM	PRJ File	1 KB
GPS_A_50_LSCV_kernel.shp	8/30/2016 1:59 PM	SHP File	15 KB
GPS_A_50_LSCV_kernel.shx	8/30/2016 1:59 PM	SHX File	1 KB
GPS_A_95_href_kernel.dbf	8/30/2016 1:58 PM	DBF File	1 KB
GPS_A_95_href_kernel.prj	8/30/2016 1:58 PM	PRJ File	1 KB
GPS_A_95_href_kernel.shp	8/30/2016 1:58 PM	SHP File	31 KB
GPS_A_95_href_kernel.shx	8/30/2016 1:58 PM	SHX File	1 KB

Figure 10: Example output of code to create kernel home range shapefiles. The file "GPS_A.csv" is the original GPS locations file. The 4 files beginning with "GPS_A_50_LSCV_kernel" are for the 50% KDE produced using the LSCV method of bandwidth selection. The 4 files beginning with "GPS_A_95_href_kernel" are for the 95% KDE produced using the href method of bandwidth selection.

The kernel home range code also produces a results matrix that shows summary data for all home range polygons created. This file can be found in the Environment tab in RStudio and will be named "results". The results matrix contains ID name, type of home range created (HR_TYPE), percent density contour (KERNRL_LEVEL), the number of points (POINTS), the area of the home range in hectares (AREA_HA), the smoothing parameter selected (h_value), and the method used to select the smoothing parameter (h_meth) (Fig. 11).



	ID	HR_TYPE	KERNEL_LEVEL	POINTS	AREA_HA	h_value	h_meth
1	GPS_A	KERNEL	50	31	6.62551711425781	28.9312611853917	LSCV
2	GPS_B	KERNEL	50	43	2.43628580322266	21.3963273350323	LSCV
3	GPS_C	KERNEL	50	42	33.7155131347656	71.5509728317923	LSCV

Figure 11: Example of output results file generated creating kernel home ranges at the 50% density contour, using the LSCV method of bandwidth selection.

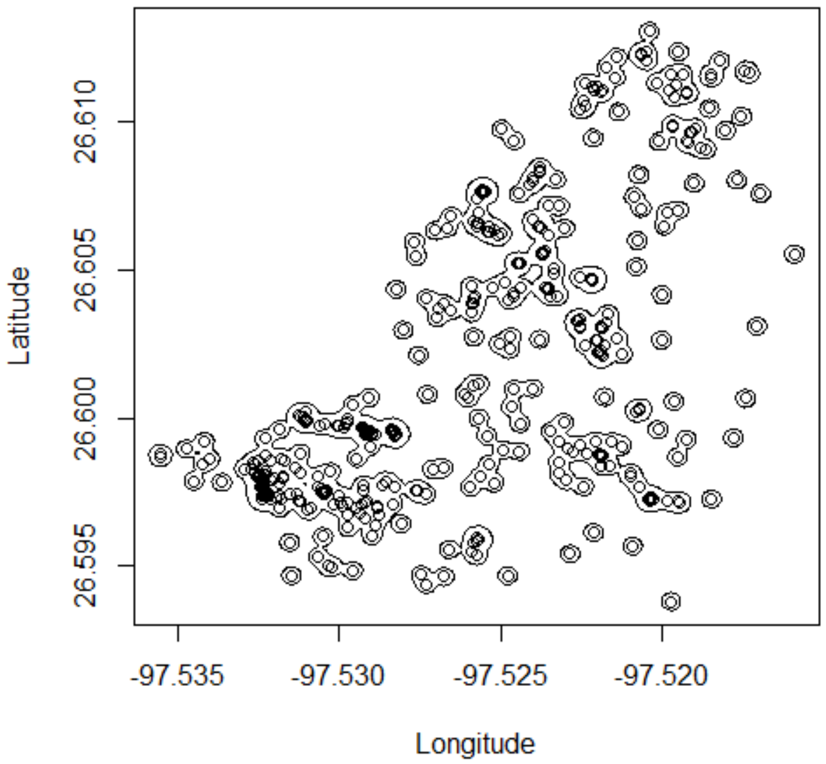


Figure 12: Example of 95% Kernel density home range created using the Least Squares Cross Validation (LSCV) procedure for bandwidth selection. This figure shows the output that the "Create_Kernel" R file sends to the "Plots" tab in RStudio.

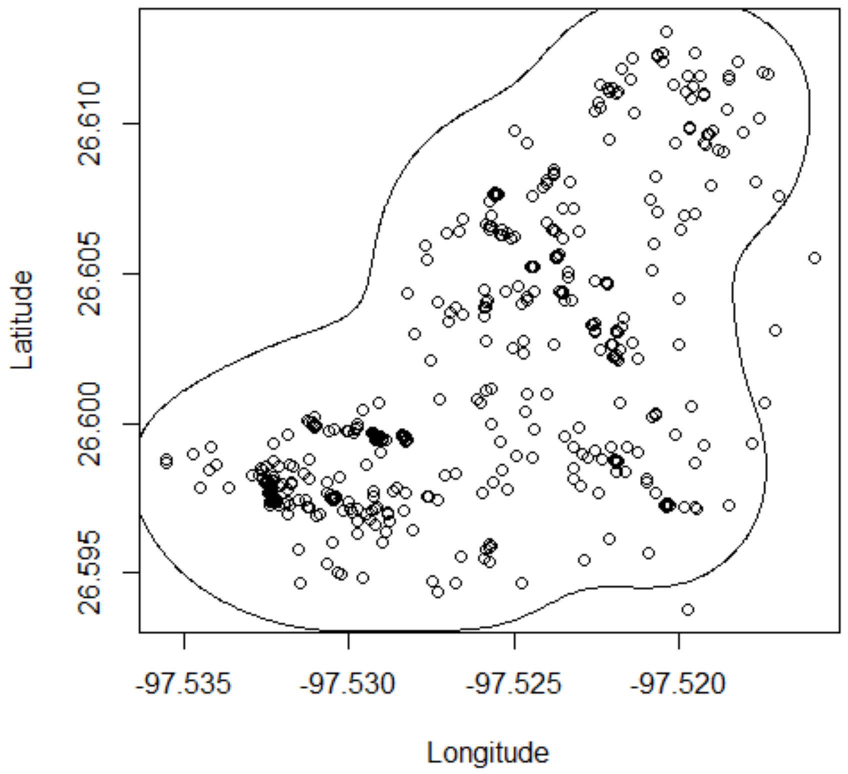


Figure 13: Example of 95% Kernel Density home range produced using the reference bandwidth (href) method of bandwidth selection. Note that although the home range contours are truncated by the extent of the plot area the shapefiles produced will be complete.

One quirk to generating kernel home range estimates with the above method is that you will always see the following warning message:

```
Warning messages:
1: In kernelUD(newlist[[i]], h = h, kern = "bivnorm", grid = grid) :
  xy should contain only one column (the id of the animals)
id ignored
```

This warning message is caused because we used Spatial Points Data Frames, rather than Spatial Points objects, to create the home ranges in adehabitatHR. I have experimented with this extensively, and have come to the conclusion that there is absolutely nothing wrong with using Spatial Points Data Frames with these functions. If adehabitat, rather than adehabitatHR, is loaded into R however, it will produce an error message rather than a warning message. Error messages indicate that the function did not run as intended and that something is seriously wrong. Warning messages indicate that the function did run successfully, but that there may be one or two irregularities worth examining before proceeding.

MAKING KERNEL HOME RANGES IN RHR

The package “adehabitatHR” is one of the most popular R packages for home range analysis, but there are other options available. A relatively new package is “rhr”, which stands for “Reproducible Home Ranges”. This package allows you to generate kernel home ranges in a similar manner as adehabitatHR, but also allows additional methods of bandwidth estimation. The code block titled “Create KDE rhr” through the objects in “newlist” in a similar way as the other code blocks. User defined input consists of the following 3 lines:

```
per = 50
hmeth = rhrHpi # select one (no quotes): rhrHref, rhrHlscv, rhrHpi,
rhrHrefScaled
h_meth = "rhrHpi" # Same as above but with quotations. You must specify both.
```

As with the previous code block, “per” specifies the percent density contour you wish to draw (e.g. 50% KDE, 95% KDE). The values “hmeth” and “h_meth” specify the method used to select bandwidth. The methods selected (rhrHref, hrhHlscv, rhrHpi, and rhrHrefScaled) need to be identical for both “hmeth” and “h_meth”, however, quotation marks must be placed around the name for “h_meth” but not for “hmeth”.

After this has been accomplished, highlight the code and click Run. In the Plot window, two different plots will be created for each individual. The first will show the KDE as a raster with the specified kernel level indicated by a solid black line and animal relocation points given as small crosses (Fig. 14). The second plot will show only the isopleths drawn for whichever kernel density contour (per) was selected (Fig. 15). The isopleth will be exported as a shapefile, complete with projection, to the working directory in an identical manner as in the previous code block.

The results file for this code block is similar to that created using the adehabitatHR package, but with one additional field. Because some of the bandwidth selection methods rely on two parameters, making reproducible home ranges requires two values to be saved for each home range estimate. These are given as “h_value1” and h_value2”. For methods that require only one value (e.g., rhrHref) both fields will be present in the results file but they will be identical to each other. For methods that require two values (e.g., rhrHpi) the two values will be different. The results file will appear as a matrix in the Environment window with the name “rhr_results” (Fig. 16).

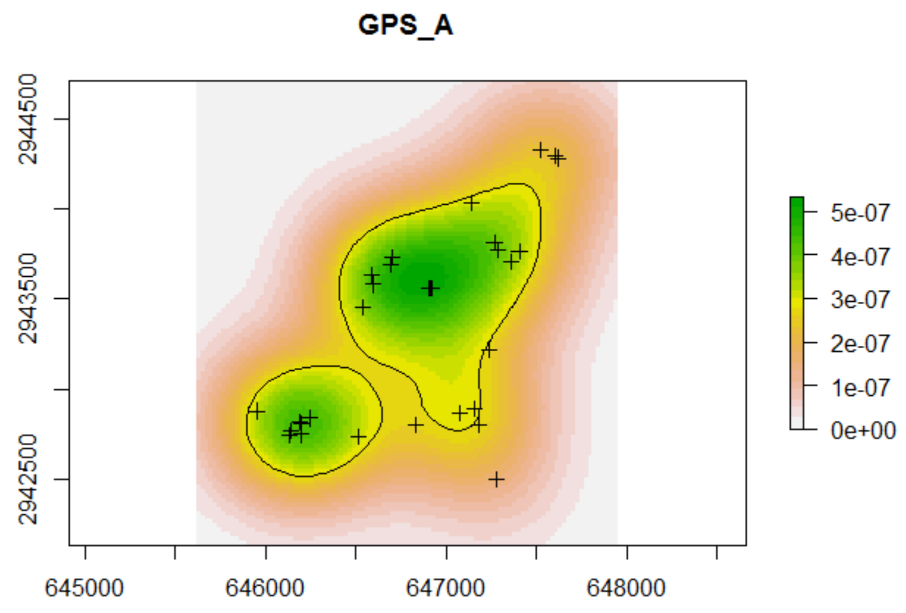


Figure 14: Kernel density estimate as a raster file with 95% density isopleth drawn in black and animal relocations shown as black crosses.

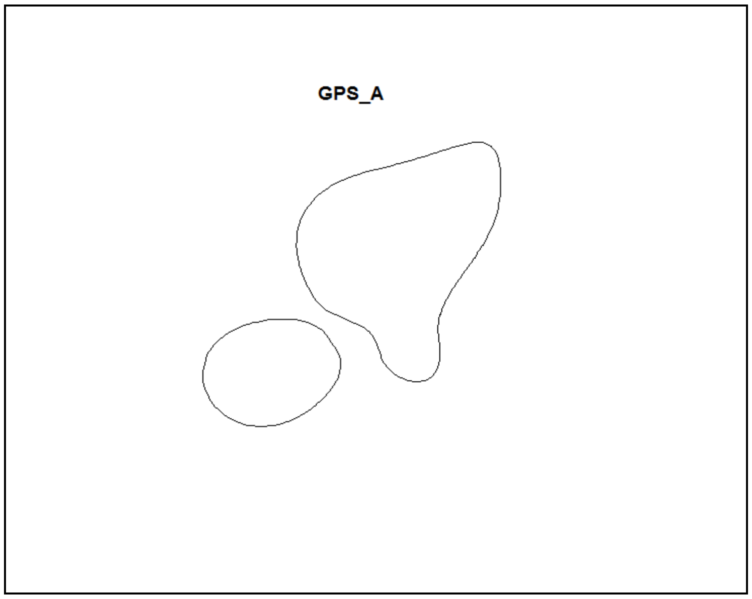
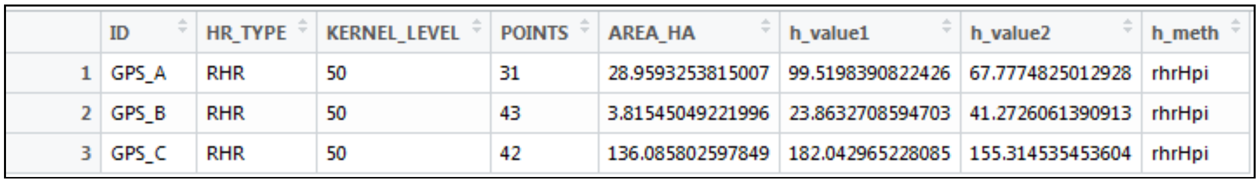


Figure 15: Kernel density estimate, drawn at the 95% level.



	ID	HR_TYPE	KERNEL_LEVEL	POINTS	AREA_HA	h_value1	h_value2	h_meth
1	GPS_A	RHR	50	31	28.9593253815007	99.5198390822426	67.7774825012928	rhrHpi
2	GPS_B	RHR	50	43	3.81545049221996	23.8632708594703	41.2726061390913	rhrHpi
3	GPS_C	RHR	50	42	136.085802597849	182.042965228085	155.314535453604	rhrHpi

Figure 16: Results file created using package "rhr" and the "rhrHpi" method of bandwidth selection. Note that "h_value1" and "h_value2" are different because two unique values are required for this method of bandwidth selection.

ADDING FIELDS TO GPS LOCATIONS FILES

In addition to generating home range shapefiles, there may be fields that you want to add to your original GPS locations files. For example, if your GPS collars collect coordinates in Latitude Longitude format, you may want to re-project them as UTM coordinates and display both coordinate systems as columns in the input file. Additionally, you may want to convert time zones to display date and time both in Coordinated Universal Time and in the local time zone. Perhaps you want a robust and ecologically-meaningful way of separating night from day using the sun's azimuth above the earth.

The code block “Create New Fields” is the most customizable code block in this manual. The functions I included are mainly intended to provide examples of the types of operations that can be automated using R. I would encourage anyone with a bit of experience in R to play around with this to generate new fields for their particular datasets. The basic process for adding a new field to all location files is to insert a line into the “for” loop, somewhere between lines 33 and 49. The name of the new field appears to the right of the “$”, after “infile”. The arguments for the function you are running will be located to the right of the “<-“ on that line. It takes some practice to get this working for complicated functions, but it is worth learning.

Before using this code, a few important points need to be made. First, the function I created to calculate travel distance uses the Pythagorean theorem. Therefore, it can only be used with Cartesian coordinate systems, such as UTM. If you re-project points from Latitude-Longitude to UTM, the code will work fine without modification. Secondly, the function that calculates sun azimuth requires, as input, the approximate latitude and longitude of your study area in decimal degrees. Usually, reporting the coordinates of some point near your study area (e.g., closest town) is good enough. Specify the latitude and longitude you would like to use under “ref_lat” and “ref_long” respectively, in the user defined input portion of the code block.

User defined input is relatively simple for this R file. Specify the name of the x axis for your new coordinate system under “newx”. Specify the name of the y axis for your new coordinate system under “newy”. Specify the new time zone code under “new.timezone”. Specify the output projection under “output.projection”. This step is actually somewhat redundant if you already used the code block “Re-project Points to UTM”, however, I included it in case the user wants to change points to some other coordinate system.

BROWNIAN BRIDGE MOVEMENT MODELS

After you run the code block "Create New Fields", you should have two lists in your working environment. The first list is "newlist", which consists of a list of Spatial Points Data Frames projected in whatever coordinate system was selected at the "Reprojecting Points" step. The second list is called "newlist_df" and it consists of a list of data frames (not Spatial Points Data Frames) populated with all fields specified in the "Create New Fields" code block. Many of the functions that were added to the "Create New Fields" code block are just to demonstrate the cool things you can do with R. A few of the new fields, however, are necessary for creating Brownian bridge movement models. Specifically, coordinates in UTM (or some other Cartesian coordinate system) are necessary, as is a field specifying the time lag in minutes between consecutive GPS fixes. The Brownian bridge movement model is an approach to generating utilization distributions that is specifically intended for auto-correlated location data. It is used when GPS fixes are taken at such high frequencies that consecutive points cannot be considered independent. Of the sample files in Appendix B, only the files "GPS_D" and "GPS_E" have this structure. The Brownian bridge movement model is inappropriate for files "GPS_A", "GPS_B" and "GPS_C", however, you can still create Brownian Bridges from these files if you are interested in seeing how the results differ from those created for truly auto-correlated data.

I would recommend the following steps for using the "Create BBMM" code block for the first time. First, save the sample data files named "GPS_D" and "GPS_E" as csv files in a file folder with no other files (i.e., remove "GPS_A", "GPS_B", and "GPS_C"). Next, run through the following code blocks: "Create List of Spatial Points Data Frames", "Re-project Points to UTM", and "Create New Fields". Don't worry about creating MCP or KDE home range estimates or plotting points to remove outliers for these sample data sets. Next, load the code block titled "Create BBMM" into the Source browser notebook.

The options involved in creating Brownian bridge movement models are considerably more complicated than those for any of the previous steps. The function "brownian.bridge" estimates probability of use across a spatial grid of points based upon the locations of animal relocation points and the time lag between these points. In order for this function to work, you will need to specify this grid of points. There are two ways to do this: the first is to specify only the cell size and to allow the BBMM package to estimate the extent of the availability grid, the second is to manually create a grid of points and tell the function to estimate probability of use for each point in this grid. The first method requires less user input, but it often produces an availability grid that is too small to capture the full utilization distribution. I recommend using both methods on the sample data files before using this code block on your own data. To do this, modify the following lines of user-defined input:

```
X = "UTM_X" # The column name of your x coordinate
Y = "UTM_Y" # The column name of your y coordinate
loc.error = 20
maxlag = 60
levels = 95
cell.size = 10 # change to NULL if area.grid is used
grid = NULL # if area.grid is created Null status will be removed
```

The values “X”, and “Y” should be self-explanatory. The value “loc.error” refers to the location error expected for your relocation points. It can either be a single value or a vector of values. The value “maxlag” is the maximum time lag in minutes that is to be allowed for Brownian bridge calculation. The value “levels” is the percent contour to be drawn around points in the availability grid. It is analogous to the density contour in a kernel home range. The value “cell.size” is the size of each point in the grid. Smaller values of “cell.size” will result in a higher-resolution picture of utilization distribution, but increase computation time. The value “grid” is the name of the availability grid to be used if you are manually specifying an availability grid. If you want to specify only “cell.size” and have the function estimate the availability grid for you, this value needs to be NULL.

Next, if you want the package to estimate the availability grid for you, skip all the lines of user-defined input below the line of hashtags (#). Specifically, you will be skipping these lines:

```
# Optional user defined input. Skip if "cell size" method of grid creation is
used.
##############################################################################
LeftX=646000
RightX=648000
TopY=2945000
BottomY=2942000
size=10

xcoords<-seq(from=LeftX, to=RightX, by=size)
ycoords<-seq(from=BottomY, to=TopY, by=size)
x<-rep(xcoords, times=length(ycoords))
y<-rep(ycoords, each=length(xcoords))
grid<-data.frame(x,y)
cell.size = NULL   # over-rides earlier cell size if run
##############################################################################
```

The above code generates a uniform rectangular grid of points. It is actually a very useful bit of code for various operations and it is worth playing around with to understand how it works. Basically, the values "LeftX", "RightX", "TopY", and "BottomY" specify the minimum X value, maximum X value, maximum Y value, and minimum Y value respectively. The value "size" specifies how far apart (in meters) you want the points to be. Note that the last line of code in this block re-sets the value "cell.size" to NULL. This is because the "brownian.bridge" function requires *either* cell size or an availability grid (but not both) to be specified.

After skipping the above code, highlight everything below the second line of hashtags and click run. Depending on the number of files you have and the cell size you specified, this could take a while to run. Eventually, you should see plots similar to Figures 16 and 17 appearing in your Plots window.

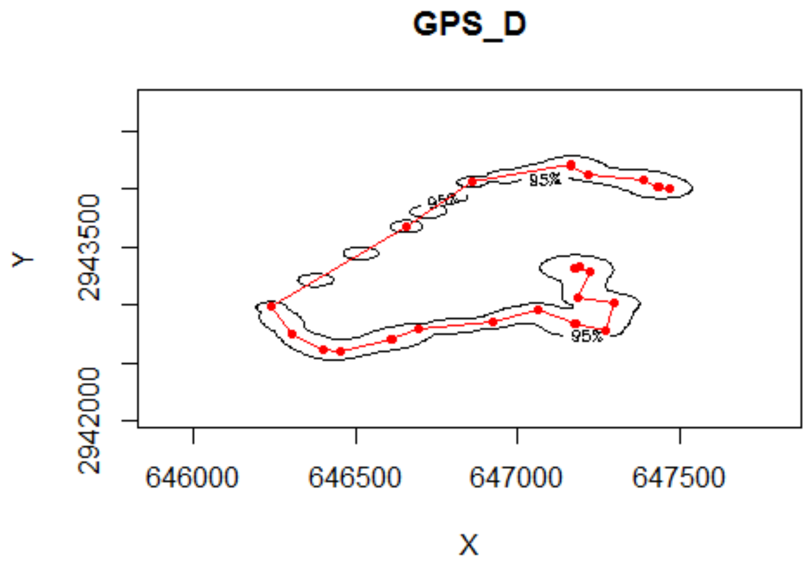


Figure 16: Plot of a Brownian bridge movement model with 95% probability contour drawn in black, movement trajectory represented by a red line, and relocation points represented by red dots. This movement trajectory was adequately captured by the availability grid created by BBMM.

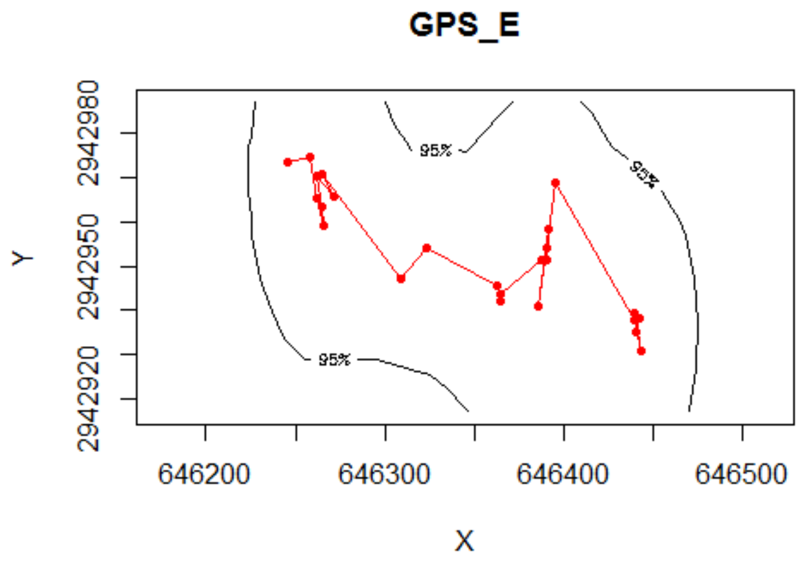


Figure 17: Plot of a Brownian bridge movement model with 95% probability contour drawn in black, movement trajectory represented by a red line, and relocation points represented by red dots. This movement trajectory was not adequately captured by the availability grid created by BBMM.

Notice that the plot for “GPS_E” is cut off by the margins of the plot image. This is not simply a quirk of the R plotting function that will go away once we generate shapefiles (as was the case with kernel polygons). Any shapefiles exported using this function will also be cut off. The problem lies in the method used to specify the availability grid. Usually, specifying only the cell size and allowing the BBMM package to generate the availability grid works fine. In this case, however, the availability grid is not large enough to capture the full 95% probability contour.

In cases like this, we will need to specify the availability grid manually. Just looking at Figure 17, we could probably guess what our maximum and minimum X and Y coordinates should be to capture the full utilization distribution. The code provided specifies a minimum x value of 646000, a maximum x value of 648000, a maximum y value of 2945000, and a minimum y value of 2942000. To manually specify an availability grid with this extent, we simply highlight the entire “Create BBMM” code block, including the section bracketed by hashtags, and click run. This should produce plots similar to Figures 18 and 19.

Figure 19 doesn’t look too impressive since the availability grid created was too large. However, this won’t affect the shapefile generated for this probability contour. Unless you are planning on using the plots created in R for a publication or poster, I wouldn’t worry about this too much. You can simply export the shapefile using this code block and view it in ArcGIS.

The “Create BBMM” code block generates 3 output files per individual. The first is a Spatial Lines shapefile showing whichever probability contour was specified under the value “levels”. As with all other shapefiles generated in this manual, this actually consists of 4 separate files, one of which specifies the projection. The second file is a points shapefile showing points in the availability grid with probability of use given in the “z” column. Points with probability of use $\leq$ 0.00000001 are removed. These points can be viewed in ArcGIS and color coded according to the “z” column to give a picture of the internal anatomy of the home range (Fig. 20). The third file is a csv file showing the same values as the points shapefile, but lacking spatial projection information.

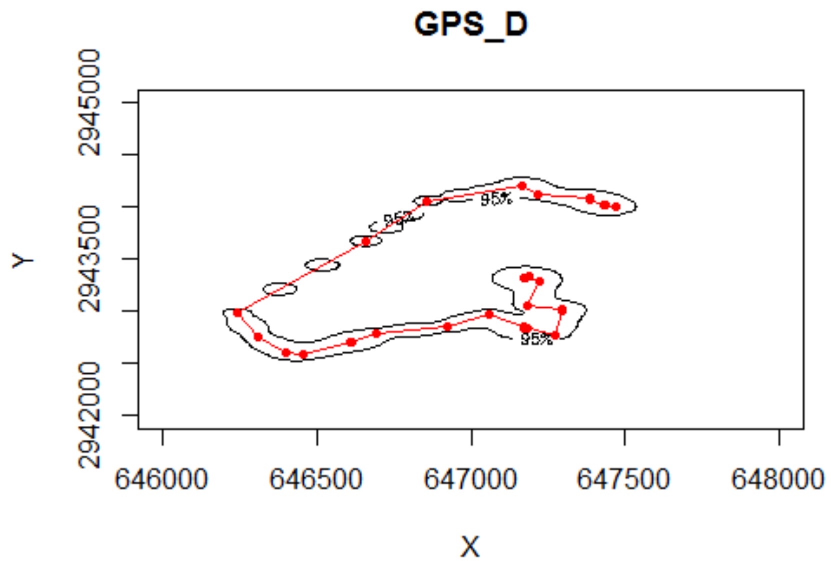


Figure 18: Plot of a Brownian bridge movement model with 95% probability contour drawn in black, movement trajectory represented by a red line, and relocation points represented by red dots. Availability grid was specified manually.

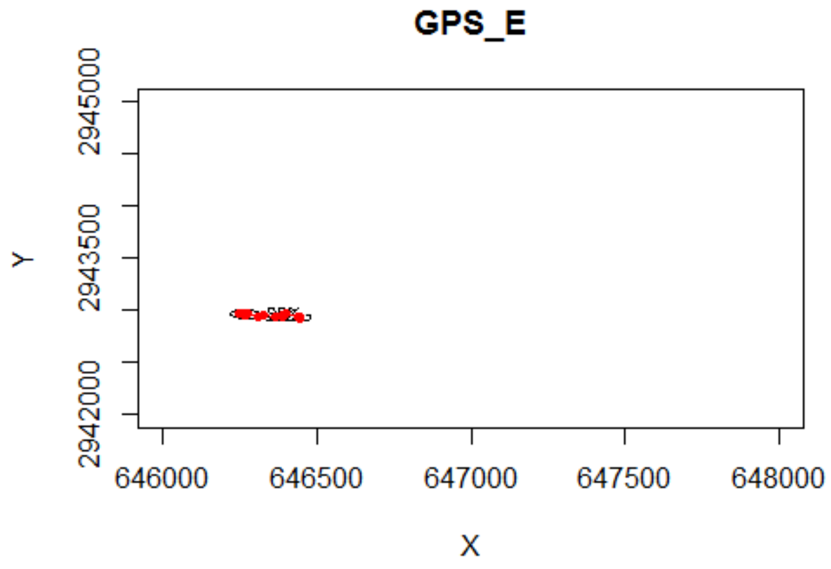


Figure 19: Plot of a Brownian bridge movement model with 95% probability contour drawn in black, movement trajectory represented by a red line, and relocation points represented by red dots. Availability grid was specified manually.

Both the Spatial Lines polygons showing probability contours (e.g., 50% and 95%) and the point grid shapefiles can be viewed in ArcGIS. The points in particular can provide an interesting picture of animal space use since they show how probability of use varies throughout the animal's home range. The "z" values in these files specify probability of use, and you can make a color-coded representation of an animal's utilization distribution by setting a color scale to this variable (Fig. 20).

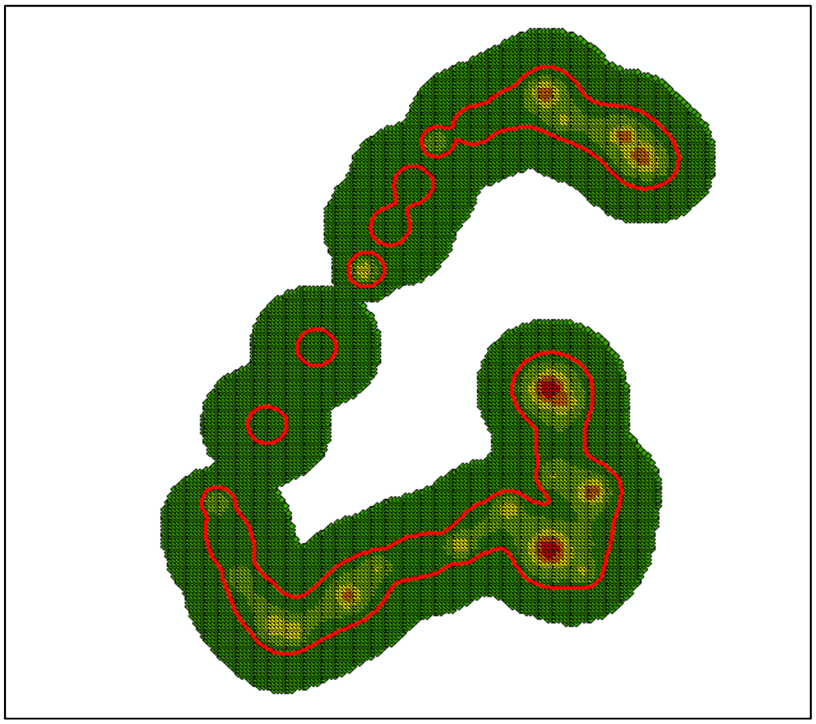


Figure 20: Map produced in ArcGIS showing Brownian bridge points shapefile and Brownian bridge 95% probability contour. The 95% probability contour is shown by the solid red line. Points that are colored red have a higher probability of use than points colored yellow or green.

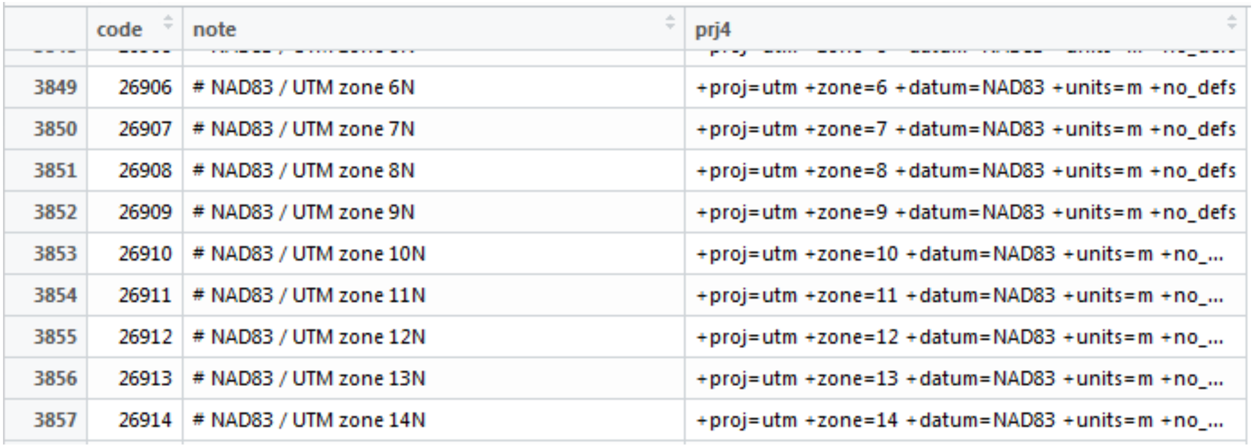
FINDING CODES FOR COORDINATE REFERENCE SYSTEMS

Specifying the Coordinate Reference System (CRS) is probably the trickiest part to using R for spatial data analysis. Specifying the CRS requires that packages "rgdal" and "sp" be loaded. Many vignettes for packages relating to spatial ecology (e.g., adehabitatHR) omit any discussion of CRS. You can generate home range estimates and study animal movement using such packages without considering CRS, but eventually this will cause problems. If you export your data to shapefiles, and later view these shapefiles in ArcGIS, they might appear far from your study area. The best way to analyze actual wildlife telemetry data is to specify the correct CRS from the beginning, and use the function "spTransform" to transform these as needed. An explanation of how to determine the CRS of your data can be found at: https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/OverviewCoordinateReferenceSystems.pdf.

Hopefully, you either know the coordinate reference system used for your data or can easily find it. If your x any y coordinates are given in decimal latitude longitude coordinates, you will probably use WGS84, which is specified by the following code: "+init=epsg:4326". You can make a data frame of all CRS codes recognized by R using the following command:

```
EPSG<-make_EPSG()
```

This produces a data frame with 5,078 rows showing all CRSs recognized by R (Fig. 21). If you are trying to transform your coordinates to an uncommon CRS you can search this data frame to find the code needed. Most likely, your GPS data will be delivered as decimal latitude and longitude coordinates, and you will be transforming it to Universal Transverse Mercator (UTM).



	code	note	prj4
3849	26906	# NAD83 / UTM zone 6N	+proj=utm +zone=6 +datum=NAD83 +units=m +no_defs
3850	26907	# NAD83 / UTM zone 7N	+proj=utm +zone=7 +datum=NAD83 +units=m +no_defs
3851	26908	# NAD83 / UTM zone 8N	+proj=utm +zone=8 +datum=NAD83 +units=m +no_defs
3852	26909	# NAD83 / UTM zone 9N	+proj=utm +zone=9 +datum=NAD83 +units=m +no_defs
3853	26910	# NAD83 / UTM zone 10N	+proj=utm +zone=10 +datum=NAD83 +units=m +no_...
3854	26911	# NAD83 / UTM zone 11N	+proj=utm +zone=11 +datum=NAD83 +units=m +no_...
3855	26912	# NAD83 / UTM zone 12N	+proj=utm +zone=12 +datum=NAD83 +units=m +no_...
3856	26913	# NAD83 / UTM zone 13N	+proj=utm +zone=13 +datum=NAD83 +units=m +no_...
3857	26914	# NAD83 / UTM zone 14N	+proj=utm +zone=14 +datum=NAD83 +units=m +no_...

Figure 21: Screen capture of a small portion of the data frame of all possible CRSs available in R. The field "code" specifies the epsg code which can be used to specify the CRS (e.g., "+init=epsg:26908" specifies NAD83/ UTM zone 8N). The field "note" describes the CRs, and the field "prj4" lists all the arguments for that particular CRS.

Scrolling through the data frame of possible CRSs for R can be overwhelming, but there are ways to narrow this down substantially. Let's say I wanted to subset this data frame to show only those records with "NAD83" included somewhere in the notes. This would be the syntax for doing so:

```
tmp<-EPSG[grep("NAD83", EPSG$note),]
```

This code creates a new data frame called “tmp” that shows only those rows with “NAD83” in the notes. You don’t need to understand too much about how this code works, but if you play around with it you can see that you can further subset the data frame by modifying the code above. For example:

```
tmp2<-EPSG[grep("proj=utm", tmp$prj4),]
```

This creates a new data frame called “tmp2” that selects rows from “tmp” that have the text “proj=utm” somewhere in the prj4 column. If you are transforming latitude/longitude coordinates to UTM in South Texas, I would recommend keeping the values given under “input.projection” and “output.projection” exactly as they are in all code blocks given in Appendix A.

SPECIFYING DATE AND TIME FORMAT

The code blocks in this manual treat date and time as a single date/time field in POSIXct format with relevant time zone information. You don’t really need to know what POSIXct format is to get the code to work, but you should be able to specify the correct date/time format so that the time lag field is calculated correctly and so you can use packages to accurately calculate things like moon phase and sun angle. This manual uses the function “parse_date_time” in the R package “lubridate” to combine date and time fields into a single field and assign it the correct POSIXct format. I would recommend taking a look at the “lubridate cheat sheet” available at http://blog.yhat.com/static/pdf/R_date_cheat_sheet.pdf. In short, you want to make sure that you specify the correct format under “datetime.code” in the user defined input portion of “Create List of Spatial Points Data Frames”. The format provided for the sample data is “dmy_HMS” which means that day comes first, followed by month, year, (a space), hour, minute, and second. If you were to accidentally change this to “mdy_HMS” your dates would be incorrect.

A lot of GPS collars provide date and time in both UTC and the local time zone. In my experience, however, these files often to not correctly account for daylight savings time. Unless you are confident that the local time field in your input locations files correctly accounts for daylight savings time, I would recommend discarding the local time zone field provided by the GPS collars and recalculating this on your own using “lubridate”. This is fairly straightforward and is actually done for you in the code block “Create New Fields”. You simply have to specify the correct time zone code in the user defined input section under “new.timezone”. The time zone code provided in the example data (“CST6CDT”) is Central time, with daylight savings time included.

TROUBLESHOOTING

My biggest criticism of R is that the error messages are extremely difficult to interpret. A common mistake R users make is failing to specify the correct file pathway to the input data. Let’s say I used back slashes instead of forward slashes under “input” for the code block “Create List of Spatial Points Data Frames”. In other words, I typed `“F:\R\R for GPS collar data\Test”` instead of `“F:/R/R for GPS collar data/Test”`. This would produce the following error message:

```
Error in datalist[[i]] : subscript out of bounds
```

This is not exactly helpful. Rather than try to google specific error messages to figure out what is going on, I would recommend double-checking both the input files and the user defined input code of each code block to make sure they are as similar as possible to the examples. If you have files with different column headings for x and y coordinates, for example, this will definitely cause problems.

Another common problem is not having the required packages loaded. The error messages in this case will be a little easier to interpret. Let's say I did not have the package "sp" installed when I ran "Create List of Spatial Points Data Frames". This produces the following error message:

```
Error in coordinates(xy) <- c(x, y) : could not find function "coordinates<-"
```

What this error message means is that you tried to use the function "coordinates" that was not available in your working environment. It was not available because the package containing this function (sp) was not loaded.

Having random junk in your input data files can also cause major problems. If a column is meant to contain data in a specific format (e.g., numerical), including just a single cell containing a text string (e.g., "unknown", or "missing") will throw a wrench in things. It is usually ok to have NA operators in your data, as R is able to recognize common NA operators, however, you shouldn't mix and match NA operators. For example, you don't want to have some cells with "NA" and others with "N/A" or "na". The result of random junk in your input files is often that R will read a specific field as a factor rather than as a character or numerical field. This can cause problems if, for example, "Latitude" is meant to be a numerical value but R recognizes it as a factor. This will make it impossible to perform any mathematical operations on this field and will prevent you from being able to plot points or create Spatial Points Data Frames.

APPENDIX A

Install Required Packages

```
# Install required packages
# Run this code before using any other code blocks

usePackage <- function(p)
{
  if (!is.element(p, installed.packages()[,1]))
    install.packages(p, dep = TRUE)
  require(p, character.only = TRUE)
}


pkg<-c("sp","lubridate", "rgdal", "maptools",  "adehabitatHR", "rgdal", "rast
er", "BBMM", "lunar", "oce", "move")
lapply(pkg, usePackage)
lapply(pkg, library, character.only=TRUE)
install.packages("rhr", repos=c("http://78.47.85.98/R",
                                "http://cran.rstudio.com/"), dep = TRUE)
library(rhr)
```

Create List of Spatial Points Data Frames

```
# Create List of Spatial Points Data Frames
# Animal locations must be saved as separate csv files in a single folder
# Names for x and y coordinates and Date and Time fields must be identical
# Created by John Leonard 08_23_2016

# User defined input. Modify lines below.
########################################################################
input = "J:/R/R for GPS collar data/Test Collar Data"
x = "Longitude" # The column name of your x coordinate
y = "Latitude" # The column name of your y coordinate
datetime.code = "dmy_HMS" #modify this using documentation in Lubridate
Datecolumn= "GMT.Date"
Timecolumn= "GMT.Time"
timezone = "UTC"
input.projection = "+init=epsg:4326"

na.operator = "N/A"
#######################################################################

# Load required packages
require(sp)
require(lubridate)

# Set working directory and create empty list to store csv files.
setwd(input)
datalist <- list()
files <- list.files(input)
files

# Read in CSV files.
for(file in files) {
  stem <-gsub("\\.csv$","",file)
  datalist[[stem]] <- (read.csv(file, stringsAsFactors=FALSE, na.strings=na.o
perator))
}

# Loop through csv files and create a list of SpatialPointsDataFrames
newlist<-list()
#for loop for going through each file
for (i in 1:length(datalist)){
  infile = datalist[[i]]
  infile = subset(infile, infile[,x]!="NA")
  infile = subset(infile, infile[,y]!="NA")
  Date = infile[,Datecolumn]
  Time = infile[,Timecolumn]
  infile$Date.Time<-parse_date_time(paste(Date, Time), datetime.code,
                                  tz=timezone)
  xy = infile[,c(x, y)]
```

```r
  coordinates(xy)<-c(x,y)
  proj4string(xy)<-CRS(input.projection)
  infile2<-SpatialPointsDataFrame(xy, infile)
  name = names(datalist[i])
  newlist[[i]]<-infile2
  names(newlist)[i]<-name

}

rm(Date, file, files, i, infile2, name, Time, xy, Datecolumn, input,
    pkg, stem, Timecolumn, timezone,usePackage) # remove unwanted objects
```

Plot Points to Remove Outliers

```
# Plot all points to remove outliers
# To be used on the "newlist" object
# Simple Version. Only removes outliers by coordinates
# Created by John Leonard 08_23_2016

for (i in 1:length(newlist)){
  plot(newlist[[i]]@coords, main=names(newlist[i]))
}

# First make duplicate of newlist called newlist2 just to be safe

newlist2<-newlist

# Modify code below to select maximum or minimum x or y values
# Code below makes "27" the maximum latitude value for GPS_A
newlist2$GPS_A<-subset(newlist2$GPS_A, newlist2$GPS_A@coords[,y]<27)

# Code below makes "-97.55" the minimum longitude value for GPS_B
newlist2$GPS_B<-subset(newlist2$GPS_B, newlist2$GPS_B@coords[,x]>-97.55)
newlist2$GPS_C<-subset(newlist2$GPS_C, newlist2$GPS_C@coords[,y]>26.55)
# Plot again to see if all outliers have been removed

for (i in 1:length(newlist2)){
  plot(newlist2[[i]]@coords, main=names(newlist2[i]))
}

# If it looks good, convert newlist2 back into newlist.
# Note: This cannot be undone!
newlist<-newlist2
```

Re-project Points to UTM

```
# Re-project Points to UTM
# Created by John Leonard 08_23_2016

# User defined input. Modify line below.
######################################################################
output.projection = "+proj=utm +zone=14 +datum=NAD83"
######################################################################

# load required packages
require(sp)
require(lubridate)
require(rgdal)

#for loop for going through each file
for (i in 1:length(newlist)){
  infile = newlist[[i]]
  infile = spTransform(infile, CRS(output.projection))
  name = names(newlist[i])
  newlist[[i]]<-infile
  names(newlist)[i]<-name
}

rm(i, name) # remove unwanted objects
```

Export Points

```
# Export Points
# Export points as shapefiles with projection
# Note: All POSIXct fields must be converted to character
# Created by John Leonard 08_23_2016


for (i in 1:length(newlist)){
  infile<-newlist[[i]]
  infile$Date.Time<-as.character(infile$Date.Time)
  name=names(newlist[i])
  writePointsShape(infile, name, factor2char=TRUE)
  cat(showWKT(proj4string(newlist[[i]])), file=paste0(name,".prj"))
}

rm(i, name) # remove unwanted objects
```

Create MCP

```
# Create MCP
# Plots Minimum Convex Polygon Home Ranges and exports shapefiles
# Spatial coordinates should be UTM for correct area calculation
# Created by John Leonard 08_23_2016

# User defined input. Modify line below.
##################################################################
per= 100 # percent MCP to create (e.g. 95, 100, etc.)
##################################################################

# Load required packages
require(sp)
require(maptools)
require(adehabitatHR)
require(rgdal)


# Initialize Matrix to store results
mcpresults = matrix(nrow=0, ncol=5)
colnames(mcpresults)<-c("ID", "HR_TYPE", "MCP_LEVEL", "POINTS", "AREA_HA")

for (i in 1:length(newlist)){
  xy.mcp = SpatialPoints(newlist[[i]]@coords)
  mcp.out = mcp(xy.mcp, percent=per, unout="ha")
  plot(xy.mcp, main= names(newlist[i]))
  plot(mcp.out, add=TRUE)
  name=names(newlist[i])
  area=mcp.out$area
  row=nrow(newlist[[i]])
  mcpresults=rbind(mcpresults,c(name, "MCP",per, row, area))
  writePolyShape(mcp.out, paste0(name,"_",per,"_MCP"), factor2char=TRUE)
  cat(showWKT(proj4string(newlist[[i]])), file=paste0(name,"_",
                                                       per,"_MCP.prj"))
}

rm(area, i, mcp.out, name, per, row, xy.mcp)
```

Create KDE adehabitatHR

```
# Create KDE
# Create and export kernel home ranges with projection
# Important!!! Use adehabitatHR NOT adehabitat.
# Created by John Leonard 08_23_2016

# User defined input. Modify lines below.
#######################################################################
per =95 # percent utilization distribution estimated
h="href" # bandwidth setting ("href", "LSCV", or user specified number)
grid=1000
#######################################################################

# Load required packages
require(sp)
require(maptools)
require(adehabitatHR)
require(rgdal)

# Initilizing matrix for storing results
results = matrix(nrow=0, ncol=7)
colnames(results)<-c("ID", "HR_TYPE", "KERNEL_LEVEL", "POINTS", "AREA_HA",
                     "h_value", "h_meth")
for (i in 1:length(newlist)){
  name= names(newlist[i])
  ud<-kernelUD(newlist[[i]],h=h, kern="bivnorm", grid=grid)
  ver=getverticeshr(ud,per)
  area=ver$area #get homerange polygon area
  h_value=ud@h$h
  h_meth=ud@h$meth
  row=nrow(newlist[[i]])
  plot(newlist[[i]]@coords, main= names(newlist[i]))
  plot(ver, add=TRUE)
  writePolyShape(ver,paste0(name,"_",per,"_",h,"_kernel"), factor2char=TRUE)
  cat(showWKT(proj4string(newlist[[i]])),
    file=paste0(name,"_",per,"_",h,"_kernel.prj"))
  results=rbind(results,c(name, "KERNEL",per, row, area, h_value, h_meth))
}

rm(area, grid, h, h_meth, h_value, infile,
    name, per, row, ud, ver) # remove unwanted objects

# Didn't work? Make sure adehabitatHR and NOT adehabiat is loaded!
```

Create KDE rhr

```
# Creating Kernel Home Ranges with rhr
# Creates and exports kernel density isopleths as shapefiles
# Allows greater flexibility in selecting bandwidth
# Created by John Leonard 08_23_2016

# User defined input. Modify lines below.
############################################################################
per = 50
hmeth = rhrHpi # select one: rhrHref, rhrHlscv, rhrHpi, rhrHrefScaled
h_meth = "rhrHpi" # Same as above but with quotations. You must specify both.
############################################################################

# Load packages
require(rhr)
require(sp)
require(maptools)
require(rgdal)

rhr_results = matrix(nrow=0, ncol=8)
colnames(rhr_results)<-c("ID", "HR_TYPE", "KERNEL_LEVEL", "POINTS",
                         "AREA_HA", "h_value1", "h_value2", "h_meth")
for (i in 1:length(newlist)){
  name= names(newlist[i])
  xy<-SpatialPoints(newlist[[i]]@coords)
  bandwidth= hmeth(xy)$h
  kde<-rhrKDE(xy, h = bandwidth, levels = per)
  name = names(newlist[i])
  row = nrow(newlist[[i]])
  area = (rhrArea(kde, levels = per)$area/10000)
  h_value1=bandwidth[1]
  h_value2=bandwidth[2]
  plot(kde, main = name)
  plot(xy, add=TRUE)
  iso<-rhrIsopleths(kde, levels = per)
  plot(iso, main = name)
  writePolyShape(iso,paste0(name,"_",per,"_",h_meth), factor2char=TRUE)
  cat(showWKT(proj4string(newlist[[i]])),
      file=paste0(name,"_",per,"_",h_meth,".prj"))
  rhr_results=rbind(rhr_results, c(name, "RHR",per,row,
                                   area, h_value1,h_value2, h_meth))
}

rm(area, bandwidth, h_meth, h_value1, h_value2, i,
    iso, kde, name, per, row, xy, hmeth) # remove unwanted objects
```

Create New Fields

```
# Create New Fields
# Populates GPS locations files with new fields and writes to csv files
# "newlist" is a list of Spatial Points Data Frames with projection
# traveldist function only works with Cartesian coordinate system
# newlist_df will be a list of data frames (not Spatial Points Data Frames)
# Created by John Leonard 08_23_2016

# User defined input. Modify lines below.
##########################################################################
ref_lat = 26.55  # Approximate Latitude for sun azimuth calculation
ref_long = -97.42 # Approximate Longitude for sun azimuth calculation
newx="UTM_X"
newy="UTM_Y"
new.timezone="CST6CDT"
output.projection = "+proj=utm +zone=14 +datum=NAD83"
#########################################################################

require(sp)
require(rgdal)
require(lubridate)
require(lunar)
require(oce)

traveldist<-function(x,y){sqrt((append(0,diff(x))^2+append(0,diff(y))^2))}

newlist_df<-list()
#for loop for going through each file
for (i in 1:length(newlist)){
  infile = newlist[[i]]
  infile = spTransform(infile, CRS(output.projection))
  xy = data.frame(coordinates(infile))
  colnames(xy)[1]<-newx
  colnames(xy)[2]<-newy
  infile = data.frame(infile@data)
  infile = cbind(infile, xy)
  infile$Local.Date.Time<-with_tz(infile$Date.Time, tzone=new.timezone)
  infile$Year<-strftime(infile$Date.Time, format="%y")
  infile$Month<-strftime(infile$Date.Time, format="%m")
  infile$Day<-strftime(infile$Date.Time, format="%d")
  infile$Hour<-strftime(infile$Date.Time, format="%H")
  Timelag.sec<-c(NA, (diff(as.numeric(infile$Date.Time))))
  infile$Timelag.min<-Timelag.sec/60
  infile$Traveldist<-traveldist(infile[,newx], infile[,newy])
  infile$Velocity<-infile$Traveldist/infile$Timelag.min
    sun<-sunAngle(infile$Date.Time, longitude = ref_long,
                latitude = ref_lat, useRefraction = TRUE)
  infile$sun.azimuth<-sun$azimuth
  infile$moonphase<-lunar.phase(infile$Date.Time, name=TRUE)
  name = names(newlist[i]) #Create a name based on file name
  newlist_df[[i]]<-infile
  names(newlist_df)[i]<-name

  write.csv = write.table(infile, file = paste0(name,"_modified.csv"),
                        sep = ",",
    col.names = TRUE, row.names = FALSE,qmethod = "double")
}

rm(infile, xy, i, name, newx, newy, ref_lat, ref_long,
    sun, Timelag.sec, write.csv) # Remove unwanted objects
```

Create BBMM

```
# Create Brownian Bridge Movement Models
# Recommended to use this after "Create New Fields"
# newlist_df is list of Data Frames not Spatial Points Data Frames
# Should only be used on high-frequency telemetry files
# Created by John Leonard 08_23_2016

# User defined input. Modify lines below.
##################################################################
X = "UTM_X" # The column name of your x coordinate
Y = "UTM_Y" # The column name of your y coordinate
loc.error = 20
maxlag = 60
levels = c(95)
cell.size = 10 # change to NULL if area.grid is used
grid = NULL # if area.grid is created Null status will be removed
##################################################################

# Optional user defined input. Skip if "cell size"
# method of grid creation is used.
##################################################################
LeftX=646000
RightX=648000
TopY=2945000
BottomY=2942000
size=10

xcoords<-seq(from=LeftX, to=RightX, by=size)
ycoords<-seq(from=BottomY, to=TopY, by=size)
x<-rep(xcoords, times=length(ycoords))
y<-rep(ycoords, each=length(xcoords))
grid<-data.frame(x,y)
cell.size = NULL   # over-rides earlier cell size if run
##################################################################

require(BBMM)
require(raster)
require(maptools)
require(rgdal)


for (i in 1:length(newlist_df)){
  infile = newlist_df[[i]]
  name = names(newlist_df[i])
  BBMM<-brownian.bridge(infile[,X], infile[,Y],
        time.lag = infile$Timelag.min[-1],location.error=loc.error,
        cell.size=cell.size, area.grid=grid)
  x<-BBMM$x[BBMM$probability >= 0.00000001]
  y<-BBMM$y[BBMM$probability >= 0.00000001]
  z<-BBMM$probability[BBMM$probability >= 0.00000001]
  tmp.df<-data.frame(x,y,z)
  contours<-bbmm.contour(BBMM, levels= levels, locations =
        cbind(infile[,X], infile[,Y]),plot = TRUE)
  title(main=name)
  out.raster <- rasterFromXYZ(tmp.df,crs=CRS(output.projection),
                              digits=2)
  raster.contour <- rasterToContour(out.raster,levels=contours$Z)
  writeLinesShape(raster.contour, paste0(name,"_",levels,"_BBMM"),
                  factor2char=TRUE)
  cat(showWKT(proj4string(newlist[[1]])), file=paste0(name,"_",
                              levels,"_BBMM.prj"))
  write.csv(tmp.df, file= paste0(name, "_BBMM.csv"))
  tmp.xy<-tmp.df[,c(1:2)]
  coordinates(tmp.df)<-tmp.xy
  proj4string(tmp.df)<-CRS(output.projection)
  writePointsShape(tmp.df, paste0(name,"BBMM_points"),
                   factor2char=TRUE)
  cat(showWKT(proj4string(newlist[[1]])),
      file=paste0(name,"BBMM_points.prj"))
}

rm(grid, infile, tmp.xy, BBMM, BottomY, cell.size, contours, i, LeftX,     le
vels, loc.error, maxlag, name, out.raster, raster.contour, RightX, size, tmp.
df,TopY, X, x, xcoords, Y, y, ycoords, z) # remove unwanted objects
```

APPENDIX B

GPS_A

```
No,GMT.Date,GMT.Time,Latitude,Longitude
1,04.03.2014,17:17:06,27.5196477,-97.8816508
2,24.03.2015,18:00:54,26.6064405,-97.5266566
3,25.03.2015,6:01:48,26.5989913,-97.5229284
4,26.03.2015,6:02:00,26.5978741,-97.5285523
5,27.03.2015,6:01:29,26.5984423,-97.5253675
6,27.03.2015,18:00:48,26.6059578,-97.5276865
7,28.03.2015,6:01:54,26.5984083,-97.5218732
8,29.03.2015,6:01:49,26.604351,-97.5282482
9,30.03.2015,6:01:50,26.6054602,-97.5276204
10,31.03.2015,6:01:29,26.6094811,-97.5221408
11,31.03.2015,18:03:01,26.606822,-97.5265411
12,01.04.2015,6:01:48,26.611661,-97.5173016
13,02.04.2015,6:02:30,26.5985723,-97.5317179
14,03.07.2015,4:31:00,26.6052205,-97.5244903
15,03.07.2015,5:00:29,26.6052209,-97.5244126
16,03.07.2015,5:30:19,26.605227,-97.5244498
17,03.07.2015,6:01:13,26.6052318,-97.5245095
18,04.07.2015,6:01:29,26.598013,-97.5317434
19,05.07.2015,6:03:35,26.5981892,-97.5323105
20,06.07.2015,6:01:29,26.5979925,-97.5323624
21,07.07.2015,6:02:00,26.5992063,-97.5341875
22,07.07.2015,18:01:30,26.6117546,-97.5174344
23,08.07.2015,6:01:53,26.5992264,-97.5220764
24,09.07.2015,6:01:29,26.6075099,-97.5208761
25,10.07.2015,6:01:29,26.6021617,-97.5212333
26,10.07.2015,18:02:00,26.607033,-97.519495
27,11.07.2015,18:02:00,26.595673,-97.5209136
28,12.07.2015,18:01:18,26.6065042,-97.5199294
29,13.07.2015,6:01:29,26.5988166,-97.5312229
30,13.07.2015,18:02:41,26.607097,-97.5206515
31,14.07.2015,6:01:47,26.6120814,-97.5182165
32,14.07.2015,18:01:00,26.5986079,-97.5318048
```

GPS_B

```
No,GMT.Date,GMT.Time,Latitude,Longitude
1,20.03.2015,15:34:02,26.4821409,-97.748736
2,04.05.2015,9:31:00,26.604591,-97.5015835
3,04.05.2015,10:01:10,26.6046073,-97.5015783
4,04.05.2015,10:30:34,26.604535,-97.5016466
5,04.05.2015,11:00:50,26.6046899,-97.5017766
6,04.05.2015,11:31:09,26.6051792,-97.5020129
7,04.05.2015,12:01:00,26.604178,-97.5025187
8,04.05.2015,12:30:51,26.6041361,-97.501321
9,04.05.2015,13:02:54,26.6046071,-97.5015888
10,04.05.2015,13:30:38,26.6045807,-97.5016312
11,04.05.2015,14:00:24,26.6044976,-97.5016679
12,04.05.2015,14:30:11,26.6045028,-97.5016414
13,04.05.2015,15:00:30,26.6045482,-97.5016254
14,04.05.2015,15:30:44,26.6045768,-97.501622
15,04.05.2015,16:00:21,26.6045811,-97.5016239
16,04.05.2015,16:32:01,26.6045905,-97.5014692
17,05.05.2015,3:30:26,26.6054723,-97.5040255
18,05.05.2015,4:02:04,26.6059761,-97.5042102
19,05.05.2015,4:32:06,26.6049197,-97.5057943
20,18.05.2015,12:31:57,26.6039967,-97.5013199
21,18.05.2015,13:01:20,26.6040002,-97.5015209
22,19.05.2015,0:00:10,26.6046622,-97.501993
23,19.05.2015,0:32:34,26.6052456,-97.5019699
24,19.05.2015,1:01:24,26.606089,-97.5029338
25,19.05.2015,1:31:00,26.6068372,-97.5026996
26,19.05.2015,2:00:26,26.6071276,-97.5024094
27,03.06.2015,2:31:27,26.6039387,-97.501293
28,03.06.2015,3:00:29,26.6048106,-97.4999387
29,03.06.2015,3:31:00,26.6065412,-97.5000181
30,03.06.2015,4:00:13,26.6080513,-97.4999034
31,03.06.2015,4:30:20,26.6088912,-97.5001647
32,03.06.2015,5:00:30,26.6088553,-97.5000693
33,03.06.2015,5:30:23,26.609892,-97.499959
34,03.06.2015,6:03:00,26.6096874,-97.4993201
35,04.06.2015,6:02:49,26.6077894,-97.5002686
36,06.06.2015,6:01:30,26.6096849,-97.499904
37,06.06.2015,18:01:48,26.605011,-97.5026791
38,25.06.2015,6:01:59,26.6107375,-97.4908773
39,27.06.2015,6:02:40,26.6128831,-97.4943688
40,28.06.2015,18:00:54,26.6127302,-97.4941202
41,01.07.2015,6:02:17,26.6146628,-97.4875767
42,01.07.2015,18:00:29,26.6149471,-97.4906366
43,10.07.2015,6:03:00,26.6138451,-97.4892091
44,11.07.2015,18:01:30,26.6150099,-97.4904467
```

GPS_C

```
No,GMT.Date,GMT.Time,Latitude,Longitude
1,20.04.2015,14:33:29,26.4819699,-97.7649551
2,20.04.2015,15:00:16,26.5722701,-97.4482499
3,03.05.2015,6:02:31,26.6183978,-97.482038
4,04.05.2015,6:01:24,26.6184227,-97.481947
5,04.05.2015,6:30:25,26.618533,-97.4820344
6,04.05.2015,7:00:16,26.6184642,-97.4816968
7,04.05.2015,19:30:15,26.6107478,-97.5096767
8,04.05.2015,20:01:24,26.6107796,-97.5097708
9,04.05.2015,20:30:15,26.6106074,-97.5098095
10,04.05.2015,21:00:36,26.6107062,-97.5092381
11,04.05.2015,21:30:30,26.6107829,-97.5099171
12,04.05.2015,22:01:30,26.6107491,-97.5098008
13,04.05.2015,22:30:17,26.6107645,-97.5098402
14,04.05.2015,23:00:19,26.6107744,-97.5097946
15,04.05.2015,23:30:15,26.6108339,-97.5098999
16,05.05.2015,0:01:31,26.6108225,-97.5097099
17,10.05.2015,18:02:48,26.604987,-97.4996657
18,11.05.2015,6:01:47,26.6120679,-97.5019465
19,12.05.2015,6:02:31,26.6195127,-97.4995449
20,12.05.2015,18:01:01,26.6108786,-97.5012745
21,13.05.2015,18:01:18,26.6055012,-97.4944994
22,14.05.2015,6:01:59,26.6132533,-97.4947846
23,31.05.2015,6:02:30,26.6109213,-97.5002294
24,01.06.2015,18:01:57,26.6015498,-97.4935202
25,02.06.2015,6:02:25,26.6353387,-97.4893614
26,02.06.2015,6:30:22,26.6326484,-97.4853938
27,02.06.2015,7:00:25,26.6326469,-97.4796348
28,02.06.2015,7:30:54,26.6293693,-97.4767442
29,01.07.2015,6:03:14,26.6165646,-97.493789
30,01.07.2015,18:01:45,26.6114574,-97.4901754
31,02.07.2015,6:00:29,26.5997675,-97.4931211
32,02.07.2015,6:01:08,26.5997256,-97.4929938
33,02.07.2015,6:30:16,26.5985585,-97.4909666
34,02.07.2015,7:00:13,26.5953412,-97.4875568
35,02.07.2015,7:30:14,26.592486,-97.4826474
36,02.07.2015,8:00:59,26.5915851,-97.4807614
37,02.07.2015,8:32:32,26.5905198,-97.4801302
38,02.07.2015,9:01:59,26.5940041,-97.4803705
39,02.07.2015,9:31:24,26.5993019,-97.481959
40,12.07.2015,18:01:54,26.6117236,-97.5002732
41,13.07.2015,6:01:47,26.6040232,-97.5016165
42,14.07.2015,18:02:47,26.6038701,-97.5012212
43,15.07.2015,6:01:55,26.6185324,-97.481871
```

GPS_D

```
No,GMT.Date,GMT.Time,Latitude,Longitude
1,04.04.2015,6:01:30,26.6091494,-97.5188375
2,04.04.2015,6:30:17,26.609306,-97.5191711
3,04.04.2015,7:00:19,26.6093546,-97.519231
4,04.04.2015,7:30:17,26.6098715,-97.5196549
5,04.04.2015,8:00:54,26.6099107,-97.5196663
6,04.04.2015,8:30:15,26.6103555,-97.5213488
7,04.04.2015,9:00:15,26.6110596,-97.5218784
8,04.04.2015,9:30:32,26.6110683,-97.5218632
9,04.04.2015,10:00:14,26.6097845,-97.5249738
10,04.04.2015,10:30:23,26.6063357,-97.5270335
11,04.04.2015,11:00:20,26.6001222,-97.5312907
12,04.04.2015,11:30:13,26.5980463,-97.5306553
13,04.04.2015,12:00:14,26.5967446,-97.5297345
14,04.04.2015,12:30:14,26.5965947,-97.5291979
15,04.04.2015,13:00:49,26.5975529,-97.5275872
16,04.04.2015,13:30:18,26.5975422,-97.527637
17,04.04.2015,14:00:12,26.5983114,-97.5267741
18,04.04.2015,14:30:14,26.5988762,-97.5244212
19,04.04.2015,15:00:30,26.5998497,-97.5230612
20,04.04.2015,15:30:51,26.5987122,-97.5218578
21,04.04.2015,16:00:30,26.5987009,-97.5219175
22,04.04.2015,16:30:20,26.5987193,-97.5218724
23,04.04.2015,17:00:13,26.598694,-97.5218923
24,04.04.2015,17:30:24,26.5987688,-97.5219393
25,04.04.2015,18:00:19,26.598785,-97.5219366
26,04.04.2015,18:30:10,26.5981288,-97.5209618
27,04.04.2015,19:00:13,26.6002612,-97.5206945
28,04.04.2015,19:30:26,26.6003189,-97.5206804
29,04.04.2015,20:00:17,26.6007169,-97.5217935
30,04.04.2015,20:31:26,26.6027113,-97.5213921
31,04.04.2015,21:01:20,26.6030987,-97.5218722
32,04.04.2015,21:30:55,26.6030413,-97.5219005
33,04.04.2015,22:00:55,26.6030844,-97.5219022
34,04.04.2015,22:30:38,26.6030525,-97.5218271
35,04.04.2015,23:00:55,26.6030589,-97.5218459
36,04.04.2015,23:31:16,26.6032073,-97.5217183
```

GPS_E

```
No,GMT.Date,GMT.Time,Latitude,Longitude
1,18.04.2015,6:00:52,26.6000133,-97.5312327
2,18.04.2015,6:30:28,26.6000196,-97.5311133
3,18.04.2015,7:00:14,26.5999368,-97.5310689
4,18.04.2015,7:30:15,26.5998814,-97.5310327
5,18.04.2015,8:00:22,26.5999176,-97.5310392
6,18.04.2015,8:30:37,26.5999812,-97.5310676
7,18.04.2015,9:00:05,26.5999377,-97.5309728
8,18.04.2015,9:30:26,26.5999868,-97.5310391
9,18.04.2015,10:00:30,26.5997678,-97.5305979
10,18.04.2015,10:30:21,26.5998276,-97.5304599
11,18.04.2015,11:00:21,26.5997476,-97.5300599
12,18.04.2015,11:30:19,26.5997161,-97.530042
13,18.04.2015,12:00:54,26.5997281,-97.530046
14,18.04.2015,12:30:18,26.5997968,-97.5298144
15,18.04.2015,13:00:27,26.599798,-97.5297866
16,18.04.2015,13:30:35,26.5998599,-97.5297686
17,18.04.2015,14:00:57,26.599702,-97.5298366
18,18.04.2015,14:31:47,26.5998226,-97.5297872
19,18.04.2015,15:00:30,26.5999551,-97.5297307
20,18.04.2015,15:30:26,26.5996671,-97.5292929
21,18.04.2015,16:01:20,26.599604,-97.529252
22,18.04.2015,16:32:00,26.5996832,-97.5292929
23,18.04.2015,17:02:03,26.5996427,-97.5292845
24,18.04.2015,17:32:17,26.5996721,-97.5292632
25,18.04.2015,18:01:48,26.5996494,-97.5292223
26,18.04.2015,18:30:27,26.5996164,-97.5292538
27,18.04.2015,19:00:29,26.5995333,-97.5292089
28,18.04.2015,19:30:56,26.5994689,-97.5291984
29,18.04.2015,20:00:48,26.5995661,-97.5292101
30,18.04.2015,20:30:49,26.5994364,-97.5292227
31,18.04.2015,21:00:30,26.599468,-97.5291727
32,18.04.2015,21:30:32,26.599511,-97.5290198
33,18.04.2015,22:00:31,26.5994915,-97.5290901
34,18.04.2015,22:30:20,26.5994762,-97.5289939
35,18.04.2015,23:00:07,26.599465,-97.5288965
36,18.04.2015,23:30:20,26.5995674,-97.5290251
37,19.04.2015,0:00:19,26.5995419,-97.5290444
38,19.04.2015,0:30:13,26.5996365,-97.5284119
39,19.04.2015,1:00:14,26.5995811,-97.5283894
40,19.04.2015,1:30:15,26.5995823,-97.5283616
41,19.04.2015,2:00:18,26.5995214,-97.5283676
42,19.04.2015,2:30:45,26.5995531,-97.5283071
43,19.04.2015,3:00:06,26.5994356,-97.5282316
44,19.04.2015,3:30:30,26.5994335,-97.5282715
45,19.04.2015,4:00:12,26.5994684,-97.5283058
46,19.04.2015,4:30:13,26.5994157,-97.5283039
```