T-LoCoH for R
Tutorial and Users Guide

August 17, 2014

Andy Lyons

ajlyons@users.r-forge.r-project.org

http://tlocoh.r-forge.r-project.org

Table of Contents

1. Introduction

This tutorial explains how to use the T-LoCoH package for R. T-LoCoH is an algorithm for constructing home ranges and utilization distributions from a series of movement data. T-LoCoH can also be used for hull-based data exploration, for example investigating spatial patterns in time use patterns and generating 'behavior maps'. T-LoCoH will work with any set of points, but many of its features were designed specifically for time stamped locations such as those collected by a GPS device.

The primary reference for T-LoCoH is below. A copy of the paper is also available on the T-LoCoH website on R-Forge.

> Lyons, A., Turner, W.C., and WM Getz. 2013. *Home range plus: A space-time characterization of movement over real landscapes.* BMC Movement Ecology 1:2, doi:10.1186/2051-3933-1-2.

T-LoCoH is an enhanced version of LoCoH[1], and can do everything LoCoH can do. The main analytical improvement of T-LoCoH is the ability to include time into the construction and analysis of utilization distributions. This allows you to create a greater variety of space use models in which internal space is differentiated not only by the intensity of use, but also by behavioral properties such as directionality and time use.

The T-LoCoH package for R (hereafter simply referred to as *T-LoCoH*) is a software implementation of the T-LoCoH algorithm. T-LoCoH has some performance gains over the implementation of LoCoH in the Adehabitat R package (e.g., it can handle larger datasets). The package also has several multi-purpose functions such as data cleaning and creating animations from movement data.

[1] see Getz et. al., 2007; also http://locoh.cnr.berkeley.edu/

2. Installation

T-LoCoH.R is a package (extension) for R[2]. The package isn't on CRAN (yet), but it's on R-Forge. You can install it by typing at the R prompt:

```
> install.packages("tlocoh", dependencies=T, repos=c("http://R-Forge.R-project.org" ,
"http://cran.cnr.berkeley.edu"))
```

> Note: the reason you need to specify two repositories in the install.packages function is because some of the dependent packages are not in the r-forge repositories. The second repository can be any cran repository.

If installing T-LoCoH from R-Forge doesn't work, perhaps because of an internet connection problem, you can also do a manual install. For details see Appendix II, page 36.

Once T-LoCoH is installed, you can load it into memory by typing:

```
> require(tlocoh)
```

Updates and Support
T-LoCoH for R is a work in progress and updates will be made available as new functions are added and bugs fixed. To make sure that you stay informed when new versions are available, you can subscribe to the T-LoCoH mailing list by going to http://tlocoh.r-forge.r-project.org.

As with any new software, bugs are expected. Please report any bugs to T-LoCoH author Andy Lyons at ajlyons@r-forge.r-project.org.

3. T-LoCoH Objects

T-LoCoH works with two types (classes) of objects in R. Understanding what each of these types of objects contains will help you understand the workflow[3].

locoh-xy objects
A *locoh-xy* object, or *lxy* for short, contains a series of point locations. This is what you initially work with when you import your data. A *lxy* object may also contain the time-stamps for each of the points, and the id (name) of the individual / device. This means you can have the locations of multiple individuals in a single *lxy* object. This can be convenient when you want to construct home ranges for several animals at once. You could also assign different id values for different subsets of points that you want to analyze separately, such as wet season and dry season locations.

A *lxy* object may also contain what are known as *ancillary variables*, which are other variables associated with each point. These can be measurements captured by the GPS device, such as temperature or head-tilt, or derived variables that you generate in a GIS, such as distance to water or NDVI. By importing ancillary variables into T-LoCoH, you can create space-use maps

[2] http://cran.r-project.org
[3] for a detailed description of the data structure, see the vignette *T-LoCoH Data Classes*

that differentiate internal space by one of these variables, or look for associations between ancillary variables and things like speed or revisitation.

A *lxy* object contains two other things that are at the core of T-LoCoH: the parameters for a random walk null model[4], and a table of nearest neighbors. These are described in more detail below, but the advantage of storing them in the *lxy* object is that you don't have to recalculate these things each time you want to do an analysis.

A list of functions that work with *locoh-xy* objects can be found in Appendix V (page 45).

locoh-hullset objects
A *locoh-hullset* object, or *lhs* for short, is a set of little convex polygons, or hulls, around each point. Hulls are at the heart of T-LoCoH, and are the building blocks of isopleths and behavioral maps. You can think of T-LoCoH as an algorithm that transforms points into hulls. These hulls do a good job at representing space-use, and if they were constructed with time included will be 'local' both in space and time. Hulls can reveal other things about the movement pattern. For example we can compute hull metrics for revisitation, duration of use, average speed, elongation, etc.

A *lhs* object can contain multiple sets of hulls made with different parameter values, but they will all be from the same set of original locations. Each set of hulls in a *lhs* object may also have isopleths, which are aggregations of hulls, as well hull metrics. Some hull metrics are generated automatically, such as the hull area and the number of enclosed points, while other metrics are added manually after the *lhs* object has been created (such as revisitation and or eccentricity of the bounding ellipse). *lhs* objects also contain many of the data from the original *locoh-xy* object, including the point ids, ancillary variables, and date stamps.

A list of functions that work with *locoh-hullset* objects can be found in Appendix V (page 43).

4. Workflow Overview

T-LoCoH does not currently have a one-click wizard or GUI to guide you through the analysis process. Rather, you must run functions following a workflow that meets your analytical needs. A generic workflow is provided in Table 1. The initial steps will be the similar for almost all projects; steps for the analysis will vary a bit based on the objectives of the study.

Step	Choices	Functions You May Use
Preparation		
Getting ready	What research questions are you trying to address? What categories of movement do you anticipate? What outputs do you want to generate? ▪ isopleths (utilization distributions) ▪ hull metrics ▪ 'behavior' maps	published papers your brain
Import data into R Create locoh-xy object	Which ancillary variables to include with the locations?	xyt.lxy

[4] as of T-LoCoH version 1.0.5, the random walk null model is still created by not used in the default method for identifying nearest neighbors.

Step	Choices	Functions You May Use
View map of locations, distributions of step length, sampling interval		plot hist lxy.plot.freq
Clean data	What shorter-than-normal sampling interval constitutes a 'burst' of points that needs to be thinned?	lxy.plot.freq lxy.thin.bursts
Produce an animation of the movement to develop insights about the patterns	How to set up the frame: extent, label location, GIS layers, etc. How long and/or fast should the animation be? Should frames be time-based or record based?	lxy.exp.mov lxy.exp.kml
For multi-individual association analysis, harmonize the sampling interval and duration of sampling	Which common sampling interval to thin to? Which time period to examine?	lxy.plot.freq lxy.thin.byfreq
Create Hulls and Isopleths to Model Space Use		
If time stamps are present, select a value for *s*	What time scale are you interested in? View the distribution of *s* that balances time and space How 'local' do you want the hulls to be?	lxy.ptsh.add lxy.plot.sfinder lxy.plot.mtdr
Identify nearest neighbors	What method (k, r, or a)? What is the maximum value of k/r/a you'll need?	lxy.nn.add lxy.amin.add
Save your work		lxy.save
Create hulls	Value(s) of k/r/a	lxy.lhs
View summary of hullsets		summary
Sort hull (by density) and merge into isopleths	How many isopleths? Include the 100% isopleth?	lhs.iso.add
What value of k/r/a gives the "best" spatial model?	What is a real hole; what is a fake hole? Which is worse, type I or type II error?	plot lhs.plot.isoarea
Time-Use and Other Analyses		
Compute additional hull metrics (e.g., time-use metrics, elongation)	How much time must pass between separate 'visits' (i.e., inter-visit gap)?	lhs.visit.add lhs.ellipses.add hm.expr
Create other types of isopleths, such as by time-use, ancillary variables, etc.	Which metric to use for sorting hulls	lhs.iso.add plot
Examine scatterplots of hull metrics	Which pairs of metrics? Which ones are interesting?	lhs.scatter lhs.scatter.auto
View maps of hull parent points with a scatterplot legend	Colorize the scatterplot legend by manually drawing regions or a color wheel	lhs.scatter plot
Export results for additional analysis		lhs.exp.csv lhs.exp.shp

5. Exercise: Analysis of Buffalo Movement

Normally, the first step in an analysis is importing your data into R, probably in a 'flat' data structure like a spreadsheet. In addition to finding the right R functions to do this, you may also have to project the coordinates from latitude-longitude to a real-world coordinate system (like UTM), and possibly convert the time stamp from GMT to local time. For more information on these steps, see Appendix III (page 38).

The T-LoCoH package for R comes with some GPS tracking data for a single buffalo in Kruger National Park, South Africa[5]. In the following exercise, we will construct both utilization distributions, as well as look for time-use patterns for our buffalo friend, Toni.

Preparing the data

Let's begin by looking at the buffalo data:

```
> require(tlocoh)
> data(toni)
> class(toni)
[1] "data.frame"

> head(toni)
        id     long       lat           timestamp.utc
17930 toni 31.75345 -24.16950 2005-08-23 06:35:00.000
17931 toni 31.73884 -24.15402 2005-08-23 07:34:00.000
17932 toni 31.73969 -24.15359 2005-08-23 08:34:00.000
17933 toni 31.73874 -24.15329 2005-08-23 09:35:00.000
17934 toni 31.73946 -24.15336 2005-08-23 10:34:00.000
17935 toni 31.73898 -24.15363 2005-08-23 11:35:00.000

> plot(toni[ , c("long","lat")], pch=20)
```

Looking at the first few rows of the data frame (which is what the `head` function does), we can see that we need to project the coordinates from latitude-longitude to a real-world coordinate system like UTM. Kruger National Park falls in UTM zone 36 south. The following commands will project the coordinates into that zone.

```
> require(sp)
> require(rgdal)

> toni.sp.latlong <- SpatialPoints(toni[ , c("long","lat")],
proj4string=CRS("+proj=longlat +ellps=WGS84"))

> toni.sp.utm <- spTransform(toni.sp.latlong, CRS("+proj=utm +south +zone=36
+ellps=WGS84"))

> toni.mat.utm <- coordinates(toni.sp.utm)

> head(toni.mat.utm)
         long     lat
[1,] 373372.0 7326443
[2,] 371872.3 7328144
[3,] 371958.1 7328192
[4,] 371861.3 7328225
[5,] 371934.5 7328218
[6,] 371886.0 7328187
```

Looks good so far, but we should probably change the column labels because the coordinates are not long latitude and longitude anymore:

[5] Kruger African Buffalo, GPS tracking, South Africa. Downloaded from http://www.movebank.org, July 2012. Collection of Kruger Park Buffalo data funded by NSF Grant DEB-0090323 to Wayne M. Getz. Principal Investigator: Paul Cross

```
> colnames(toni.mat.utm) <- c("x","y")
> head(toni.mat.utm)
            x       y
[1,] 373372.0 7326443
[2,] 371872.3 7328144
[3,] 371958.1 7328192
[4,] 371861.3 7328225
[5,] 371934.5 7328218
[6,] 371886.0 7328187
```

So now we have a matrix, `toni.mat.utm,` with the buffalo locations in UTM coordinates. The last thing we need to do is make sure the timestamps are in the correct time zone. It isn't always obvious what time zone data time values are saved in, but in our case the column heading gives it away: `timestamp.utc`. We need to convert the time stamps from UTC time zone (aka GMT) to the time zone of the study site, which is GMT+2. Let's first see what data type the current timestamps are in, which is not obvious just by looking at them:

```
> class(toni$timestamp.utc)
[1] "factor"
```

A factor is an efficient way to save character data, but we need to convert the values in that column to a date-time object which specifies the time zone. After it’s a date-time object, we can then transform the times to the new time zone.

Let's begin by looking at the first few timestamps, converting the factor to a character:

```
> head(as.character(toni$timestamp.utc))
[1] "2005-08-23 06:35:00.000" "2005-08-23 07:34:00.000" "2005-08-23 08:34:00.000"
[4] "2005-08-23 09:35:00.000" "2005-08-23 10:34:00.000" "2005-08-23 11:35:00.000"
```

Next, we'll create a POSIXct vector, specifying what we believe to be the time zone. We'll also view the first few items to make sure R converted the character data correctly:

```
> toni.gmt <- as.POSIXct(toni$timestamp.utc, tz="UTC")

> toni.gmt[1:3]
[1] "2005-08-23 06:35:00 UTC" "2005-08-23 07:34:00 UTC" "2005-08-23 08:34:00 UTC"
```

It looks like R read the character data correctly. So next, we can convert to the local time. Using the sample code on page 40, we first take note that our computer recognizes the name of the local timezone (Kruger National Park) as 'Africa/Johannesburg'.

```
> local.tz <- "Africa/Johannesburg"

> toni.localtime <- as.POSIXct(format(toni.gmt, tz=local.tz), tz=local.tz)

> toni.localtime[1:3]
[1] "2005-08-23 08:35:00 SAST" "2005-08-23 09:34:00 SAST" "2005-08-23 10:34:00
SAST"
```

Create a locoh-xy object

We now have all the pieces we need to create a *locoh-xy* object: the locations in a real-world coordinate system, the timestamps in the local time zone, the name of the individual/device (in our case the points are just from one animal, 'toni'). However there are two other optional variables we are not going to include in this example: unique numeric id values for each point (which can be useful for linking the hulls to other information for additional analysis) and ancillary variables. See the help page for the `xyt.lxy()` function for more information about importing these variables.

Duplicate Points Pt. 1

Duplicate locations are common in GPS datasets, either because the animal was stationary for a period of time greater than the sampling frequency, left and returned to the same location, or because of rounding errors. So duplicate locations are generally not a problem provided they were at different times (or there are no time stamps). Duplicate time stamps are another issue - when you create a LoCoH-xy object, you'll get an error message if duplicate time-stamps are detected for the same individual, which are most likely the result of a data processing / management error.

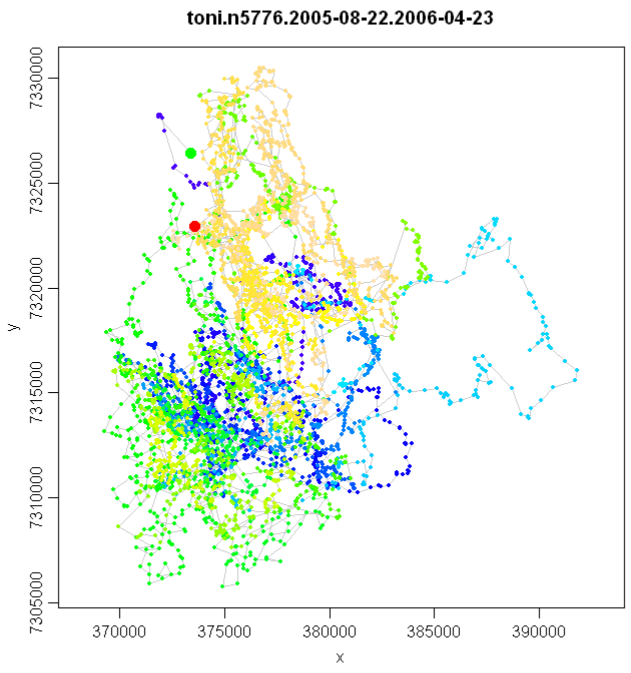
```
> toni.lxy <- xyt.lxy(xy=toni.mat.utm, dt=toni.localtime, id="toni",
proj4string=CRS("+proj=utm +south +zone=36 +ellps=WGS84"))
  595 duplicate xy-time-id rows removed
```

It seems our data had quite a few duplicate rows, with the same date stamp and location coordinates. Let's see how many are left by using the `summary()` function:

```
> summary(toni.lxy)
Summary of t.LoCoH-xy object: toni.lxy
***Locations
     id num.pts dups
   toni    5776    9
***Time span
     id      begin        end     period
   toni 2005-08-23 2006-04-23 243.3 days
***Spatial extent
     x: 369305.5 - 391823.9
     y: 7305737.9 - 7330491.3
   proj: +proj=utm +south +zone=36 +ellps=WGS84
***Random walk parameters
     id time.step.median    d.bar
   toni       3600 (1hs) 173.7575
***Nearest-neighbor set(s):
   none saved
```

The summary report shows there are still nine duplicate locations, but these have different time stamps so they were allowed to remain. We'll come back to those duplicate locations later on when making hulls. The summary report also show the parameters for the random walk null model, which we'll use later on to compute the time-scaled-distance (TSD) between points. But first, let's look at some of the properties of these locations.

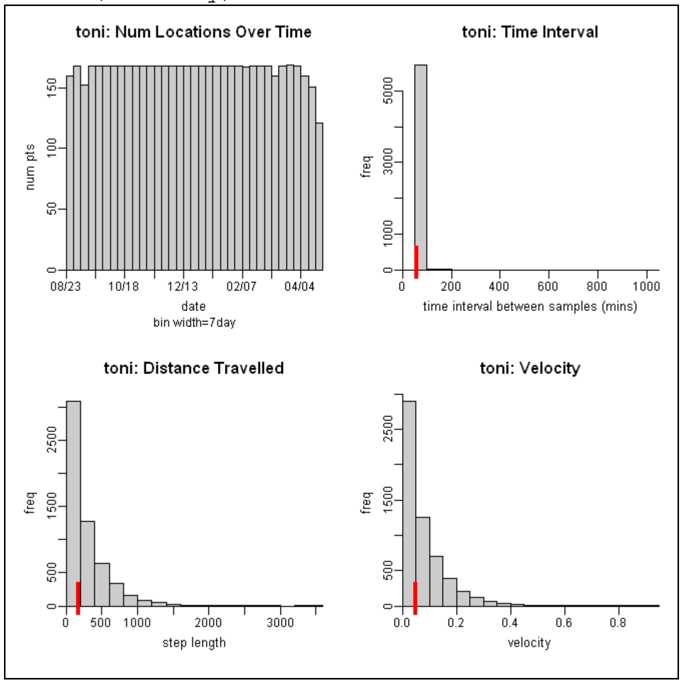
```
> plot(toni.lxy)
```



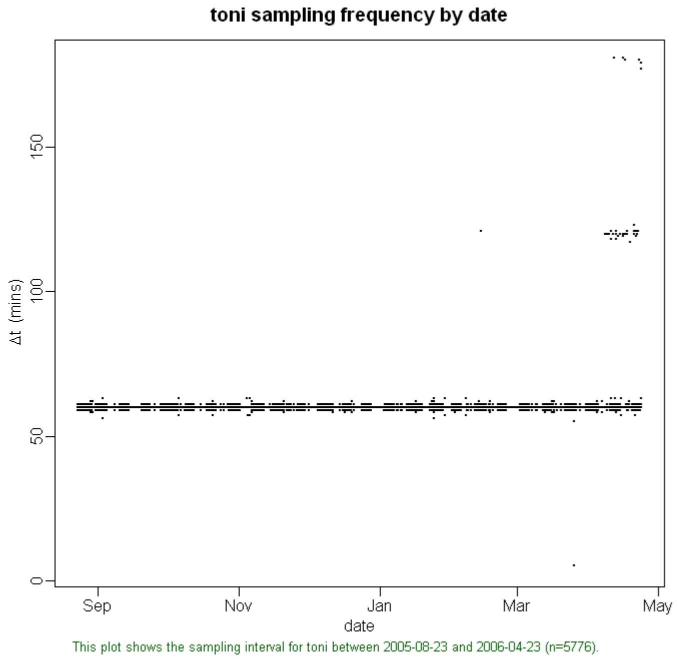
The plot of locations tells us firstly that there are no points that are obvious mistakes (for example points collected when the collar was sitting in the office, or processing mistakes where a digit was lost). We could also overlay GIS data to make sure our points are where we expect them (see Appendix IV. Displaying Spatial Data in the Background , page 42). We also take note that this animal spent most of its time in a fairly tight area, with only a few excursions. But we also see a little bit of north-south movement, and the colors suggest that the two ends of its range were used at different times—could this be a seasonal shift?

Next, let's look at the distribution of locations by date, step length, and sampling interval:

```
> hist(toni.lxy)
```



```
> lxy.plot.freq(toni.lxy, deltat.by.date=T)
```



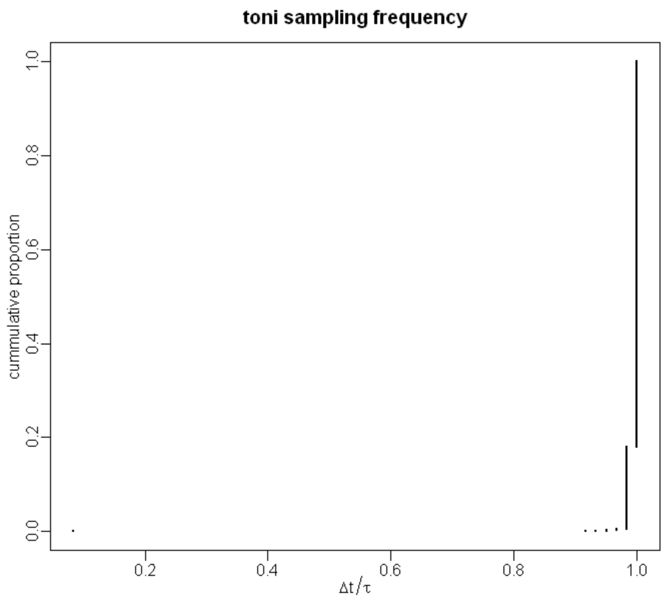
FYI: many of the plot functions in T-LoCoH by default display little green descriptive text at the bottom. To hide this text, pass the parameter `desc=0` ← ← ←

The plots of sampling frequency by date and the number of locations over time show that the sampling was pretty consistently one hour apart. There are a few gaps particularly toward the end of the period, but overall this dataset has pretty good temporal consistency which means our time-use analyses will not be biased by gaps in the data.

Removing 'Bursts'

Some GPS datasets have the opposite problem of gaps: 'bursts' of points that are closely spaced in time[6] - for example several points within a couple of minutes in a dataset that otherwise has a sampling interval of 1 hour. We need to thin out these bursts of points to avoid bias. We can see whether our dataset has point bursts by plotting the cumulative percentage of sampling intervals.

```
> lxy.plot.freq(toni.lxy, cp=T)
```



This plot shows the cummulative percentage of sampling frequencies of toni between 2005-08-23 and 2006-04-23 (n=5776) sorted smallest to largest. Sampling frequencies are expressed as a fraction of the median sampling frequency tau. Only sampling frequencies between 0 and 1 of tau are displayed (n=4890).

Our dataset seems to have only one burst of points (the little dot in the lower left corner), so we'll go ahead and thin it out. We'll set a threshold of 0.2, meaning that any group of points (or even a pair) that are less than 0.2 of the median sampling interval will be considered a cluster and thinned down to one location.

```
> toni.lxy <- lxy.thin.bursts(toni.lxy, thresh=0.2)
```

[6] note this is a different use of the word 'burst' than some other movement analysis packages (such as the Move package), where a 'burst' refers to a period of time when the GPS unit was programmed to turn itself on and record positions. In those cases, the closely timed locations are by no means an error.

Taking Subsets of a LoCoH-xy Object

If your dataset is huge, for example 100,000 points, and you don't have the patience to wait for R to create so many hulls (which may not even be necessary if you're just trying to create utilization distributions), you can create a subset of your LoCoH-xy object using the `lxy.subset()` function. This function lets you subset by 1) individual (in the case where your LoCoH-xy object includes the locations of multiple individuals), 2) index (useful for taking every nth point), or 3) date. To take every 3rd point, for example, you could use the following:

```
> toni.lxy.every3rd <- lxy.subset(toni.lxy, idx = seq(from=1, to=5775, by=3))

> summary(toni.lxy.every3rd)
Summary of t.LoCoH-xy object: toni.lxy.every3rd
***Locations
     id num.pts dups
   toni    1925    0
***Time span
     id      begin        end     period
   toni 2005-08-23 2006-04-23 243.1 days
***Spatial extent
   x: 369305.494533287 - 391722.010161986
   y: 7305737.86452739 - 7330430.5328576
***Random walk parameters
     id time.step.median    d.bar
   toni      10800 (3hs) 462.7666
***Nearest-neighbor set(s):
   none saved
```

Note when you subset a LoCoH -xy object, any nearest neighbors that were saved are lost and have to be recreated.

Creating an Animation

Animations can be a good way to view the temporal and spatial patterns in data. T-LoCoH provides two functions to animate movement data, using Google Earth and QuickTime. See Appendix VI (page 48) for details.

Determine the Space-Time Balance

Note: If your data have no time stamps, or you are not interested in including time in the analysis, you can skip this section and let *s*=0 in subsequent steps.

T-LoCoH calculates the 'distance' between points using a hybrid space-time metric called Time Scaled Distance (TSD, see Glossary). If we we're only interested in constructing a space-use model without any consideration of time (i.e., traditional utilization distributions), we could skip this step and simply set s=0 in the next step (identifying nearest neighbors). But including time is useful for at least two reasons. First, the utilization distributions we construct when time is included will do a much better job in capturing temporal partitioning of space. Think for example about path intersections, which might appear to be a dense blob of points but in fact are used at different times; including time in the hull construction will pull out the temporal partitioning.

The other advantage of including time in our hull construction is that it allows us to examine rates of revisitation and the duration of visits to an area.

T-LoCoH uses a scaling parameter *s* that controls to the degree to which local hulls are local in time as well as space. When *s*=0, time has no influence in picking nearest neighbors, and the hulls around each point are constructed merely from the closest points in space. At the opposite end of the spectrum, when *s* is big the 'closest' points are merely the points closest in time without regard to space (e.g., a scanning time window). There are two ways to pick s:

Method 1. Pick a value of *s* such that 40-80% of the hulls are time-selected (recommended to start with)

Type at the console:

```
> toni.lxy <- lxy.ptsh.add(toni.lxy)
```

From the graph read the value of *s* where the proportion of time selected hulls is between 0.4 and 0.8

Method 2. Select *s* based on a time interval of interest

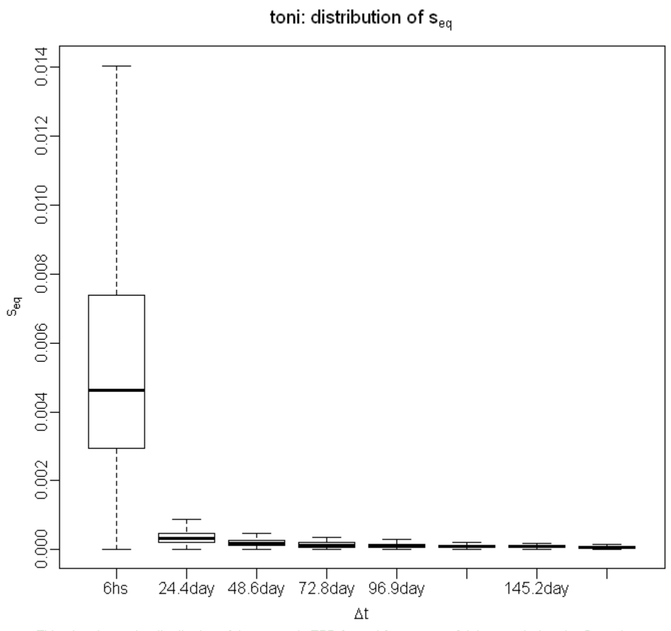
To pick a value of *s* based on a time interval of interest, you need to first think about the time scale related to your research question. If you're interested in daily foraging behavior, for example, you might want two points that are a day apart to be not considered nearest neighbors even if they are close together in space. On the other hand, if you're interested in monthly movement patterns, points a few days apart should all be candidates as nearest neighbors, and only points that are several weeks apart should really be penalized as nearest neighbors.

TIP: If you're not sure which time interval may be interesting to look at, you can try to see the 'natural' frequencies in the data by plotting the distance of each point to the centroid of the entire dataset over time:

```
> lxy.plot.pt2ctr(toni.lxy)
```

s will also depend on your map units, so after you think about your time scale the best thing is to plot the distribution of s values that equalizes space and time for a range of time scales:

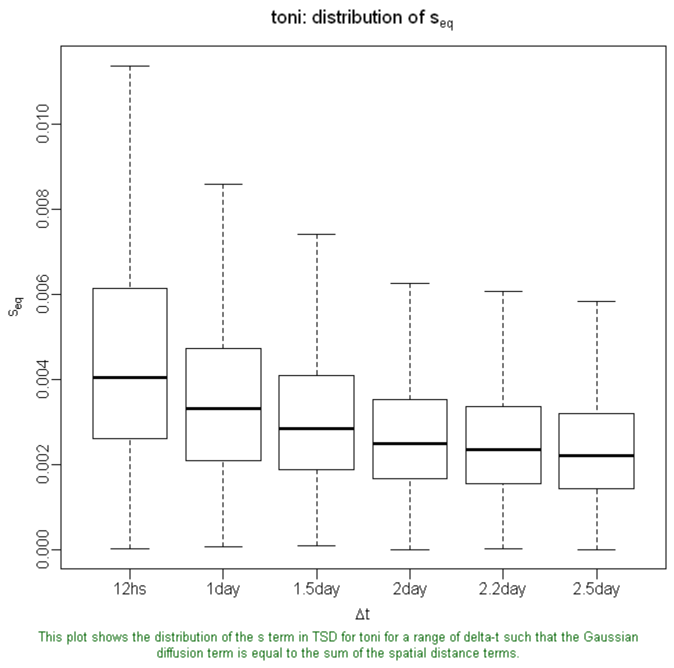
```
> lxy.plot.sfinder(toni.lxy)
```



This plot shows the distribution of the s term in TSD for toni for a range of delta-t such that the Gaussian diffusion term is equal to the sum of the spatial distance terms.

It looks like for the 24 hour period that we are interested in, the diffusion distance term will be equal to the spatial terms for half of all pairs of points if *s*=0.001. We can't see this too clearly so let's make the plot again, but this time we'll specify the time intervals we're interested in. `lhs.plot.sfinder()` has a parameter that lets you specify which time intervals to analyze, remembering that the time intervals are always entered as a number of seconds (i.e. 1 hour = 3600 seconds).

```
> lxy.plot.sfinder(toni.lxy, delta.t=3600*c(12,24,36,48,54,60))
```



Using this plot as a guide (in particular the distribution of s_{eq} for 24 hours), we can see diffusion distance (time) and space are balanced on average when *s* is about 0.003. If we pick a value of *s* which is a lot more than 0.003, we'll wind up with the temporal difference between points dominating the selection of nearest neighbors, which is not what we want. So a value around the median 0.003, is reasonable to start with. After we pick some nearest neighbors using this value of s, we can plot the actual balance between space and time for actual nearest neighbors.

Identify Nearest Neighbors

The next task is to identify nearest neighbors. There are three methods for identifying nearest neighbors[7]. The *k-method* simply finds the k^{th} nearest neighbors around each point (where 'nearest' is determined by the TSD metric, which in turn is influenced by the value of *s*). The *r-method* finds all nearest neighbors with a radius *r*, and the a-method finds all neighbors whose cumulative distance is less than or equal to *a*. You get to decide which method to use, as well as the value for *k*, *r*, or *a*. A good rule of thumb is to start with the k-method, see what those isopleths look like, and then if you see outlying points connected with really large hulls that cut across where the animal clearly did not go, try the a-method (which is less influenced by outliers).

At this point we have no idea how many nearest neighbors will give the best space use model, so let's go ahead and identify 25 neighbors per point. We identify nearest neighbors using the `lxy.nn.add()` function which returns the original object with the nearest neighbors identified (so we won't have to identify them again):

```
> toni.lxy <- lxy.nn.add(toni.lxy, s=0.003, k=25)
```

[7] see Getz et. al 2007

The `summary()` function shows us the nearest neighbor table saved:

```
> summary(toni.lxy)
Summary of t.LoCoH-xy object: toni.lxy
***Locations
     id num.pts dups
   toni    5775    9
***Time span
     id      begin        end     period
   toni 2005-08-23 2006-04-23 243.3 days
***Spatial extent
   x: 369305.494533287 - 391823.930056777
   y: 7305737.86452739 - 7330491.33815097
***Random walk parameters
     id time.step.median    d.bar
   toni       3600 (1hs) 173.7452
***Nearest-neighbor set(s):
   1 toni|s0.003|n5775|kmax25|rmax163.7|amax1534.3
```

The name of the nearest neighbor set created (shown in bold above) tell us something about this set of neighbors. The `s0.003` tells us the value of s that was used in nearest neighbor identification. We already know that this set of nearest neighbors includes a minimum of 25 neighbors for each and every point (**`kmax25`**), but we can also see that this set of neighbors has enough neighbors for each point to construct hulls using the *r-method* for values of up to *r* = 163 (map units, in this case meters), and it will also be sufficient for the *a-method* with values of *a* up to 1534 (map units). If, for example, we decide later on to construct hulls using the *a-method* with a larger value of *a* (say *a* = 2000), we would need to rerun this function and identify some more nearest neighbors:

Checking the Effects of s

If we weren't sure about which value of *s* to use, we could also identify nearest neighbors using a range of *s* values and examine the ratios of diffusion distance to TSD using the functions and the time span of the resulting hulls and using `lxy.plot.mtdr()` and `lxy.plot.tspan()`. That would look like:

```
> toni.lxy <- lxy.nn.add(toni.lxy, s=c(0.0003, 0.003, 0.03, 0.3), k=25)
> lxy.plot.mtdr(toni.lxy, k=10)
> lxy.plot.tspan(toni.lxy, k=10)
```

Save Your Data to Disk

Now that we have done so much work importing our data and identifying nearest neighbors, this would be a good time to save our LoCoH-xy object to disk. R has a great function called `save()` that we can use, but T-LoCoH has an even better function, `lxy.save()`, designed specifically for locoh-xy objects and automatically generates a filename for you:

```
> lxy.save(toni.lxy, dir=".")

LoCoH-xy toni.lxy saved to:
   c:\LoCoH\toni.n5775.2005-08-22.2006-04-23.lxy.01.RData
```

The `dir` parameter specifies where the file should be saved. In the example above, `dir="."` stands for the working directory; you can also use `dir="~"` to save the file in the R home directory, or specify a sub-directory with something like `dir="~/buffalo"`. If a file already exists, it will increment the number at the end of the filename. To load this object back into memory another time, you can use R's `load()` function.

```
> load(file.choose())
```

Create Hullsets

The building blocks of all T-LoCoH analyses are hulls, which are simply minimum convex polygons constructed around each point from a sect of nearest neighbors. Since we've already identified 25 nearest neighbors for each point, so we can create hulls with up to 25 nearest neighbors each. We don't yet know how many neighbors will give the best results—we want hulls that are big enough that they don't leave lots of tiny holes in areas the animal uses, but not so big that they cover large swaths where the animal was never seen. So let's create hullsets for a range of k values from 6 to 24. We'll use the `lxy.lhs()` function which returns a locoh-hullset collection.

Duplicate Points Pt. 2

When identifying nearest neighbors, points that lie on top of each other will always be selected as nearest neighbors when s=0 (i.e., time is excluded), but may not necessarily be considered nearest neighbors when s>0. Problems arise however when creating local hulls from a set of nearest neighbors that contains duplicate locations, because you can't create a minimum convex polygon from points with the same coordinates. T-LoCoH offers two options to deal with this. You can ignore duplicate points, although this could result in some points not getting a hull at all, if the nearest neighbors don't include at least two unique points. Alternately, duplicate locations can be randomly offset during the hull construction phase, resulting in very small (dense) hulls. The `offset.dups` parameter in the `lxy.lhs()` function defaults to offsetting points by 1 map unit; to ignore them set `offset.dups = 0`. Note that if you have duplicate points and offset them in a random direction, then any ancillary variables that relate to location (e.g., distance to water) will be off. Also, you won't get the exact same set of hulls if you re-run the analysis. To detail with this, you should offset duplicate points before generating hulls.

```
> toni.lhs <- lxy.lhs(toni.lxy, k=3*2:8, s=0.003)
```

There are a lot of computations when creating hullsets, so this could take 15 minutes or so. When done, we can use the `summary()` command again to see what it contains:

```
> summary(toni.lhs, compact=T)

Summary of LoCoH-hullset object: toni.lhs
T-LoCoH version: 1.0.5
[1] toni.pts5775.k6.s0.003.kmin0
[2] toni.pts5775.k9.s0.003.kmin0
[3] toni.pts5775.k12.s0.003.kmin0
[4] toni.pts5775.k15.s0.003.kmin0
[5] toni.pts5775.k18.s0.003.kmin0
[6] toni.pts5775.k21.s0.003.kmin0
[7] toni.pts5775.k24.s0.003.kmin0
```

Next, we'll create isopleths for our hullset. Isopleths are aggregations of hulls sorted in such a way as to reveal something about space use. The default settings for `lhs.iso.add()` sorts hulls according to density, so the isopleths reflect the likelihood of occurrence which is a proxy for intensity of use.

```
> toni.lhs <- lhs.iso.add(toni.lhs)
```

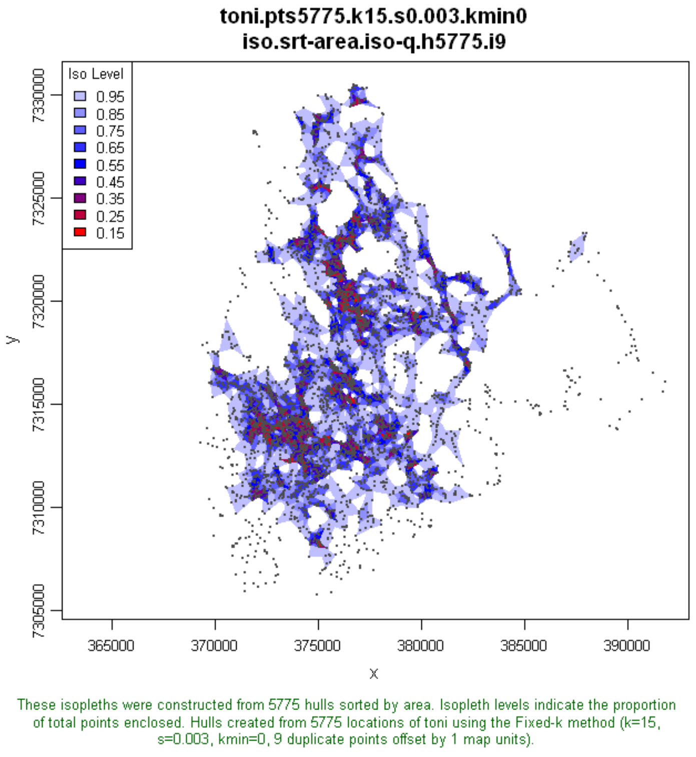
This will create isopleths for each of the seven values of k we tested. We can plot the isopleths using the `plot()` command. We could make a plot with the seven isopleths in separate panels (`figs.per.page=7`), but they would turn out awfully small. Instead we'll use the `record=T` parameter which will turn on plot recoding so we can flip through the plots using the PgUp/PgDown keys (Windows) or ⌘+left/right arrows (Mac). We'll also turn off the 'user-friendly isopleth titles' (`ufipt=F`) because we're scientists not wimps.

```
> plot(toni.lhs, iso=T, record=T, ufipt=F)
```

Which one do you think looks best? Again, we're looking for the set of hulls where heavily used area doesn't look like Swiss cheese, but also doesn't cut across unused areas. This is essentially finding a balance between Type I error (including area that isn't part of the home range) and Type II error (omitting area the animal used). Since we don't have perfect knowledge of everywhere the animal went, and 'home range' is more of an artificial construct we invented to answer questions as opposed to an 'actual' area on the ground, we have to determine this balance for ourselves based on our knowledge of the system and the implications (to our research) of making errors of omission versus errors of commission.

I think I like *k*=15, because it's the smallest value of *k* that fills in most of the holes in the core area. Let's see what that looks like with the original locations overlaid—we'll make them very small (`cex.allpts=0.1`) and light gray (`col.allpts="gray30"`) so they don't hide the detail.

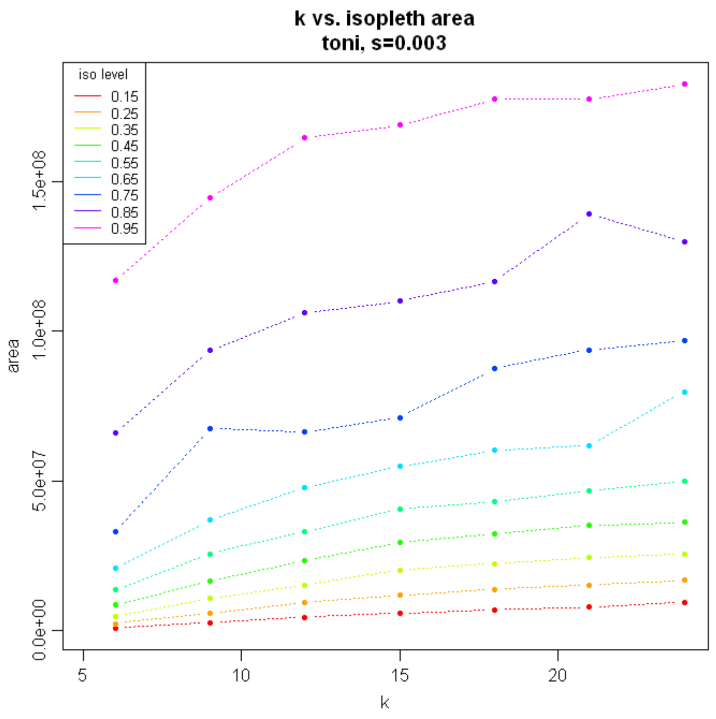
```
> plot(toni.lhs, iso=T, k=15, allpts=T, cex.allpts=0.1, col.allpts="gray30",
ufipt=F)
```


toni.pts5775.k15.s0.003.kmin0
iso.srt-area.iso-q.h5775.i9
Iso Level
0.95
0.85
0.75
0.65
0.55
0.45
0.35
0.25
0.15
y
7305000
7310000
7315000
7320000
7325000
7330000
x
365000
370000
375000
380000
385000
390000
These isopleths were constructed from 5775 hulls sorted by area. Isopleth levels indicate the proportion of total points enclosed. Hulls created from 5775 locations of toni using the Fixed-k method (k=15, s=0.003, kmin=0, 9 duplicate points offset by 1 map units).

You'll notice that there are some points that aren't included in any of the isopleths. That's because the largest isopleth (light blue) level is 0.95, meaning that it only encloses 95% of the points. And because these isopleths are sorted according to density, the 5% of the points not covered by the 95% isopleth are the least dense. If we wanted to see the 100% isopleth, we could specify the `iso.levels` parameter in `lhs.iso.add()`. But 95% is typically considered the 'home range' so we'll stop here.

In addition to the isopleth maps, two others piece of information that can help us decide which value of *k* to use are 1) the isopleth area curves and 2) the isopleth edge:area curves for the different values of *k* we've tested. Let's plot these.

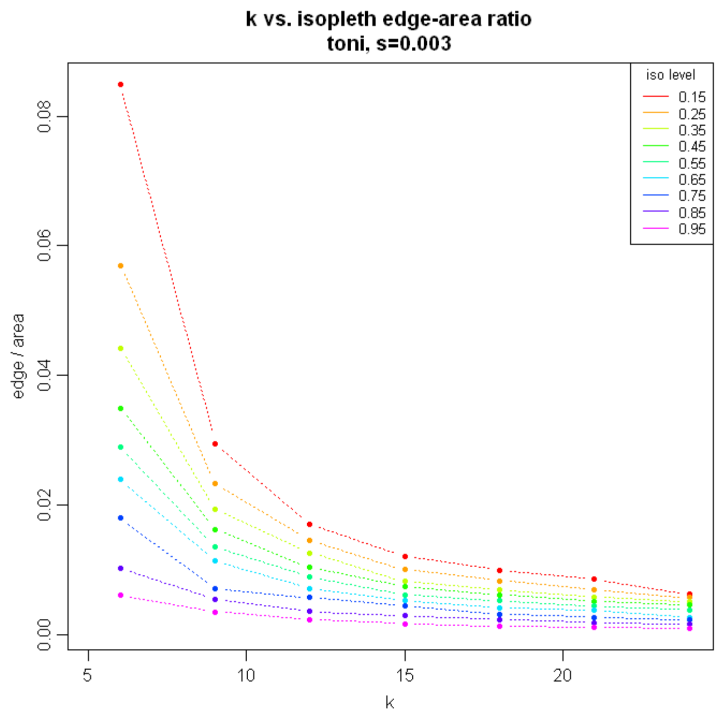
```
> lhs.plot.isoarea(toni.lhs)
```



The isopleth area plot shows us the area of each isopleth for the different values of *k*. What we're looking for here are sharp jumps in area that indicate that a slightly larger value of *k* resulted in a big increase in area – a sign that a big swath of new area was included (which is probably false commission). If we see such a jump, we want to pick a *k* value that is lower than the jump. In our example, we see a bit of jump in the 95% isopleth between *k*=15 and *k*=18, so we probably should look at values of k<=18. If we wanted to continuing refine our space-use model, the next logical step would be to create hullsets for individual values from say k=12 to k=18, and then look at those individually before making a final selection.

Now let's plot the isopleth edge:area ratios.

```
    > lhs.plot.isoear(toni.lhs)
```



The edge:area curves are simply the ratio of the edge (i.e., total perimeter) to the area for each isopleth level. High values are indicative of a Swiss cheese pattern – lots of small holes. In particular, we want to at the edge:area ratio in the 'core' areas, for example the 30th % isopleth (green line above), because we presume that tiny holes in the core area are probably the result of too small a value of *k* (unless we have a reason to suspect that the animal's core area really is quite patchy). Our curves for Toni reflect what we saw on the map – Swiss cheese looking core areas for small values of *k*, that largely fill in around *k*=15.

If we're pretty sure we like the hulls and isopleths when k=15, we can pull out just those hulls for the rest of the analysis as follows

```
    > toni.lhs.k15 <- lhs.select(toni.lhs, k=15)
```

The a-method

The *k*-method is simple and intuitive – every hull is constructed from the same number of nearest neighbors. However this doesn't always give good results particularly if the data have both dense areas and sparse areas (fairly common with free-ranging species). The reason for this is because you may need a large value of *k* to fill in the little holes in the denser core areas, but this value of *k* may result in humongous hulls in the outlying areas where the points are thin and widely scattered, overestimating the area used in those area..

The 'a' in a-method stands for adaptive, because this method is designed to reduce the number of neighbors used in areas where the points are thin and scattered. Neighbors are identified by summing up their cumulative distance from the parent point and stopping when you reach *a*. This may result in a many points being labeled as neighbors in a dense area (which is what you probably want), and just a few points in outlying areas (which is also what you want).

The same general workflow applies: 1) identify nearest neighbors (in the LoCoH-xy object), 2) create hullsets for a range of a values, 3) look at the maps and plot the isopleth areas and edge:area curves, and 4) test additional a values as needed. However deciding which values of *a* to test is not a straightforward or intuitive as picking k, because 1) *a* is a cumulative distance from the parent point to the nearest neighbors, and 2) if time is included, the 'distance' between points is the TSD space-time metric which is not a physical distance.

To help you pick a range values of *a* to test, you can use the `auto.a()` function, which will compute the *a* value such that *p*% of points get at least *n* nearest neighbors, where *p* and *n* are supplied by you. So for example, if we like the results with *k*=15, we can use `auto.a()` to tell us the value of *a* that will result in 98% of all points having 15 or more nearest neighbors:

```
> toni.lxy <- lxy.nn.add(toni.lxy, s=0.003, a=auto.a(nnn=15, ptp=0.98))

> summary(toni.lxy)
Summary of t.LoCoH-xy object: toni.lxy
***Locations
     id num.pts dups
   toni    5775    9
***Time span
     id      begin        end     period
   toni 2005-08-23 2006-04-23 243.3 days
***Spatial extent
   x: 369305.494533287 - 391823.930056777
   y: 7305737.86452739 - 7330491.33815097
***Random walk parameters
     id time.step.median    d.bar
   toni       3600 (1hs) 173.7452
***Nearest-neighbor set(s):
   1 toni|s0.003|n5775|kmax25|rmax414.5|amax12653.4
                   meth ptp   nnn a.h   tct    aVal
       auto.a #1     nn 0.98   15   1  1.05 12652.9
```

The output from the `summary()` function tells us that for *s*=0.003, if *a*=12653, 98% of the points will have at least 15 nearest neighbors. This gives us an idea of what range of *a* values to test. It would be reasonable for example to create hullsets for *a* = 8000, 9000, …15000, then look at the isopleths plots and narrow down the range of values still further until we're happy with the space use model.

First we identify enough nearest neighbors such that each point has enough neighbors identified for a=15000.

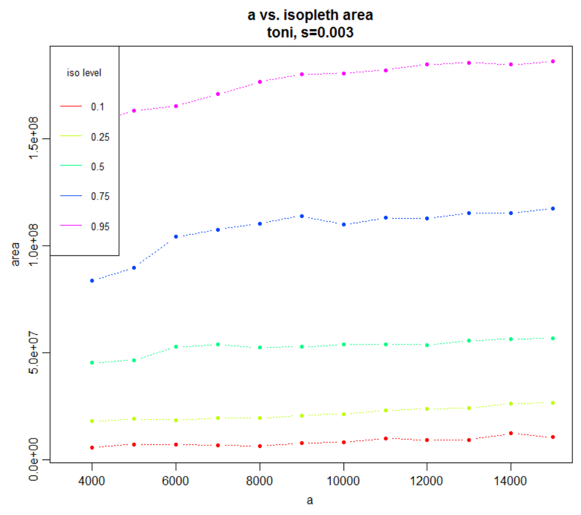
```
> toni.lxy <- lxy.nn.add(toni.lxy, s=0.003, a=15000)
Finding nearest neighbors for id=toni (n=5775), num.parent.pts=5775,
mode=Fixed-a, a=15000, s=0.003, method=diffusion
 - there is already a set of nearest neighbors for this set of parent points
   and value of s.
 - additional neighbors will be identified and appended as needed
 - finding an initial value of k using 30 randomly selected points...Done. Initial k=44
 - computing cumulative distances for k=44
 - 119298 additional neighbors appended
 - computing values of kmax, rmax, and amax...Done
 - set of neighbors (re)named: toni|s0.003|n5775|kmax25|rmax404.9|amax15003.8

Done. Nearest neighbor set(s) created / updated:
   toni|s0.003|n5775|kmax25|rmax404.9|amax15003.8
Total time: 4.4 mins
```

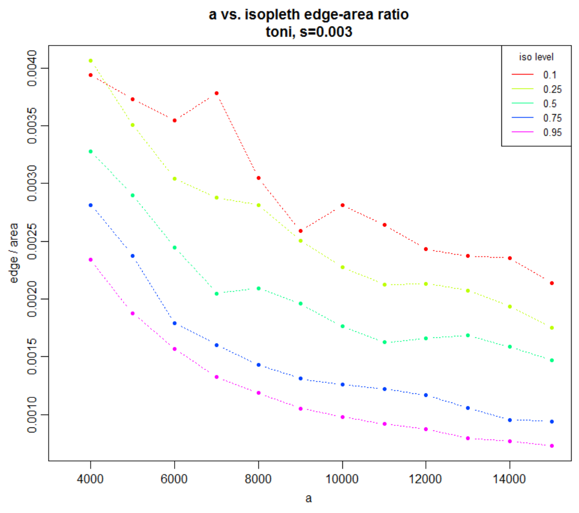
Next, we create a new hullset object using *a* values from 4000 to 15000. Note we can tell R to also create density isopleths using the `iso.add` argument (as opposed to creating isopleths separately as we did earlier).

```
> toni.lhs.amixed <- lxy.lhs(toni.lxy, s=0.003, a=4:15*1000, iso.add=T)
```

Finally, we can plot the isopleth area and edge:area curves. As before, we're looking for look for spurious jumps in the area and spurious edginess in the core area. If we were going to continue with the a-method, we'd need to look at isopleth maps also.

```
> lhs.plot.isoarea(toni.lhs.amixed)
> lhs.plot.isoear(toni.lhs.amixed)
```





Compute Additional Hull Metrics

The isopleths we just constructed are a model of how the animal used space. This can be a useful product in and of itself, if we're doing reserve design for example, or can also an input into another analysis, such as quantification of site fidelity, range shift, or evaluation of a resource utilization function.

But we can also do more with the building blocks of the isopleths – the individual hulls. Because of the way nearest neighbors were selected, the hulls are localized in both time and space. Therefore hulls are a good unit of analysis to examine any covariate (or cause) of movement. Instead of sorting and merging them by density to produce utilization distributions, we can sort and merge them by some other metric, such as directionality or revisitation rate, to produce what one might call 'behavior maps' or 'time use maps'.

Several hull metrics are computed when a hullset is first created—you can see them listed with the `summary()` function. Other hull metrics are not always needed and have to be created with a separate function.

Elongation

The eccentricity of the bounding ellipsoid, which can be used as a proxy of elongation or directionality, is created with the `lhs.ellipses.add()` function. Computing ellipses is slow (~25 minutes for Toni), but if you want to try it type this[8]:

```
> toni.lhs.k15 <- lhs.ellipses.add(toni.lhs.k15)
Total time: 24 mins

> summary(toni.lhs.k15)
Summary of T-LoCoH-hullset object: toni.lhs.k15
***Hulls created on: Wed Aug 01 11:11:26 2012
***Individuals analyzed:
     id num.pts dup.loc dup.offset               begin                 end
   toni    5775       9          1 2005-08-23 08:35:00 2006-04-23 15:09:00
***locoh mode: Fixed-k
   k's analyzed: 15
***s: 0.003
***Elements saved:
                              hulls nn ellipses
   toni.pts5775.k15.s0.003.kmin0  5775  0     5775
***Hull metrics computed: area, bearing, ecc, nep, nnn, par, perim,
   scg.enc.mean, scg.enc.sd, scg.nn.mean, scg.nn.sd, tspan
```

The `summary()` function shows us the number of ellipses that have been computed, and that eccentricity as one of the available hull metrics. We can view a sample ellipse by setting `ellipses=TRUE` in the `plot()` function. Let's also tell it to plot hulls, nearest neighbors, and all the other points by setting `hulls`, `nn`, and `allpts` to `TRUE`. By adding the `ptid` parameter, it will 'zoom in' to the hull around the parent point with a specific point id value, and if we set `ptid="auto"` it will pick a hull at random. If we want to plot all the ellipses, we could omit the `ptid` parameter.

[8] don't want to wait? you can also download the toni.lhs.k15 object with ellipses computed by entering
```
mycon <- url("http://tlocoh.r-forge.r-project.org/toni.n5775.s003.k15.elps.iso.lhs.RData")
load(mycon); close(mycon)
```

```
> plot(toni.lhs.k15, hulls=T, ellipses=T, allpts=T, nn=T, ptid="auto")
```

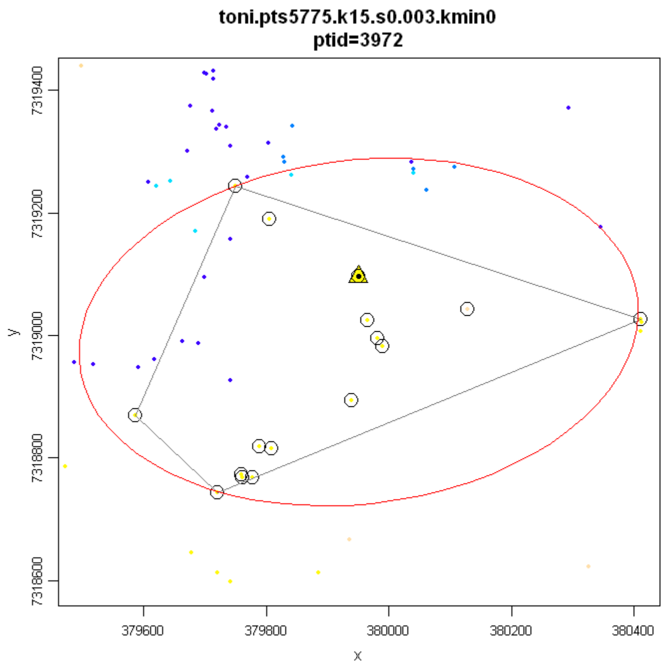


Figure 1. Sample hull for a single point. Point colors represent continuity in time. The parent-point is shown by a triangle; nearest neighbors identified using TSD with *s*=0.003 are circled. Non-circled points within the hull are closer to the parent point in space but were bypassed as nearest neighbors due to their separation in time. The ellipse outlined in red is the bounding ellipse whose eccentricity is one of the metrics of hull elongation.

Below, we will look at the spatial patterns in elongation, but first let's compute some more hull metrics.

Time-Use Metrics

Other examples of optional hull metrics are time-use hull metrics for revisitation (to the hull) and the average duration of each visit. These two dimensions of time-use can tell us something about the behavior of the animal as well as the resources it uses.

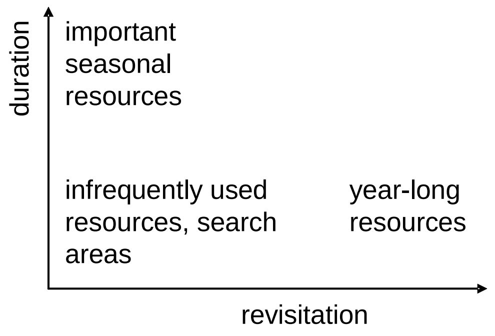

duration
important seasonal resources
infrequently used resources, search areas
year-long resources
revisitation

Figure 2. Time-use space defined by revisitation and duration

When we include time into nearest neighbor selection, we can compute revisitation and duration for any hull precisely because most hulls will enclose points that were bypassed as nearest neighbors because they occurred at different times (Figure 1). In other words, these points represent other visits to the hull. We can simply count up the number of times the animal visited the area inside the hull as a measure of revisitation, and how many times on average it was found in the hull during a single visit as a measure of the mean visit duration.

To compute revisitation and average visit duration, we need to define what a 'visit' is by specifying the inter-visit gap period (IVG). The IVG is a unit of time that must pass before another occurrence in the hull is considered a separate visit. This controls for boundary effects (e.g., the animal steps outside the hull area briefly but comes right back).

The time-use metrics are created with the `lhs.visit.add()` function. In this example we're interested in daily cycles to movement, however if we specify an inter-visit of 24 hours, then two occurrences at a water hole 23 hours apart would be considered the same visit. So let's specify an inter-visit gap period of 12 hours (remembering that intervals of time are always entered in seconds):

```
> toni.lhs.k15 <- lhs.visit.add(toni.lhs.k15, ivg=3600*12)

> summary(toni.lhs.k15)
Summary of T-LoCoH-hullset object: toni.lhs.k15
***Hulls created on: Wed Aug 01 11:11:26 2012
***Individuals analyzed:
     id num.pts dup.loc dup.offset               begin                 end
   toni    5775       9          1 2005-08-23 08:35:00 2006-04-23 15:09:00
***locoh mode: Fixed-k
   k's analyzed: 15
***s: 0.003
***Elements saved:
                                 hulls nn ellipses
   toni.pts5775.k15.s0.003.kmin0  5775  0     5775
***Auxillary hull metric parameters analyzed:
   ivg: 43200
***Hull metrics computed: area, bearing, ecc, mnlv.43200, nep, nnn,
   nsv.43200, par, perim, scg.enc.mean, scg.enc.sd, scg.nn.mean, scg.nn.sd,
   tspan
***Isopleths saved:
     [1] iso.srt-area.iso-q.h5775.i9
```

The summary report shows us the new hull metrics that have been created. The new hull metric `nsv.43200` is the measure of revisitation and stands for 'number of separate visits for an inter-visit gap period of 43200 seconds. `mnlv.43200` stands for mean number locations per visit, and is the measure of average duration.

Review

So far what we've done is:

1. Thought about our research question, and the temporal scale of the movement pattern(s) we're interested in
2. Imported the movement data into R, transformed the coordinates from latitude-longitude to UTM, and converted the date stamps from GMT to the local time zone.
3. Viewed histograms of the step length and sampling interval, and removed any short-interval 'bursts' in the data
4. Created an animation of the movement data
5. Picked a value of 's' that balances the influence of space and time for our time scale of interest
6. Created hulls for a range of *k* values, then visually selected k=15 as the smallest *k* which fills up little holes in what appears to be the core area
7. Created density isopleths (i.e., utilization distribution) for k=15
8. Computed additional hull metrics (ellipses and time-use for an inter-visit gap of 12 hours)

This would be a good time to save our work:

```
> lhs.save(toni.lhs.k15)
```

Examining Hull Metrics

We've created some cool hull metrics that reflect behavioral patterns, including directionality (eccentricity of bounding ellipse), revisitation rate, and the mean visit duration. Now let's explore these. There are a few ways we can explore hull metrics:

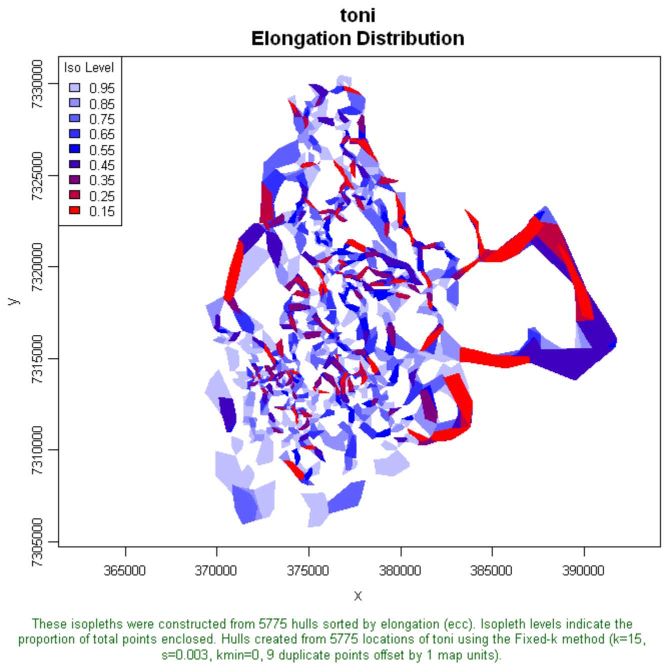
1. Create and plot isopleths after sorting hulls on a behavioral metric
2. Plot hull parent points colored by the value of a behavioral metric
3. View histograms of pairs of hull metrics
4. Plot hull parent points colored by their location in a 2 dimensional scatterplot space
5. Export all the hull metrics as a table and examine correlations using fancy statistical analysis (e.g., regression)

Behavior Isopleths

Isopleths usually reflect a gradient of the proportion of known locations, what we might call *density isopleths*. This is essentially what a traditional utilization distribution is, it tells you where the animal spent the most time.

However because isopleths are created from hulls, and hulls have so many more properties than just point density, we can construct isopleths that reflect other things, such as behavior or time-use. These will no longer be density isopleths but something akin to a behavior map. Let's begin by creating and plotting isopleths from hulls sorted by elongation, which is an imperfect measure of directional movement but might reveal something interesting. To do this, we use the `lhs.iso.add` function again, but this time we'll specify how we want the hulls to be sorted.

```
> toni.lhs.k15 <- lhs.iso.add(toni.lhs.k15, sort.metric="ecc")

> plot(toni.lhs.k15, iso=T, iso.sort.metric="ecc")
```

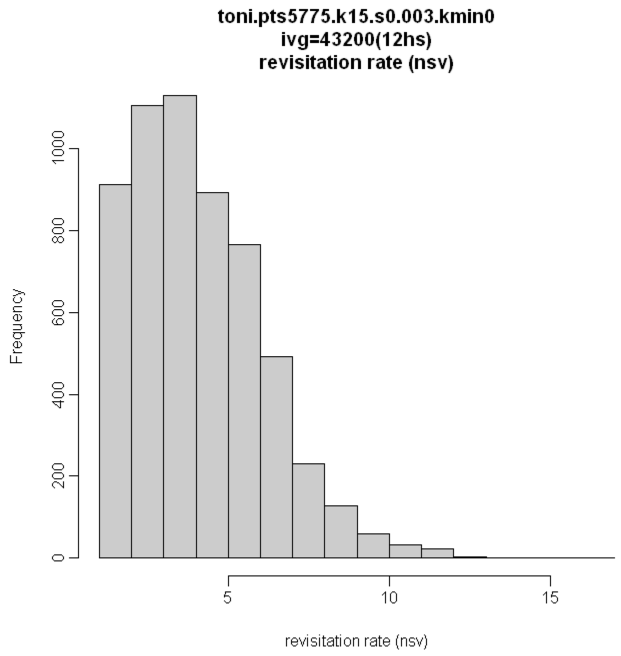

toni
Elongation Distribution
Iso Level
0.95
0.85
0.75
0.65
0.55
0.45
0.35
0.25
0.15
y
x
7305000
7310000
7315000
7320000
7325000
7330000
365000
370000
375000
380000
385000
390000
These isopleths were constructed from 5775 hulls sorted by elongation (ecc). Isopleth levels indicate the proportion of total points enclosed. Hulls created from 5775 locations of toni using the Fixed-k method (k=15, s=0.003, kmin=0, 9 duplicate points offset by 1 map units).

We can see the areas with the greatest elongation (red areas) include those areas where toni was at the periphery of her home range. You may be wondering why the core area looks like Swiss cheese now—didn't we select *k*=15 precisely because it did a good job filling in those little holes? Yes we did, as seen in the plot of the utilization distribution on page 18. The holes we see in the core area of the elongation isopleths are there because 1) the biggest isopleth shown is only 0.95, so 5% of the points are not included, and 2) the hulls have been sorted by elongation, not density, so the 5% of points that are not enclosed by these isopleths fall in the 5% least elongated (which tend to be in the core areas where the hulls are more circular).

This plot also reveals some large hulls on the right side where the animal had a single excursion. These large hulls are not an error, but they definitely over-estimate that portion of the range. They're large because we used *k*=15, so every hull was made from its 15 nearest neighbors, and the points are thin in that part of the range. Before, these large hulls didn't matter too much because we were looking at the density isopleths, and *k*=15 did a reasonable job for most of the range and these big hulls were not even included in the 95% density isopleth. However this is a good example where we should have used the *a*-method, which effectively reduces the number of neighbors used in hull construction in outlying areas.

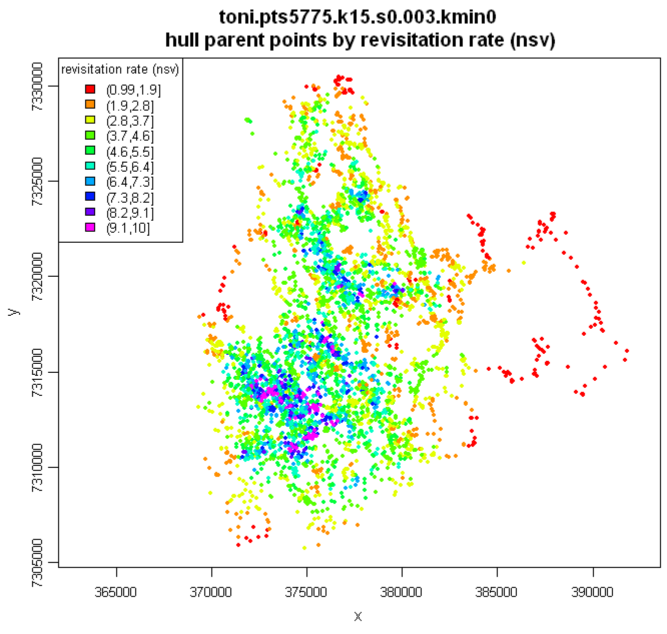
Next, lets look at the spatial patterns of revisitation. Let's look at both the histogram of revisitation, as well as a map of hull parent points colored by revisitation rate (`nsv`).

```
> hist(toni.lhs.k15, metric="nsv")
```



The histogram tells us that the majority of hulls were revisited between 1 and 4 times. There's a tail of hulls that had seven or more visits. To see where those highly revisited hulls are, let's plot the hull parent points classified by the `nsv` metric.

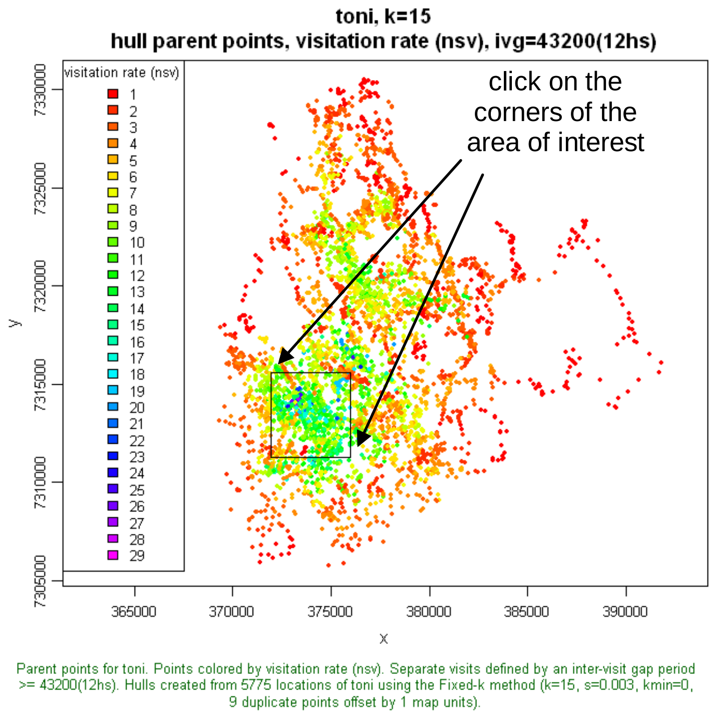
```
> plot(toni.lhs.k15, hpp=T, hpp.classify="nsv", ivg=3600*12, col.ramp="rainbow")
```



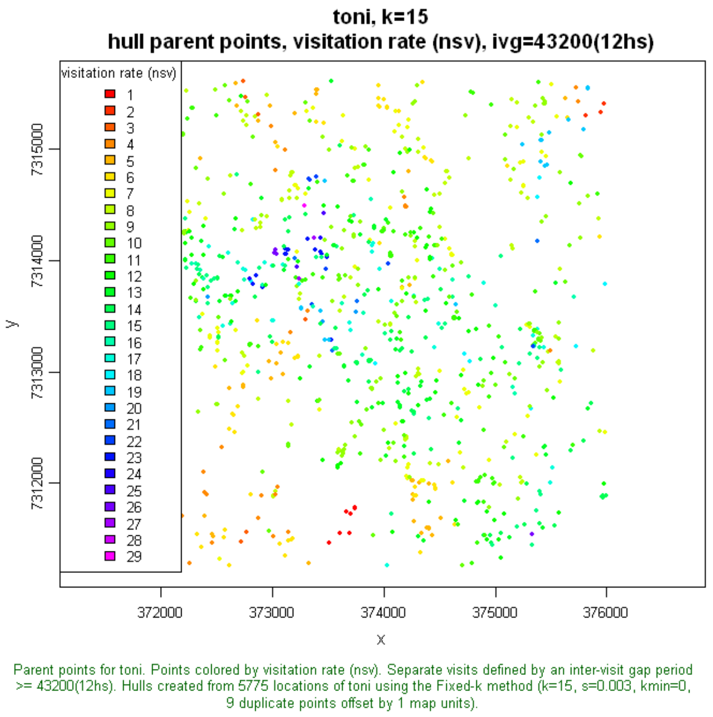
Parent points for toni. Points colored by revisitation rate (nsv). Hulls created from 5775 locations of toni using the Fixed-k method (k=15, s=0.003, kmin=0, 9 duplicate points offset by 1 map units).

Our eyes should zoom in on the dark blue and purple dots. They seem to fall in a few different clusters, suggesting these areas contain resources that toni needed repeatedly. How about we zoom into one of those areas? We can do that that by passing to the plot function an area of interest (aoi) parameter containing the coordinates of the bounding box we want to focus on. We could type the coordinates in manually, but an easier way is with the mouse. The aoi() function will prompt you to click on the current plot window (make sure you leave it open) twice. The first click should be the upper left corner of the box, and the second click should be the lower right corner. We'll save these coordinates in a new variable and then pass that to the plot function.

```
> toni.aoi <- aoi()

Click *two* points on the active plotting window that define a box
```


toni, k=15
hull parent points, visitation rate (nsv), ivg=43200(12hs)
visitation rate (nsv)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
click on the corners of the area of interest
y
7305000 7310000 7315000 7320000 7325000 7330000
x
365000 370000 375000 380000 385000 390000
Parent points for toni. Points colored by visitation rate (nsv). Separate visits defined by an inter-visit gap period >= 43200(12hs). Hulls created from 5775 locations of toni using the Fixed-k method (k=15, s=0.003, kmin=0, 9 duplicate points offset by 1 map units).

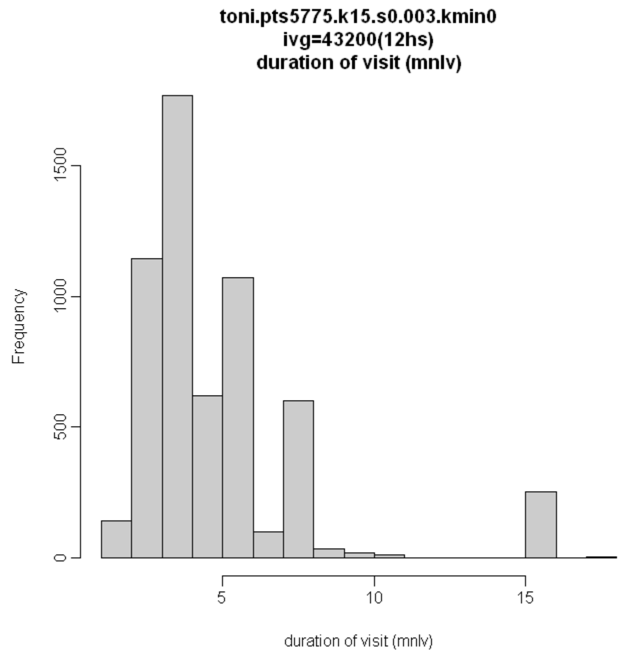
```
> plot(toni.lhs.k15, hpp=T, hpp.classify="nsv", col.ramp="rainbow", aoi=toni.aoi)
```



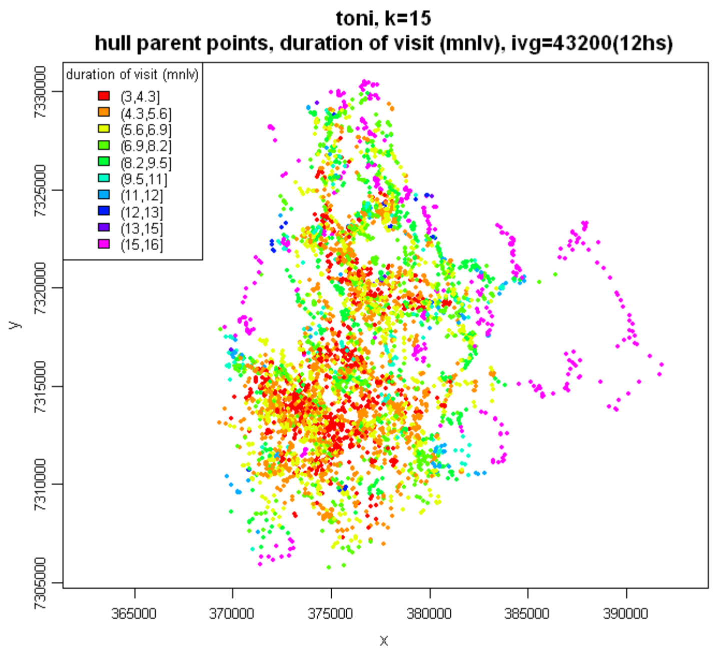
Looking at the color of the dots tells us that there are a lot of hulls in this area that were revisited more than 10 times, and if we remember the histogram of revisitation rates we created earlier, this is certainly a 'hotspot' of revisitation. If we had some GIS data or a satellite image to plot in the background (see Appendix IV), we could take an educated guess why they come back so much. We might also learn something if we were to create a scatterplot of revisitation vs. the hour of day and/or season (which can provide clues if it’s a water point).

Now let's look at the duration of each visit:

```
> hist(toni.lhs.k15, metric="mnlv", ivg=3600*12)
```



```
> plot(toni.lhs.k15, hpp=T, hpp.classify="mnlv", col.ramp="rainbow")
```

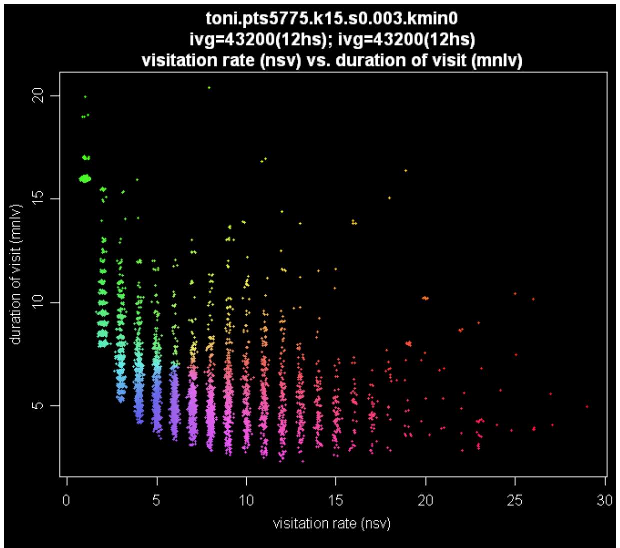


Parent points for toni. Points colored by duration of visit (mnlv). Separate visits defined by an inter-visit gap period >= 43200(12hs). Hulls created from 5775 locations of toni using the Fixed-k method (k=15, s=0.003, kmin=0, 9 duplicate points offset by 1 map units).

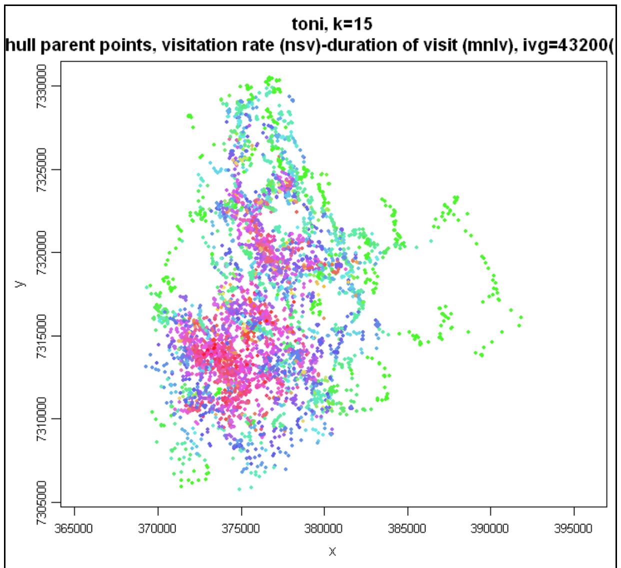
This is interesting—the hotspot of high revisitation seems to also be a cluster of low duration visits. This is a telltale signature of watering holes – animals have to come back daily, but stay there just long enough to fill up because they but don't want to hang around and risk getting eaten. To fact-check our hypothesis, we should probably zoom into the area and maybe also repeat the analysis with different values of IVG. The purple dots around the edge of the habitat would indicate those are long duration areas, but actually that's an artefact of our choice of method. Because we used the *k*-method with *k*=15, every hull is constructed from 15 nearest neighors. Furthermore if there are no repeat visits to the area, the hull will consist of 15 temporally contiguous points—a single 'visit' with 15 locations. We would have been better to use the *a*-method which results in fewer neighbors in outlying areas.

Lastly, we'll create a scatterplot of the hull revisitation and duration, and use this as a map legend. Note how we use a spiral color pattern to help us see where the points fall in the scatterplot, and we've also given the scatter plot a black background to make the colors stand out more. When we create the hull metric scatterplot, we save it as an object (hsp) and then feed that object into the plot function for the hull parent points.

```
> hsp <- lhs.plot.scatter(toni.lhs.k15, x="nsv", y="mnlv", col="spiral",
bg="black")

> plot(toni.lhs.k15, hpp=T, hsp=hsp, hpp.classify="hsp")
```



Each point represents a hull (n=5775). On the x-axis is visitation rate (number of separate visits). On the y-axis is duration of visit (mean number of locations in the hull per visit). Separate visits defined by an inter-visit gap period >= 43200(12hs). x values have been 'jiggled' by 0.1 to better see point density. y values have been 'jiggled' by 0.05 to better see point density.


toni, k=15
hull parent points, visitation rate (nsv)-duration of visit (mnlv), ivg=43200(
y
x

Parent points for toni. Points colored by their location in nsv-mnlv space. Separate visits defined by an inter-visit gap period >= 43200(12hs). Hulls created from 5775 locations of toni using the Fixed-k method (k=15, s=0.003, kmin=0, 9 duplicate points offset by 1 map units).

The colors on the map separate surprisingly well, in other words the distribution of colors don't appear random at all. We should probably zoom in to some of the area as we did before to double-check the amount of variability, but it appears these two dimensions of time-use do a relatively good job in dividing the landscape in to discrete regions. What other hull metrics might be useful for classifying the movement of a buffalo? To see a list of all possible metrics, type `hm.expr()`. To see a bunch of scatterplots

```
> lhs.plot.scatter.auto(toni.lhs.k15)
```

6. References

Getz, W., S. Fortmann-Roe, et al. (2007). *LoCoH: Nonparameteric Kernel Methods for Constructing Home Ranges and Utilization Distributions*. PLoS ONE 2(2): e207.

Getz, W. and C. Wilmers (2004). *A local nearest-neighbor convex-hull construction of home ranges and utilization distributions*. Ecography 27: 489.

Lyons, A., Turner, W.C., and WM Getz. 2013. *Home range plus: A space-time characterization of movement over real landscapes. BMC Movement Ecology 1:2*, *doi:10.1186/2051-3933-1-2*.

Appendix I. Glossary

ancillary variable	additional variables that go with each location and/or hull. Ancillary variables can either be collected by the GPS device (e.g., temperature, pulse rate) or computed later (e.g., distance to water).
auxiliary parameter	a required 'extra' parameter that goes with (and thus defines) a hull metric. For example, the hull metric 'number of separate visits' (i.e., revisitation) requires an auxiliary parameter called 'ivg' (inter-visit gap)
burst	A group of sequential locations that are close together in time relative to the median sampling interval for the entire dataset. A burst can be the result of hyperactive hardware (i.e., error) or intended by programming the GPS recorder.
core	A concept in behavioral ecology of the primary area used by an individual. In practice, often taken to be an area that captures 50% of the observed (or predicted) occurrences.
ffmpeg	A cross-platform open source software program that (among other things) takes a series of still images and converts them to a video animation
frame	A single image in an animation
home range	‘that area traversed by the individual in its normal activities of food gathering, mating and caring for young. Occasional sallies outside the area, perhaps exploratory in nature, should not be considered as in part of the home range.’ Burt (1943, p. 351); in practice the 'home range' is taken to be an area which encloses 95% of known locations
hsp region	A manually digitized (i.e., drawn with the mouse) region of a hull scatterplot
hull scatterplot	A scatterplot of two hull metrics, typically created with the `lhs.scatter()` function. In addition to displaying a plot, the `lhs.scatter()` function returns an 'hsp' object containing the parameters for the scatterplot so it can be easily created again. `lhs.scatter()` also has options to apply a color pattern (either a spiral pattern or manually digitized regions). A 'hsp' object created by `lhs.scatter()` can also be saved with the Locoh-hullset object (`lhs.hsp.add`), and can be used as a legend for map of hull parent points, or (in the case of manually digitized regions) as a filter.
hull	A minimum convex polygon constructed around each location
hullset	see *locoh-hullset*
isopleth level	The upper bound of an isopleth, usually expressed as a quantile of points enclosed. For example an isopleth with isopleth level = 0.95 would enclose 95% of the locations.

isopleth	A contour line that defines a subset of points based on probability of occurrence or some other metric
lhs	common name given to *locoh-hullset* objects
locoh-hullset	One of the main types of objects used in the T-LoCoH package, containing one or more sets of hulls and associated hull metrics and isopleths (page 4)
locoh-xy	An object used by the T-LoCoH package, containing locations, time stamps, nearest neighbor lookup tables, and ancillary variables for one or more individuals (see page 3).
lxy	common name given to *locoh-xy* objects
MPI	Minimum Proportion Inclusion – a principle for selecting the smallest acceptable value for the k/a/r parameter. The principle says the lower bound for k/a/r is the value such that *p*% of all points are included in at least one hull constructed from at least *n* points, where *p* and *n* are selected by the analyst in reference to the research question and characteristics of the data.
MSHC	Minimum Spurious Hole Covering – a principle for selecting a value for the k/a/r parameter. The principle says you should pick a value that is large enough that the resulting hulls cover up 'spurious holes', which are holes that are not 'real'.
PNG	Portable Network Graphics, a lossless file format for bitmap images
s	A parameter used in the TSD equation that determines how much time influences the TSD 'distance' between two points. When *s*=0, time plays no role and TSD is equivalent to Euclidean distance. As *s* gets large, the time interval between points dominates spatial distance.
THE	True Hole Exclusion – a principle for selecting a value for the k/a/r parameter. The principle says you should a value that is small enough to exclude true holes in the movement pattern.
TSD	Time Scale Distance. A 'distance' metric or measurement that combines separation in time with distance in space. TSD for any two points *i* and *j* is given by: $\Psi_{ij} = \sqrt{\Delta x_{ij}^{\,2} + \Delta y_{ij}^{\,2} + \left(s v_{\max} \Delta t_{ij}\right)^2}$ where v_{max} is the maximum observed velocity between any two consecutive points, and *s* is a dimensionless scaling parameter provided by the analyst (s≥0).
utilization distribution	Spatial contours that differentiate a home range area based on the estimated probability of occurrence (i.e., density of observations)

Appendix II. Manually Installing the T-LoCoH Package

Hopefully you were able to install the T-LoCoH package using the following R command:

```
install.packages("tlocoh", dependencies=TRUE, repos=c("http://R-Forge.R-project.org",
"http://cran.cnr.berkeley.edu"))
```

If that command failed, perhaps because you don't have a good internet connection or there isn't a build for your machine, you can manually install the package as follows.

1. Install the Dependent Packages

Like most R-packages, T-LoCoH requires a handful of other packages to work. The above command should install the dependent packages, but you can also install them by pasting in the following three lines into your R console:

```
dep.pkg <- c("pbapply", "sp", "FNN", "rgeos", "rgdal", "maptools", "png")

pkgs.not.installed <- dep.pkg[!sapply(dep.pkg, function(p) require(p, character.only=T))]

install.packages(pkgs.not.installed, dependencies=TRUE)
```

Package **gpclib** is a special case because a binary version does not exist for either Windows or Mac. You therefore have to install **gpclib** from source (see command below). On Windows, installing packages from source requires first installing RTools[9].

```
## Mac OS:
if (!require(gpclib)) install.packages("gpclib")

## Windows:
if (!require(gpclib)) install.packages("gpclib", type="source")
```

gpclib is only called upon if the polygon union function in **rgeos** can't deal with something (rare). So if you can't get **gpclib** to install, don't panic, T-LoCoH will still work in most cases.

2. Download the T-LoCoH Package

Download the package from the following website. If you're on Windows, download the windows binary version (zip file). Mac and Linux users can download the source version (tar.gz file).

https://r-forge.r-project.org/R/?group_id=1622

After you've downloaded the correct version of the package, do not expand the zip or gz file.

[9] http://cran.r-project.org/bin/windows/Rtools/

3. Install the T-LoCoH Package

Windows: Install the package by going to the 'Packages' menu on the R console and selecting 'Install packages from local zip files'.

Mac: Select ‘Package Installer’, then select ‘Local Source Files’. Click the ‘Install’ button and pick the file you downloaded.

4. Load T-LoCoH into Memory

Once T-LoCoH is installed, you can load it into memory by typing:

```
require(tlocoh)
```

Appendix III. Importing GPS Data into R

How you get your GPS locations into R depends on how it is stored (e.g., csv files, Excel spreadsheet, ODBC database). There are many ways to import tabular data into R, too many to describe here. Suffice to say it is often a pain. For a more user-friendly graphical interface for R that comes with menu commands for importing data, see the R package `Rcmdr`.

At the end of the importing process, you should have an R data frame or matrix for the x and y coordinates. The x and y coordinates of your points are the *only* two objects that are absolutely necessary to use T-LoCoH. If your points are time-stamped, as most GPS data is, and you plan to use the time features of T-LoCoH, then you also need a vector of the same length as your locations containing the date-time values of each point. This vector can either contain date formatted strings (see below), or a vector of the R class *POSIXct* or *POSIXlt.*

If you have points from multiple individuals that you want to analyze separately, you can either create separate data frames for each animal (which will become separate *locoh-xy* objects), or put them all together in one combined data frame, and make a character vector (or factor) of equal length containing the names of the individual associated with each point.

Importing Coordinates from a Shapefile

If your locations are saved in a point shapefile, you can import them into R using the maptools package as follows.

```
> require(maptools)
> fn <- "C:/GIS/raccoon_pts_2007.shp"
> shp.pts <- readShapePoints(fn, verbose=T)
Shapefile type: Point, (1), # of Shapes: 851

> xys <- coordinates(shp.pts)
> head(xys)
  coords.x1 coords.x2
0  30.16230 -15.08048
1  30.21131 -15.11760
2  30.21768 -15.15861
3  30.21988 -15.18552
4  30.20558 -15.19026
5  30.16669 -15.19447
```

If there is a field in the attribute table for the time stamps, you can grab those with a couple more steps:

```
> names(shp.pts)
 [1] "AREA"       "PERIMETER"  "PPPOINT_"
 [4] "PPPOINT_ID" "PPPTTYPE"   "PPPTTYPETX"
 [7] "PPPTNAME"   "DATE"   "LONG"       "LAT"
> timestamps <- shp.pts$PPTNAME
```

Importing Date-Times
Importing date-time values into R can be a painful, but it’s a necessary hurdle if you want to use the time-enhanced features of T-LoCoH. Due to the variety of ways spreadsheets and databases save dates and times, you may want to try exporting your date-time as a character field formatted in a style that R will recognize as a time stamp. To see what a date-time format looks like that R will be able to convert to a date-time field, type the following command:

```
> Sys.time()
[1] "2012-01-28 09:02:31 PST"
```

Your goal is then to format the dates in your spreadsheet or database to match this format, then export the values as text, after which R should be able to import and convert them successfully.

In MS Access you can use the format() function to format a date field to match the above (i.e., in a query):

```
    format(PointDate, "yyyy-mm-dd hh:nn:ss")
```

In Excel, you can go to Format – Cells and give the date-time cells a custom format:

```
    yyyy-mm-dd hh:mm:ss
```

Time Zones
R saves the time zone of time values. If the time zone is omitted when you import your data into R, R will presume the data is in either the time zone of your computer or UTC (GMT). Many satellite GPS loggers record the time stamp in GMT, so this is often fine. But you may want to convert your time values to the local time of the study site, particularly if you're interested in how the hour of the day may be associated with movement patterns.

Suppose we have a character vector of times called *gps.times*, that doesn't have a time zone specified but we know in our heart of hearts that the time values are actually in UTC. Next, suppose we want to convert these times to local time at the study site so we can see if the hour of day affects movement, and our study site is in GMT+2. We can convert a character vector of UTC times to an R object of class POXITct in GMT+2 in the following way:

First, let's view what we have:

```
> head(gps.times)
[1] "2010-01-25 12:02:46"
[2] "2010-01-25 13:03:10"
[3] "2010-01-25 14:02:40"
[4] "2010-01-25 15:02:46"
[5] "2010-01-25 16:03:03"
[6] "2010-01-25 17:02:36"
```

The format of the time values looks good because it matches the R format for times, and we note again that there is no time zone recorded. We'll first convert this character vector into a vector of *POSIXct* objects (R's class for time values), explicitly specifying that the time values are in UTC:

```
> dt.utc <- as.POSIXct(gps.times, tz = "UTC")
> dt.utc[1]
[1] "2010-01-25 12:02:46 UTC"
```

Finally, we can convert this POSIXct object into another POSIXct object with a different time zone. Unfortunately, R doesn't recognize "UTC+2" as a valid time zone name ☹, so first we need to check the time zone names that are recognized by our specific version of R.

```
> tzfile <- file.path(R.home("share"), "zoneinfo", "zone.tab")

> tzones <- read.delim(tzfile, row.names = NULL, header = FALSE, col.names = 
c("country", "coords", "name", "comments"), as.is = TRUE, fill = TRUE, comment.char 
= "#")

> sort(tzones$name)
  [1] "Africa/Abidjan"
  [2] "Africa/Accra"
  [3] "Africa/Addis_Ababa"
  [4] "Africa/Algiers"
```

That's quite a list, but we can quickly find the time zone name for our study site. With this piece of information, we finish it off with:

```
> dt.bw <- as.POSIXct(format(dt.utc, tz = "Africa/Gaborone"), tz="Africa/Gaborone")
> dt.bw[1]
[1] "2010-01-25 14:02:46 CAT"
```

If all this seems a like a lot of work just to convert time zones, that's because it is. Picking apart the last function, the 'tz' parameter within the format() function converts the time values in *dt.utc* from UTC to Central Africa Time (GMT+2) (i.e., it adds two hours to each time). However because format() returns a character vector rather than a POSIXct object, the time zone is not stored. The tz parameter in the outer as.POSIXct() function tells R that these date values are in Central Africa Time; without this second tz parameter the time zone recorded would default back to either UTC or the current time zone on your computer. That's when it's appropriate to scream *aaaRRRRggghh*! But at least now we can now feed *dt.bw* into the function that creates an *lxy* object and we shouldn't have to worry about it again.

Projecting Coordinates into Real World Units

While T-LoCoH can analyze locations in any coordinate system, it is strongly recommended that all coordinates be projected into a coordinate system with 'real world' units, such as meters or feet. Geographic coordinates (longitude-latitude) are <u>not</u> a real world coordinate system, because degrees are neither meaningful or consistent measures of length or area (i.e., 0.005 degrees will be a different length on the ground depending if you're at the equator versus Argentina).

Most GIS programs can project coordinates into real world coordinate systems. You can also do this in R using the *sp* package. In the next example, we will convert our coordinates into UTM, which is a common coordinate system. UTM divides the planet into 36 zones based on longitude[10]. To transform the coordinates, we need to first create a 'sp' object (a common R class for spatial data).

```
> require(sp)
> head(pts.latlong.df)
         x         y
1 21.03120 -33.96367
2 21.03638 -33.96733
3 21.04206 -33.96638
4 21.03891 -33.96523
5 21.03483 -33.96283
6 21.03304 -33.95979

> pts.latlong.sp <- SpatialPoints(pts.latlong.df, proj4string=CRS("+proj=longlat
+ellps=WGS84"))

> pts.utm.sp <- spTransform(pts.latlong.sp, CRS("+proj=utm +south +zone=34
+ellps=WGS84"))

> pts.utm.df <- coordinates(pts.utm.sp)
> head(pts.utm.df)
            x       y
[1,] 502882.8 6241872
[2,] 503360.7 6241466
[3,] 503885.5 6241571
[4,] 503594.6 6241699
[5,] 503217.4 6241964
[6,] 503052.3 6242302
```

[10] to find out which UTM zone your data fall in, visit http://www.dmap.co.uk/utmworld.htm or plug one of your coordinates into a geographic/utm coordinate converter such as http://home.hiwaay.net/~taylorc/toolbox/geography/geoutm.html

Appendix IV. Displaying Spatial Data in the Background

Displaying Shapefiles

The plot and animation functions[11] in T-LoCoH have the ability to display shapefiles (a common file format for vector GIS data) in the background. You can specify the symbology attributes, including fill and border color, plot character, size, and line style.

The first step to making this work is to prepare the GIS data. This includes 1) projecting the data into the same coordinate system as the movement data, and 2) saving it as shapefiles. Note that the R functions that read shapefiles can have difficulty if there is a feature with a NULL shape or other irregularities, so if you run into that problem you may also have to use your GIS software to delete all records with NULL shapes.

Secondly, you need to tell T-LoCoH what shapefiles to display, and how to display them. There are two ways to do this: 1) creating a csv file with symbology info, or 2) passing a list.

Option 1. Create a csv file

The first option is to create a csv file (comma separated value) with columns containing the required information to display shapefiles. This option works well if you'll be plotting the same base layers often, because once the csv file is set up you can specify which layers to include by simply providing their assigned names, like 'roads' or 'vegetation'. The csv file should have the following columns:

Column heading	Contents
layer	layer name
fn	shape file name (either absolute, relative to the working directory, or relative to the folder where the csv file resides)
type	layer type: *polygon*, *line*, or *point*
lty	line type (for line features only): 0=blank, 1=solid, 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash
pch	plot character number (for point features only), see http://rgraphics.limnology.wisc.edu/images/miscellaneous/pch.png
cex	size of plot character (used for point layers only)
lwd	line width (used for line layers only)
border	border color (polygons only), TIP: to hide the border set to NA
color	color of the object

csv files are plain text and can be edited either with a text editor like notepad or a spreadsheet program like Excel. To see an example of a csv file with the above columns, you can copy the `kruger_gis.csv` sample file that comes with T-LoCoH, which you can locate by typing:

```
> system.file("shps", "kruger_gis.csv", package="tlocoh")
[1] "C:/Program Files/R/R-2.15.0/library/tlocoh/shps/kruger_gis.csv"
```

Once you have a) prepared your GIS data and b) created a csv file with the symbology parameters, you can add GIS layers to your plots by passing 1) the `shp.csv` parameter to tell

[11] including `plot.locoh.lxy()`, `plot.locoh.lhs()`, `lhs.exp.mov()`, and `lxy.exp.mov()`

T-LoCoH where the csv file is, and 2) the `layers` parameter to specify the names of the layers to display.

```
> plot(lxy, shp.csv="yosemite_gis.csv", layers=c("roads", "water"))
```

Option 2. Pass a list object
If you don't want to take the trouble to create a csv file, you can tell T-LoCoH which shapefiles to display in the background by setting the `layers` parameter to a specially formatted list object. In this case, `layers` should be a list with one element for each layer to be displayed, with each element equal to list with named elements matching the columns of the csv file (above).

```
> layers.lst <- list(roads=list(layer="roads", fn="c:/gis/roads.shp", type="line",
pch=0, lty=2, lwd=1, col="#8B4513CC"))

> plot(lxy, layers=layers.lst)
```

Displaying TIFF Files

The functions for plotting and creating animations of *locoh-hullsets* and *locoh-xy* objects have parameters which allow you to display a raster image (such as a satellite image) in the background. To use these parameters, you must prepare the image(s) as follows:

- The pixel values in the image must be 'prestretched' for display. Quite often, the pixel values in a remotely sensed image have a very narrow range and must be stretched to avoid appearing very dark and/or with very little contrast. GIS programs will often stretch the pixel values automatically so they look better on the screen. T-LoCoH doesn't support on-the-fly contrast stretching, so you must save the image with the pixel values already stretched (this is an option in many remote sensing packages, you can also do it in R).
- The image must be in the same projection system as the other spatial data being plotted including the locations.
- The image should be saved as a GeoTIFF (*.tiff) or another raster format that can be read by the RGdal package[12].

After your image is prepared, the parameters to display the image are as follows:

Parameter	Purpose
`tiff.fn`	Filename of the image
`tiff.bands`	`tiff.bands` is a vector with exactly one or exactly three integers that correspond to the bands in the image[13]. In the case of a multi-band image, these numbers define which bands will be displayed using the red, green, and blue color guns respectively. In a Landsat TM image, for example, the first four bands are blue, green, red, and infrared. To display a TM GeoTIFF image as 'natural colors', you would set `tiff.bands=c(3,2,1)`.

[12] for a complete list of supported raster formats, run `gdalDrivers()`
[13] even though the argument name is 'tiff.bands', this argument is used for all images including those that are not saved as tiff

	To display a single band from a multi-band image, pass a single value for `tiff.bands`. You can also let `tiff.fn` be a single-band image (in which case `tiff.bands` will be ignored). See also `tiff.col`.
`tiff.col`	When displaying a **single band** image, `tiff.col` should be the color values used to display the image. If the single-band image contains five land cover categories, for example, you could set `tiff.col= rainbow(5)`. If the single-band image contains continuous elevation values (i.e., a DEM), you could display them with 256 shades of gray with `tiff.col=gray(0:255/255)` or perhaps topographic colors with `topo.colors(255)` or `terrain.colors(n)`. `tiff.col` is ignored if displaying three bands from a multi-band image (see `tiff.bands`).
`tiff.pct`	When `tiff.pct=T`, the script will create a indexed 256-color version of the image, which may result in quicker drawing time particularly if several plots are being drawn
`tiff.buff`	`tiff.buff` can be used to expand the range of values on the x and y axis so that you see a bit of the background image beyond the extent of the points. This is a value in map units, so for example if your image is in UTM (meters) and `tiff.buff=500`, the plot would 'zoom out' so you see the area containing the locations plus a 500m buffer.
`tiff.fill.plot`	Fill the plot area (all the way to the axes). `TRUE | FALSE`

Appendix V. T-LoCoH Functions

locoh-xy Functions

Function	Description
Create a locoh-xy object	
xyt.lxy	Create a *lxy* object
move.lxy	Convert a Move object from the move[14] package to a *lxy* object
Manipulate and Manage locoh-xy objects	
summary	View a summary of an *lxy* object
lxy.save	Save a *lxy* object to disk using an intelligent file name
lxy.repair	Repair a *lxy* object
lxy.merge	Merge two *lxy* objects together
lxy.subset	Create a new *lxy* object containing a subset of points
lxy.id.new	Assign new id value(s)
lxy.anv.add	Add an ancillary variable
lxy.proj.add	Add projection information
lxy.reproject	Reproject locations
lxy.gridanv.add	Add ancillary values from one or more rasters
Clean and Thin Data	
lxy.thin.bursts	Thin out short-timed 'bursts' of points which were an artifact of the recording hardware
lxy.thin.byfreq	Selectively remove points to achieve a common time period and/or sampling frequency for a *lxy* object containing the locations for multiple individuals.
Selecting Space-Time Balance	
lxy.ptsh.add	
lxy.plot.ptsh	
lxy.plot.sfinder	Plot the values of s which equalize the spatial and temporal terms in TSD
lxy.plot.tspan	Plot the distribution of the time span of nearest neighbors for different values of s
lxy.plot.mtdr	Plot distribution of the ratio the maximum theoretical distance ratio for nearest neighbors
Identify Nearest Neighbors	
lxy.amin.add	Compute a value that ensures all points have enough neighbors
lxy.nn.add	Identify nearest neighbors
Plotting functions	
plot	Plot an *lxy* object
hist	Create histograms of the properties of a *lxy* object, including step length, speed, and sampling interval
lxy.plot.freq	Plot the number of observations and/or sampling interval over time
lxy.plot.pt2ctr	Plot the distance of each point to the centroid to help find the periodicity of 'natural' cycles in the data

[14] http://cran.r-project.org/web/packages/move/

Function	Description
Export locoh-xy objects	
lxy.exp.csv	Export a *lxy* object to a csv file
lxy.exp.mov	Prepare frames for animation; create QuickTime video (or other video formats with a different encoder tools)
lxy.exp.kml	Export locations and time stamps to kml for animation in Google Earth
lxy.exp.shp	Export to shapefile format

locoh-hullset Functions

Function	Description
Creating Hullsets	
lxy.lhs	Create a *lhs* object from a *lxy* object
lxy.lhs.wiz	*lhs* creation wizard
Manipulating Hullsets	
summary	View a summary of a *lhs* object
lhs.save	Save a *lhs* object to disk using an intelligent filename
lhs.select	Take of subset of hullsets
lhs.merge	Merge hullsets together
lxy.anv.add.R	Add an ancillary variable
lhs.anv.del	Delete ancillary variable(s)
Plotting	
plot	Plot hulls, hull parent points, isopleths, and/or ellipses
Hull Metrics	
hm.expr	View all possible hull metrics
lhs.visit.add	Compute hull metrics for revisitation and duration
lhs.ellipses.add	Compute bounding ellipses
lhs.revisit.add	Compute interval-specific revisitation metrics
lhs.revisit.del	Delete interval-specific revisitation metrics
Isopleths	
lhs.iso.add	Create isopleths
lhs.iso.del	Delete saved isopleths
lhs.iso.rast	Convert isopleths to raster
Hull Metrics – Plots	
hist	View a histogram of hull metrics
lhs.plot.scatter	Create a scatterplot of two hull metrics
lhs.plot.scatter.auto	Create scatterplots for a whole bunch of pairs of hull metrics
lhs.plot.isoarea	Plot the area of each isopleth for each value of the k/a parameter
lhs.plot.isoear	Plot the edge:area ratio of each isopleth for each value of the k/a parameter
lhs.mf.plot	Multi-frame plot
lhs.hsp.add	Save a hull scatterplot in the *lhs*
lhs.plot.revisit	Plot revisitation
lhs.hsp.del	Delete a saved hull scatterplot

Function	**Description**
Exporting	
lhs.exp.shp	Export hulls, hull parent points, and/or isopleths as shapefiles
lhs.exp.mov	Create a Quicktime animation from a LoCoH-hullset object
lhs.exp.csv	Export hull metrics as a CSV file
hulls	Extract hulls as a list of SpatialPolygonsDataFrame objects
isopleths	Extract isopleths as a list of SpatialPolygonsDataFrame objects
Filtering	
lhs.filter.anv	Create subsets of hulls based on an ancillary variable
lhs.filter.hsp	Create subsets of hulls based on manually defined regions in a hull metric scatterplot space
Association Analysis	
lhs.so.add.R	Add metric(s) for temporally overlapping hulls
lhs.to.add.R	Add metric(s) for spatially overlapping hulls
lhs.pep.add	Compute a hull metric for proportion of enclosed points
Other	
lhs.dr.add	Identify directional routes, which are segments between temporally contiguous points whose hulls are also highly elongated

Large datasets functions*

Function	**Description**
Large Datasets	
lxy.lhs.batch	Create separate *lhs* objects saving each one to disk as a separate file (for large datasets)
lhs.exp.isodata	Extract and compile isopleth attributes from multiple hullsets saved to disk
isodata.plot	Plot dataframe compiled by *lhs.exp.isodata*

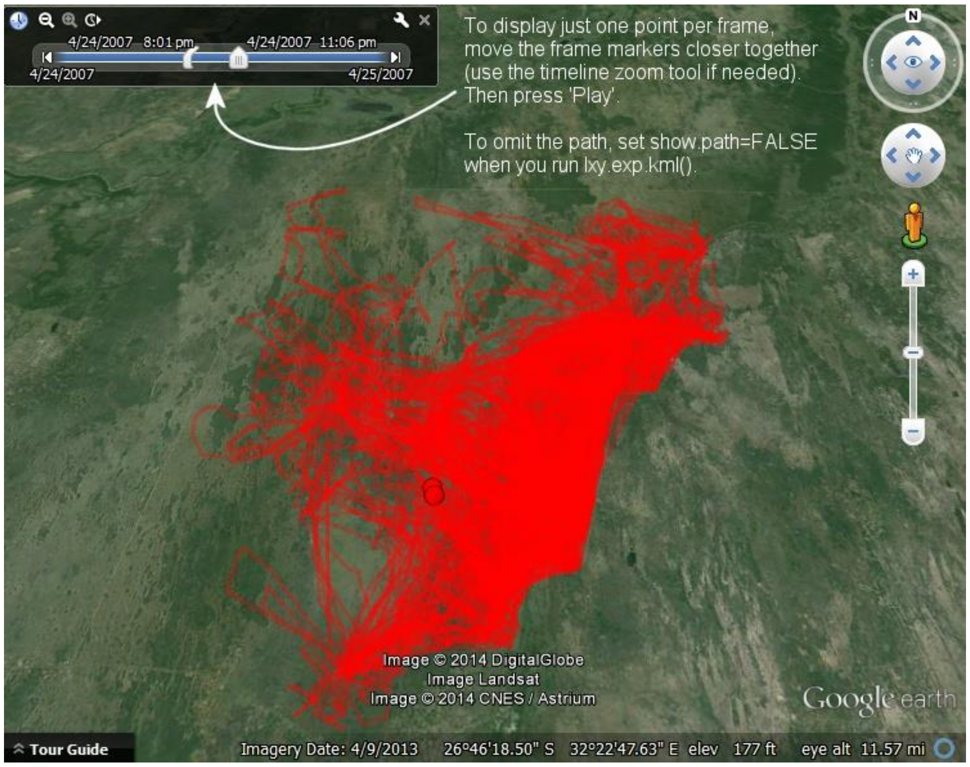
*to help deal with the memory limits of R

Appendix VI. Creating an Animation

Animations can be a good way to see patterns in the data, including cycles, synchronicity, foraging vs. searching behavior, etc. This obviously requires that your data have time stamps saved.

Animating with Google Earth

T-LoCoH has two functions that can be used to visualize movement data through an animation. For a quick animation using Google Earth, run `lxy.exp.kml()`. This will export a *locoh-xy* object as a kml file which you can then animate using Google Earth. When you open the kml (or kmz file if compression was turned on) in Google Earth, the animation toolbar should appear. Adjust the time span of the display by moving the time start marker, then click the Play button. Options in the xy.exp.kml include setting the color and opacity of the path (or hiding it completely), setting the color of the place mark symbols, and exporting everything nth point.


4/24/2007 8:01 pm
4/24/2007 11:06 pm
4/24/2007
4/25/2007
Image © 2014 DigitalGlobe
Image Landsat
Image © 2014 CNES / Astrium
Google earth
Tour Guide
Imagery Date: 4/9/2013 26°46'18.50" S 32°22'47.63" E elev 177 ft eye alt 11.57 mi

To display just one point per frame, move the frame markers closer together (use the timeline zoom tool if needed). Then press 'Play'.

To omit the path, set show.path=FALSE when you run lxy.exp.kml().

Creating an Animation File

Alternately, the `lxy.exp.mov()` function can export a *locoh-xy* object as an animation file (Quicktime or MP4). The way T-LoCoH does this is to create each frame one-by-one, save them to disk as temporary PNG files, launch an encoding program that converts the frames into an animation, and then delete the temporary PNG files. T-LoCoH uses a great little open-source video encoder called *ffmpeg*, which among other things can take a series of

New in T-LoCoH version 1.17
You can now export an animation to **MP4** format (using the h.264 codec). The option gives excellent quality and is somewhat easier to embed online or in presentations. To export to mp4, set `fmt="mp4"`

still images and convert them into an animation. This means you have to download[15] and install the version of *ffmpeg* appropriate for your operating system before you can create animations in T-LoCoH. On Windows, *ffmpeg* is a single executable file (ffmpeg.exe) that you should save either in the R working directory or a directory on the system path environment variable (e.g., c:\windows)[16]. If you can't install *ffmpeg*, or you prefer to use a different encoder, you can also use T-LoCoH to create the individual frames and encode them using a different application (e.g., Quicktime Pro)[17].

The function that exports *locoh-xy* objects as animations is `lxy.exp.mov()`. There are generally three steps in the animation workflow; we will use `lxy.exp.mov()` for all three steps, telling it what to do by changing the parameters.

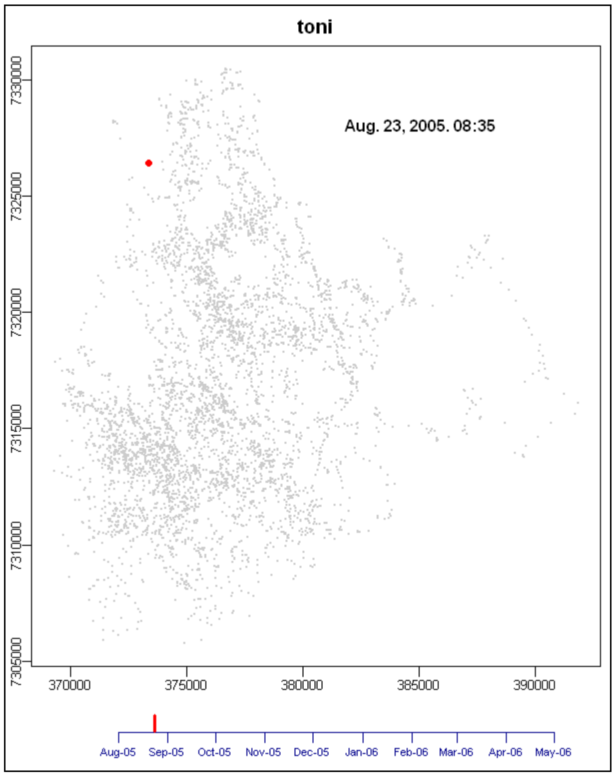
1. Design the frames
2. Select the encoding parameter: frame rate, duration, and/or skip factor
3. Create the frames and encode the video

The first task in creating an animation is to design the frame. This includes picking the frame size, deciding whether any GIS layers or images should be displayed in the background, whether a label for the date stamp should be displayed (and if so where), whether the frame should have a title, axes, etc.

To help us design the frame, let's first run `lxy.exp.mov()` with the default parameters setting `screen.test=TRUE` (so it won't create PNG files) and `max.frames=1`.

```
> lxy.exp.mov(toni.lxy, screen.test=TRUE, max.frames=1)
```

[15] http://ffmpeg.org/download.html
[16] alternately you can specify the full path to *ffmpeg.exe* with the `ffmpeg` argument in lxy.exp.mov()
[17] to create frames as a series of still images for encoding later, set the `tmp.dir` argument to a directory of your choice, `create.mov=FALSE`, and `tmp.files.delete=FALSE`



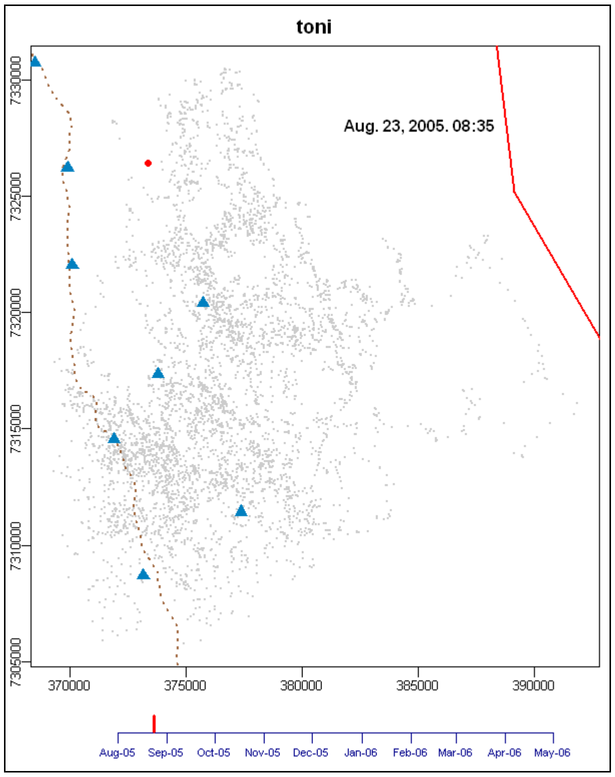
This isn't bad. The little red dot is the location in the current frame and the little gray dots are all the other locations[18]. If we wanted to hide the little date bar at the bottom, we could set `date.bar=0`, but this is a pretty long dataset so we'll keep it.

Next, let's add some GIS layers to help give some reference. T-LoCoH can display GIS data in shapefile format (see Appendix IV. Displaying Spatial Data in the Background , page 42). The T-LoCoH package comes with a few layers for the part of Kruger National Park where our buffalo Toni was collared, including the park boundary, watering troughs, and the main road. To add these layers, we need to add the `shp.csv` parameter to specify the location of a CSV file that contains the symbology properties for each layer (we'll use the csv file that comes with the package), as well as the `layers` parameter to specify the name(s) of layer(s) we want displayed (for more information, see page 42). We'll also set `crop.layers.to.extent=FALSE`, which can speed up the frame creation but is not needed.

```
> lxy.exp.mov(toni.lxy, screen.test=TRUE, max.frames=1, shp.csv="kruger_gis.csv",
layers="roads,troughs,boundary", crop.layers.to.extent=FALSE)
```

[18] if we didn't like the gray dots we could hide them by setting `col.xys.background=NA`

Now we see the water troughs as little blue triangles, the road as a dotted brown line, and the park boundary as a red line. If we wanted to adjust the appearance of these features we could edit the kruger_gis.csv file (see page 42).



New in T-LoCoH v1.17
You can also display raster images in the background of an animation. See page 42 or the help page for lxy.exp.mov on how to set `tiff.fn`, `tiff.bands`, `tiff.col`, `tiff.buff`, and `tiff.fill.plot`

Next, let's get rid of the title 'toni' title at the top of the frame by setting `title.show=FALSE` parameter (we'll make it part of the caption). And let's also move the date label to the lower right corner by specifying values for `dt.label.x` and `dt.label.y` parameters.

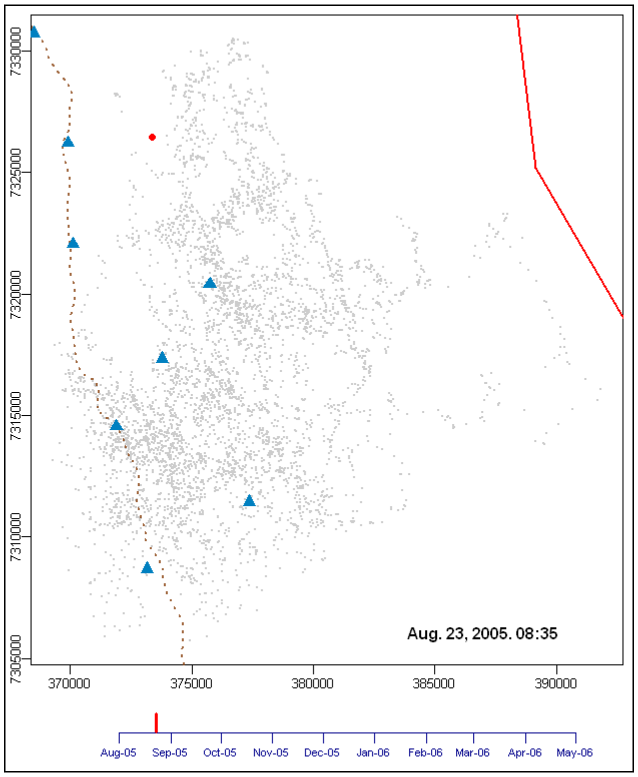
```
> lxy.exp.mov(toni.lxy, screen.test=TRUE, max.frames=1, shp.csv="kruger_gis.csv",
layers="roads,troughs,boundary", crop.layers.to.extent=F, dt.label.x=387000,
dt.label.y=7306000, title.show=FALSE)
```

Tip: A trick for getting the coordinates from the current plot window (for the label placement) is to use R's `locator()` function. This function will prompt you to click on the plot window then show you the coordinates where you clicked. Pass it a '1' to tell it that you only need one point.

```
> locator(1)
```

That's better. We can change the background color of the date label with the `dt.label.bg` argument. A background color is particularly useful if we are displaying an image in the background.

We're almost ready to generate the animation. What we've done so far is to create a sample frame in a plot window. The next step is to create a few sample frames as PNG images files, and if those are ok then we can do the final step which is to produce all of the frames and encode them into a video. To indicate that we want to make PNG files instead of a plot window, we omit the `screen.test` parameter, tell it to make 5 frames, and pass the name of a directory in the `tmp.dir` parameter. We want to inspect these PNG files after they've been created, so we'll specify "tmp" as the directory which will be created in the working folder. Normally the script will delete the PNG files when done, so we need to tell it not to delete them using the `tmp.files.delete` parameter.



Other Parameters You Can Use to Customize Your Animation

To see descriptions for all the parameters you can adjust to customize your animation, type `?lxy.exp.mov`. Below is a list of most of the parameters grouped by what they do:

- Animate multiple individuals simultaneously: `all.ids.at.once, all.ids.col.unique, all.ids.col, all.ids.legend, all.ids.legend.cex`
- Colors: `col.xys.active, col.xys.background, cex.xys.active, cex.xys.background, col.by.hour.of.day, col.hod`
- Time parameters: `dt.start, dt.end, frame.method, frame.rtd`
- Map extent: `xlim, ylim` (see also `tiff.buff`)
- Date label: `dt.label, dt.label.col, dt.label.bg, dt.label.x, dt.label.y, tz.local.check, tz.local`
- Date bar: `date.bar, date.bar.bins, col.db, cex.axis.db`
- Title: `title, title.show`
- Axes: `axes.show, axes.ticks, axes.titles`
- Display GIS layers: `shp.csv, layers`
- Display images: `tiff.fn, tiff.bands, tiff.col, tiff.buff, tiff.fill.plot`
- Speed-up drawing of background elements: `bg2png, crop.layers.to.extent, tiff.pct`
- Frame and font size: `width, height, png.pointsize`
- Margins: `mar.map, mgp.map`
- Frame generation: `max.frames, screen.test, tmp.dir, tmp.files.delete`
- Animation settings: `duration, fps, skip, ffmpeg, create.mov, prompt.continue, beep`
- Output file: `fn.mov, fn.mov.dir, fn.mov.exists`

We also need to think about the pixel size of each frame. Our frame is taller than wider, so let's specify a moderate width of 480 pixels and allow R to pick the height as needed. Finally, to create animation we need to set two of the following three parameters: `duration` (in seconds), `fps` (frames per second), and `skip` (display every nth frame). We'll fine-tune these in the next step, but for now we'll tentatively tell it to encode at 10 frames per second and set skip=1 (i.e., show every frame). Finally, because we're not quite ready to encode the video, we'll set `create.mov=FALSE`. After the PNG file(s) have been saved to disk (in a directory specified by the `tmp.dir` parameter), let's preview them using an image viewer.

```
> lxy.exp.mov(toni.lxy, max.frames=5, shp.csv="kruger_gis.csv",
layers="roads,troughs,boundary", crop.layers.to.extent=F, dt.label.x=387000,
dt.label.y=7306000, title.show=F, tmp.dir="tmp", tmp.files.delete=FALSE, width=480,
fps=10, skip=1, create.mov=FALSE)

Ready to generate animation
  Num frames=5. Duration=0.5secs. fps=10. Skip=1. Frame size: 480x592
  Temp folder: tmp
  Delete temp files: FALSE
  Record background as PNG: TRUE
  Mov file: <skipped>
Continue? y/n y
  Creating frames...
```

Use an image viewer to look at the PNG files it created. Presuming we're happy with the design of the frames, we next need to select parameter values for the temporal extent of each frame. Each frame can represent a single location, or a fixed interval of time. With the single location option, each frame will show the next point in the series, regardless of its time. If each frame represents a fixed interval of time, then some frames may have no location displayed if there were no points collected during that time interval (i.e., a data gap). Alternately, some frames could have more than one location. The default is to create location-based frames for a single individual, and time-based frames when animating multiple individuals simultaneously. We already checked above that our movement data for toni has very few gaps, so we will stick with the default for a single individual (location-based frames). If we wanted to be purist and switch the time-based frames, we would set `frame.method="time"`.

The last thing is to fine-tune the frame rate and skip factor. We'll take out the `max.frames` parameter so it will compute the duration for all frames, but we'll keep `create.mov=FALSE` because we're still not quite ready to actually launch the encoder. We'll also remove the `tmp.dir` and the `tmp.files.delete` parameter, so it will use the default settings which is to create the frames in a temp folder that we don't need to be concerned about and delete them when done.

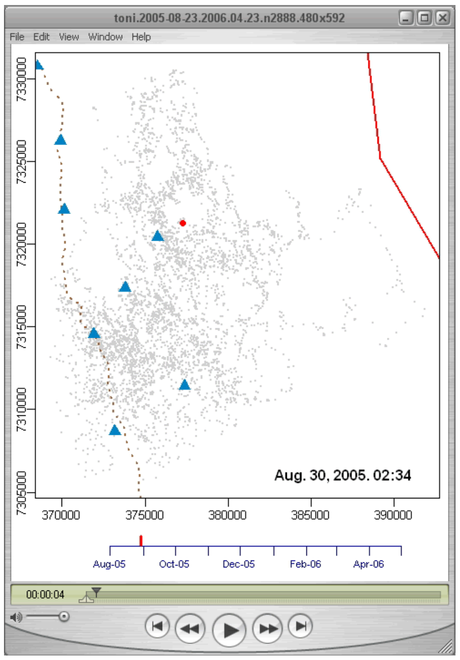
```
> lxy.exp.mov(toni.lxy, shp.csv="kruger_gis.csv", layers="roads,troughs,boundary",
crop.layers.to.extent=F, dt.label.x=387000, dt.label.y=7306000, title.show=F,
width=480, fps=10, skip=1, create.mov=FALSE)

Ready to generate animation
  Num frames=5775. Duration=577.5secs. fps=10. Skip=1. Frame size: 480x592
  Temp folder: C:\Users\Andy\AppData\Local\Temp\RtmpsVP043
  Delete temp files: TRUE
  Record background as PNG: TRUE
  Mov file: <skipped>
Continue? y/n
```

The output tells us that if we use a skip factor of 1 (all frames included) and a frame rate of 10 fps, the resulting animation will be 577 seconds, which is almost 10 minutes. That might be ok, but if we want to reduce that down to 2-3 minutes, which is the average attention span of a graduate student at a lab meeting, we can adjust the frame rate and/or skip factor. Since the sampling interval for this animal is 1 hour, we could set `skip=2` (show every other frame) which would cut the total duration in half and we probably wouldn't miss too much (depending on what the animation was for). We could also speed it up by setting `fps=20` (frames per second), which should reduce the duration in half again. Let's see what those changes give us.

```
> lxy.exp.mov(toni.lxy, shp.csv="kruger_gis.csv", layers="roads,troughs,boundary",
crop.layers.to.extent=F, dt.label.x=387000, dt.label.y=7306000, title.show=F,
width=480, fps=20, skip=2, create.mov=FALSE)

Ready to generate animation
  Num frames=2888. Duration=144.4secs. fps=20. Skip=2. Frame size: 480x592
  Temp folder: C:\Users\Andy\AppData\Local\Temp\RtmpsVP043
  Delete temp files: TRUE
  Record background as PNG: TRUE
  Mov file: <skipped>
Continue? y/n n
```

This looks acceptable. So we run the function one last time and change `create.mov=TRUE`. (If we wanted to do a test-run we could run it with `max.frames=100`).

```
lxy.exp.mov(toni.lxy, shp.csv="kruger_gis.csv", layers="roads,troughs,boundary",
crop.layers.to.extent=F, dt.label.x=387000, dt.label.y=7306000, title.show=F,
width=480, fps=20, skip=2, create.mov=TRUE)

Ready to generate animation
  Num frames=2888. Duration=144.4secs. fps=20. Skip=2. Frame size: 480x592
  Temp folder: C:\Users\Andy\AppData\Local\Temp\RtmpsVP043
  Delete temp files: TRUE
  Record background as PNG: TRUE
  Mov file: c:/locoh/toni.2005-08-23.2006.04.23.n2888.480x592.mov
Continue? y/n y
```



After about 25 minutes or so (most of which is spent creating the individual frames), you should have a Quicktime file[19]. Note we didn't specify a file name for the mov file—we could have specified a file name using the `fn.mov` parameter, but the script picks a file name which although a bit verbose tells you a lot about the content of the animation: the name of the animal(s), dates, number of frames, and frame size.

The default settings for the encoder use Quicktime's animation codec, which is a lossless compression method that also 'scrubs' really well, meaning you can drag the time indicator forward and backward in time without a lag in the screen refresh. Exporting as MP4 (`fmt="mp4"`) also produces good quality and is easier to embed in PowerPoint (especially on Windows) or a webpage

Play the video – what do you see? Does Toni seem to have favorite patches of vegetation she returns to? Do the waterholes seem to influence her movement? Does there seem to be any difference in movement pattern during the dry winter (April – October) versus the wet summer (November – March)?

[19] download the final animation at http://tlocoh.r-forge.r-project.org/toni.2005-08-23.2006.04.23.n2888.480x592.mov