# Using the move package

Marco Smolla                Bart Kranstauber

**Abstract**

This vignette gives examples of how to use the **move** package. It explains how to load data in the **move** functions using `read.csv()` and `move()`, transform data into a different projection method with `spTransform()`, and calculate the utilization density using `brownian.bridge.dyn()`. Further functions to extract information from the Move object and plot the track and utilization density will be explained as well.

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

## Introduction

The first set of functions the move package includes lets you import, analyse and visualize tracking data. A possible workflow could look like this:

1. import data
2. transform coordinates to correct aeqd (Azimuthal Equidistant) projection
3. calculate utilization densities using the dynamic Brownian Bridge Movement Model
4. plot the track and the utilization density

## Importing data

### Movebank data

There are two ways to import data into the **move** functions. The easiest way is to work with downloaded files from `movebank.org`. With the following function you just have to define the path to the file like `move(x="path")`.

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

### Non-Movebank data

If you want to use data that are not from movebank, you can use the `move()` function as well. In this case you first need to import the '`data.frame`' with the ordinary `read.table()`. Afterwards you define which columns store the x and y coordinates and the timestamps. You also need to define the format of the timestamps and projection. The projection method must be a '`CRS`' and the timestamps a '`POSIXct`' object.

```
> file <- read.table(system.file("data","leroy.csv",package="move"),
+           header=TRUE, sep=",", dec=".")
> data <- move(x=file$location.long, y=file$location.lat,
+           time=as.POSIXct(file$timestamp, format="%Y-%m-%d %H:%M:%S", tz="UTC"),
+           data=file, proj=CRS("+proj=longlat"))
```

In both cases you create an object of the class '`Move`'. It stores among others the timestamps and coordinates which are necessary for the following functions.

## Transform coordinates projection

For technical reasons the coordinates of the Move object must be in the `aeqd`, or Azimuthal Equidistant, projection. 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(data)

[1] " +proj=longlat +ellps=WGS84"

> data2 <- spTransform(x=data, CRSobj="+proj=aeqd", center=TRUE)
> proj4string(data2)

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

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

## Using the dynamic Brownian Bridge Movement Model

To calculate the utilization density (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 (you can only set one of them).
When the `brownian.bridge.dyn()` function issues the warning, the extent of your raster is too small, that is that 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. 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.

```
> p <- brownian.bridge.dyn(object=data2, location.error=23.5, dimSize=45, ext=.3)
```

Running the `brownian.bridge.dyn()` creates an object of the '`DBBMM`' class, which amonge others stores the raster of the map with the values from the UD.

## Plotting data

### Plotting the track (Move object)

To plot the track of the animal, simply use the following functions. You can just use `plot()` on a '`Move`' object and get the track with dots and lines. If you only want to plot points or lines use the respective functions. With the `add` and `col` arguments you can add the plots to another plot or change the colors.
If you want to plot the track of the animal on a real map, use the `plot()` function with the `google` argument set `TRUE`.
Figure 1 shows what the different functions create.

```
> attach(mtcars)
> par(mfrow=c(2,2))
> plot(data2)
> points(data2, col="blue")
> lines(data2, col="green")
> plot(data, google=TRUE)
```

### Plotting the utilization density (DBBMM object)

You might want to plot the '`DBBMM`' object you created earlier. Use the `plot()` function to produce a fixed cell size ratio graphic from the raster, or the `image()` function to produce 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. The same works if you want to add the track as lines or points, or both.
Figure 2 shows what this could look like.

```
> attach(mtcars)
> par(mfrow=c(2,2))
> plot(p)
> image(p)
> plot(p)
> contour(p, levels=c(.5,.9), add=TRUE)
> image(p)
> lines(data2, add=TRUE, col="black")
```

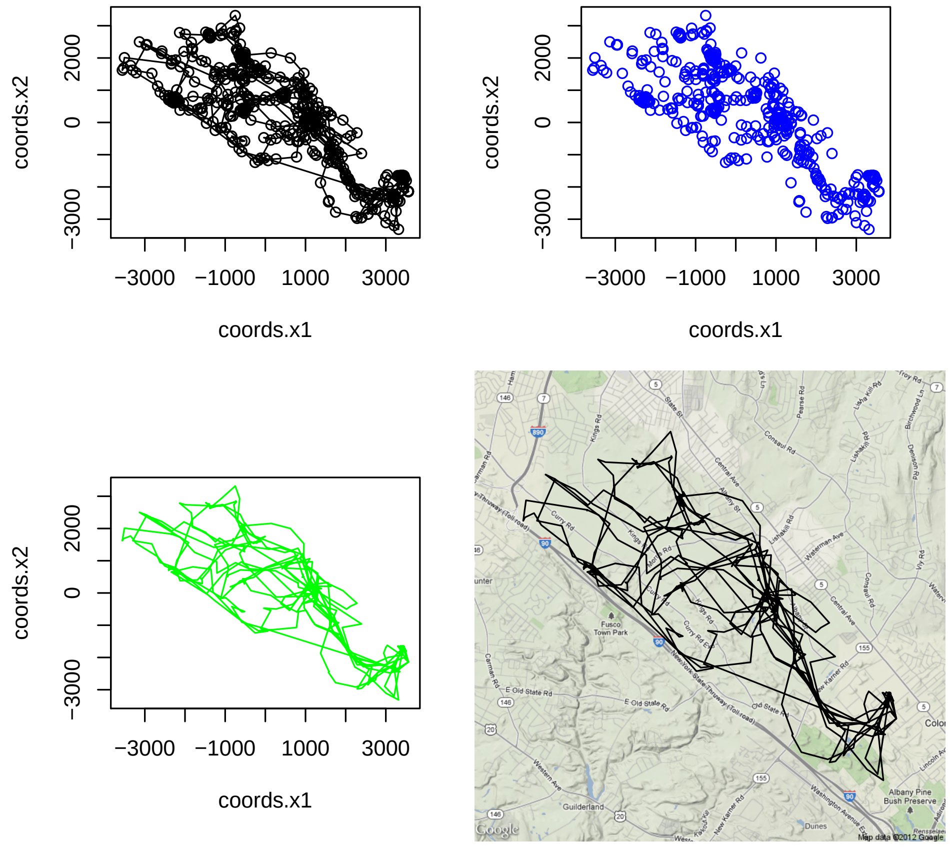

Figure 1: Different ways (lines, points, lines and points, google map) to display the track.

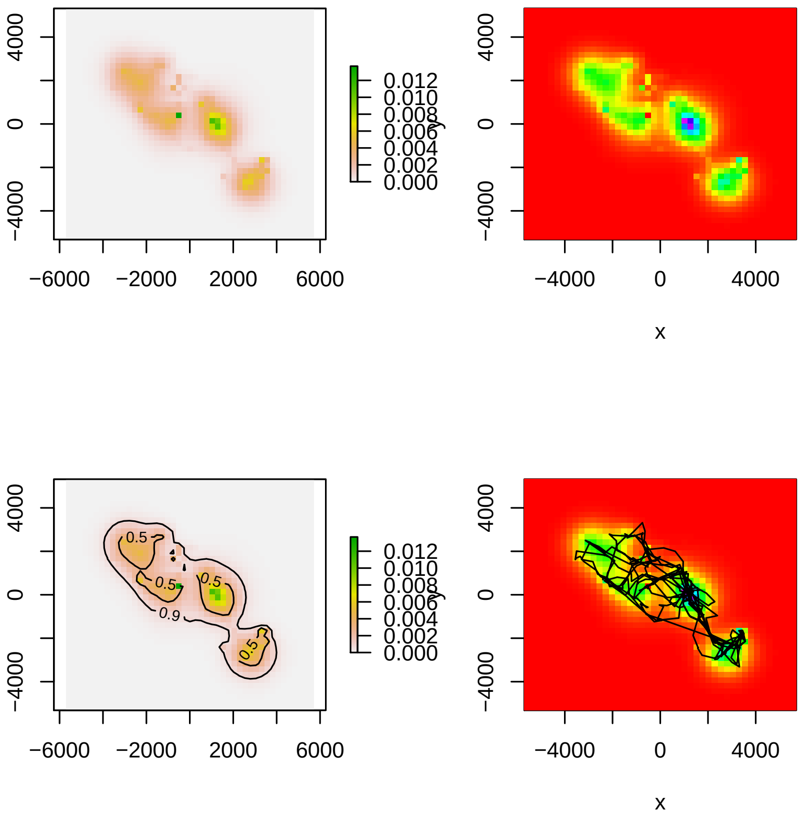

Figure 2: Different ways to display the utilization density (left `plot()`, right `image()`), including the `contour()` (left) and `lines()` (right) function.

## Extract information

The ‘`Move`’ and the ‘`DBBMM`’ object store a lot of data. There are a couple of functions you can use to easily extract them.

### ... from a Move object

`as.data.frame()` : extracts the spatial data frame
`coordinates()` : extracts the coordinates of the Move object
`n.locs()` : returns the number of tracked locations
`proj4string()` : returns the projection of the coordinates
`time.lag()` : calculates the time differences between coordinates (in minutes)

### ... from a DBBMM object

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

**Affiliation:**

Marco Smolla
Max-Planck-Institute for Ornithology, Radolfzell, Germany
E-mail: `msmolla@orn.mpg.de`

Bart Kranstauber
Max-Planck-Institute for Ornithology, Radolfzell, Germany
E-mail: `kranstauber@orn.mpg.de`