An introduction to the 'move' package

Marco Smolla · Bart Kranstauber

Abstract

The move package provides a series of functions to import, visualize and analyze animal movement related data in an object-oriented way. The introduced classes allow one to store data that are related to an individual in one object, such as coordinates and timestamps, name, species, age, sex, study site, etc. This makes working with multiple animals, or multiple seasons much easier. Furthermore, the package includes functions to calculate home ranges (Minimum Convex Polygon Bootstrap) and utilization distributions (UDs). The latter uses the dynamic Brownian Bridge Movement Model to calculate animal UDs.

Keywords: animal track, time series, utilization density, gps.

Introduction

The amount of recorded animal movement data is increasing. The data offer information about land use, animal distribution, interactions between the environment and much more. The enormous amount of data needs to be analyzed with mathematical algorithms. The programming language R, developed by the R Development Core Team, offers an easy to learn environment to do statistical analyses. It even supports object-oriented programming. The package we present here offers additional functions that are specially made to analyze animal movement data.

Movement data that are associated with individual animals or a group of animals can be imported into R and stored as a single object. These objects can include all kinds of information. The minimum requirement however are timestamps, coordinates, and a unique ID per animal. Further attributes of the objects can be sensor types, body size, sex, weight, etc. A lot of data sets are available on a database called Movebank (www.movebank.org). If a user has the permission to see a data set, one can download it as a .csv (comma separated values) file. Files from Movebank can be directly imported into R.

A more recent update of the Move package has added a web-interface. Now it is possible to directly download whole data sets from within R. The only requirement is an account on Movebank and the rights to see the data (see the "browsing Movebank" vignette for more information).

Besides importing data from Movebank, it is also possible to use personal data sets, as long as they meet the minimum requirements of timestamps, coordinates, and animal IDs.

A series of functions allow one to plot, summarize, and analyze the movement data. Individual 'Move' objects can be stacked to a MoveStack object which includes a series of animals and the additional data that are associated with them. This allows one to work with many animals at the same time.

A new feature, called burst, allows one to subset track information. This can be interesting,

for example, if a track should be analyzed depending on specific behaviors (migratory, non-migratory, resting behavior).

One of the main features of the move package is the calculation of the utilization distribution (UD), a measure to estimate the home range of an animal. In this version of the package, we use the dynamic Brownian Bridge Movement Model (dBBMM) to calculate the UD. The UD can be calculated for individuals or for a group (stack) of individuals. This results in the object types 'DBBMM' or 'DBBMMStack'. These objects store all information that are needed to visualize the UD, including the raster it was calculated for, and the values per raster cell. The following sections introduce the different functions we implemented for the 'Move', 'MoveStack', 'DBBMM', and 'DBBMMStack' objects. We demonstrate how functions can be used to import, analyze, and visualize animal movement data. The aim of this vignette is to enable all users unfamiliar with the package to handle all basic functions.

1. Data import

The import of data into R can happen in two ways: either by importing .csv data files (using the `move()` function) or by using the web interface of the move package (using the `getMovebankData()` function). The web interface allows one to access and download data sets from Movebank.

1.1. Import of Movebank data

Files from Movebank (www.movebank.org) can be imported with `move(x="path")`, where "path" is a character string that locates the file on the hard drive. It may look like:

```
> data <- move(x="/User/Documents/test.csv").
```

The following line imports the "leroy.csv.gz" file that is part of the move package.

```
> leroy <- move(system.file("extdata","leroy.csv.gz",package="move"))
```

1.2. Import of non-Movebank data

If data sets are not published on Movebank but stored on the hard drive they can be imported with the same `move()` function. Because the column names may differ from the Movebank standard, it is necessary to indicate the timestamp and coordinates columns, the projection method of the coordinates (e.g. longitude-latitude), and the data frame that includes all of the imported data. The standard `move()` method also allows one to directly enter the used sensor type and the animal name.

Import of a non-Movebank csv file that is attached to this package:

```
> data <- read.csv(system.file("extdata","ricky.csv.gz",package="move"))
> ricky <- move(x=data$location.long, y=data$location.lat, time=as.POSIXct(data$timestamp,
+             format="%Y-%m-%d %H:%M:%OS", tz="UTC"), proj=CRS("+proj=longlat +ellps=WGS84
+             data=data, animal=data$individual.local.identifier, sensor=data$sensor)
> ricky
```

```
class       : Move
features    : 8958
extent      : -73.94032, -73.8683, 42.82073, 42.851  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
variables   : 18
names       :     X, event.id, eobs.battery.voltage, eobs.fix.battery.voltage, eobs.horizo
min values  :     2, 44295482,                 3449,                     3081,
max values  : 10760, 44529025,                 3740,                     3559,
timestamps  : 2010-02-09 17:01:23 ... 2010-03-31 17:31:26 Time difference of 50 days  (sta
sensors     : gps
indiv. data : behavioural.classification, eobs.status, eobs.type.of.fix, manually.marked.o
indiv. value: NA A 3 NA true gps Martes pennanti 1016 Urban fisher GPS tracking 18N Easter
date created: 2015-03-19 13:13:27
```

Entering the object name calls the `show()` function of 'Move' objects. This function provides
an overview of the object. It returns the object class ('Move'), the animal ID (ricky), num-
ber of coordinates (8961), the extent of the coordinates, the coordinates reference system (or
projection, here longitude-latitude), the number of columns of the imported data.frame, and
the names of the columns of the data.frame, as well as the first and last timestamp and the
duration of the observation. The `show()` works in a similar way for all other object types.

1.3. Handling multiple animals

If the data-set includes more than one animal, it is necessary to indicate the column with the
animal IDs. The `move()` automatically creates a stack of Move objects called 'MoveStack'.
Sometimes it is desirable to work with stacked Move objects. Most functions of the Move
package are capable of working with stacks. Stacks are either created when .csv files with
multiple animal IDs are imported, or when a '('list) of '('Move) objects is passed on to the
stacking function `moveStack()`.

```
> list <- list(leroy, ricky)
> stack <- moveStack(list)
> stack
```

```
class       : MoveStack
features    : 9877
extent      : -73.94032, -73.84366, 42.70898, 42.851  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
variables   : 22
names       :           timestamp, eobs.battery.voltage, eobs.horizontal.accuracy.estimate
min values  : 2009-02-11 12:16:45,                 3449,                               2.05
max values  : 2009-03-04 09:16:59,                 3740,                              97.02
timestamps  : 2009-02-11 13:16:45 ... 2010-03-31 19:31:26 Time difference of 413 days  (st
sensors     : gps
indiv. data : eobs.fix.battery.voltage, manually.marked.outlier, visible, sensor.type, ind
min ID Data : NA NA true gps Martes pennanti   74 NA Urban fisher GPS tracking Eastern Sta
```

```
max ID Data : NA NA true gps Martes pennanti 1016 NA Urban fisher GPS tracking Eastern Sta
individuals : Leroy, Ricky.T
unused rec. : 1071
date created: 2015-03-19 13:13:27
```

The show function works also for the stack object. It combines the information of all objects
that are stored in the 'MoveStack'.
It is also possible to break down a 'MoveStack' into single 'Move' objects using the `split()`
function. It returns a list of all 'Move' objects.

```
> unstacked <- split(stack)
> #use show(unstacked) to see all objects of unstacked
```

1.4. Import of Movebank data sets using the web interface

The Move package includes a bunch of functions to access the Movebank database. With
`getMovebankData()`, whole data sets can be downloaded and stored as a 'Move' or 'MoveStack'
object (we can **not** offer a **working example** in this vignette due to fact that Movebank
does not allow dummy accounts). You can use the function as follows:

```
> bobby <- getMovebankData(study="BCI Ocelot", animalName="Bobby", login=abc)
```

With this command the data associated with the animal ID "Bobby" from the study "BCI
Ocelot" is stored in a 'Move' object (because it is only one animal) called `bobby`.
The following command will download the full study:

```
> ocelot <- getMovebankData(study="BCI Ocelot", login=abc)
```

This function will create a 'MoveStack' object. Note, it is necessary to provide a login object
to access the Movebank database. More information about the Movebank browsing functions
are in the additional vignette "browsing Movebank".

2. Visualizing and analyzing Move objects

In the previous chapter we presented the `moveStack()` and `split()` functions to stack and unstack 'Move' objects. In this chapter we present further functions to work with the two object types.

2.1. Show and summarize

An important function to work with objects is the `show()` function. Depending on the implementation it gives an overview of the object one is working with. The function can either be called by entering the name of the object directly, or by calling the function with the object:

```
> show(leroy)
```

```
class        : Move
features     : 919
extent       : -73.93067, -73.84366, 42.70898, 42.7687  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
variables    : 17
names        :            timestamp, eobs.battery.voltage, eobs.horizontal.accuracy.estimate
min values   : 2009-02-11 12:16:45,                 3596,                              3.07
max values   : 2009-03-04 09:16:59,                 3666,                             97.02
timestamps   : 2009-02-11 12:16:45 ... 2009-03-04 09:16:59 Time difference of 21 days  (sta
sensors      : gps
indiv. data : eobs.fix.battery.voltage, manually.marked.outlier, visible, sensor.type, ind
indiv. value: 0 NA true gps Martes pennanti 74 Leroy Urban fisher GPS tracking Eastern Sta
unused rec. : 1071
date created: 2015-03-19 13:13:27
```

Both ways return the same information. For Move objects, this is the class name, number of coordinates, the map extent, the coordinate reference system, number of additional variables, names of the additional variables and their minimum and maximum values, the sensor type, further data that are specific to the animal, the number of recordings that were skipped, and the date of file creation.

The `summary()` can be used to summarize important information of 'Move' or 'MoveStack' objects. `Summary()` calls the function `distance()`, `time()`, `speed()`, and `angle()` and returns a summary of these values.

```
> summary(ricky)

$Ricky.T
  TravDist  MaxDist   MinDist FarthDist AverDist    SDDist    SEDist
1 375988.9 2128.647 0.1577597  3.478313 41.97711 52.99414 0.7593597


$Ricky.T
        Duration  AverDur     SDDur  dupl multseason
1 1200.501 hours 0.1340293 0.7330165 FALSE      FALSE
```

```
$Ricky.T
  AverSpeed  VarSpeed MaxSpeed
1 0.2899869 0.1067791 9.772032

$Ricky.T
  AverAzimuth VarAzimuth SEAzimuth
1   -153.1399  0.9906986  16.37971
```

```
> #summary(stack) ##works also for stacks
```

The following functions return more information of the object: `n.locs()` returns the number of locations of an 'Move' or 'MoveStack' object

```
> n.locs(ricky)
```

```
[1] 8958
```

`timeLag()` returns the time differences between successive coordinates from 'Move' or 'MoveStack' objects (the `units` argument determines the units of the data output; it is minutes by default)

```
> head(timeLag(leroy, units="mins"))
```

```
[1] 14.88333 14.18330 14.45007 15.04998 14.90000 15.39998
```

`timestamps()` returns all time-stamps of an 'Move' or 'MoveStack' object

```
> head(timestamps(leroy))
```

```
[1] "2009-02-11 12:16:45 UTC" "2009-02-11 12:31:38 UTC"
[3] "2009-02-11 12:45:48 UTC" "2009-02-11 13:00:16 UTC"
[5] "2009-02-11 13:15:19 UTC" "2009-02-11 13:30:13 UTC"
```

2.2. Plot

A simple plotting function is implemented both for 'Move' and 'MoveStack' objects. The `plot()` function uses arguments from the generic plot function to change plot properties. In the example below the arguments `col` (color), `lwd` (line width), `pch` (point character), and `xlab` and `ylab` (x and y labels) are used to adjust the graphs (see Figure. 1).

```
> par(mfcol=1:2)
> plot(leroy, type="o", col=3, lwd=2, pch=20, xlab="location_long",
+      ylab="location_lat")
> plot(stack, col=c(6,5), lwd=2, xlab="location_long", ylab="location_lat")
```
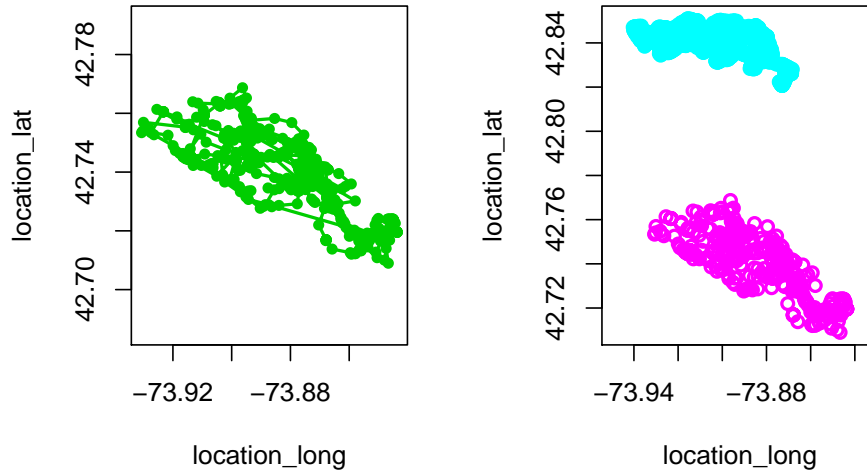
Figure 1: Using the `plot()` function to plot the track and its coordinates of a 'Move' (left) and a 'MoveStack' (right) object.

2.3. Google plot

A useful addition to plot the pure track is to plot it on a map. The easiest ways to do this are with OpenStreetMap, Stamen Map, Cloudmade or Google Maps and to use the ggmap package, which is based on ggplot.

Note, the function requires two further packages (`ggmap` and `mapproj`). The `get_map()` function makes a call to the online databases and retrieves a map which is defined by the bounding box (`bbox`). The source argument allows one to choose between OpenStreetMap ('`osm`'), Stamen Maps ('`stamen`'), Cloudmade ('`cloudmade`') and Google Maps ('`google`'). The `ggmap()` function plots the map in a new graphics device and `geom_path()` adds the track of the 'Move' object (see Figure 2).

```
> require(ggmap) #these packages are necessary to work with google maps
> require(mapproj)
> leroy_df <- as(leroy, "data.frame")
> m <- get_map(bbox(extent(leroy)*1.1), source="stamen", zoom=12)
> ggmap(m)+geom_path(data=leroy_df, aes(x=location.long, y=location.lat))
```

2.4. Subset and transform objects

Sometimes only a subset of all coordinates of an animal track is needed. With the subset function a section of all coordinates are returned as a new 'Move' or 'MoveStack' object.
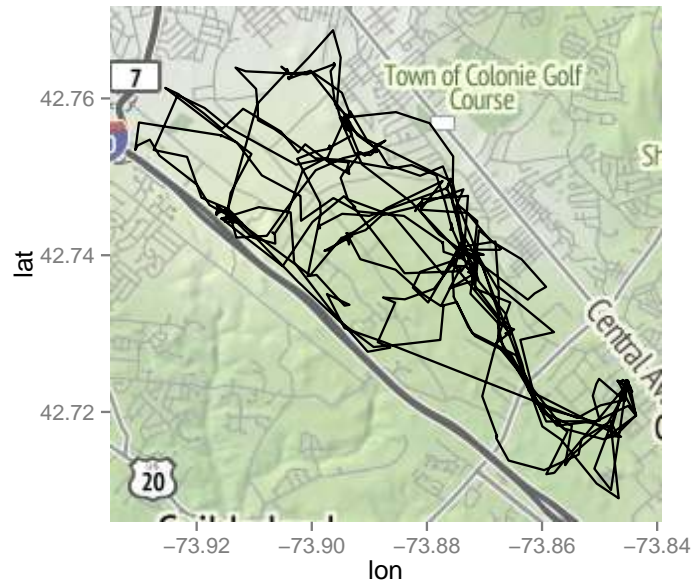
```
> ricky[1:25]
```

Figure 2: Plotting a track on a map using ggmap().

```
class       : Move
features    : 25
extent      : -73.90426, -73.90235, 42.84125, 42.84189  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
variables   : 18
names       :  X, event.id, eobs.battery.voltage, eobs.fix.battery.voltage, eobs.horizonta
min values  :  2, 44528398,                 3518,                     3388,
max values  : 32, 44528427,                 3662,                     3493,
timestamps  : 2010-02-09 17:01:23 ... 2010-02-09 20:58:51 Time difference of 4 hours  (sta
sensors     : gps
indiv. data : behavioural.classification, eobs.status, eobs.type.of.fix, manually.marked.o
indiv. value: NA A 3 NA true gps Martes pennanti 1016 Urban fisher GPS tracking 18N Easter
date created: 2015-03-19 13:13:27


> stack[800:1100] #see the names of both animals in second last row


class       : MoveStack
features    : 301
extent      : -73.91325, -73.86532, 42.72992, 42.84189  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
variables   : 22
names       :           timestamp, eobs.battery.voltage, eobs.horizontal.accuracy.estimate
min values  : 2009-03-01 11:45:50,                 3500,                               3.07
max values  : 2009-03-04 09:16:59,                 3725,                              67.33
timestamps  : 2009-03-01 12:45:50 ... 2010-02-11 06:40:38 Time difference of 347 days  (st
```

```
sensors     : gps
indiv. data : eobs.fix.battery.voltage, manually.marked.outlier, visible, sensor.type, ind
min ID Data : NA NA true gps Martes pennanti   74 NA Urban fisher GPS tracking Eastern Sta
max ID Data : NA NA true gps Martes pennanti 1016 NA Urban fisher GPS tracking Eastern Sta
individuals : Leroy, Ricky.T
unused rec. : 1071
date created: 2015-03-19 13:13:27
```

If it is necessary to directly interact with different data of the same object, it can be advantageous to work with a data.frame. The `as()` function can transform a 'Move' or 'MoveStack' to a data.frame:

```
> head(as(leroy, "data.frame"))

            timestamp location.long location.lat eobs.battery.voltage
45 2009-02-11 12:16:45     -73.89880     42.74370                 3615
46 2009-02-11 12:31:38     -73.89872     42.74369                 3623
47 2009-02-11 12:45:48     -73.89869     42.74364                 3627
48 2009-02-11 13:00:16     -73.89862     42.74374                 3632
49 2009-02-11 13:15:19     -73.89871     42.74368                 3642
50 2009-02-11 13:30:13     -73.89885     42.74365                 3647
   eobs.horizontal.accuracy.estimate eobs.key.bin.checksum
45                             14.85            2992317972
46                              5.38            1723246055
47                              5.38            1910450098
48                              6.40            2286746099
49                              7.68            3797866101
50                              8.70            3956003832
   eobs.speed.accuracy.estimate     eobs.start.timestamp eobs.status
45                         5.65 2009-02-11 12:14:59.000           A
46                         4.69 2009-02-11 12:30:01.000           A
47                         4.19 2009-02-11 12:45:01.000           A
48                         5.97 2009-02-11 13:00:02.000           A
49                         6.34 2009-02-11 13:15:01.000           A
50                         6.15 2009-02-11 13:30:01.000           A
   eobs.temperature eobs.type.of.fix eobs.used.time.to.get.fix ground.speed
45               20                3                       106         2.10
46               26                3                        97         0.51
47               24                3                        47         0.16
48               24                3                        14         0.23
49               25                3                        18         0.48
50               23                3                        12         0.17
   heading height.above.ellipsoid utm.easting utm.northing utm.zone
45  125.17                   79.3    590130.0      4732942      18N
46    3.28                   94.2    590136.3      4732940      18N
47   91.10                   82.5    590138.5      4732935      18N
48  335.54                  153.7    590144.0      4732947      18N
```

```
49   359.79                    73.7    590137.0      4732940     18N
50   29.49                     71.2    590126.0      4732936     18N
     study.local.timestamp sensor         timestamps eobs.fix.battery.voltage
45   2009-02-11 07:16:45     gps 2009-02-11 12:16:45                         0
46   2009-02-11 07:31:38     gps 2009-02-11 12:31:38                         0
47   2009-02-11 07:45:48     gps 2009-02-11 12:45:48                         0
48   2009-02-11 08:00:16     gps 2009-02-11 13:00:16                         0
49   2009-02-11 08:15:19     gps 2009-02-11 13:15:19                         0
50   2009-02-11 08:30:13     gps 2009-02-11 13:30:13                         0
     manually.marked.outlier visible sensor.type individual.taxon.canonical.name
45                        NA    true         gps                 Martes pennanti
46                        NA    true         gps                 Martes pennanti
47                        NA    true         gps                 Martes pennanti
48                        NA    true         gps                 Martes pennanti
49                        NA    true         gps                 Martes pennanti
50                        NA    true         gps                 Martes pennanti
     tag.local.identifier individual.local.identifier            study.name
45                     74                          Leroy Urban fisher GPS tracking
46                     74                          Leroy Urban fisher GPS tracking
47                     74                          Leroy Urban fisher GPS tracking
48                     74                          Leroy Urban fisher GPS tracking
49                     74                          Leroy Urban fisher GPS tracking
50                     74                          Leroy Urban fisher GPS tracking
          study.timezone
45 Eastern Standard Time
46 Eastern Standard Time
47 Eastern Standard Time
48 Eastern Standard Time
49 Eastern Standard Time
50 Eastern Standard Time
```

For technical reasons the coordinates of the Move object must be in `aeqd` projection, which stands for Azimuthal Equidistant. To check the projection of your coordinates you can use the `proj4string()` method. If your data are not in the right projection, use the following command to change it.

```
> proj4string(leroy)
```

```
[1] "+proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0"
```

```
> leroy_t <- spTransform(x=leroy, CRSobj="+proj=aeqd", center=TRUE)
> proj4string(leroy_t)
```

```
[1] "+proj=aeqd +ellps=WGS84 +lon_0=-73.8871629 +lat_0=42.73884025"
```

The data are now in the correct projection and the coordinate system is now centered to the center of the track.

2.5. Bursting tracks

It can be interesting to compare different parts of the track with each other. For example, how do data points differ between winter and summer, or between behaviors like migrating, non-migrating, resting? To indicate which point of the data set belongs to which category, a track is 'bursted'. This means, that an additional column is introduced to the data set that is associated with a category and all other track information.

A track is bursted by supplying a vector with the length of the number of coordinates. The vector is then used as a factor to be associated with the 'Move' object. The returned object belongs to the class 'MoveBurst'.

```
> behavior <- c(rep(1:8,each=111), rep(1, 30))
> leroy_b <- move::burst(x=leroy, f=behavior)
> class(leroy_b)

[1] "MoveBurst"
attr(,"package")
[1] "move"
```

Bursted tracks can be plotted with the basic `plot()` and the more complex `plotBursts()` function. Both functions use colors to indicate to which burstID a segment belongs.

The `plot()` function simply plots the different segments as colored lines. (see in the code example a way to plot points instead of colored line segments)

The `plotBursts()` plots a circle for each segment right at the midpoint of the segment. The circles have a size that is calculated with an extra size function. By default, it is the relative time of the segment compared to the whole track. It is possible to calculate the size with a different function using the `sizeFun` argument of the function. (Figure 3)

```
> par(mfrow=c(2,2))
> plot(leroy_b, type="l", lwd=2)
> plot(midPoint(coordinates(leroy_b)[-n.locs(leroy_b), ],
+                coordinates(leroy_b)[-1, ]), col=leroy_b@burstId, pch=20)
> plotBursts(leroy_b, breaks=3, add=FALSE, pch=19)
```
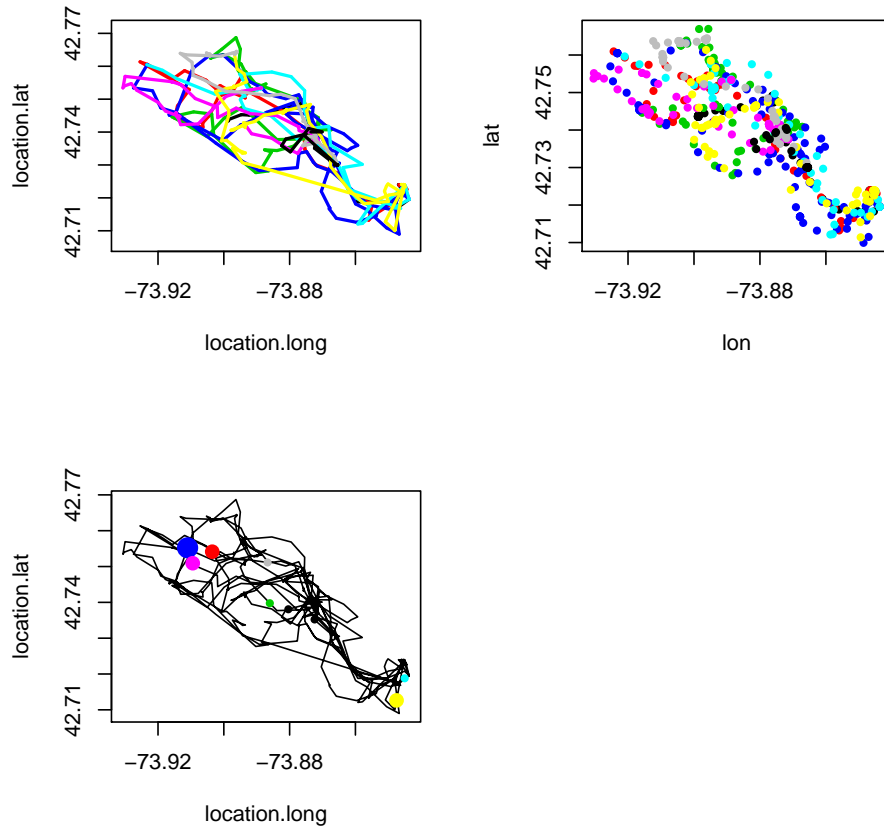
Figure 3: Plotting the track and adding the burst information as colored lines, points, or circles (the color corresponds to the burstID; the size of the circles, three groups by default, represents the relative amount of time per segment)

3. Dynamic Brownian Bridge Movement Model

To calculate the utilization distribution (UD) with the dynamic Brownian Bridge Movement Model use the `brownian.bridge.dyn()` function. You need to specify the Move object from which you want to calculate the UD, the location error of your localization method (in map units), the raster options, and the extension of the raster.

You can either set the number of the raster cells along the longest dimension of your map by setting a numeric value for the `dimSize` argument, or - if you know the extent of your map - you can set the size of the raster cells with a numeric value for the `raster` argument (note, you can only set one of them).

When the `brownian.bridge.dyn()` function issues the warning that the extent of your raster is too small, this is because a large part of the UD is at the borders of the raster. You can change the extent of the raster by setting the `ext` argument. If you want to extend the raster in all four directions equally, choose one number. You can use a vector of two numbers to extend the x and the y dimension differently, or even a vector with four numbers to extend differently in all four directions.

```
> r <- spTransform(ricky[1:500,], center=T)
> ricky_dbbmm <- brownian.bridge.dyn(r, dimSize=150, location.error=23,
+                                     ext=.3, time.step=60, margin=13)
```

Running the `brownian.bridge.dyn()` creates an object of the 'DBBMM' class, which among others stores the raster of the map with the values from the UD. You can also run the function with a 'MoveStack'. The returned object will then be a 'DBBMMStack'.

The 'DBBMM' and 'DBBMMStack' objects can be summarized with the `show()` function (using the show function directly or by using the object name):

```
> ricky_dbbmm


class       : DBBMM
dimensions  : 122, 150, 18300  (nrow, ncol, ncell)
resolution  : 31.03099, 31.03099  (x, y)
extent      : -2327.14, 2327.509, -1892.768, 1893.013  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=aeqd +ellps=WGS84 +lon_0=-73.8864686 +lat_0=42.83130625
data source : in memory
names       : layer
values      : 1.962427e-37, 0.02436245  (min, max)


> raster(ricky_dbbmm)


class       : RasterLayer
dimensions  : 122, 150, 18300  (nrow, ncol, ncell)
resolution  : 31.03099, 31.03099  (x, y)
extent      : -2327.14, 2327.509, -1892.768, 1893.013  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=aeqd +ellps=WGS84 +lon_0=-73.8864686 +lat_0=42.83130625
```

```
> par(mfrow=c(1,2))
> plot(ricky_dbbmm, xlab="location_long", ylab="location_lat")
> plot(ricky_dbbmm, xlab="location_long", ylab="location_lat")
> lines(spTransform(ricky[1:500,], center=TRUE), col=3, lwd=2)
> #plot(ricky_dbbmm, xlab="location_long", ylab="location_lat")
> #points(spTransform(ricky[1:500, ], center=TRUE), col=8)
```
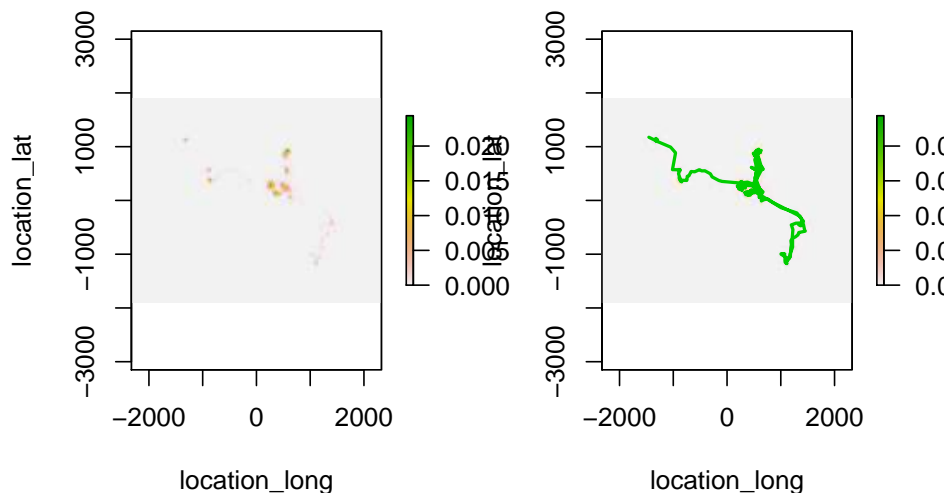

Figure 4: Plotting the UD (left) and adding the track as a line (center) or as points (right)

The second method (`raster()`) returns all important information of the raster that is stored in the 'DBBMM' object.
If only certain objects of a 'DBBMMStack' are needed, the stack can be split into a 'list' of 'DBBMM' objects using the `split()` function.

3.1. Plotting UDs

'DBBMM' objects can be plotted with the `plot()` function. This produces a fixed cell size ratio graphic from the raster (see Figure 4). A second function - `image()` - produces a variable cell size ratio graphic (not prone to distortions after resizing the graphics window) from the raster.

To plot contour lines from the raster, use the `contour()` function and set the percentage levels that you want to print. With `add=TRUE` you can add the contour to a previous plot (see Figure 5).

`raster()` : returns the information of the stored raster
`outerProbability()` : calculates the probabilities of the UD at the border of the raster

```
> plot(ricky_dbbmm, xlab="location_long", ylab="location_lat")
> contour(ricky_dbbmm, levels=c(.5, .95), col=c(6,2), add=TRUE, lwd=2)
```
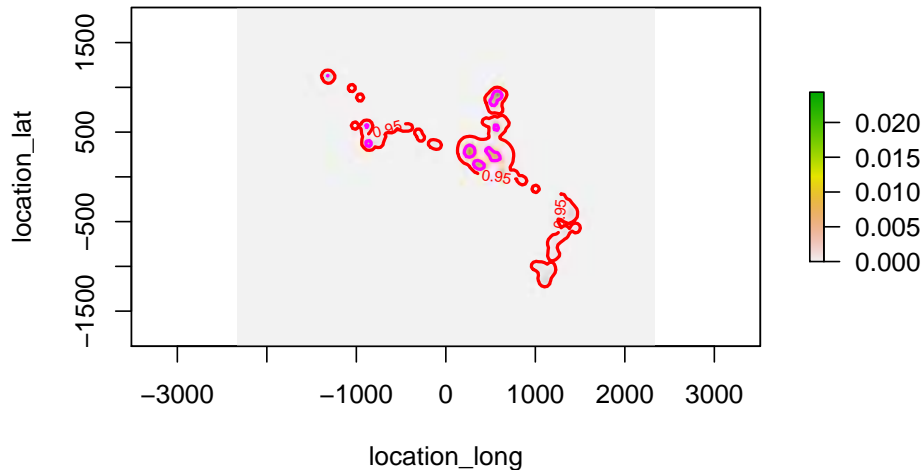

Figure 5: Plotting the 95% and 50% contour lines on top of the UD.

3.2. Calculating UD area size

To determine the area size of the UD, we need to calculate the contour raster from the UD raster. This means, that each raster has not longer its original value but the value of the contour it belongs to. A cell with a very high value in the UD raster will have a very low value in the contour raster. We will use the `getVolumeUD()` function to calculate the contour raster and then create a new raster that includes only the cells that belong to the 95% contour. Cells that belong to the contour will get the value 1 while the others get 0. In the last step we can just sum the values of the raster. The area is the number of cells (which we just calculated) multiplied by the actual size of the raster cells.

```
> ricky_cont <- getVolumeUD(ricky_dbbmm)
> ricky_cont <- ricky_cont<=.95
> area <- sum(values(ricky_cont))
> area
```

```
[1] 654
```

3.3. Storing and loading objects

If one or more objects should be stored and loaded at another time into R, there is an easy way to save them. The `save()` function saves all listed objects from the workspace (in the

example below it is only ricky_dbbmm). The file format is `RData`. More than one variable or the whole workspace can be stored in a `RData` file. Complicated calculations (e.g. Brownian bridges), can be calculated once and then stored.

`RData` files can be loaded using the `load()` function. It loads all objects from the file again into the workspace. In the following example we save the object `ricky_dbbmm` as `test.RData`, remove the object (it does not appear any more in the workspace), and load it again.

```
> save(x=ricky_dbbmm, file="~/Desktop/test.RData")
> rm(ricky_dbbmm)
> load(file="~/Desktop/test.RData")
```

3.4. Store a track as a KML file

'`Move`' objects can also be saved as KML (keyhole markup language) files. KML files can easily be loaded into GoogleEarth and GoogleMaps. For this the `plotKML` package is needed. Once installed the `kml()` function creates a file with the ending `.kml` and the name of the object. The kml file is then stored in the current working directory.

The following code creates the file `leroy.kml` from the '`Move`' object named `leroy`. If GoogleEarth is installed, a double click on that file opens the locations of the track in GoogleEarth.

```
> #install.packages('plotKML')
> require(plotKML)
> kml(leroy)
```

4. Home Range Bootstrap using Minimum Convex Polygons

The move package includes a method to calculate minimum convex polygons (MCP) using `hrBootstrap()`. A minimum convex polygon describes an area that is formed by the points of a track. The convex is because no line that connects two points within this polygon crosses the border of the polygon.

We implemented this method to calculate MCPs from the `adehabitatHR` package (more information) using a bootstrap method. This means that the area size of the MCP is calculated with an (exponentially) increasing number of coordinates. The MCP size continues to increase until it reaches a plateau.

Because the function takes random coordinates to calculate the MCP, every calculation is repeated 100 times by default. For every number of coordinates the quantiles are calculated and plotted in a line diagram. A black horizontal line indicates the real MCP size (calculated with all coordinates).

```
> hrBootstrap(x=leroy, rep=25, unin='km', unout='km2')
```
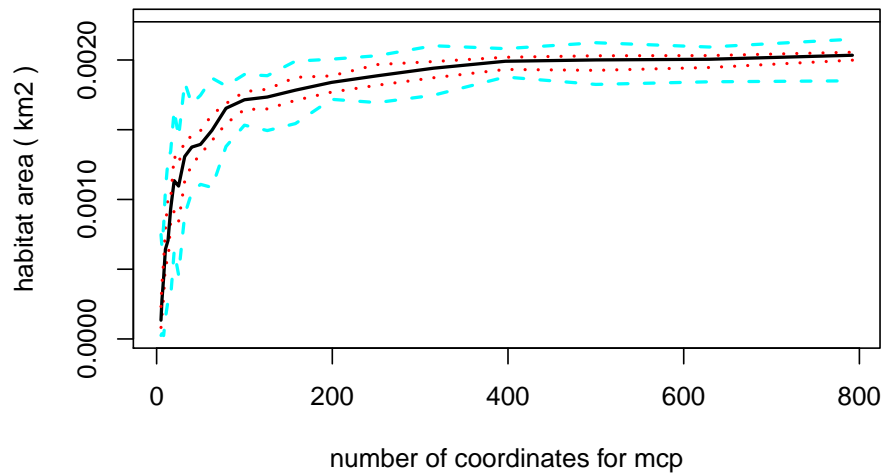

Figure 6: The returned plot forms the `hrBootstrap()` for a track with 25 repetitions.

Affiliation:

Marco Smolla
Max-Planck-Institute for Ornithology, Radolfzell, Germany
E-mail: marco.smolla@postgrad.manchester.ac.uk

Bart Kranstauber
Max-Planck-Institute for Ornithology, Radolfzell, Germany
E-mail: bart.kranstauber@uni-konstanz.de