Analysis of Animal Movements in R: the `adehabitatLT` Package

Clement Calenge,
Office national de la chasse et de la faune sauvage
Saint Benoist – 78610 Auffargis – France.

July 2016

Contents

1 History of the package adehabitatLT

The package `adehabitatLT` contains functions dealing with the analysis of animal movements that were originally available in the package `adehabitat` (Calenge, 2006). The data used for such analysis are generally relocation data collected on animals monitored using VHF or GPS collars.

I developped the package `adehabitat` during my PhD (Calenge, 2005) to make easier the analysis of habitat selection by animals. The package `adehabitat` was designed to extend the capabilities of the package `ade4` concerning studies of habitat selection by wildlife.

Since its first submission to CRAN in September 2004, a lot of work has been done on the management and analysis of spatial data in R, and especially with the release of the package `sp` (Pebesma and Bivand, 2005). The package `sp` provides classes of data that are really useful to deal with spatial data...

In addition, with the increase of both the number (more than 250 functions in Oct. 2008) and the diversity of the functions in the package `adehabitat`, it soon became apparent that a reshaping of the package was needed, to make its content clearer to the users. I decided to "split" the package `adehabitat` into four packages:

- `adehabitatHR` package provides classes and methods for dealing with home range analysis in R.

- `adehabitatHS` package provides classes and methods for dealing with habitat selection analysis in R.
- `adehabitatLT` package provides classes and methods for dealing with animals trajectory analysis in R.
- `adehabitatMA` package provides classes and methods for dealing with maps in R.

We consider in this document the use of the package `adehabitatLT` to deal with the analysis of animal movements. All the methods available in `adehabitat` are also available in `adehabitatLT`. Contrary to the other brother packages, the classes of data returned by the functions of `adehabitatLT` are the same as those implemented in the original package `adehabitat`. Indeed, the structure of these classes were described in a paper (Calenge et al. 2009).

Package `adehabitatLT` is loaded by

```
> library(adehabitatLT)
```

2 What is a trajectory?

2.1 Two types of trajectories

We designed the class `ltraj` to store the movements of animals monitored using radio-tracking, GPS, etc. The rationale underlying the structure of this class is described precisely in Calenge et al. (2009). We summarize this rationale in this vignette.

Basically, the trajectory of an animal is the curve described by the animal when it moves. The sampling of the trajectory implies a step of discretization, i.e., the division of this continuous curve into a number of discrete "steps" connecting successive relocations of the animal (Turchin, 1998). Two main classes of trajectories can be distinguished:

- **Trajectories of type I** are characterized by the fact that the time is not precisely known or not taken into account for the relocations of the trajectory;
- **the trajectories of type II** are characterized by the fact that the time is known for each relocation. This type of trajectory may in turn be divided into two subtypes:

– **regular trajectories**: these trajectories are characterized by a constant time lag between successive relocations;

– **irregular trajectories**: these trajectories are characterized by a variable time lag between successive relocations;

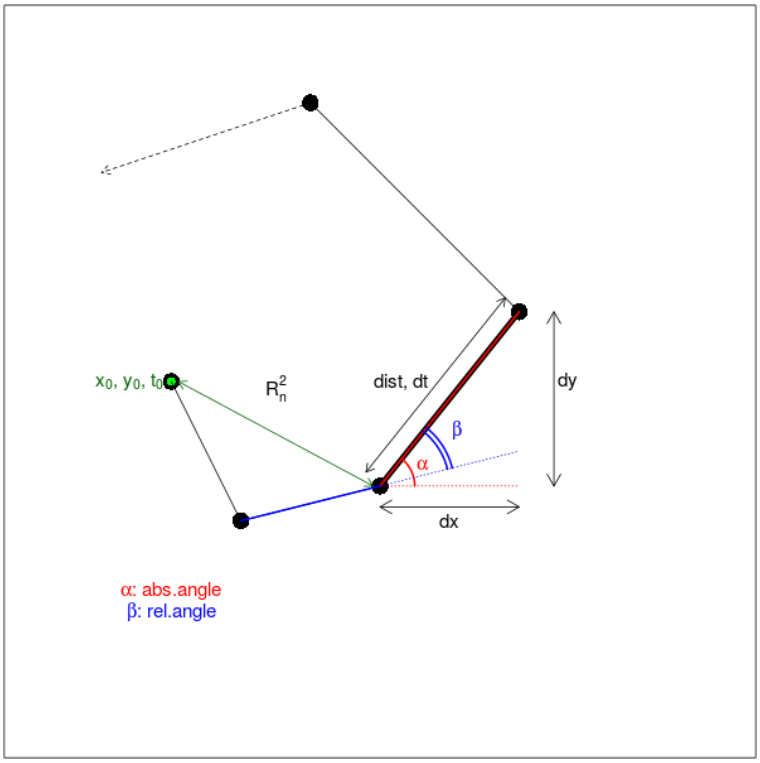
Note that the functions of `adehabitatLT` are mainly designed to deal with type I or type II regular trajectories. Irregular trajectories are harder to analyze, as the descriptive parameters of these trajectories (see below) may not be compared when computed on different time lags.

2.2 Descriptive parameters of the trajectory

Marsh and Jones (1988) noted that a good description of the trajectory is achieved when the following criteria are fullfilled:

- the description is achieved with a minimum set of relatively easily measured parameters;
- the relationships between these parameters are defined precisely (e.g., with the help of a model);
- the parameters and the relationships between them are sufficient to reconstruct characteristic tracks without loosing any of their significant properties.

Based on a literature review (see Calenge et al. 2009), we have chosen to characterize all the trajectories by the following parameters:



- `dx, dy, dt`: these parameters measured at relocation i describe the increments of the x and y directions and time between the relocations i and $i+1$. Such parameters are often used in the framework of stochastic differential equation modelling (e.g. Brillinger et al. 2004, Wiktorsson et al. 2004);

- `dist`: the distance between successive relocations is often used in animal movement analysis (e.g. Root and Kareiva 1984, Marsh and Jones 1988);

- `abs.angle`: the absolute angle α_i between the x direction and the step built by relocations i and $i+1$ is sometimes used together with the parameter `dist` to fit movement models (e.g. Marsh and Jones 1988);

- `rel.angle`: the relative angle β_i measures the change of direction between the step built by relocations $i-1$ and i and the step built by relocations i and $i+1$ (often called "turning angle"). It is often used together with the parameter `dist` to fit movement models (e.g. Root and Kareiva 1984, Marsh and Jones 1988);

- `R2n`: the squared distance between the first relocation of the trajectory and the current relocation is often used to test some movements models (e.g. the correlated random walk, see the seminal paper of Kareiva and Shigesada, 1983).

2.3 Several bursts of relocations

Very often, animal monitoring leads to several "bursts" of relocations for each monitored animal. For example, a GPS collar may be programmed to return one relocation every ten minutes during the night and no relocation during the day. Each night corresponds to a burst of relocations for each animal. We designed the class `ltraj` to take into account this burst structure.

2.4 Understanding the class `ltraj`

An object of class `ltraj` is created with the function `as.ltraj` (see the help page of this function). We will take an example to illustrate the creation of an object of class `ltraj`. First load the dataset `puechabonsp` from the package `adehabitatMA`:

```
> data(puechabonsp)
> locs <- puechabonsp$relocs
> locs <- as.data.frame(locs)
> head(locs)
```

```
   Name Age Sex   Date      X       Y
1 Brock   2   1 930701 699889 3161559
2 Brock   2   1 930703 700046 3161541
3 Brock   2   1 930706 698840 3161033
4 Brock   2   1 930707 699809 3161496
5 Brock   2   1 930708 698627 3160941
6 Brock   2   1 930709 698719 3160989
```

The data frame `locs` contains the relocations of 4 wild boar monitored using radio-tracking at Puechabon (Near Montpellier, South of France). First the date needs to be transformed into an object of the class `POSIXct`.

Remark: The class `POSIXt` is designed to store time data in R (see the very clear help page of `POSIXt`). This class extends two sub-classes:

- **the class** `POSIXlt`: This class stores a date in a list containing several elements related to this date (day of the month, day of the week, day of the year, month, year, time zone, hour, minute, second).

- **the class** `POSIXct`: This class stores a date in a vector, as the number of seconds passed since January, 1st, 1970 at 1AM. This class is more convenient for storing dates into a data frame.

We will use the function `strptime` (see the help page of this function) to convert the date in `locs` into a `POSIXlt` object, as then `as.POSIXct` to convert it into the class `POSIXct`:

```
> da <- as.character(locs$Date)
> head(da)

[1] "930701" "930703" "930706" "930707" "930708" "930709"

> da <- as.POSIXct(strptime(as.character(locs$Date),"%y%m%d", tz="Europe/Paris"))
```

We can then create an object of class `ltraj` to store the wild boar movements:

```
> puech <- as.ltraj(xy = locs[,c("X","Y")], date = da, id = locs$Name)
> puech

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id burst nb.reloc NAs date.begin   date.end
1 Brock Brock       30   0 1993-07-01 1993-08-31
2 Calou Calou       19   0 1993-07-03 1993-08-31
3  Chou  Chou       40   0 1992-07-29 1993-08-30
4  Jean  Jean       30   0 1993-07-01 1993-08-31


 infolocs provided. The following variables are available:
[1] "pkey"
```

The result is a list of class ltraj containing four bursts of relocations corresponding to four animals. The trajectory is of type II and is irregular. There are no missing values.

This object is actually a list containing 4 elements (the four bursts). Each element is a data frame. Have a look, for example, at the first rows of the first data frame:

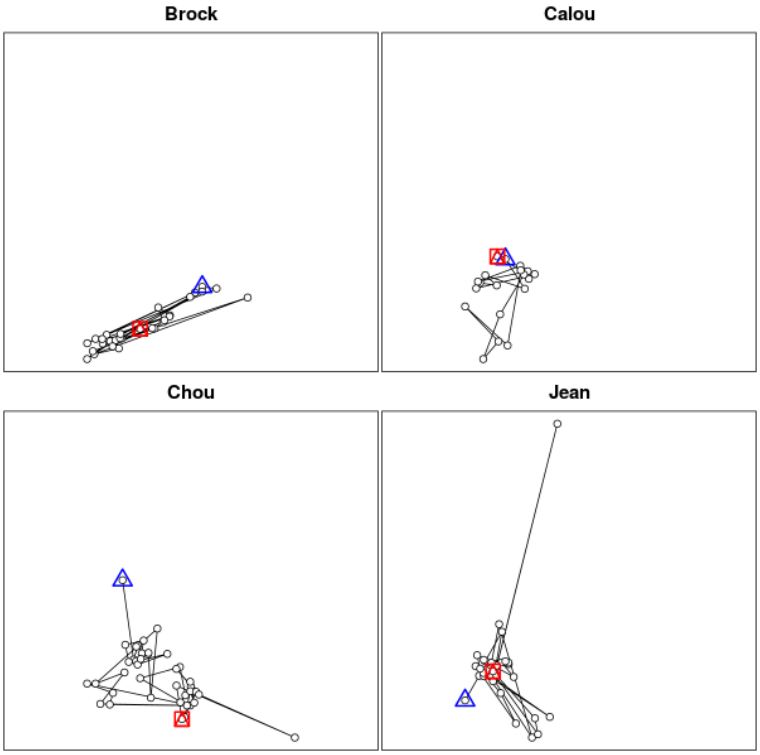
```
> head(puech[[1]])
```

```
       x       y       date    dx   dy      dist     dt     R2n   abs.angle
1 699889 3161559 1993-07-01   157  -18  158.0285 172800       0 -0.11415127
2 700046 3161541 1993-07-03 -1206 -508 1308.6252 259200   24973 -2.74292194
3 698840 3161033 1993-07-06   969  463 1073.9320  86400 1377077  0.44574032
4 699809 3161496 1993-07-07 -1182 -555 1305.8135  86400   10369 -2.70260603
5 698627 3160941 1993-07-08    92   48  103.7690  86400 1974568  0.48088728
6 698719 3160989 1993-07-09   272   26  273.2398 345600 1693800  0.09529869
   rel.angle
1         NA
2 -2.6287707
3 -3.0945230
4  3.1348390
5 -3.0996920
6 -0.3855886
```

The function `as.ltraj` has automatically computed the descriptive parameters described in section 2.2 from the x and y coordinates, and from the date. Note that `dx,dy,dist` are expressed in the units of the coordinates `x,y` (here, metres) and `abs.angle,rel.angle` are expressed in radians.

A graphical display of the bursts can be obtained simply by:

```
> plot(puech)
```



2.5 Two points of views: steps (ltraj) or points (data.frame)?

We noted in the previous section that, in `adehabitatLT`, we consider the trajectory as a collection of successive "steps" ordered in time. We will see later in this vignette that most functions of `adehabitatLT` deal with trajectories considered from this point of view. However, several users (in particular, many thanks to Mathieu Basille and Bram van Moorter) noted that although this point of view may be useful to manage and analyse trajectories, it may be too restrictive to allow an easy management of such data.

Actually, the trajectory data may also be considered as a set of successive *points* (the relocations) ordered in time. At first sight, the distinction between these two models may seem trivial, but it is important to consider it in several cases.

For example, any slight change in the coordinates/date of a relocation will change the value of all derived statistics (`dt`, `dist`, etc.). In the previous versions of adehabitat, it was possible to change directly the values of coordinates/dates in the object, and then to compute again the steps characteristics thanks to the function `rec` (it is still possible in the present version, but not recommended).

For example, consider the object `puech` created in the previous section. Have a look at the first relocations of the first burst:

```
> head(puech[[1]])
```

```
       x       y       date    dx   dy      dist     dt     R2n   abs.angle
1 699889 3161559 1993-07-01   157  -18  158.0285 172800       0 -0.11415127
2 700046 3161541 1993-07-03 -1206 -508 1308.6252 259200   24973 -2.74292194
3 698840 3161033 1993-07-06   969  463 1073.9320  86400 1377077  0.44574032
4 699809 3161496 1993-07-07 -1182 -555 1305.8135  86400   10369 -2.70260603
5 698627 3160941 1993-07-08    92   48  103.7690  86400 1974568  0.48088728
6 698719 3160989 1993-07-09   272   26  273.2398 345600 1693800  0.09529869
   rel.angle
1         NA
2 -2.6287707
3 -3.0945230
4  3.1348390
5 -3.0996920
6 -0.3855886
```

Imagine that we realize that the X coordinate of the second relocation is actually equal to 700146 instead of 700046:

```
> puech2 <- puech
> puech2[[1]][2,1] <- 700146
> head(puech2[[1]])
```

```
       x       y       date    dx   dy      dist     dt     R2n   abs.angle
1 699889 3161559 1993-07-01   157  -18  158.0285 172800       0 -0.11415127
2 700146 3161541 1993-07-03 -1206 -508 1308.6252 259200   24973 -2.74292194
3 698840 3161033 1993-07-06   969  463 1073.9320  86400 1377077  0.44574032
4 699809 3161496 1993-07-07 -1182 -555 1305.8135  86400   10369 -2.70260603
5 698627 3160941 1993-07-08    92   48  103.7690  86400 1974568  0.48088728
6 698719 3160989 1993-07-09   272   26  273.2398 345600 1693800  0.09529869
   rel.angle
1         NA
2 -2.6287707
3 -3.0945230
4  3.1348390
5 -3.0996920
6 -0.3855886
```

The coordinate has been changed, but the step characteristics are now incorrect. The function `rec` recompute these statistics according to these changes:

```
> head(rec(puech2)[[1]])
```

```
       x       y       date    dx   dy      dist     dt     R2n   abs.angle
1 699889 3161559 1993-07-01   257  -18  257.6296 172800       0 -0.06992472
2 700146 3161541 1993-07-03 -1306 -508 1401.3208 259200   66373 -2.77062747
3 698840 3161033 1993-07-06   969  463 1073.9320  86400 1377077  0.44574032
4 699809 3161496 1993-07-07 -1182 -555 1305.8135  86400   10369 -2.70260603
5 698627 3160941 1993-07-08    92   48  103.7690  86400 1974568  0.48088728
6 698719 3160989 1993-07-09   272   26  273.2398 345600 1693800  0.09529869
   rel.angle
1         NA
2 -2.7007027
3 -3.0668175
4  3.1348390
5 -3.0996920
6 -0.3855886
```

Although the function `rec` can be useful for sporadic use, it is limited when a larger number of modifications is required on the relocations (e.g. filtering incorrect relocations when "cleaning" GPS monitoring data). This is where the class `ltraj` does not fit. For such work, it is more convenient to see the trajectory as a set of points located in both space and time. And for such operations, it is sometimes more convenient to work with data frames. Two functions are provided to quickly convert a ltraj to and from data.frames: the functions `ld` and `dl`.

The function `ld` allows to quickly convert an object of class `ltraj` to the class `data.frame`. Consider for example the object `puech` created in the previous section. We can quickly convert this object towards the class `data.frame`:

```
> puech2 <- ld(puech)
> head(puech2)
```

```
       x       y       date    dx   dy      dist     dt     R2n   abs.angle
1 699889 3161559 1993-07-01   157  -18  158.0285 172800       0 -0.11415127
2 700046 3161541 1993-07-03 -1206 -508 1308.6252 259200   24973 -2.74292194
3 698840 3161033 1993-07-06   969  463 1073.9320  86400 1377077  0.44574032
4 699809 3161496 1993-07-07 -1182 -555 1305.8135  86400   10369 -2.70260603
5 698627 3160941 1993-07-08    92   48  103.7690  86400 1974568  0.48088728
6 698719 3160989 1993-07-09   272   26  273.2398 345600 1693800  0.09529869
   rel.angle    id burst             pkey
1         NA Brock Brock Brock.1993-07-01
2 -2.6287707 Brock Brock Brock.1993-07-03
3 -3.0945230 Brock Brock Brock.1993-07-06
4  3.1348390 Brock Brock Brock.1993-07-07
5 -3.0996920 Brock Brock Brock.1993-07-08
6 -0.3855886 Brock Brock Brock.1993-07-09
```

Note that the data frame contains all the descriptors of the steps. In addition,

two variables `burst` and `id` allow to quickly convert this object back towards the class `ltraj`, with the function `dl`:

```
> dl(puech2)

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id burst nb.reloc NAs date.begin   date.end
1 Brock Brock       30   0 1993-07-01 1993-08-31
2 Calou Calou       19   0 1993-07-03 1993-08-31
3  Chou  Chou       40   0 1992-07-29 1993-08-30
4  Jean  Jean       30   0 1993-07-01 1993-08-31


 infolocs provided. The following variables are available:
[1] "pkey"
```

Using `dl` and `ld` can be extremey useful during the first steps of the analysis, especially during data "cleaning".

3 Managing objects of class `ltraj`

3.1 Cutting a burst into several segments

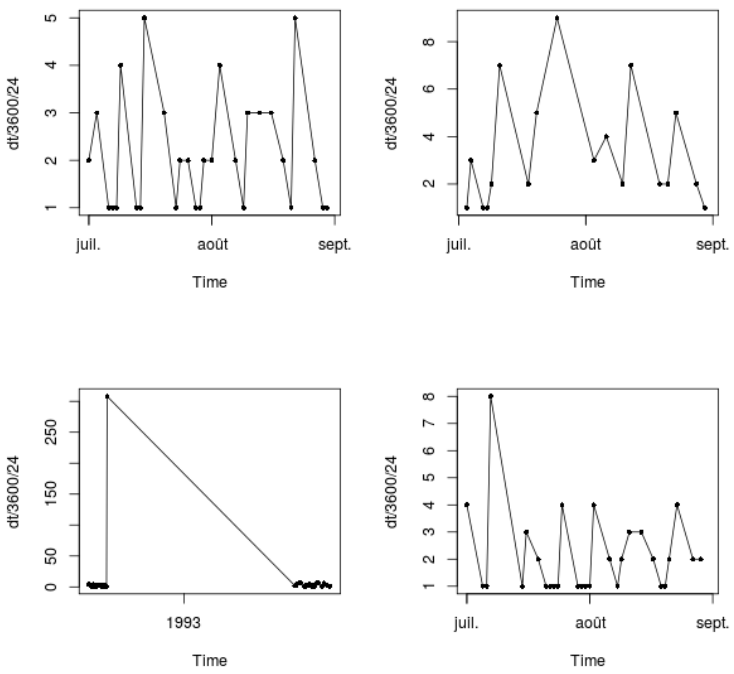
Now, let us analyse the object `puech` created in the previous section. We noted that the object `puech` was not regular:

```
> is.regular(puech)

[1] FALSE
```

The function `is.regular` returns a Boolean... such a result can be obtained from a regular trajectory where just one relocation is missing, or from a completely irregular trajectory... we need more precision!

Have a look at the value of `dt` according to the date, using the function `plotltr`. Because `dt` is measured in seconds and that no more than one relocation is coellected every day, we convert this time lag into days by dividing it by 24 (hours/day) $\times$ 3600 (seconds / hour):

```
> plotltr(puech, "dt/3600/24")
```



The wild boar Chou was monitored during two successive summers (1992 and 1993). We need to "cut" this burst into two "sub-burst". We will use the function `cutltraj` to proceed. We first define a function `foo` that returns `TRUE` when the time lag between two successive relocations is greater than 100 days:

```
> foo <- function(dt) {
+     return(dt> (100*3600*24))
+ }
```

Then, we use the function `cutltraj` to cut any burst relocations with a value of `dt` such that `foo(dt)` is true, into several bursts for which no value of `dt` fullfills this criterion:

```
> puech2 <- cutltraj(puech, "foo(dt)", nextr = TRUE)
> puech2

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs
```

```
Characteristics of the bursts:
     id   burst nb.reloc NAs date.begin   date.end
1 Brock Brock.1       30   0 1993-07-01 1993-08-31
2 Calou Calou.1       19   0 1993-07-03 1993-08-31
3  Chou  Chou.1       16   0 1992-07-29 1992-08-28
4  Chou  Chou.2       24   0 1993-07-02 1993-08-30
5  Jean  Jean.1       30   0 1993-07-01 1993-08-31


 infolocs provided. The following variables are available:
[1] "pkey"
```

Now, note that the burst of `Chou` has been splitted into two bursts: the first burst corresponds to the monitoring of Chou during 1992, and the second burst corresponds to the monitoring of Chou during 1993. We can give more explicit names to these bursts:

```
> burst(puech2)[3:4] <- c("Chou.1992", "Chou.1993")
> puech2

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id     burst nb.reloc NAs date.begin   date.end
1 Brock   Brock.1       30   0 1993-07-01 1993-08-31
2 Calou   Calou.1       19   0 1993-07-03 1993-08-31
3  Chou Chou.1992       16   0 1992-07-29 1992-08-28
4  Chou Chou.1993       24   0 1993-07-02 1993-08-30
5  Jean    Jean.1       30   0 1993-07-01 1993-08-31


 infolocs provided. The following variables are available:
[1] "pkey"
```

Note that the function `id()` can be used similarly to replace the IDs of the animals.

3.2 Playing with bursts

The bursts in an object `ltraj` can be easily managed. For example, consider te object `puech2` created previously:

```
> puech2

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id     burst nb.reloc NAs date.begin   date.end
1 Brock   Brock.1       30   0 1993-07-01 1993-08-31
2 Calou   Calou.1       19   0 1993-07-03 1993-08-31
3  Chou Chou.1992       16   0 1992-07-29 1992-08-28
4  Chou Chou.1993       24   0 1993-07-02 1993-08-30
5  Jean    Jean.1       30   0 1993-07-01 1993-08-31


 infolocs provided. The following variables are available:
[1] "pkey"
```

Imagine that we want to work only on the males (Brock, Calou and Jean). We can subset this object using a classical extraction function:

```
> puech2b <- puech2[c(1,2,5)]
> puech2b

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id   burst nb.reloc NAs date.begin   date.end
1 Brock Brock.1       30   0 1993-07-01 1993-08-31
2 Calou Calou.1       19   0 1993-07-03 1993-08-31
3  Jean  Jean.1       30   0 1993-07-01 1993-08-31


 infolocs provided. The following variables are available:
[1] "pkey"
```

Or, if we want to study the animals monitored in 1993, we may combine this object with the monitoring of Chou in 1993:

```
> puech2c <- c(puech2b, puech2[4])
> puech2c
```

```
*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id     burst nb.reloc NAs date.begin   date.end
1 Brock   Brock.1       30   0 1993-07-01 1993-08-31
2 Calou   Calou.1       19   0 1993-07-03 1993-08-31
3  Jean    Jean.1       30   0 1993-07-01 1993-08-31
4  Chou Chou.1993       24   0 1993-07-02 1993-08-30


 infolocs provided. The following variables are available:
[1] "pkey"
```

It is also possible to select the bursts according to their id of their burst id (see the help page of `Extract.ltraj` for additional information, and in particular the example section).

The function `which.ltraj` can also be used to identify the bursts satisfying a condition. For example, imagine that we want to identify the bursts where the distance between successive relocations was greater than 2000 metres at least once:

```
> bu <- which.ltraj(puech2, "dist>2000")
> bu

    id  burst results
1 Jean Jean.1      18
2 Jean Jean.1      19
```

This data frame contains the ID, burst ID and relocation numbers satisfying the specified criterion. We can then extract the bursts satisfying this criterion:

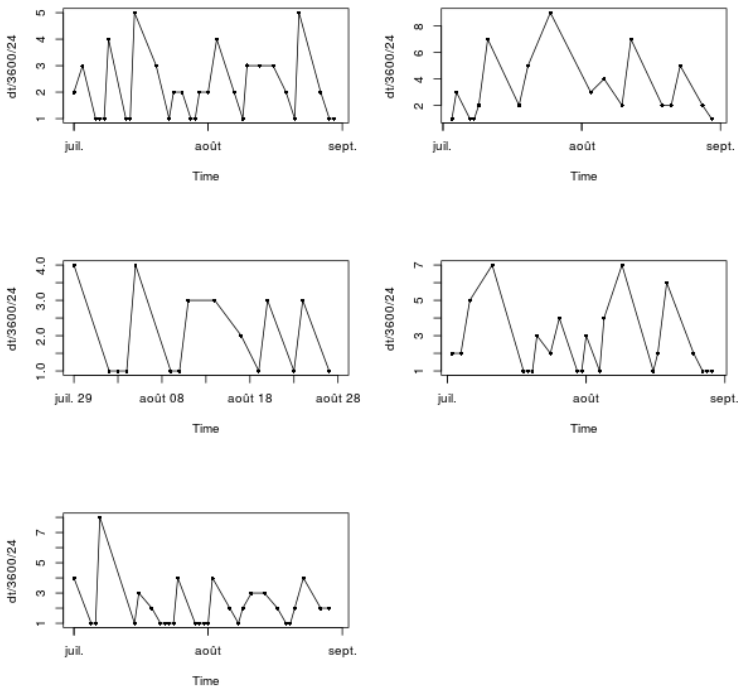
```
> puech2[burst(puech2)%in%bu$burst]

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
    id  burst nb.reloc NAs date.begin   date.end
1 Jean Jean.1       30   0 1993-07-01 1993-08-31
```

```
 infolocs provided. The following variables are available:
[1] "pkey"
```

3.3 Placing the missing values in the trajectory

Now, look again at the time lag between successive relocations:

```
> plotltr(puech2, "dt/3600/24")
```



The relocations have been collected daily, but there are many days during which this relocation was not possible (storm, lack of field workers, etc.). We need to add missing values to define a regular trajectory. To proceed, we will use the function `setNA`. We have to define a reference date:

```
> refda <- strptime("00:00", "%H:%M", tz="Europe/Paris")
> refda
```

```
[1] "2017-11-28 CET"
```

This reference date will be used to check that each date in the object of class `ltraj` is separated from this reference by an integer multiple of the theoretical

dt (here, one day), and place the missing values at the times when relocations should theoretically have been collected. We use the function setNA:

```
> puech3 <- setNA(puech2, refda, 1, units = "day")
> puech3

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id     burst nb.reloc NAs date.begin   date.end
1 Brock   Brock.1       62  32 1993-07-01 1993-08-31
2 Calou   Calou.1       60  41 1993-07-03 1993-08-31
3  Chou Chou.1992       31  15 1992-07-29 1992-08-28
4  Chou Chou.1993       60  36 1993-07-02 1993-08-30
5  Jean    Jean.1       62  32 1993-07-01 1993-08-31


 infolocs provided. The following variables are available:
[1] "pkey"
```

The trajectories are now regular, but there are now a lot of missing values!

3.4 Rounding the timing of the trajectories to define a regular trajectory

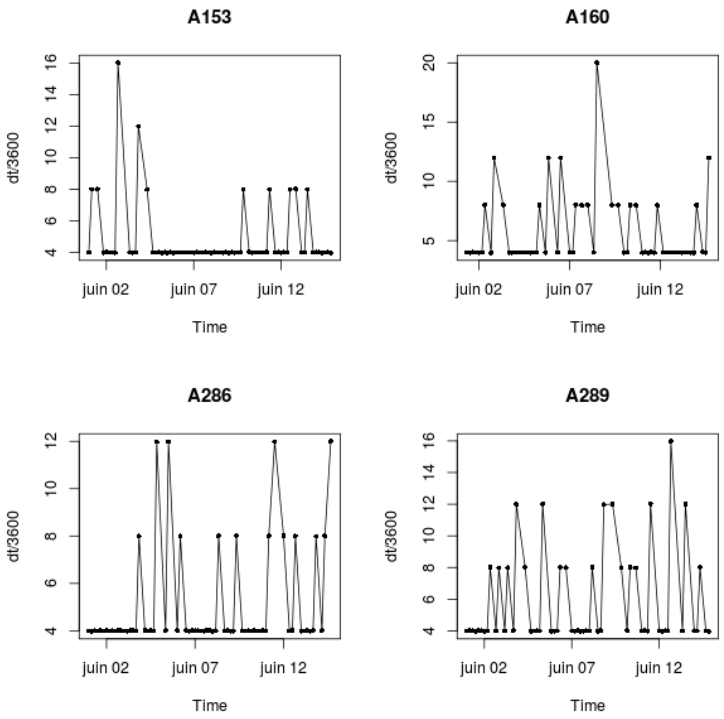
In some cases, despite the fact that the relocations were expected to be collected to return a regular trajectory, a minor delay is sometimes observed in this timing (e.g. the GPS collar needs some time to relocate). For example, consider the monitoring of four ibex in the Belledonne Mountains (French Alps):

```
> data(ibexraw)
> ibexraw

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
    id burst nb.reloc NAs          date.begin            date.end
1 A153  A153       71   0 2003-06-01 00:00:56 2003-06-14 20:01:33
2 A160  A160       59   0 2003-06-01 08:01:35 2003-06-14 16:02:20
```

```
3 A286   A286        68    0 2003-06-01 00:02:45 2003-06-14 16:01:41
4 A289   A289        58    0 2003-06-01 00:01:31 2003-06-14 20:02:32
```

There is a variable time lag between successive relocations. Look at the time lag between successive relocations:

```
> plotltr(ibexraw, "dt/3600")
```



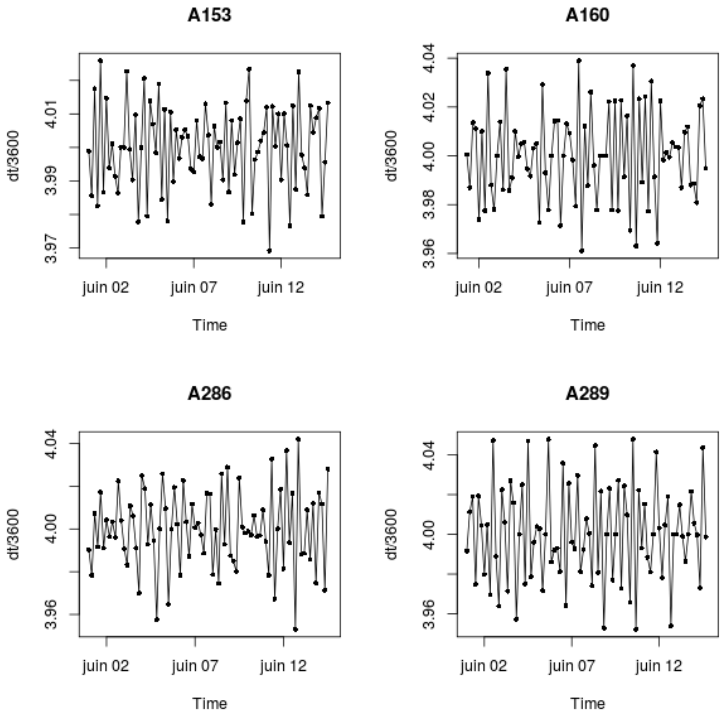
The relocations should have been collected every 4 hours, but there are some missing values. Use the function `setNA` to place the missing values, as in the section 3.3. We define a reference date and place the missing values:

```
> refda <- strptime("2003-06-01 00:00", "%Y-%m-%d %H:%M", tz="Europe/Paris")
> ib2 <- setNA(ibexraw, refda, 4, units = "hour")
> ib2

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs
```

```
Characteristics of the bursts:
    id burst nb.reloc NAs          date.begin            date.end
1 A153  A153       84  13 2003-06-01 00:00:56 2003-06-14 20:01:33
2 A160  A160       81  22 2003-06-01 08:01:35 2003-06-14 16:02:20
3 A286  A286       83  15 2003-06-01 00:02:45 2003-06-14 16:01:41
4 A289  A289       84  26 2003-06-01 00:01:31 2003-06-14 20:02:32
```

Even when filling the gaps with NAs, the trajectory is still not regular. Now, look again at the time lag between successive relocations:

```
> plotltr(ib2, "dt/3600")
```



We can see that the time lag is only slightly different from 4 hour. The function `sett0` can be used to "round" the timing of the coordinates:

```
> ib3 <- sett0(ib2, refda, 4, units = "hour")
> ib3

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
```

```
Regular traject. Time lag between two locs: 14400 seconds

Characteristics of the bursts:
    id burst nb.reloc NAs          date.begin            date.end
1 A153  A153       84  13 2003-06-01 00:00:00 2003-06-14 20:00:00
2 A160  A160       81  22 2003-06-01 08:00:00 2003-06-14 16:00:00
3 A286  A286       83  15 2003-06-01 00:00:00 2003-06-14 16:00:00
4 A289  A289       84  26 2003-06-01 00:00:00 2003-06-14 20:00:00
```

The trajectory is now regular.

Important note: The functions `setNA` and `sett0` are to be used to define a regular trajectory from a nearly regular trajectory. **It is NOT intended to transform an irregular trajectory into a regular one** (many users of `adehabitat` asked this question).

3.5 A special type of trajectories: same duration

In some cases, an object of class `ltraj` contains several regular bursts of the same duration characterized by relocations collected at the same time (same time lags between successive relocations, same number of relocations). We can check whether an object of class "ltraj" is of this type with the function `is.sd`. For example, consider again the movement of 4 ibexes monitored using GPS, stored in an object of class `ltraj` created in the previous section:

```
> is.sd(ib3)

[1] FALSE
```

This object is not of the type `sd` (same duration). However, theoretically, all the trajectories should have been sampled at the same time points. It is regular, but there are mismatches between the time of the relocations:

```
> ib3

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Regular traject. Time lag between two locs: 14400 seconds

Characteristics of the bursts:
    id burst nb.reloc NAs          date.begin            date.end
1 A153  A153       84  13 2003-06-01 00:00:00 2003-06-14 20:00:00
2 A160  A160       81  22 2003-06-01 08:00:00 2003-06-14 16:00:00
3 A286  A286       83  15 2003-06-01 00:00:00 2003-06-14 16:00:00
4 A289  A289       84  26 2003-06-01 00:00:00 2003-06-14 20:00:00
```

This is caused by the fact that there are missing relocations at the beginning and/or end of the monitoring for several animals (A160 and A286). We can use the function `set.limits` to define the time of beginning and ending of the trajectories. This function adds NAs to the beginning and ending of the monitoring when required:

```
> ib4 <- set.limits(ib3, begin = "2003-06-01 00:00",
+                   dur = 14, units = "day", pattern = "%Y-%m-%d %H:%M",
+                   tz="Europe/Paris")
> ib4

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone unspecified: dates printed in user time zone *
Regular traject. Time lag between two locs: 14400 seconds

Characteristics of the bursts:
    id burst nb.reloc NAs date.begin   date.end
1 A153  A153       85  14 2003-06-01 2003-06-15
2 A160  A160       85  26 2003-06-01 2003-06-15
3 A286  A286       85  17 2003-06-01 2003-06-15
4 A289  A289       85  27 2003-06-01 2003-06-15
```

All the trajectory are now covering the same time period:

```
> is.sd(ib4)

[1] TRUE
```

Remark: in our example, all the bursts are covering exactly the same time period (all begin at the same time and date and all stop at the same time and date). However, the function `set.limits` is much more flexible. Imagine for example that we are studying the movement of an animal during the night, from 00:00 to 06:00. If we have one burst per night, then it is possible to define an object of class `ltraj`, type `sd`, containing several nights of monitoring, even if the nights of monitoring do not correspond to the same date. If we consider that all the bursts cover the same period, then it is still possible to use the function `set.limits` to define an object of type `sd` (this is explained deeply on the help page of `set.limits`).

It is then possible to store some parameters of `sd` objects into a data frame (with one relocation per row and one burst per column), using the function `sd2df`. For example, considering the distance between successive relocations:

```
> di <- sd2df(ib4, "dist")
> head(di)
```

```
       A153      A160     A286       A289
1  41.04875        NA 214.2008   15.65248
2        NA        NA 428.8508  293.60007
3        NA  517.4882 606.9802  837.70401
4        NA 1533.7881 637.1538 1108.75065
5        NA  608.6912 216.0000   72.61543
6 244.66303 2264.8366 216.8156  383.65870
```

This data frame can then be used to study the interactions or similarities between the bursts.

3.6 Metadata on the trajectories (Precision of the relocations, etc.)

Sometimes, additional information is available for each relocation, and we may wish to store this information in the object of class `ltraj`, to allow the analysis of the relationships between these additional variables and the parameters of the trajectory.

This meta information can be stored in the attribute `infolocs` of each burst. This should be defined when creating the object `ltraj`, *but can also be defined later* (see section 4.6 for an example). For example, load the dataset `capreochiz`:

```
> data(capreochiz)
> head(capreochiz)
```

```
         x        y                date Dop Status Temp Act Conv
1 967.3994 1137.488 2004-02-13 17:02:18 5.7  3DDif   10   0    0
2 961.9346 1141.413 2004-02-14 00:31:23 3.6  3DDif    3   0    0
3 961.9340 1141.401 2004-02-14 12:02:17 6.9  3DDif    8   0    0
4 961.9426 1141.409 2004-02-15 00:00:47 5.0  3DDif    3   0    0
5 961.9472 1141.405 2004-02-15 00:30:13 4.1  3DDif    2   0    0
6 961.9430 1141.409 2004-02-15 12:01:11 8.1  3DDif    5   0    0
```

This dataset contains the relocations of one roe deer monitored using a GPS collar in the Chize forest (Deux-Sevres, France). This dataset contains the x and y coordinates (in kilometres), the date, and several variables characterizing the precision of the relocations. Note that the date is already of class `POSIXct`. We now define the object of class `ltraj`, storing the variables `Dop, Status, Temp, Act, Conv` in the attribute `infolocs` of the object:

```
> capreo <- as.ltraj(xy = capreochiz[,c("x","y")], date = capreochiz$date,
+                    id = "Roe.Deer",
+                    infolocs = capreochiz[,4:8])
> capreo
```

```
*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
        id    burst nb.reloc NAs          date.begin            date.end
1 Roe.Deer Roe.Deer     2355   0 2004-02-13 17:02:18 2004-08-31 06:00:47


 infolocs provided. The following variables are available:
[1] "Dop"    "Status" "Temp"   "Act"    "Conv"
```

The object `capreo` can be managed as usual. The function `infolocs()` can be used to retrieve the attributes `infolocs` of the bursts building up a trajectory:

```
> inf <- infolocs(capreo)
> head(inf[[1]])

  Dop Status Temp Act Conv
1 5.7  3DDif   10   0    0
2 3.6  3DDif    3   0    0
3 6.9  3DDif    8   0    0
4 5.0  3DDif    3   0    0
5 4.1  3DDif    2   0    0
6 8.1  3DDif    5   0    0
```

The function `removeinfo` can be used to set the attribute `infolocs` of all bursts to `NULL`.

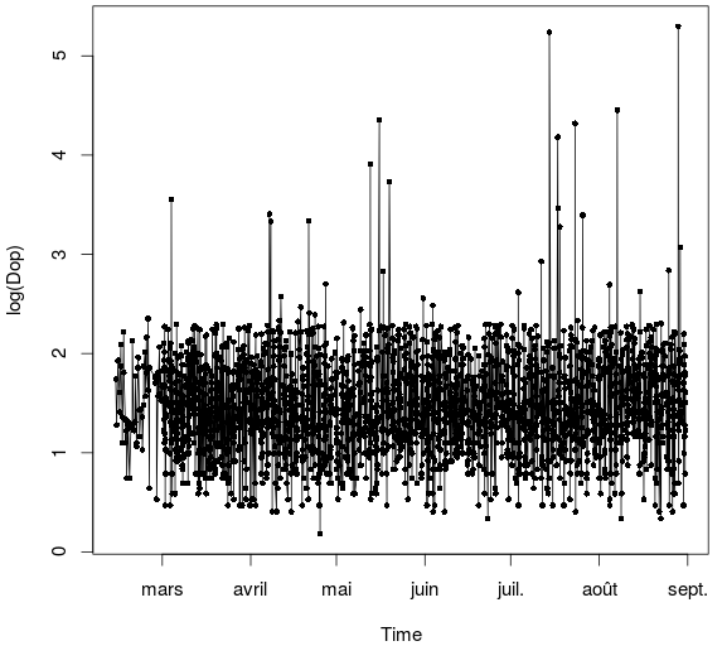
Note that it is required that:

- all the bursts are characterized by the same variables in the attribute `infolocs`. For example, it is not possible to store only the variable `Dop` for one burst and only the variable `Status` for another burst into the same object;

- each row of the data frame stored as attributes `infolocs` correspond to one relocation (that is, the number of rows of the attribute should be the same as the number of relocations in the corresponding burst).

Most functions of the package `adehabitatLT` do manage this attribute. For example, the functions `cutltraj` and `plotltr` can be used by calling variables stored in this attribute (as well as many other functions). For example:

```
> plotltr(capreo, "log(Dop)")
```



4 Analyzing the trajectories

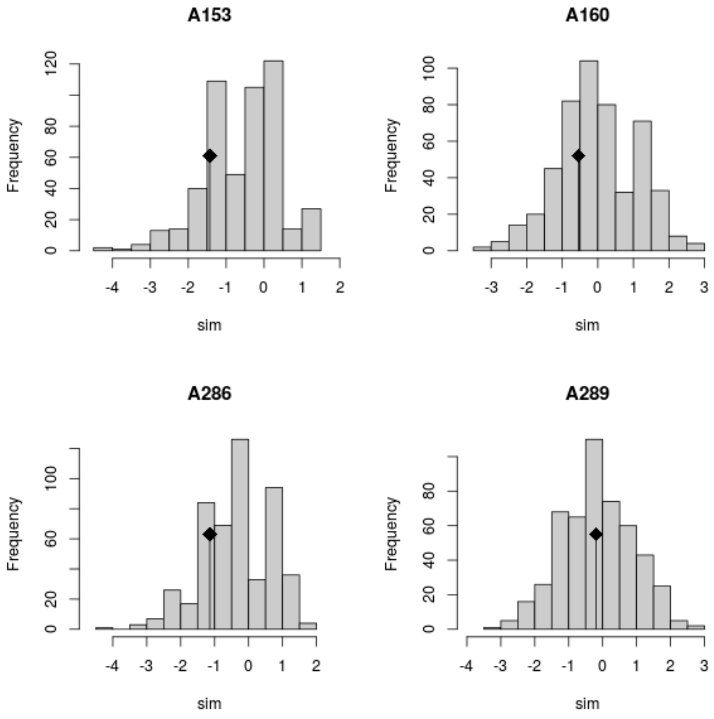
In this section, we will describe several tools available in `adehabitatLT` to analyse a trajectory.

4.1 Randomness of the missing values

A first important point is the examination of the distribution of the missing values in the trajectory. Missing values are frequent in the trajectories of animals collected using telemetry (e.g., GPS collar may not receive the signal of the satellite at the time of relocation, due for example to the habitat structure obscuring the signal, etc.). As noted by Graves and Waller (2006), the analysis of the patterns of missing values should be part of trajectory analysis.

The package `adehabitatLT` provides several tools for this analysis. For example, consider the object `ib4` created in section 3.5, and containing 4 bursts describing the movements of 4 ibexes in the Belledonne moutain. We can first test whether the missing values occur at random in the monitoring using the function `runsNAltraj`:

```
> runsNAltraj(ib4)
```



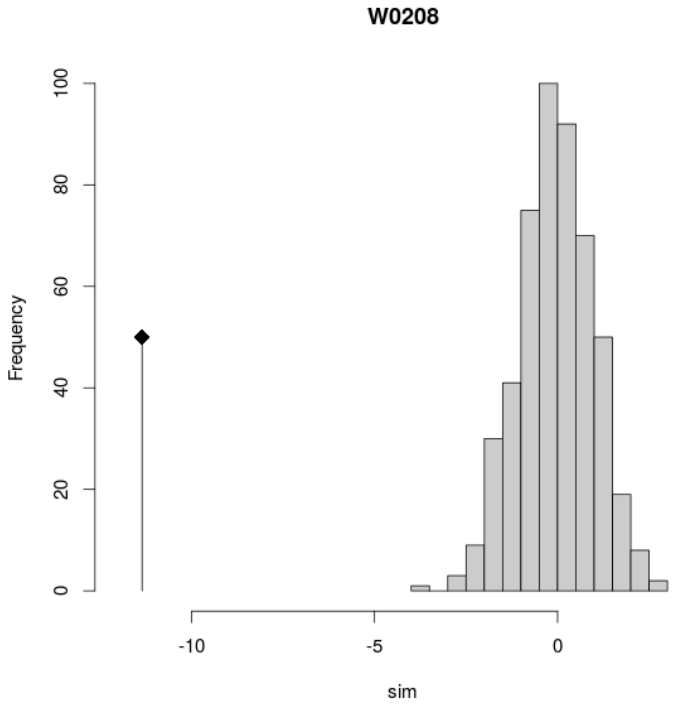
In this case, no difference appears between the number of runs actually observed in our trajectories and the distribution of the number of runs under the hypothesis of a random distribution of the NAs. The hypothesis of a random distribution of the NAs seems reasonable here. The statistics used here for the test is the number of runs in the sequence of relocations. For example, the sequence reloc-NA-NA-reloc-reloc-reloc-NA-NA-NA-reloc contains 5 runs, 3 runs of successful relocations and 2 runs of missing values. Under the hypothesis of random distribution of the missing values in the sequence, the theoretical expectation and standard deviation of the number of runs is known. The runs test is a randomization test that compares the standardized value of the number of runs (i.e. (value-expectation)/(standard deviation)) to the distribution of values obtained after randomizing the distribution of the NA in the sequence. Thus, a negative value of this standardized number of runs indicates that the missing values tend to be clustered together in the sequence.

But now, consider the distribution of the missing values in the case of the monitoring of one brown bear using a GPS collar:

```
> data(bear)
> bear

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: UTC *
Regular traject. Time lag between two locs: 1800 seconds

Characteristics of the bursts:
     id burst nb.reloc NAs          date.begin            date.end
1 W0208 W0208     1157 157 2004-04-19 16:30:00 2004-05-13 18:30:00
```

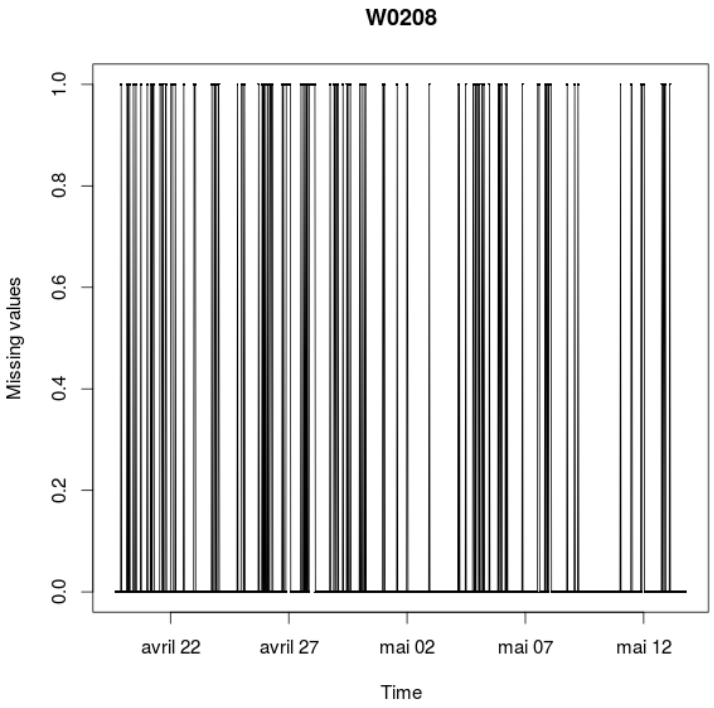
This trajectory is regular. The bear was monitored during one month, with one relocation every 30 minutes. We now test for a random distribution of the missing values for this trajectory:

```
> runsNAltraj(bear)
```



In this case, the missing values are not distributed at random. Have a look at the distribution of the missing values:

```
> plotNAltraj(bear)
```



Because of the high number of relocations in this trajectory, this graph is not very clear. So a better way to study the distribution of the missing values is to work directly on the vector indicating whether the relocations are missing or not. That is:

```
> missval <- as.numeric(is.na(bear[[1]]$x))
> head(missval)

[1] 0 0 0 0 0 0
```

This vector can then be analyzed using classical time series methods (e.g. Diggle 1990). We do not pursue on this aspect, as this is not the aim of this vignette to describe time series methods.

4.2 Should we consider the time?

4.2.1 Type II or type I?

Until now, we have only considered trajectories of type II (time recorded). However, a common approach to the analysis of animal movements is to consider the movement as a discretized curve, and to study the geometrical properties of this curve (e.g., Turchin 1998; Benhamou 2004). That is, even if the data collection

implied the recording of the time, it is often more convenient to consider the monitored movement as a trajectory of type I. There are two ways to define a type I trajectory with the functions of `adehabitatLT`. The first is to set the argument `typeII=FALSE` when calling the function `as.ltraj`. The second is to use the function `typeII2typeI`. For example, considering the trajectory of the bear loaded in the previous section, we can transform it into a type I object by:

```
> bearI <- typeII2typeI(bear)
> bearI

********** List of class ltraj **********

Type of the traject: Type I (time not recorded)

Characteristics of the bursts:
     id burst Nb.reloc NAs
1 W0208 W0208     1157 157


 infolocs provided. The following variables are available:
[1] "pkey"
```

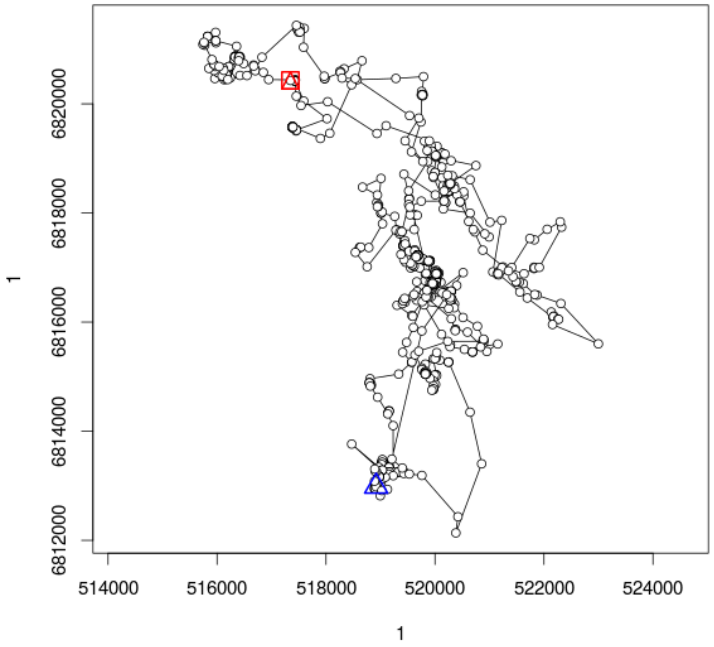
Nothing has changed, except that the time is replaced by an integer vector ordering the relocations in the trajectory.

4.2.2 Rediscretizing the trajectory in space

Several authors have advised to rediscretize type I trajectories so that they are built by steps with a constant length (e.g. Turchin 1998). This is a convenient approach to the analysis, as all the geometrical properties of the trajectory can be summarized by studying the variation of the relative angles.

The function `redisltraj` can be used for this rediscretization. For example, look at the trajectory of the brown bear stored in `bearI` (created in the previous section):

```
> plot(bearI)
```

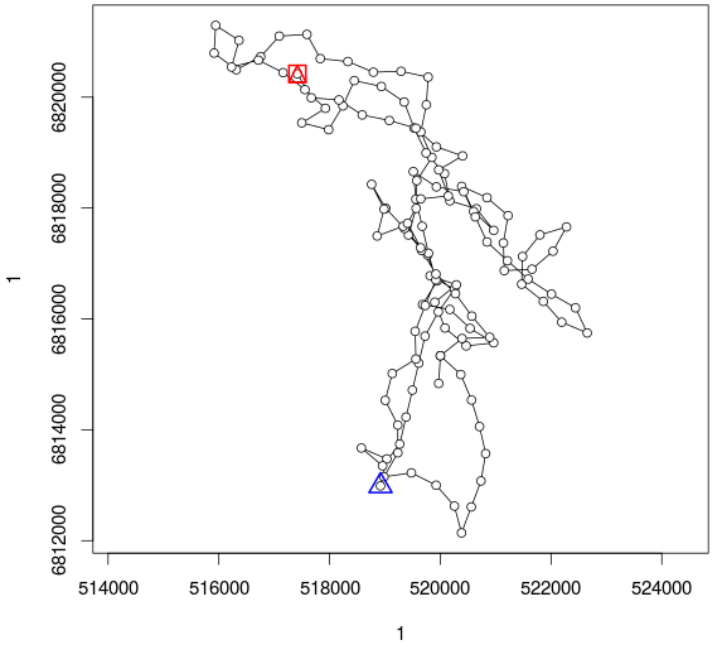


Now, rediscretize this trajectory with constant step length of 500 metres:

```
> bearIr <- redisltraj(bearI, 500)
> bearIr

********** List of class ltraj **********

Type of the traject: Type I (time not recorded)

Characteristics of the bursts:
     id      burst Nb.reloc NAs
1 W0208 W0208.R500      131   0


 infolocs provided. The following variables are available:
[1] "pkey"
```

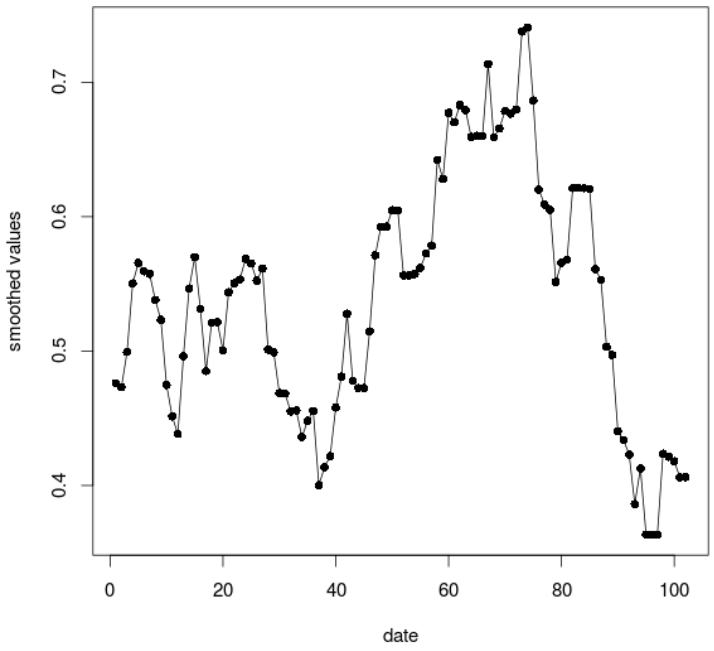
The number of relocations has increased. Have a look at the rediscretized trajectory:

```
> plot(bearIr)
```



Then, the geometrical properties can be studied by examining the distribution of the relative angles. For example, the function `sliwinltr` can be used to smooth the cosine of relative angle using a sliding window method:

```
> sliwinltr(bearIr, function(x) mean(cos(x$rel.angle)), type="locs", step=30)
```



The beginning of the trajectory is characterized by mean cosine close to 0.5 (tortuous trajectory). Then the movements of the animal is more linear (i.e., less tortuous). A finer analysis should now be done on these data. So that we need to get the relative angles from this rediscretized trajectory:

```
> cosrelangle <- cos(bearIr[[1]]$rel.angle)
> head(cosrelangle)
```

```
[1]  1.00000000  0.17250586 -0.95704496 -0.02868422  0.90540259  1.00000000
```

This vector can now be analyzed using classical time series analysis methods. We do not pursue this analysis further, as this is beyond the scope of this vignette.

4.2.3 Rediscretizing the trajectory in time

A common way to deal with missing values consist in linear interpolation. That is given relocations $r_1 = (x_1, y_1, t_1)$ and $r_3 = (x_3, y_3, t_2)$, it is possible to estimate the relocation r_2 collected at t_2 with:

$$x_2 = \frac{(t_2 - t_1)}{(t_3 - t_1)}(x_3 - x_1)$$

$$y_2 = \frac{(t_2 - t_1)}{(t_3 - t_1)}(y_3 - y_1)$$

Such an interpolation may be of limited value when many relocations are missing (because it supposes implicitly that the animal is moving along a straight line). However, it can be useful to "fill in" a small number of relocations. Such an interpolation can be carried out with the function `redisltraj`, setting the argument `type = "time"`. In this case, this function rediscretizes the trajectory so that the time lag between successive relocations is constant. For example, consider the data set `porpoise`:

```
> data(porpoise)
> porpoise

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: UTC *
Regular traject. Time lag between two locs: 86400 seconds

Characteristics of the bursts:
        id    burst nb.reloc NAs          date.begin            date.end
1      GUS      GUS       64   0 2004-01-20 11:00:00 2004-03-23 11:00:00
2    David    David      101   5 1999-08-02 18:00:00 1999-11-10 17:00:00
3 Mitchell Mitchell       97   9 1999-08-09 23:00:00 1999-11-13 22:00:00
4     Eric     Eric       96  36 2000-08-19 13:00:00 2000-11-22 12:00:00
```

We focus on the first three animals. David and Mitchell contain a small number of missing relocations, so that it can be a good idea to interpolate these relocations, to facilitate later calculations. We first remove the missing values with `na.omit` and then interpolate the trajectory so that the time lag between successive relocations is constant:

```
> (porpoise2 <- redisltraj(na.omit(porpoise[1:3]), 86400, type="time"))

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: UTC *
Regular traject. Time lag between two locs: 86400 seconds

Characteristics of the bursts:
        id    burst nb.reloc NAs          date.begin            date.end
1    David    David      100   0 1999-08-02 18:00:00 1999-11-09 18:00:00
2      GUS      GUS       63   0 2004-01-20 11:00:00 2004-03-22 11:00:00
3 Mitchell Mitchell       96   0 1999-08-09 23:00:00 1999-11-12 23:00:00
```

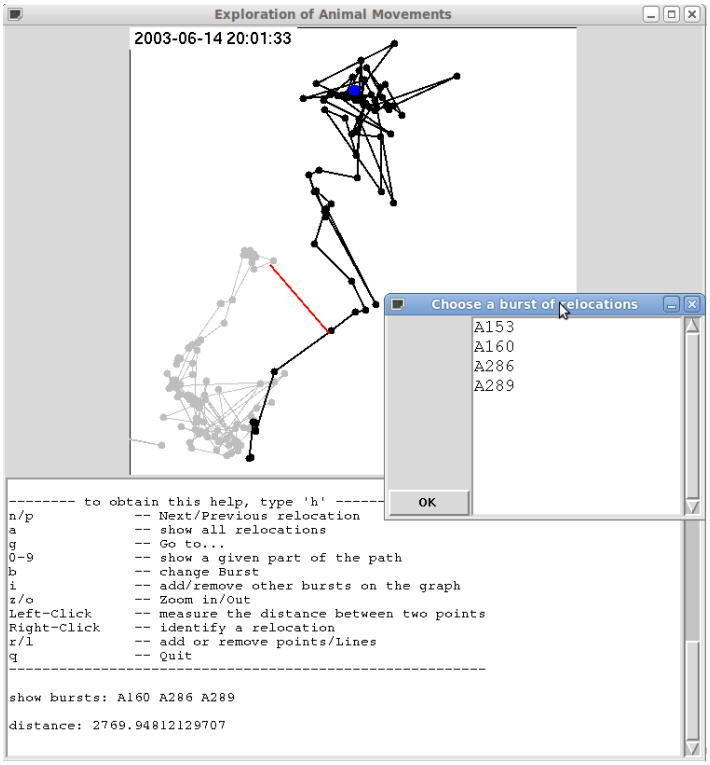
```
 infolocs provided. The following variables are available:
[1] "pkey"
```

The object `porpoise2` now does not contain any missing values. Note that the rediscretization step should be expressed in seconds.

4.3 Dynamic exploration of a trajectory

The package `adehabitatLT` provides a function very useful for the dynamic exploration of animal movement: the function `trajdyn`. This function allows to dynamically zoom/unzoom, measure distance between several bursts or several relocations, explore the trajectory in space and time, etc. For example, the ibex data set is explored by typing:

```
> trajdyn(ib4)
```



Note that this function can draw a background defined by an object of class `SpatialPixelsDataFrame` or `SpatialPolygonsDataFrame`.

4.4 Analyzing autocorrelation

Dray et al. (2010) noted that the analysis of the sequential autocorrelation of the descriptive parameters presented in section 2.2 is essential to the understanding of the mechanisms driving these movements. The approach proposed by these authors is implemented in `adehabitatLT`. In this section, we describe the functions that can be used to carry out this kind of analysis.

A positive autocorrelation of a parameter means that values taken near to each other tend to be either more similar (positive autocorrelation) or less similar (negative autocorrelation) than would be expected from a random arrangement.

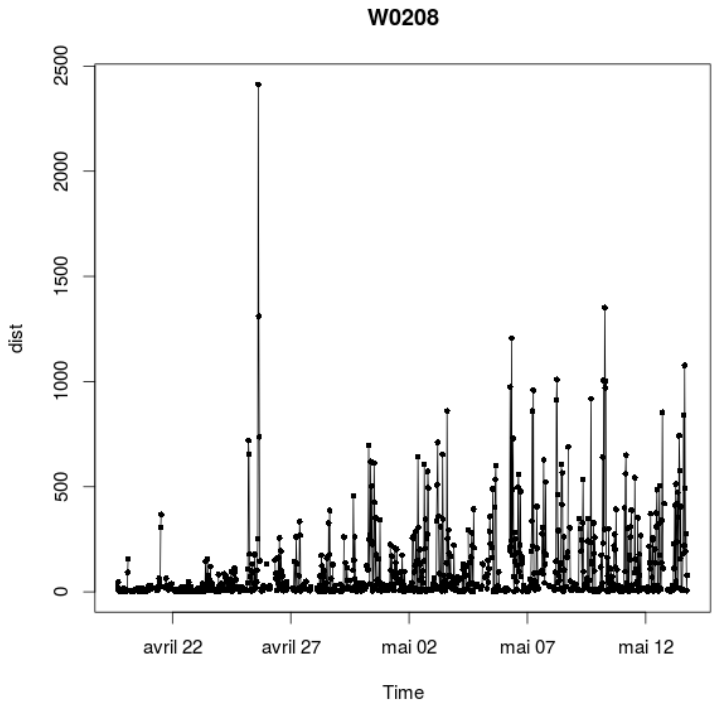
4.4.1 Testing for autocorrelation of the linear parameters

The independence test of Wald and Wolfowitz (1944) is implemented in the generic function `wawotest`. Basically, this function can be used to test the sequential autocorrelation in a vector. However, the method `wawotest.ltraj` allows to test autocorrelation for the three *linear* parameters `dx`, `dy` and `dist` for each burst in an object of class `ltraj`. For example, consider again the monitoring of movements of the bear:

```
> wawotest(bear)

249 NA removed
249 NA removed
249 NA removed
[[1]]
               dx             dy      dist
a    1.180527e+02  2.089499e+02 405.90227
ea -1.000000e+00 -1.000000e+00  -1.00000
va   8.927603e+02  8.612297e+02 878.76704
za   3.984481e+00  7.154119e+00  13.72629
p    3.381395e-05  4.211076e-13   0.00000
```

Note that this function removes the missing values before the test. The row `p` indicates the P-value of the test. We can see that the three linear parameters are strongly positively autocorrelated. There are periods during which the animal is traveling at large speed and periods when the animals are walking at lower speed. Note that this was already apparent on the graph showing the changes with time of the distance between successive relocations:

```
> plotltr(bear, "dist")
```



This was even clearer on the graph showing the moving average of the distance (with a window of 5 days):

```
> sliwinltr(bear, function(x) mean(na.omit(x$dist)),
+           5*48, type="locs")
```

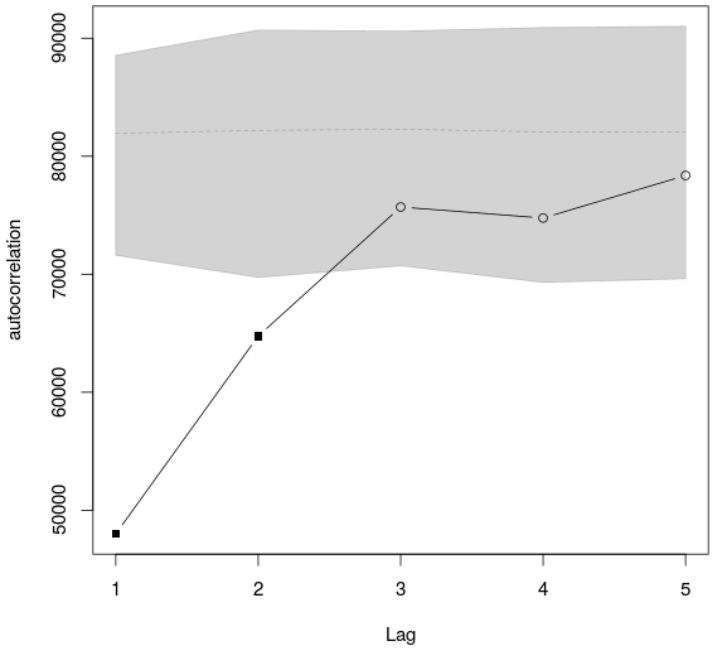
(not executed in this report).

4.4.2 Analyzing the autocorrelation of the parameters

The autocorrelation function (ACF) $\rho(a)$ measures the correlation between a parameter measured at time t and the same parameter measured at time $t - a$ in the same time series (Diggle, 1990). This allows to analyze the autocorrelation, and to identify the scales at which this autocorrelation occurs. Dray et al. (2010) noted that the autocorrelation function measured at lag 1 ($\rho(1)$) is mathematically equivalent to the independence test of Wald and Wolfowitz (1944).

Dray et al. (2010) extended the mathematical bases underlying the ACF to handle the missing data occurring frequently in the trajectories. Their approach is implemented in the function `acfdist.ltraj` (the management of NAs is described in detail on the help page of this function). For example, consider again the monitoring of a brown bear:

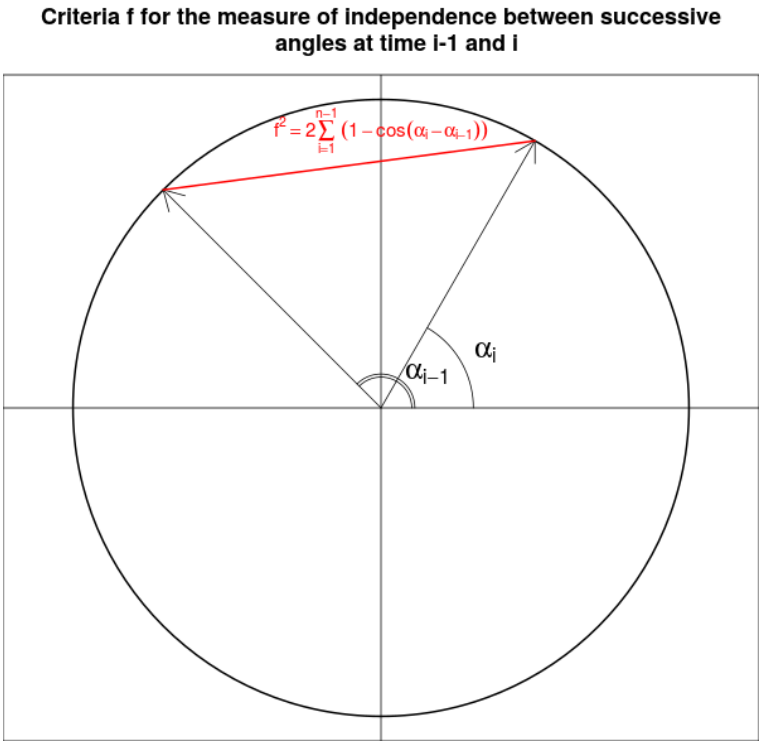
```
> acfdist.ltraj(bear, lag=5, which="dist")
```



We have calculated here the ACF for the distance for a time lag up to 5 relocations. The interested reader can try to calculate the ACF for trajectories up to 100 relocations to see the cyclic patterns occurring in this trajectory.

4.4.3 Testing autocorrelation of the angles

The test of the autocorrelation of the angular parameters (relative or absolute angles, see section 2.2) is based on the chord distance between successive angles (see Dray et al. 2010 for additional details):



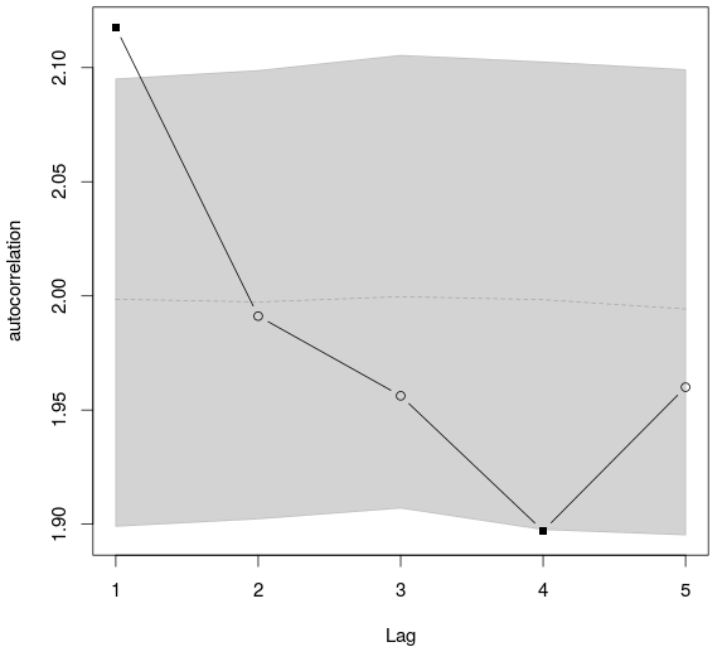
For example, the function `testang.ltraj` is a randomization test using the mean squared chord distance as a criteria. For example, considering again the trajectory of the bear, we can test the autocorrelation of the relative angles between successive moves:

```
> testang.ltraj(bear, "relative")

[[1]]
Monte-Carlo test
Call: as.randtest(sim = res$sim[-1], obs = res$sim[1], alter = alter)

Observation: 1471.667

Based on 999 replicates
Simulated p-value: 0.052
Alternative hypothesis: two-sided

    Std.Obs Expectation    Variance
  -1.910304 1544.766108 1464.266878
```

4.4.4 Analyzing the autocorrelation of angular parameters

Dray et al. (2010) extended the ACF to angular parameters by considering the chord distance as a criteria to build the ACF. This approach is implemented in the function `acfang.ltraj`. Considering again the `bear` dataset:

```
> acfang.ltraj(bear, lag=5)
```



We can see that the relative angle observed at time i is significantly correlated with the angle observed at time $i-1$.

4.5 Segmenting a trajectory into segments characterized by a homogenous behaviour

4.5.1 The method of Gueguen (2001)

We implemented a new approach to the segmentation of movement data, relying on a Bayesian partitioning of a sequence. This approach was originally developed in molecular biology, to partition DNA sequences (Gueguen 2001). We describe this approach in this section.

Biologically, a positive autocorrelation in any of the descriptive parameters may mean that the animal behaviour is changing with time (there are periods

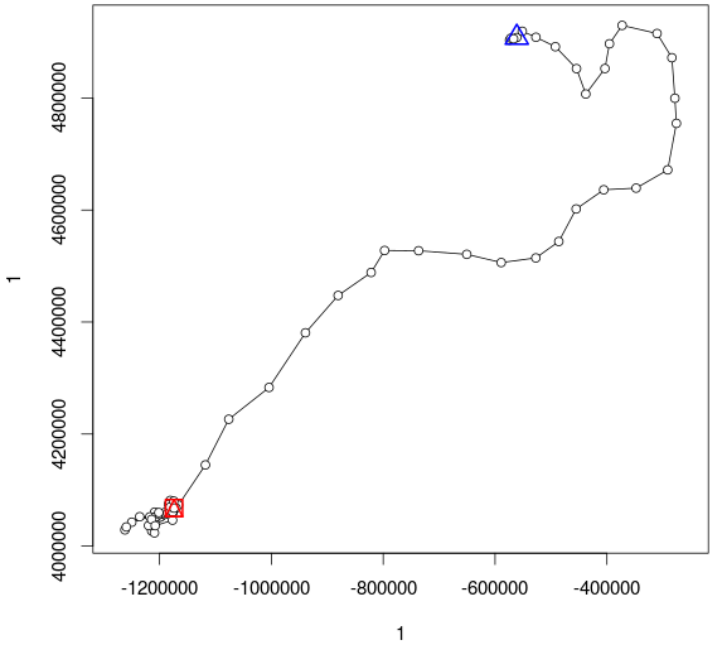
during which the animal is feeding, other during which the animal is resting, etc.). The idea is then to segment the trajectory of the animal into homogenous segments.

We will use the movements of a porpoise monitored using an Argos collar to illustrate this method. First we load the data:

```
> data(porpoise)
> gus <- porpoise[1]
> gus

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: UTC *
Regular traject. Time lag between two locs: 86400 seconds

Characteristics of the bursts:
   id burst nb.reloc NAs          date.begin            date.end
1 GUS   GUS       64   0 2004-01-20 11:00:00 2004-03-23 11:00:00
```

The trajectory is regular and is built by relocations collected every 24 hours during two months. Plot the data:

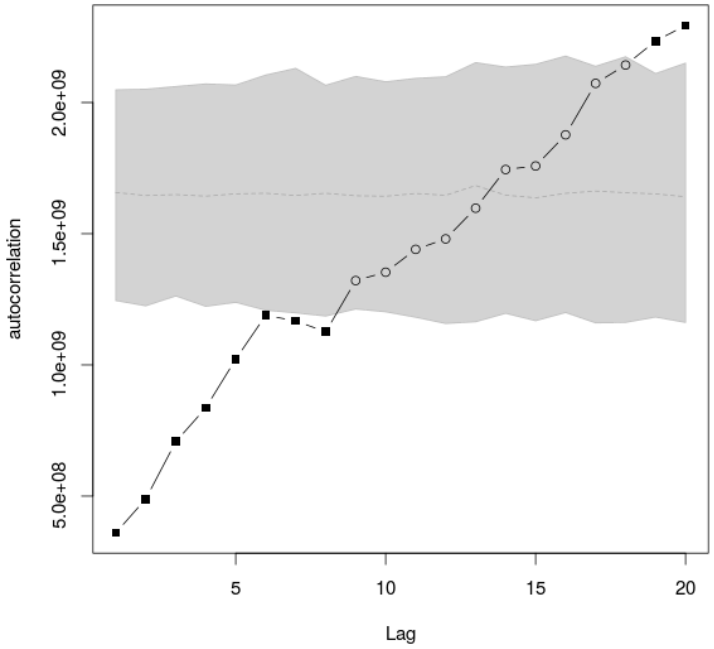
```
> plot(gus)
```



Visually, **the trajectory seems to be built by three segments**. At the very beginning of the trajectory, the animal is performing very short moves. Then, the animal is travelling faster toward the southwest, and finally, the animal is again performing very small moves.

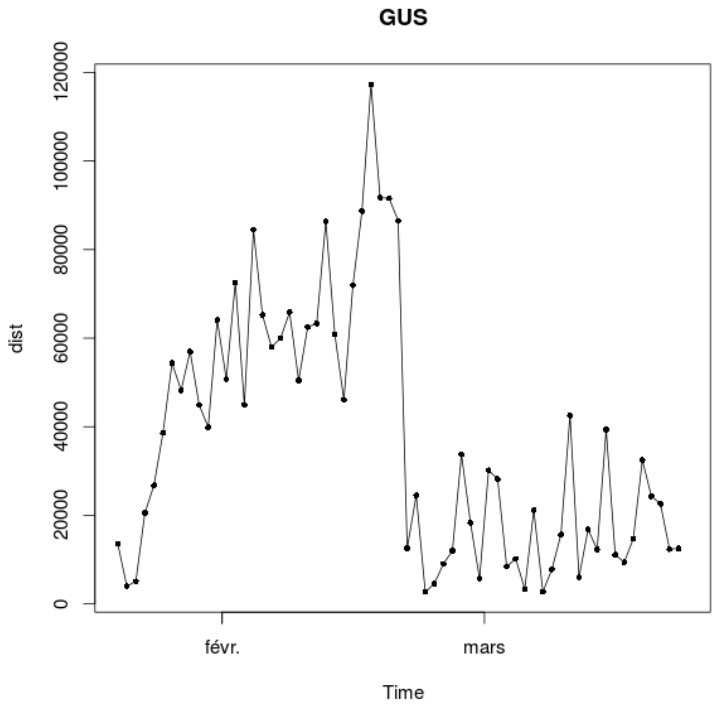
We can draw the ACF for the distance between successive relocations to illustrate the autocorrelation pattern from another point of view:

```
> acfdist.ltraj(gus, "dist", lag=20)
```



There is a strong autocorrelation pattern present in the data, up to lag 8. We can plot the distances between successive relocations according to the date

```
> plotltr(gus, "dist")
```



Now, let us suppose that the distances between successive relocations have been generated by a normal distribution, with different means corresponding to different behaviours. Let us built 10 models corresponding to 10 values of the mean distance ranging from 0 to 130 km/day:

```
> (tested.means <- round(seq(0, 130000, length = 10), 0))

 [1]      0  14444  28889  43333  57778  72222  86667 101111 115556 130000
```

Based on the visual exploration of the distribution of distance, we set the standard deviation of the distribution to 5 km. We can now define 10 models characterized by 10 different values of means and with a standard deviation of 5 km:

```
> (limod <- as.list(paste("dnorm(dist, mean =",
+                         tested.means,
+                         ", sd = 5000)")))

[[1]]
[1] "dnorm(dist, mean = 0 , sd = 5000)"

[[2]]
```

```
[1] "dnorm(dist, mean = 14444 , sd = 5000)"

[[3]]
[1] "dnorm(dist, mean = 28889 , sd = 5000)"

[[4]]
[1] "dnorm(dist, mean = 43333 , sd = 5000)"

[[5]]
[1] "dnorm(dist, mean = 57778 , sd = 5000)"

[[6]]
[1] "dnorm(dist, mean = 72222 , sd = 5000)"

[[7]]
[1] "dnorm(dist, mean = 86667 , sd = 5000)"

[[8]]
[1] "dnorm(dist, mean = 101111 , sd = 5000)"

[[9]]
[1] "dnorm(dist, mean = 115556 , sd = 5000)"

[[10]]
[1] "dnorm(dist, mean = 130000 , sd = 5000)"
```

The approach of Gueguen (2001) allows, based on these a priori models, to find both the number and the limits of the segments building up the trajectory. Any model can be supposed for any parameter of the steps (the distance, relative angles, etc.), provided that the model is Markovian.

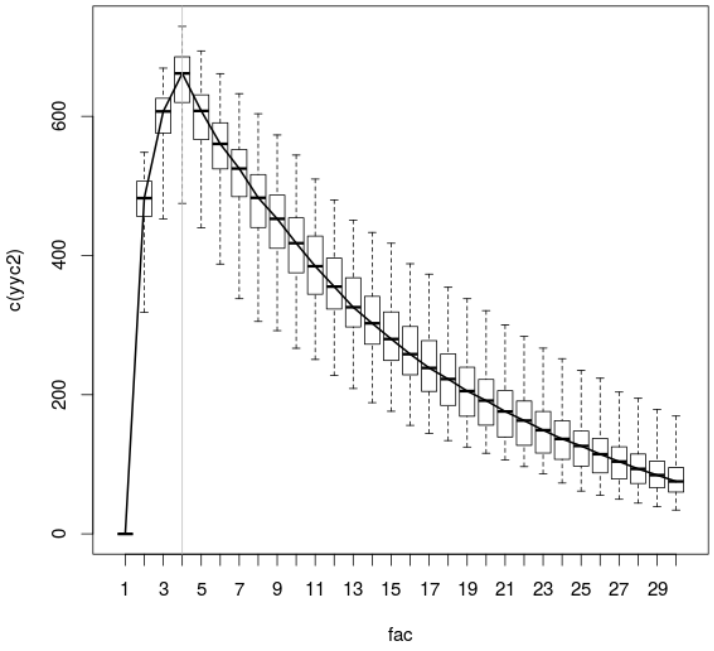
Given the set of *a priori* models, for a given step of the trajectory, it is possible to compute the probability density that the step has been generated by each model of the set. The function `modpartltraj` computes the matrix containing the probability densities associated to each step (rows), under each model of the set (columns):

```
> mod <- modpartltraj(gus, limod)
> mod

*************************************
* Probabilities computed for a trajectory
* with the following models:

 [1] "mod.1"  "mod.2"  "mod.3"  "mod.4"  "mod.5"  "mod.6"  "mod.7"  "mod.8"
 [9] "mod.9"  "mod.10"
```

Then, we can estimate the optimal number of segments in the trajectory, given the set of *a priori* models, using the function `bestpartmod`, taking as argument the matrix `mod`:

```
> bestpartmod(mod)
```

Maximum likelihood for K = 4



This graph presents the value of the log-likelihood (y) that the trajectory is actually made of K segments (x). Note that this log-likelihood is actually corrected using the method of Gueguen (2001) (which implies the Monte Carlo simulation of the independence of the steps in the trajectory – explaining the boxplots –, see the help page of `bestpartmod` for further details on this procedure). In this case, the method indicates that 4 segments are a reasonable choice for the segmentation. **This is a surprise for us, as we rather expected 3 segments** (actually, the number of segments returned by the function depend on the models supposed *a priori*).

Finally, the function `partmod.ltraj` can be used to compute the segmentation. The mathematical rationale underlying these two functions is the following: given an optimal k-partition of the trajectory, if the i^{th} step of the trajectory belongs to the segment k predicted by the model d, then either the

relocation $i-1$ belongs to the same segment, in which case the segment containing $i-1$ is predicted by d, or the relocation $i-1$ belongs to another segment, and the other $(k-1)$ segments together constitute an optimal $(k-1)$ partition of the trajectory [1–$(i-1)$]. These two probabilities are computed recursively by the functions from the matrix `mod`, observing that the probability of a 1-partition (partition built by one segment) of the trajectory from 1 to i described by the model m is simply the product of the probability densities of the steps from 1 to i under the model m.

Remark: this approach relies on the hypothesis of the independence of the steps within each segment.

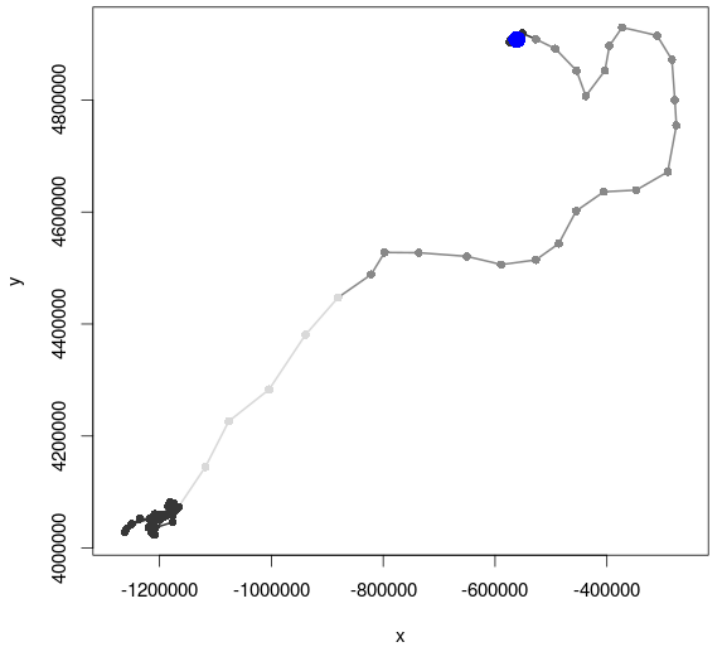
Now, use the function `partmod.ltraj` to partition the trajectory of the porpoise into 4 segments:

```
> (pm <- partmod.ltraj(gus, 4, mod))
```

```
Number of partitions: 4
Partition structure:
  relocation Num   Model
1          1 --- -------
2          |   2   mod.2
3          6 --- -------
4          |   5   mod.5
5         28 --- -------
6          |   8   mod.8
7         33 --- -------
8          |   2   mod.2
9         64 --- -------

The segments are contained in the component $ltraj of the list
```

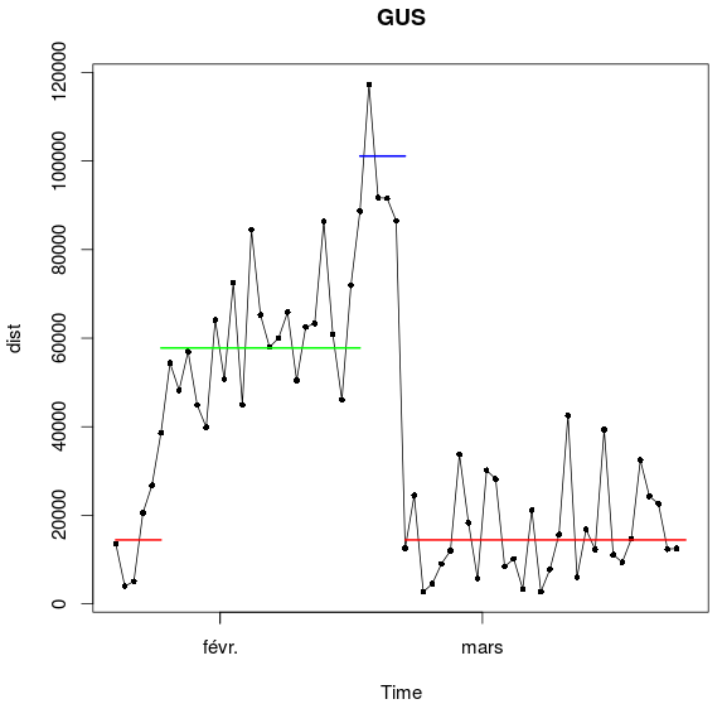
We can see that the models at the beginning of the trajectory and at the end of the trajectory are the same. Have a look at this segmentation:

```
> plot(pm)
```



This is very interesting: we already noted that the movements at the very beginning and the end of the trajectory were much slower that the rest of the trajectory, and this is confirmed by this partition. However, this segmentation illustrates a change of speed at the middle of the "migration". The end of the migration is much faster than the beginning. This is clearer on the graph showing the changes in distance between successive relocations with the date. Let us plot this graph together with the segmentation:

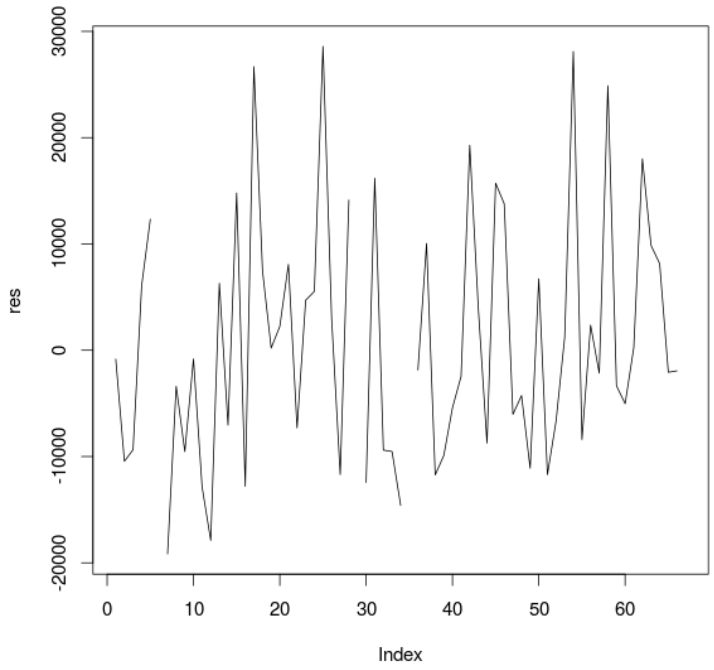
```
> ## Shows the segmentation on the distances:
> plotltr(gus, "dist")
> tmp <- lapply(1:length(pm$ltraj), function(i) {
+     coul <- c("red","green","blue")[as.numeric(factor(pm$stats$mod))[i]]
+     lines(pm$ltraj[[i]]$date, rep(tested.means[pm$stats$mod[i]],
+                                   nrow(pm$ltraj[[i]])),
+           col=coul, lwd=2)
+ })
```



The end of the migration is nearly two times faster than the beginning of the migration.

To conclude, have a look at the residuals of this segmentation:

```
> ## Computes the residuals
> res <- unlist(lapply(1:length(pm$ltraj), function(i) {
+     pm$ltraj[[i]]$dist - rep(tested.means[pm$stats$mod[i]],
+                              nrow(pm$ltraj[[i]]))
+ }))
> plot(res, ty = "l")
```



And a Wald and Wolfowitz test suggests that the residuals of this segmentation are independent, confirming the validity of the approach:

```
> wawotest(res)

4 NA removed
         a         ea         va         za          p
-5.4712718 -1.0000000 59.2571538 -0.5808456  0.7193277
```

4.5.2 The method of Lavielle (1999, 2005)

Barraquand and Benhamou (2008) proposed an approach to partition the trajectory, to identify the places where the animal stays a long time (based on the calculation of the residence time: see the help page of the function `residenceTime`). Once the residence time has been calculated for each relocation, they propose to use the method of Lavielle (1999, 2005) to partition the trajectory. We describe this method in this section.

The method of Lavielle *per se* finds the best segmentation of a time series, given that it is built by K segments. It searches the segmentation for which a contrast function (measuring the contrast between the actual series and the model underlying the segmented series) is minimized. Let Y_i be the value of

the focus variable (e.g. the speed, the residence time, etc.) at time i, and n the number of observations. We suppose that the data have been generated by the following model:

$$Y_i = \mu_i + \sigma_i \epsilon_i$$

where μ_i and σ_i are respectively the mean and the standard deviation of Y_i, and ϵ_i is a sequence of zero mean random variables with unit variance. *Note that it is not required that the ϵ_i are independent.* The model underlying the segmentation relies on the hypothesis that the parameters μ_i and σ_i are constant within a segment, but also that they vary between successive segments. Several contrast functions can be used to measure the discrepancy between the observed series and a given segmentation model, depending on assumptions on this variation:

- we can suppose *a priori* that only the mean μ_i changes between segments, and that $\sigma_i = \sigma$ is the same for all segments. Then, for a given partition of the series built by K segments with known limits, the following function can be used to measure the discrepancy between the observed series and the model supposing a mean constant within segments and changing from one segment to the next:

  $$J_K(Y) = \sum_k G_k^1(Y_{i,i\in k})$$

  where $Y_{i,i\in k}$ is the set of observations in the segment k, and

  $$G_k^1(Y_{i,i\in k}) = \sum_{i=t_1^k}^{t_{n(k)}^k} (Y_i - \bar{Y}_k)^2$$

  with t_1^k and $t_{n(k)}^k$ the indices of the first and last observations of segment k respectively, and $\bar{Y}_k$ the sample mean of the observations in the segment k;

- we can suppose *a priori* that only the standard deviation σ_i changes between segments, and that $\mu_i = \mu$ is the same for all segments. Then, for a given partition of the series built by K segments with known limits, the following function can be used to measure the discrepancy between the observed series and the model supposing a variance constant within segments and changing from one segment to the next:

  $$J_K(Y) = \sum_k G_k^2(Y_{i,i\in k})$$

  where

  $$G_k^2(Y_{i,i\in k}) = \frac{1}{n(k)} \log \left(\frac{1}{n(k)} \sum_{i=t_1^k}^{t_{n(k)}^k} (Y_i - \bar{Y})^2 \right)$$

with $n(k)$ the number of observations in the segment k and $\bar{Y}$ the mean of the whole series;

- finally, we can suppose *a priori* that both the standard deviation σ_i and the mean μ_i change between segments. Then, for a given partition of the series built by K segments with known limits, the following function can be used to measure the discrepancy between the observed series and the model supposing a mean and variance constant within segments but changing from one segment to the next:

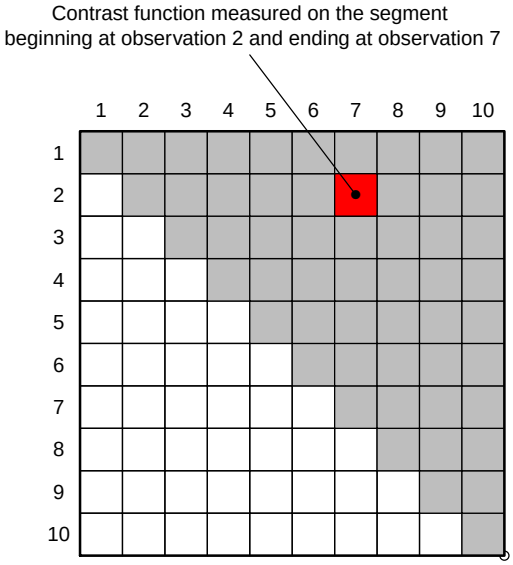
$$J_k(Y) = \sum_k G_k^3(Y_{i,i\in k})$$

where

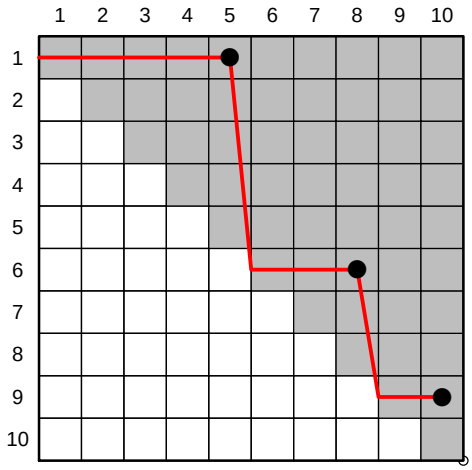
$$G_k^3(Y_{i,i\in k}) = \frac{1}{n(k)} \log \left(\frac{1}{n(k)} \sum_{i=t_1^k}^{t_{n(k)}^k} (Y_i - \bar{Y}_k)^2 \right)$$

The three contrast functions can be used to measure the contrast between a given series and the model underlying a given segmentation of this series. *For a given number of segments* K, the method of Lavielle consists in the search of the best segmentation of the series, i.e. the segmentation for which the chosen contrast function is minimized.

For example, consider a series of 10 observations. First, we calculate an upper triangular 10×10 matrix containing the value of the contrast function for all possible segments built by the successive observations of the series. This matrix is called the contrast matrix:



Given a number K of segments, the Lavielle method consists in the search of the best path from the first to the last observation in this matrix. For example, consider the following path:



This segmentation is built by 3 segments:

- the first segment begins at the first observation and ends at the fifth observation. The value of the contrast function for this segment is at the intersection between the first row and the fifth column.
- the second segment begins at the sixth observation and ends at the eighth observation. The value of the contrast function for this segment is at the intersection between the sixth row and the eighth column.
- the last segment begins at the ninth observation and ends at the tenth observation. The value of the contrast function for this segment is at the intersection between the ninth row and the tenth column.

The value of the contrast function for the whole segmentation is the sum of the values located at the dots. For a given value of the number of segments (in this example $K = 3$), a dynamic programming algorithm allows to identify the best segmentation, i.e. the path through the matrix for which the contrast function is minimized.

Remark: in practice, a minimum number of observations L_{min} in the segments should be specified. For example, if $L_{min} = 3$, no segment will contain less than

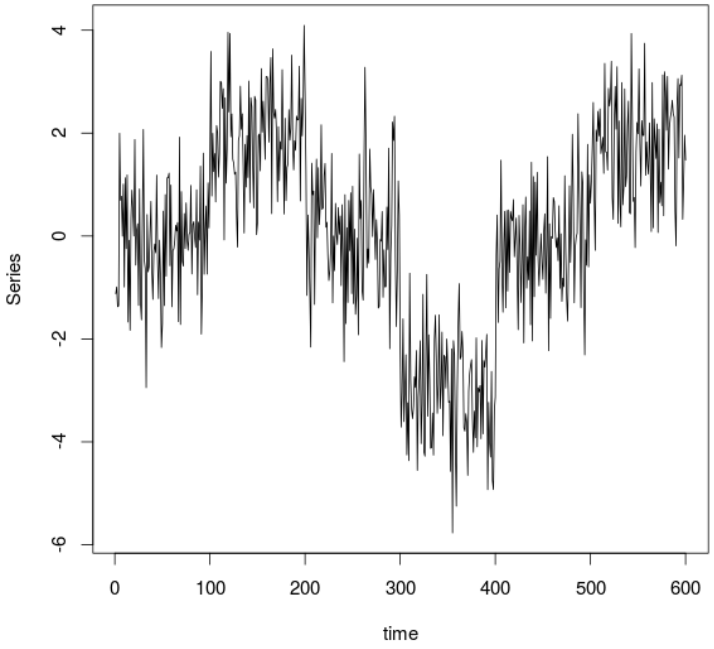
3 observations.

Now, remains the question of the choice of the optimal number of segments K_{opt}. Several approaches have been proposed to guide the choice of K_{opt}. The approaches proposed by Lavielle (2005) rely on the examination of the decrease of the contrast function $J(K)$ with the number of segments K. Indeed, the more there are segments, and the smaller the contrast function will be (because a segmentation built by more segments will fit more closely to the actual series). However, there should be a clear "break" in the decrease of this function after the optimal value K_{opt}. Therefore, the number of segments can be chosen graphically, based on the examination of the decrease of this contrast function J(K) with K. Lavielle (2005) also suggested an alternative way to estimate automatically the optimal number of segments, also relying on the presence of a "break" in the decrease of the contrast function. He proposed to choose the last value of K for which the second derivative $D(K)$ of a *standardized* constrast function is greater than a threshold S (see Lavielle, 2005 for details). Based on numerical experiments, he proposed to choose the value $S = 0.75$.

Note that the standardization of the contrast function is based on the specification of a value of K_{max}, the maximum number of segments expected by the user (see Lavielle 2005 for details). Note that the value of K_{max} chosen by the user may have a strong effect on the resulting value of K_{opt}, in particular for small time series (i.e. less than 500 observations). Barraquand (com. pers) noted from simulations that values of K_{max} set to about 3 or 4 times the expected value of K_{opt} seem to give the best results.

The Lavielle method is implemented in the function `lavielle`. For example, consider the following time series, built by 6 segments of 100 observations each, drawn from a normal distribution with a mean varying from one segment to the next (0, 2, 0, -3, 0, 2):

```
> set.seed(129)
> seri <- c(rnorm(100), rnorm(100, mean=2),
+           rnorm(100), rnorm(100, mean=-3),
+           rnorm(100), rnorm(100, mean=2))
> plot(seri, ty="l", xlab="time", ylab="Series")
```



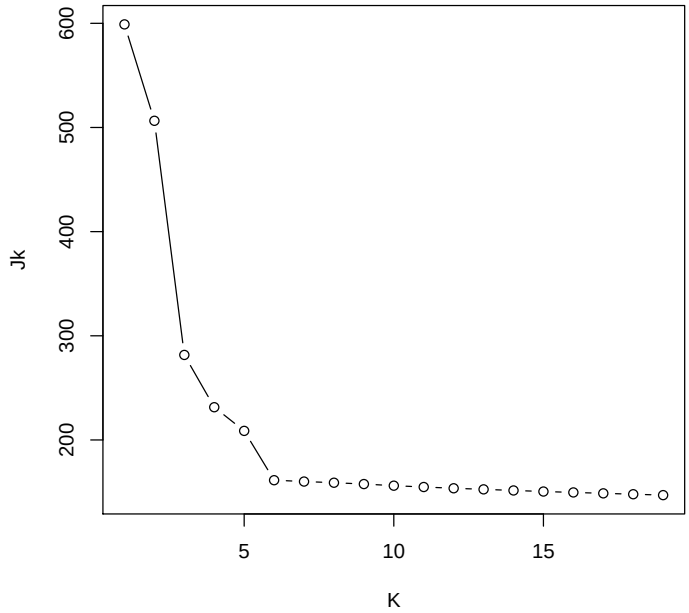
We will use the Lavielle method to see whether we can identify the limits used to generate this series. We can use the function `lavielle` to calculate the contrast matrix:

```
> (l <- lavielle(seri, Lmin=10, Kmax=20, type="mean"))

Segmentation of a series using the method of Lavielle
 $contmat: contrast matrix
 $sumcont: optimal contrast
 $matpath: matrix of the path
 $Kmax:    maximum number of segments
 $Lmin:    Minimum number of obs. to build a segment
 $ld:      The size of the subsampling grid
 $series:  the series
```

Note that we indicate that each segment should contain at least 10 observations (`Lmin=10`) and that the whole series is built by at most 20 segments (`Kmax = 20`). The result of the function is a list of class `"lavielle"` containing various results, but the most interesting one here is the contrast matrix. We can have a look at the decrease of the contrast function when the number K of segments increase, with the function `chooseseg`:

```
> chooseseg(l)
```

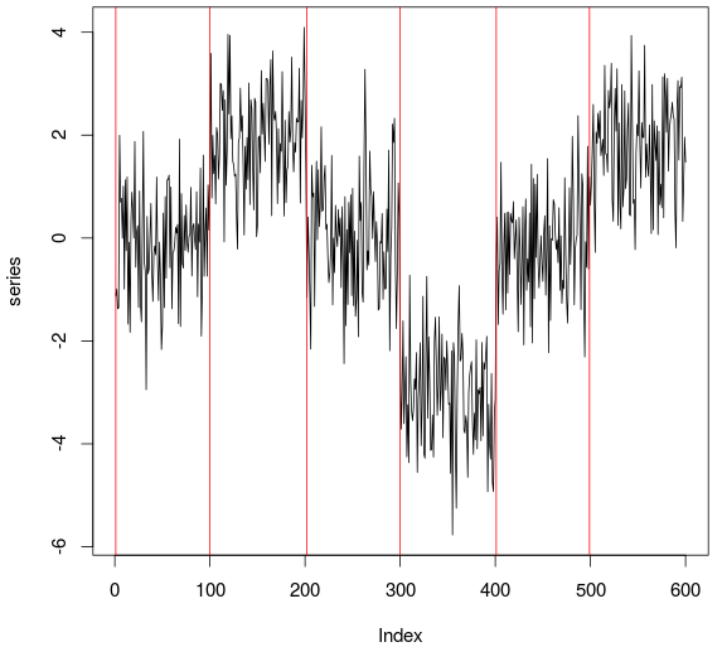
```
    K             D       Jk
1   1           Inf 599.0000
2   2 -5.5485861559 506.4326
3   3  7.3267868763 281.6300
4   4  1.1530621869 231.4410
5   5 -1.0341982627 208.7321
6   6  1.9327670882 161.3760
7   7  0.0074135521 160.0819
8   8 -0.0074135521 158.9646
9   9 -0.0107869455 157.6706
10 10  0.0107869455 156.1195
11 11  0.0040387117 154.8255
12 12  0.0064045526 153.6277
13 13 -0.0002814105 152.5826
14 14  0.0010609576 151.5307
15 15  0.0063271203 150.5042
16 16  0.0003941470 149.6284
17 17  0.0004747425 148.7620
18 18 -0.0004747425 147.9070
19 19  0.0005643351 147.0406
```



This function returns: (i) the value of the contrast function `Jk` for various values of K, (ii) the value of the second derivative of the contrast function `D`

for various values of K, and (iii) a graph showing the decrease of `Jk` with K. The slope of the contrast function is strongly negative when $K < 6$, but there is a sharp break at $K = 6$. Visually, the value of $K_{opt} = 6$ seems therefore reasonable. Moreover, the last value of K for which $D > 0.75$ is $K = 6$, which confirms our first visual choice (the second derivative is actually very strong for this value of K). We can have a look at the best segmentation with 6 segments with the function `findpath`:

```
> fp <- findpath(l, 6)
```



The Lavielle method does a pretty good job in this ideal situation. Note that `fp` is a list containing the indices of the first and last observations of the series building the segment:

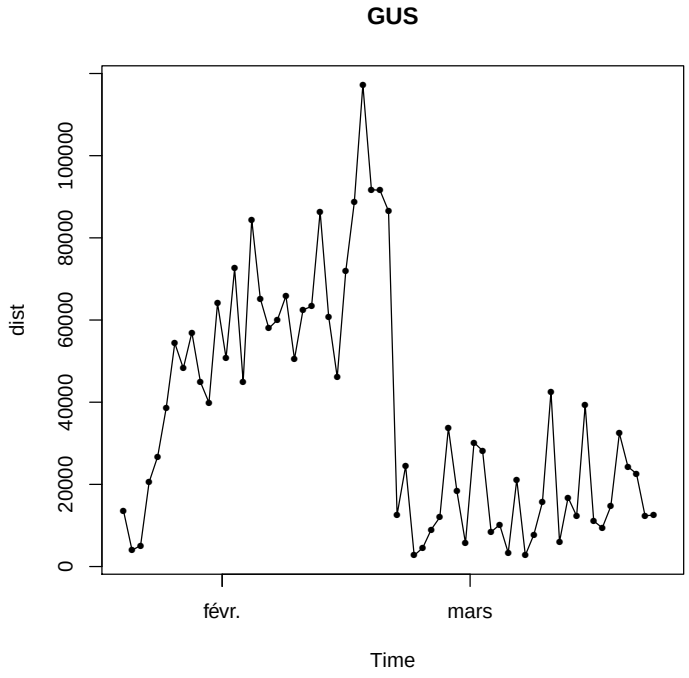
```
> fp

[[1]]
[1]   1 99

[[2]]
[1] 100 201
```

```
[[3]]
[1] 202 299

[[4]]
[1] 300 400

[[5]]
[1] 401 498

[[6]]
[1] 499 600
```

The limits found by the method are very close to the actual limits here.

The function `lavielle` is a generic function that has a method for objects of class `"ltraj"`. For example, consider again the porpoise described in the previous section. We will perform a segmentation of this trajectory based on the distances travelled by the animal from one relocation to the next.

```
> plotltr(gus, "dist")
```

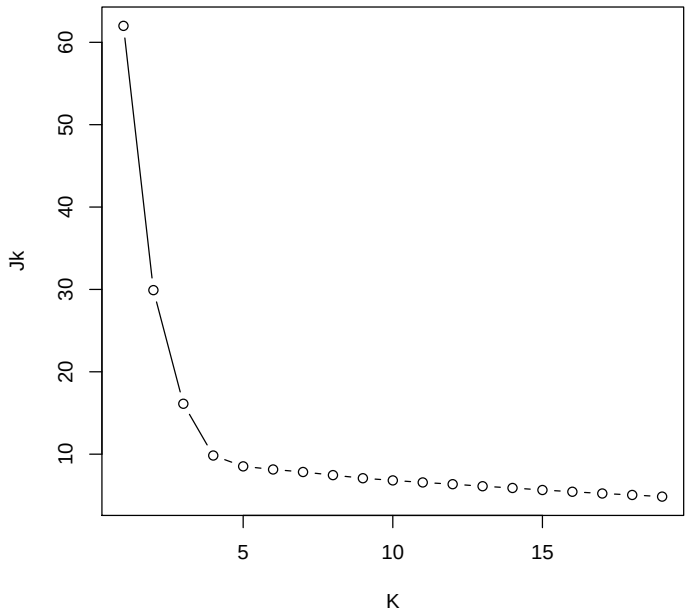


We can use the function `lavielle` to partition this trajectory into segments with homogeneous speed. A priori, based on a visual examination, there is no

reason to expect that this series is built by more than 20 segments. We will use the Lavielle method, with $K_{max} = 20$ and $L_{min} = 2$ (we suppose that there are at least 2 observations in a segment):

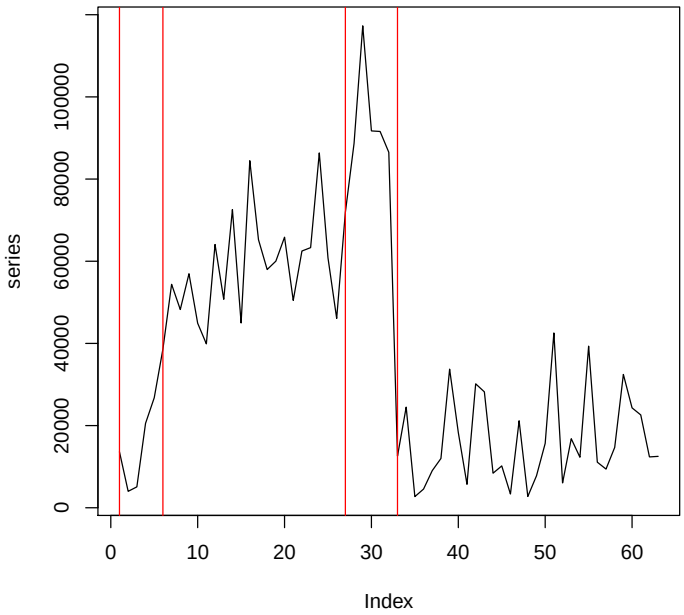
```
> lav <- lavielle(gus, Lmin=2, Kmax=20)
> chooseseg(lav)
```

```
    K             D        Jk
1   1           Inf 62.000000
2   2  6.0646620289 29.916691
3   3  2.4909974271 16.121554
4   4  1.6466711405  9.838095
5   5  0.3122344015  8.520223
6   6  0.0230477679  8.143903
7   7 -0.0242022033  7.837084
8   8  0.0011544354  7.457283
9   9  0.0371514816  7.080963
10 10  0.0056043674  6.816674
11 11  0.0116307321  6.569285
12 12 -0.0115498156  6.356970
13 13  0.0115498156  6.109825
14 14 -0.0113274369  5.897509
15 15  0.0113274369  5.651035
16 16  0.0006966657  5.438720
17 17  0.0097325552  5.228505
18 18 -0.0061039466  5.047639
19 19  0.0185150717  4.848366
```



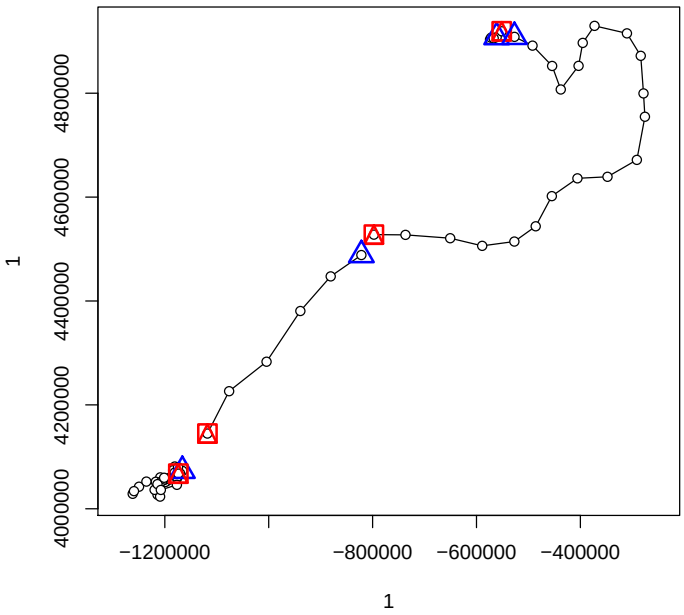
There is a clear break in the decrease of the contrast function after $K = 4$. In addition, this is also the last value of K for which $D > 0.75$. Look at the segmentation of the series of distances:

```
> kk <- findpath(lav, 4)
> kk

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: UTC *
Regular traject. Time lag between two locs: 86400 seconds

Characteristics of the bursts:
   id     burst nb.reloc NAs          date.begin            date.end
1 GUS Segment.1        5   0 2004-01-20 11:00:00 2004-01-24 11:00:00
2 GUS Segment.2       21   0 2004-01-25 11:00:00 2004-02-14 11:00:00
3 GUS Segment.3        6   0 2004-02-15 11:00:00 2004-02-20 11:00:00
4 GUS Segment.4       32   0 2004-02-21 11:00:00 2004-03-23 11:00:00
```



Note that the red lines indicate the beginning of new segments. The function `findpath` here returns an object of class `"ltraj"` with one burst per segment:

```
> plot(kk)
```



Note that this segmentation is nearly identical to the one found with the method of Gueguen (2001).

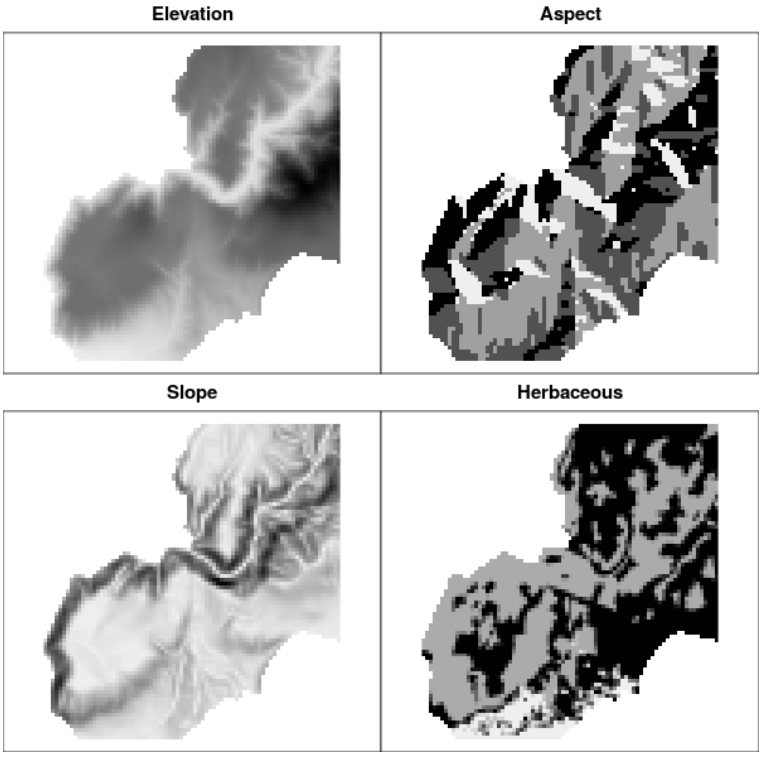
Important Remark: The Lavielle method implies the calculation of a contrast matrix with dimensions equal to $n \times n$, where n is the number of observations in the series. However, as n increases, the size of the contrast matrix may exceed the memory limit of the computer. In this case, a possibility consists in the calculation of the contrast function along a grid of size l_d. Therefore the contrast function will not be calculated for all possible segments in the series, but for all possible segments containing multiple of l_d observations. For example, if $l_d = 3$, possible segments beginning by observation 1 are (1,2,3), (1, 2, ..., 6), (1, 2, ..., 9), (1, 2, ..., 12), etc. Therefore, l_d defines the resolution of the segmentation: the segment limits can only occur for observations located at indices multiple of l_d. This sub-sampling approach is possible with the function `lavielle`, by setting the parameter `ld` to a value greater than 1. *Note that in this case,* ***`Lmin`*** *should necessarily be a multiple of* ***`ld`*** (the function fails otherwise).

Remark: The Lavielle method was originally propose to partition a trajectory based on the residence time of the animal. The residence time method is implemented in the function `residenceTime`. The use of this function will add a new variable in the component `"infolocs"` of the object of class `"ltraj"` that can be used with the Lavielle method (see the help page of this function).

4.6 Rasterizing a trajectory

In some cases, it may be useful to rasterize a trajectory. In particular, when the aim of the study is to examine the habitat traversed by the animal, this approach may be useful. For example, consider the dataset `puechcirc`, containing 3 trajectories of 2 wild boars. It may be useful to identify the habitat traversed by the animal during each step. A habitat map is available in the dataset `puechabonsp`:

```
> data(puechcirc)
> data(puechabonsp)
> mimage(puechabonsp$map)
```



We can rasterize the trajectories of the wild boars:

```
> ii <- rasterize.ltraj(puechcirc, puechabonsp$map)
```

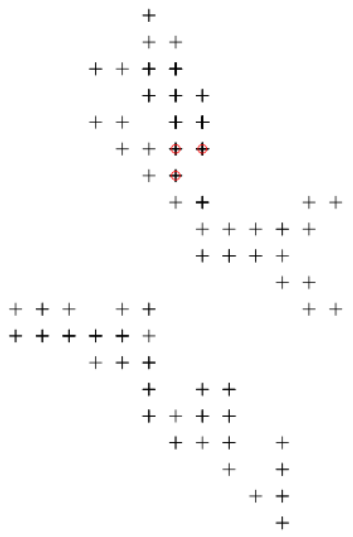
The result is a list containing 3 objects of class "SpatialPointsDataFrame" (one per animal). Let us examine the first one:

```
> tr1 <- ii[[1]]
> head(tr1)
```

```
         coordinates step
5  (700300, 3158400)    3
6  (700200, 3158400)    3
7  (700200, 3158300)    3
8  (700200, 3158300)    4
9  (700200, 3158400)    4
10 (700300, 3158400)    4
Coordinate Reference System (CRS) arguments: NA
```

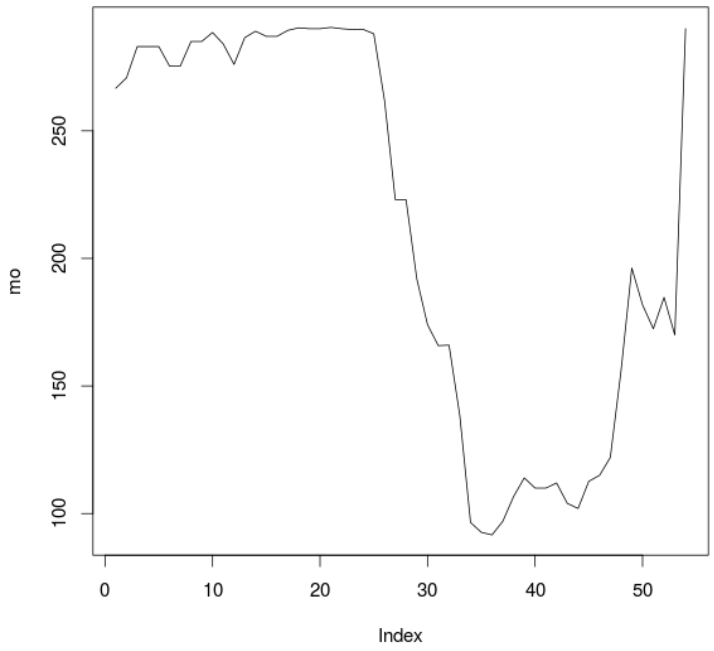
This data frame contains the coordinates of the pixels traversed by each step. For example, the rasterized trajectory for the first animal is:

```
> plot(tr1)
> points(tr1[tr1[[1]]==3,], col="red")
```



The red points indicate the pixels traversed by the third step. These results can be used to identify the habitat characteristics of each step. For example, we may calculate the mean elevation for each step. To proceed, we use the function `over` of the package sp.

```
> ov <- over(tr1, puechabonsp$map)
> mo <- tapply(ov[[1]], tr1[[1]], mean)
> plot(mo, ty="l")
```



Here, we can see that the first animal stays on the plateau at the beginning at the monitoring, then goes down to the crops, and goes back to the plateau. It is easy to repeat the operation for all the animals. We will make use of the `infolocs` attribute. We first build a list containing data frames, each data frame containing on variable describing the mean elevation traversed by the animal between relocation $i-1$ and relocation i:

```
> val <- lapply(1:length(ii), function(i) {
+
+     ## get the rasterized trajectory
+     tr <- ii[[i]]
+
+     ## over with the map
+     ov <- over(tr, puechabonsp$map)
+
+     ## calculate the mean elevation
+     mo <- tapply(ov[[1]], tr[[1]], mean)
+
+     ## prepare the output
+     elev <- rep(NA, nrow(puechcirc[[i]]))
+
+     ## place the average values at the right place
```

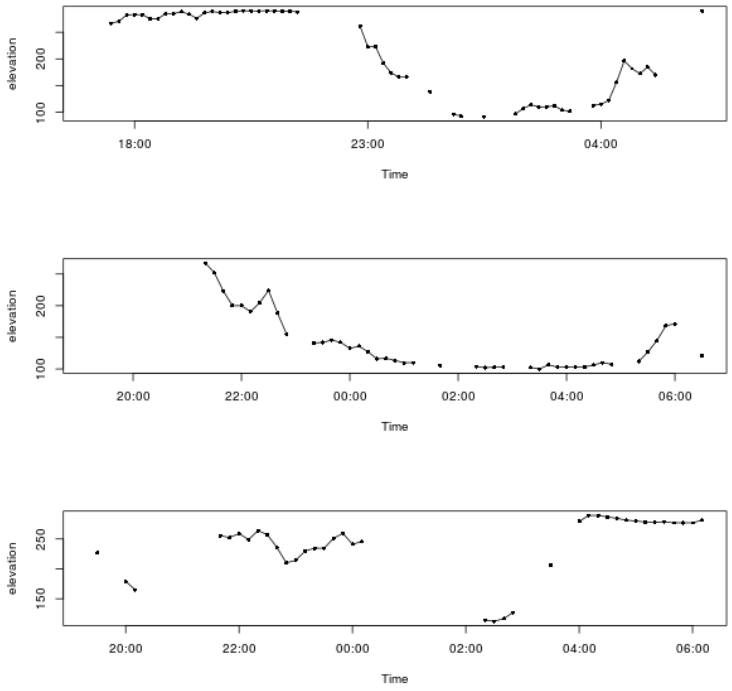
```
+     ## names(mo) contains the step number (i.e. relocation
+     ## number +1)
+     elev[as.numeric(names(mo))+1] <- mo
+     df <- data.frame(elevation = elev)
+
+     ## same row.names as in the ltraj
+     row.names(df) <- row.names(puechcirc[[i]])
+
+     return(df)
+ })
>
```

Then, we define the `infolocs` attribute:

```
> infolocs(puechcirc) <- val
```

and finally, we can plot the mean elevation as a function of date:

```
> plotltr(puechcirc, "elevation")
```



4.7 Models of animal movements

Several movement models have been proposed in the litteratured to describe animal movements. The package `adehabitatLT` contains several functions al-

lowing to simulate these models. Such simulations can be very useful to test hypotheses concerning a trajectory, because all the descriptive parameters of the steps are also generated by the functions. Actually, the package proposes 6 functions to simulate such models:

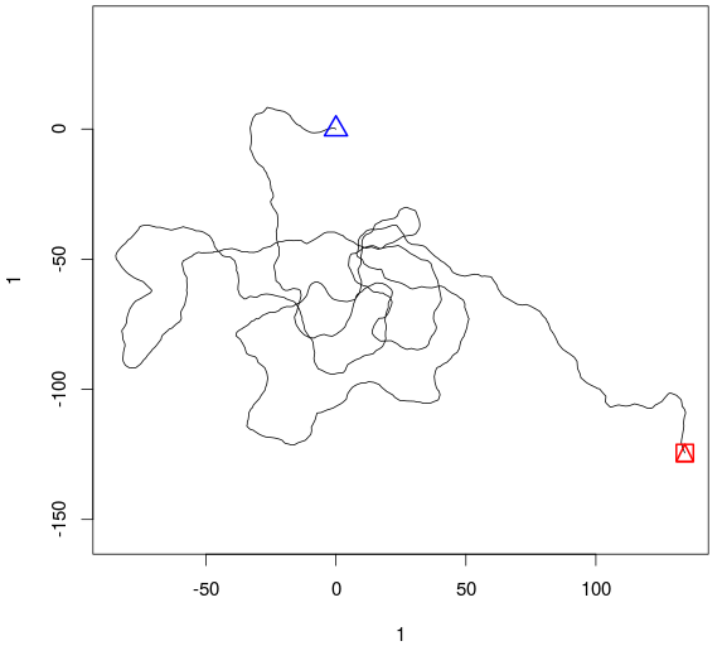
- `simm.brown` can be used to simulate a Brownian motion;
- `simm.crw` can be used to simulate a correlated random walk. This model has been often used to describe animal movements (Kareiva and Shigesada 1983);
- `simm.mba` can be used to simulate an arithmetic Brownian motion (with a drift parameter and a covariance between the coordinates, see Brillinger et al. 2002);
- `simm.bb` can be used to simulate a Brownian bridge motion (i.e. a Brownian motion constrained by a fixed start and end point);
- `simm.mou` can be used to simulate a bivariate Ornstein-Uhlenbeck motion (often used to describe the sedentarity of an animal, e.g. Dunn and Gipson 1977);
- `simm.levy` can be used to simulate a Levy walk, as described (Bartumeus et al. 2005).

All these functions return an object of class `ltraj`. For example, simulate a correlated random walk built by 1000 steps characterized by a mean cosine of the relative angles equal to 0.95 and a scale parameter for the step length equal to 1 (see the help page of `simm.crw` for additional details on the meaning of these parameters):

```
> sim <- simm.crw(1:1000, r=0.95)
> sim

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone unspecified: dates printed in user time zone *
Regular traject. Time lag between two locs: 1 seconds

Characteristics of the bursts:
  id burst nb.reloc NAs          date.begin            date.end
1 A1    A1     1000   0 1970-01-01 01:00:01 1970-01-01 01:16:40


 infolocs provided. The following variables are available:
[1] "pkey"
```

Note that the vector `1:1000` passes as an argument is considered here as a vector of dates (it is converted to the class `POSIXct` by the function, see section 2.4 for more details on this class). Other dates can be passed to the functions. Have a look at the simulated trajectory:

```
> plot(sim, addp=FALSE)
```



5 Null models of animal movements

5.1 What is a null model?

The package `adehabitatLT` provides several functions to perform a null model analysis of trajectories. Null models are frequently used in community ecology to test hypotheses about the processes that generated the data. Gotelli and Graves (1996) defined the null model as "*a pattern generating model that is based on randomization of ecological data or random sampling from a known or imagined distribution. The null model is designed with respect to some ecological or evolutionary process of interest. Certain elements of the data are held constant, and others are allowed to vary stochastically to create new assemblage patterns. The randomization is designed to produce a pattern that would be expected in*

the absence of a particular ecological mechanism".

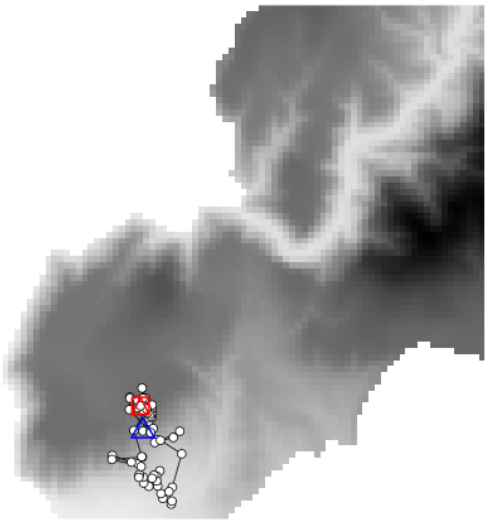
Gotelli (2001) gave a more detailed description of the null model approach: "*to build a null model, an index of community structure, such as the amount of niche overlap (...), is first measured for the real data. Next, a null community is generated according to an algorithm or set of rules for randomization, and this same index is measured for the null community. A large number of null communities are used to generate a frequency histogram of index values expected if the null hypothesis is true. The position of the observed index in the tails of this null distribution is then used to assign a probability value to the pattern, just as in a conventional statistical analysis.*

Although mostly used in community ecology, this approach was also advocated for trajectory data, e.g. in the analysis of habitat selection (Martin et al. 2008) and the study of the interactions between animals (Richard et al. 2012). This approach can indeed be very useful to test hypotheses related to animal movements. The package adehabitatLT propose several general null models that can be used to test biological hypotheses.

5.2 The problem

For example, consider the first animal in the object `puechcirc` loaded previously. We plot this trajectory on an elevation map:

```
> data(puechcirc)
> data(puechabonsp)
> boar1 <- puechcirc[1]
> xo <- coordinates(puechabonsp$map)
> ## Note that xo is a matrix containing the coordinates of the
> ## pixels of the elevation map (we will use it later to define
> ## the limits of the study area).
>
> plot(boar1, spixdf=puechabonsp$map, xlim=range(xo[,1]), ylim=range(xo[,2]))
```



At first sight, it seems that the animal occupies a large range of values of elevation. We may want to test whether the variance of elevation values at animal relocations could have been observed under the hypothesis of random habitat use. The question is therefore: what do we mean by "random habitat use"? We can precise our aim here. We may consider the animal as a random sample of all animals living on the study area. If we focus on the second scale of habitat selection as defined by Johnson (1980), i.e. on the selection of the home range in the study area, under the hypothesis of random habitat use, the observed trajectory could have been located anywhere on the study area.

Therefore, if we want to know if the actual elevation variance could have been observed under the hypothesis of random habitat use, one possibility is to simulate this kind of random habitat use a large number of times, and to calculate the elevation variance for each simulated data set. If the observed value is far from the distribution of simulated values, this would allow to rule out the null model.

One possibility to simulate this kind of "random habitat use" for this animal could be to randomly rotate and shift the observed trajectory over the study area. Rotating and shifting the trajectory as a whole allows to keep the trajectory structure unchanged (therefore taking into account the internal constraints

of the animal, see Martin et al. 2008). The function `NMs.randomShiftRotation` from the package `adehabitatLT` allows to define this type of null model (but other null models are also available in `adehabitatLT`, see section 5.7), and the function `testNM` allows to simulate it.

5.3 The principle of the approach

The basic principle of the null model approach implemented in the package is the following:

- first write a treatment function that will be used to calculate a criterion characterizing the process under study from the simulated data (see below for how to write such a function);

- then choose a null model and define the constraints to be satisfied by simulated datasets. Define this null model with one function `NMs.*` (see section 5.7 or the help page of `testNM` for a list of the available null models), and simulate N random data sets generated by this model using the function `testNM`;

- finally compare the observed criterion with the distribution of simulated values to determine whether the observed data set could have been generated by the null model.

The function `testNM`, used to simulate the model, generates at each step of the simulation process a data frame `x` with three columns: the coordinates x, y, and the date (whatever the chosen type of null model). The treatment function defines how to calculate the criterion used in the test from this data frame. The treatment function can be any function defined by the user, but **must** take arguments `x` and `par`, where:

- `x` is the data frame generated by the null model;

- `par` can be any R object (e.g. a list) containing the parameters required by the treatment function. Note that this argument is needed even if no parameters are required by the treatment function. In this case, the argument `par` will not be used by the function, but must be present in the function definition).

We will define a treatment function that calculate the elevation variance from a simulated trajectory later. For the moment, we will define a treatment function that just plot the simulated trajectories on an elevation map. Consider the following function:

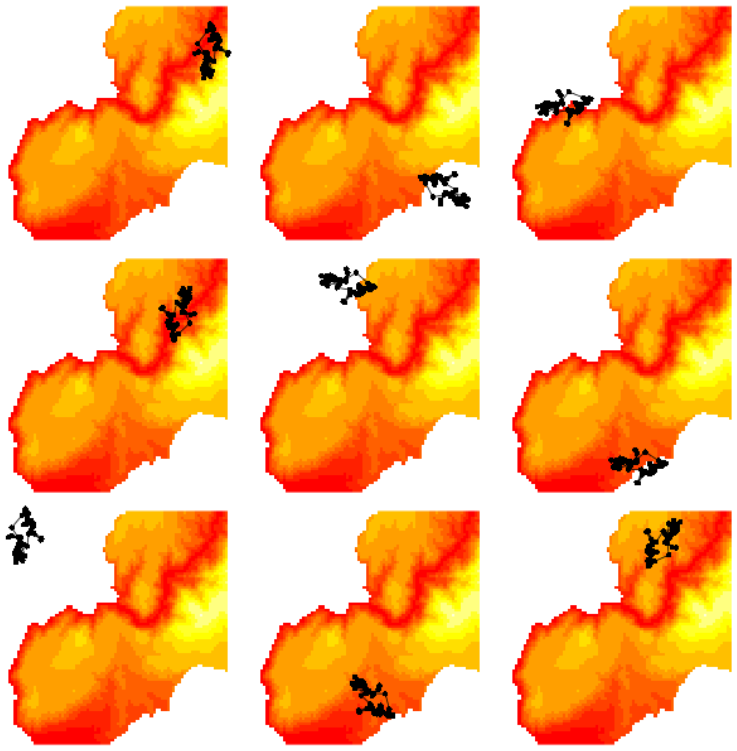
```
> plotfun <- function(x, par)
+ {
+     image(par)
+     points(x[,1:2], pch=16)
+     lines(x[,1:2])
+     return(x)
+ }
```

This function will be used to plot the simulations of the null model. In this case, the argument `par` correspond to a map of the study area. Note that this function also returns the data frame `x`. Now, define the following null model:

```
> nmo <- NMs.randomShiftRotation(na.omit(boar1), rshift = TRUE, rrot = TRUE,
+                                rx = range(xo[,1]), ry = range(xo[,2]),
+                                treatment.func = plotfun,
+                                treatment.par = puechabonsp$map[,1], nrep=9)
> nmo

**********************************
Null Model object of type randomShiftRotation  (single)
9  repetitions will be carried out
Please consider the function testNM() for the simulations
```

We have removed the missing values from the object of class `ltraj` (this function does not accept missing values in the trajectory). The arguments `rshift` and `rrot` indicate that we want to randomly rotate the trajectory and shift it over the study area. The study area is necessarily a rectangle defined by its x and y limits `rx` and `ry`. We indicate that the treatment function is the function `plotfun` that we just wrote, and that the argument `par` that will be passed to the treatment function (i.e. `treatment.par`) is the elevation map. We only define 9 repetitions of the null model. Now, we simulate the model using the function `testNM`:

```
> set.seed(90909)
> par(mfrow=c(3,3), mar=c(0,0,0,0))
> resu <- testNM(nmo, count = FALSE)
```



This figure illustrates the datasets generated by the null model. We can see that each data set corresponds to the original trajectory after a random rotation and shift over the study area. First, we can see a problem: some of the generated trajectories are located outside the study area (because the study area is necessarily defined as a rectangle in this type of null model). It here necessary to define a *constraint function* when defining the null model. Like the treatment function, the constraint function takes two arguments `x` and `par`, and *must return a logical value* indicating whether the constraint(s) are satisfied or not. In our example, we would like that all the relocations building the trajectory fall within the study area. In other words, if we overlap spatially the relocations and the elevation map, there should be no missing value. Define the following constraint function:

```
> confun <- function(x, par)
+ {
+     ## Define a SpatialPointsDataFrame from the trajectory
+     coordinates(x) <- x[,1:2]
+     ## overlap the relocations x to the elevation map par
+     jo <- join(x, par)
+     ## checks that there are no missing value
+     res <- all(!is.na(jo))
+     ## return this check
```

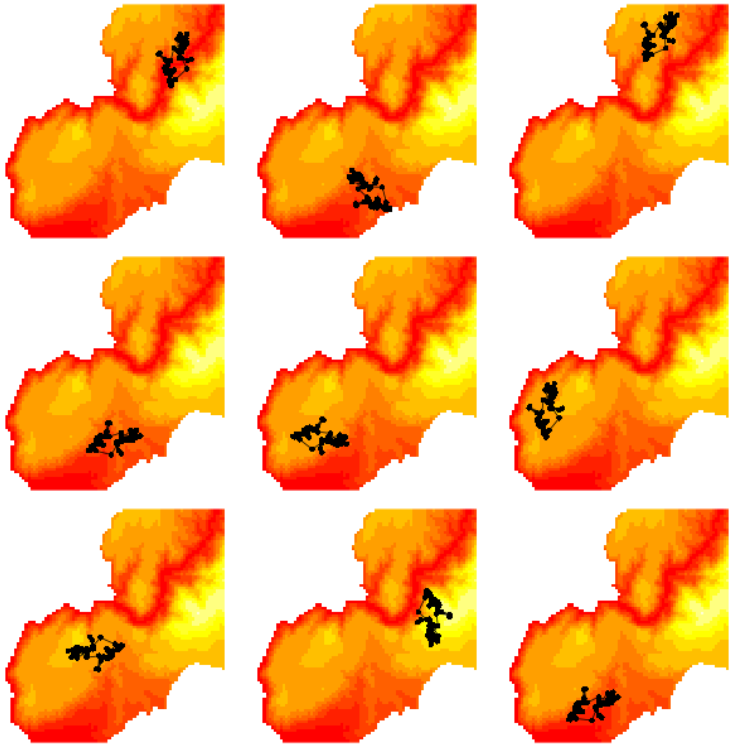
```
+     return(res)
+ }
```

Now, define again the null model, but also define the constraint function:

```
> nmo2 <- NMs.randomShiftRotation(na.omit(boar1), rshift = TRUE, rrot = TRUE,
+                                 rx = range(xo[,1]), ry = range(xo[,2]),
+                                 treatment.func = plotfun,
+                                 treatment.par = puechabonsp$map[,1],
+                                 constraint.func = confun,
+                                 constraint.par = puechabonsp$map[,1],
+                                 nrep=9)
```

Now, if we simulate the null model, only those data sets satisfying the constraint will be returned by the function:

```
> set.seed(90909)
> par(mfrow=c(3,3), mar=c(0,0,0,0))
> resu <- testNM(nmo2, count = FALSE)
```



5.4 Back to the problem

Now consider again our problem: is the variance of the elevation values at the relocations greater than expected under the hypothesis of random use of space?

We can write a treatment function that calculates the variance of elevation values from a data frame `x` containing the relocations and a SpatialPixelsDataFrame `par` containing the elevation map:

```
> varel <- function(x, par)
+ {
+     coordinates(x) <- x[,1:2]
+     jo <- join(x, par)
+     return(var(jo))
+ }
```

We can define again a null model to calculate a distribution of values of variance expected under the null model. We use the function `varel` as treatment function in the null model. To save some time, we calculate only 99 values under this null model, but the user is encouraged to try the function with a larger value:

```
> nmo3 <- NMs.randomShiftRotation(na.omit(boar1), rshift = TRUE, rrot = TRUE,
+                                 rx = range(xo[,1]), ry = range(xo[,2]),
+                                 treatment.func = varel,
+                                 treatment.par = puechabonsp$map[,1],
+                                 constraint.func = confun,
+                                 constraint.par = puechabonsp$map[,1],
+                                 nrep=99)
```

Note that we define the same constraint function as before (all relocations should be located inside the study area. We now simulate the null model:

```
> sim <- testNM(nmo3, count=FALSE)
```

Now, calculate the observed value of variance:

```
> (obs <- varel(na.omit(boar1)[[1]], puechabonsp$map[,1]))
```

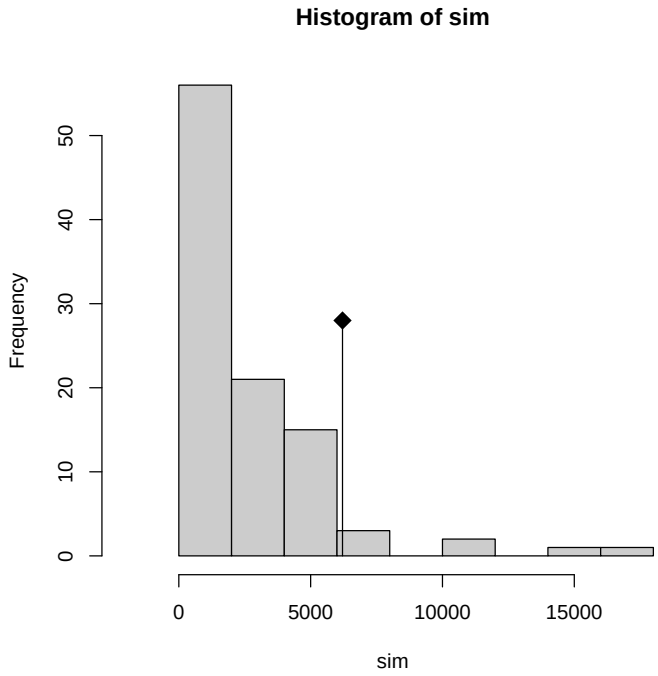
```
[1] 6208.397
```

And compare this observation with the distribution obtained under the null model, using the function `as.randtest` from the package ade4:

```
> (ran <- as.randtest(unlist(sim[[1]]), obs))
```

```
Monte-Carlo test
Call: as.randtest(sim = unlist(sim[[1]]), obs = obs)

Observation: 6208.397

Based on 99 replicates
Simulated p-value: 0.07
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
1.222352e+00 2.669105e+03 8.383787e+06
```

```
> plot(ran)
```



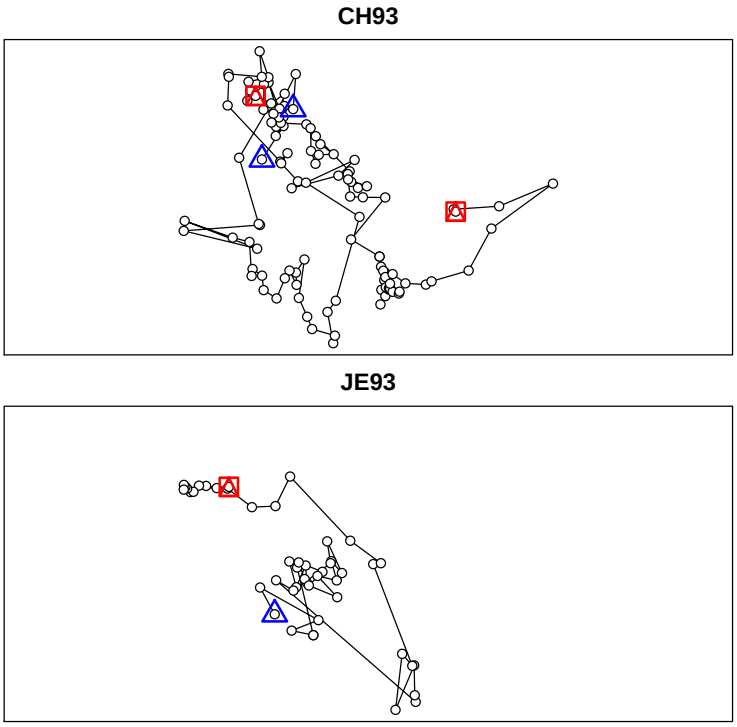
The P-value is rather low, which seems to indicate that the range of elevations used by the boar is important in comparison to what would be expected under the hypothesis of random habitat use.

5.5 Null model with several animals

Now, consider again the data set `puechcirc`. This data set contains three trajectories of two wild boars:

```
> puechcirc

********** List of class ltraj **********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Regular traject. Time lag between two locs: 600 seconds

Characteristics of the bursts:
    id    burst nb.reloc NAs          date.begin            date.end
1 CH93 CH930824       80  16 1993-08-24 17:00:00 1993-08-25 06:10:00
2 CH93 CH930827       69  17 1993-08-27 19:10:00 1993-08-28 06:30:00
3 JE93 JE930827       66  19 1993-08-27 19:20:00 1993-08-28 06:10:00
```

```
> plot(puechcirc)
```



Note that the two trajectories of CH93 are roughly located at the same place on the study area. We decide to bind these two trajectories into one:

```
> (boar <- bindltraj(puechcirc))

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
    id burst nb.reloc NAs          date.begin            date.end
1 CH93  CH93      149  33 1993-08-24 17:00:00 1993-08-28 06:30:00
2 JE93  JE93       66  19 1993-08-27 19:20:00 1993-08-28 06:10:00


 infolocs provided. The following variables are available:
[1] "pkey"
```

Now, we can reproduce the null model analysis separately for each animal. When the object of class `ltraj` passed as argument contains several trajectories,

the simulations are performed separately for each one. Therefore, to define the null model for all animals in `boar`, we can use the same command line as before, just replacing `boar1` by `boar`:

```
> nmo4 <- NMs.randomShiftRotation(na.omit(boar), rshift = TRUE, rrot = TRUE,
+                                 rx = range(xo[,1]), ry = range(xo[,2]),
+                                 treatment.func = varel,
+                                 treatment.par = puechabonsp$map[,1],
+                                 constraint.func = confun,
+                                 constraint.par = puechabonsp$map[,1],
+                                 nrep=99)
```

We now simulate this null model with the function `testNM`:

```
> sim2 <- testNM(nmo4, count=FALSE)
```

`sim2` is a list with two components (one per trajectory), each component being itself a list with `nrep=99` elements (the elevation variance calculated for each of the 99 datasets generated by the null model). We can calculate the observed elevation variance for each observed trajectory:

```
> (obs <- lapply(na.omit(boar), function(x) {
+   varel(x, puechabonsp$map[,1])
+ }))

[[1]]
[1] 5661.974

[[2]]
[1] 3265.768
```

And calculate a P-value for each animal separately:

```
> lapply(1:2, function(i) {
+   as.randtest(unlist(sim2[[i]]), obs[[i]])
+ })

[[1]]
Monte-Carlo test
Call: as.randtest(sim = unlist(sim2[[i]]), obs = obs[[i]])

Observation: 5661.974

Based on 99 replicates
Simulated p-value: 0.19
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
```

```
4.130523e-01 3.753411e+03 2.135023e+07

[[2]]
Monte-Carlo test
Call: as.randtest(sim = unlist(sim2[[i]]), obs = obs[[i]])

Observation: 3265.768

Based on 99 replicates
Simulated p-value: 0.34
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
2.343670e-01 2.701657e+03 5.793429e+06
```

In this case, none of the two tests are significant at the conventional $\alpha = 5\%$ level.

5.6 Single null models and multiple null models

In the previous section, we showed how to build a null model and simulate this null model for several trajectories separately. Now, we may find more convenient to design a global criterion (measured on all animals) to test whether the elevation variance is greater than expected under the null model. This is the principle of "multiple null models". This approach is the following:

- First define a global criterion that will be used to test the hypothesis under study;
- Define "single null models" using any of the functions `NMs.*` (see below), i.e. the randomization approach and the constraints that will be used to simulate a trajectory for each animal;
- Define a "multiple null model" from the "single null models", by defining the constraints that should be satisfied by each set of simulated trajectories and the treatment function that will be applied to each set;
- Simulate the null model `nrep` times to obtain `nrep` values of the criterion under the defined null model, using the function `testNM`
- Calculate the observed value of the criterion and calculate a P-value by comparing the observed value to the distribution of values expected under the null model.

We illustrate this approach on our example. We first define, as a global criterion, the mean elevation variance, i.e. the elevation variance averaged over all animals. We need to define a treatment function allowing to calculate this

global criterion. The treatment function should take two arguments named `x` and `par`. The argument `x` should be an object of class `"ltraj"` (i.e. the set of trajectories simulated by the null model at each step of the randomization process) and the argument `par` can be any R object (e.g. a list) containing the parameters needed by the treatment function. In our example, the treatment function will be the following:

```
> meanvarel <- function(x, par)
+ {
+     livar <- lapply(x, function(y) {
+         coordinates(y) <- y[,1:2]
+         jo <- join(y, par)
+         return(var(jo))
+     })
+     mean(unlist(livar))
+ }
```

We have defined the single null models in the previous section. We now define a multiple null model from this object using the function `NMs2NMm`:

```
> nmo5 <- NMs2NMm(nmo4, treatment.func = meanvarel,
+                 treatment.par = puechabonsp$map, nrep = 99)
```

Note that both the treatment function and the number of repetitions that we have defined in the function `NMs.randomShiftRotation` will be ignored when the multiple null model will be simulated. We can now simulate the model:

```
> sim3 <- testNM(nmo5, count=FALSE)
```

At each step of the randomization process, two trajectories are simulated (under the null model and satisfying the constraints) and the treatment function is applied to the simulated trajectories. The result is therefore a list with `nrep` component, each component containing the result of the treatment function. We now calculate the observed criterion:

```
> (obs <- meanvarel(na.omit(boar), puechabonsp$map))
```

```
[1] 280.3734
```

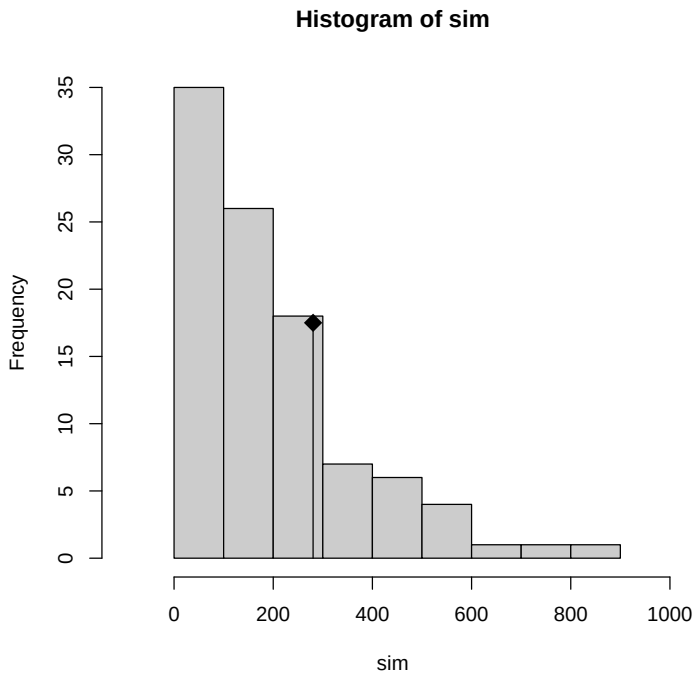
And we define a randomization test from these results, using the function `as.randtest` from the package ade4:

```
> (ran <- as.randtest(unlist(sim3), obs))
```

```
Monte-Carlo test
Call: as.randtest(sim = unlist(sim3), obs = obs)

Observation: 280.3734
```

```
Based on 99 replicates
Simulated p-value: 0.23
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
4.906214e-01 1.988151e+02 2.763395e+04
```

```
> plot(ran)
```



As a conclusion, we are unable to highlight any difference between the observed mean elevation variance and the distribution of values expected under the null model.

5.7 Several important remarks

We give in this section several important remarks:

- We have illustrated null models in this vignette with the help of the function `NMs.randomShiftRotation`. However, several other null models are available in the package. The function `NMs.randomCRW` can be used to simulate a correlated random walk, with the distribution of turning angles and distances between successive relocations calculated from

the data. The function `NMs.CRW` implements a purely parametric correlated random walk (see `?simm.crw`), where parameters are specified by the used. Finally, the function `NMs.randomCs` is similar to the function `NMs.randomShiftRotation`, but give a finer control on the distances between trajectory barycenter and a set of capture sites. The help page of `testNM` given further details and examples of use.

- We have illustrated the use of null models to test for habitat selection, but it can be used for a wide variety of hypotheses (interactions between animals, etc.).

- The main argument of the functions `NMs.*` is an object of class `ltraj`. However, in some cases, the null models can be useful to test hypotheses on other types of data. For example, imagine that we want to measure the overlap between the home-range of several animals monitored using radio-tracking, and that, for each animal, only the X and Y coordinates are available. It is therefore impossible, in theory, to define an object of class `ltraj` (since a date is needed for such objects), though it might be biologically sensible to compare the observed overlap with the overlap expected under a random distribution of the animals. It is still possible to use the functions `NMs.randomShiftRotation` and `NMs.randomCs` for this kind of analysis. In this case, the user can create a "fake" date variable (e.g. if there are `nr` rows in the data frame containing the X and Y coordinates, a "fake" date variable can be created by `fa <- 1:nr`, and can be used to create an object of class `ltraj` that can be used in null model analysis.

- Note that for multiple null models, two kinds of constraints are possible: it is possible to define constraints for the individual trajectories in the function `NMs.*` (e.g. the simulated trajectory should fall within the study area), and for the whole set of trajectories in the function `NMs2NMm` (e.g. at least 80% of the trajectories should be located in a given habitat type, see Richard et al., 2012 for an example). The two types of constraint function are taken into account when simulating the null model.

- Note that missing relocations are not allowed in null model analysis. Therefore, if there are missing relocations in the trajectory, the use of `na.omit.ltraj` is required prior to the analysis.

6 Conclusion and perspectives

Several other methods can be used to analyze a trajectory. Thus, the first passage time method, developed by Fauchald and Tveraa (2003) to identify the

areas where *area restricted search* occur is implemented in the function `fpt`. Several methods are available in the package `adehabitatHR` to estimate a home range based on objects of class `ltraj`. Thus, the Brownian bridge kernel method (Bullard 1999, Horne et al. 2007), the biased random bridge kernel method (Benhamou and Cornelis 2010, Benhamou 2011), and the product kernel algorithm (Keating and Cherry 2009) are implemented in the functions `kernelbb` and `kernelkc` respectively.

But one thing is important: at many places in this vignette, we have noted that the descriptive parameters of the steps can be analysed as a (possibly multiple) time series. The R environment provides many functions to perform such analyses, and **we stress that the package `adehabitatLT` should be considered as a springboard toward such functions**.

We included in the package `adehabitatLT` several functions allowing the analysis of animal movements. All the brother packages `adehabitat*` contain a vignette similar to this one, which explains not only the functions, but also in some cases the philosophy underlying the analysis of animal space use.

References

Bartumeus, F., da Luz, M.G.E., Viswanathan, G.M. and Catalan, J. 2005. Animal search strategies: a quantitative random-walk analysis. Ecology, 86: 3078–3087.

Barraquand, F. and Benhamou, S. 2008. Animal movements in heterogeneous landscapes: identifying profitable places and homogeneous movement bouts. Ecology, 89, 3336–3348.

Benhamou, S. and Cornelis, D. 2010. Incorporating movement behavior and barriers to improve kernel home range space use estimates. Journal of Wildlife Management, 74, 1353–1360.

Benhamou, S. 2011. Dynamic approach to space and habitat use based on biased random bridges. PLOS ONE, 6, 1–8.

Benhamou, S. 2004. How to reliably estimate the tortuosity of an animal's path: straightness, sinuosity, or fractal dimension? Journal of Theoretical Biology, 229, 209–220.

Brillinger, D., Preisler, H., Ager, A., Kie, J. and Stewart, B. 2002. Employing stochastic differential equations to model wildlife motion. Bulletin of the Brazilian Mathematical Society, 2002, 33, 385-408.

Brillinger, D., Preisler, H., Ager, A. and Kie, J. 2004. An exploratory data analysis (EdA) of the paths of moving animals. Journal of Statistical Planning and Inference, 122, 43-63.

Bullard, F. 1999. Estimating the home range of an animal: a Brownian bridge approach Johns Hopkins University.

Calenge, C. 2005. Des outils statistiques pour l'analyse des semis de points dans l'espace ecologique. Universite Claude Bernard Lyon 1.

Calenge, C. 2006. The package adehabitat for the R software: a tool for the analysis of space and habitat use by animals. Ecological modelling, 197, 516–519.

Calenge, C., Dray, S. and Royer-Carenzi, M. 2009. The concept of animals' trajectories from a data analysis perspective. Ecological Informatics, 4, 34–41.

Diggle, P. 1990. Time series. A biostatistical introduction Oxford University Press.

Dray, S., Royer-Carenzi, M. and Calenge, C. 2010. The exploratory analysis of autocorrelation in animal-movement studies. Ecological Research, 4, 34-41.

Dunn, J. and Gipson, P. 1977. Analysis of radio telemetry data in studies of home range. Biometrics, 33, 85-101

Fauchald, P. and Tveraa, T. 2003. Using first-passage time in the analysis of area-restricted search and habitat selection/ Ecology, 84, 282-288.

Gotelli, N. 2001. Research frontiers in null model analysis. Global ecology and biogeography, 10, 337–343.

Gotelli, N. and Graves, G. 1996. Null models in Ecology. Smithsonian Institution Press.

Graves, T. and Waller, J. 2006. Understanding the causes of missed global positioning system telemetry fixes. Journal of Wildlife Management, 70, 844–851.

Gueguen, L. 2001. Segmentation by maximal predictive partitioning according to composition biases. Pp 32-44 in: Gascuel, O. and Sagot, M.F. (Eds.), Computational Biology, LNCS, 2066.

Horne, J., Garton, E., Krone, S. and Lewis, J. 2007. Analyzing animal movements using Brownian bridges. Ecology, 88, 2354–2363.

Kareiva, P. and Shigesada, N. 1983. Analysing insect movement as a correlated random walk. Oecologia, 56, 234–238.

Keating, K. and Cherry, S. 2009. Modeling utilization distributions in space and time. Ecology, 90, 1971–1980.

Lavielle, M. 2005. Using penalized contrasts for the change-point problem. Signal Processing, 85, 1501–1510.

Lavielle, M. 1999. Detection of multiple changes in a sequence of dependent variables. Stochastic Processes and their Applications. 83, 79–102.

Marsh, L. and Jones, R. 1988. The form and consequences of random walk movement models. Journal of Theoretical Biology, 133, 113–131.

Martin, J., Calenge, C., Quenette, P.Y. and Allainé, D. 2008. Importance of movement constraints in habitat selection studies. Ecological Modelling, 213, 257–262.

Pebesma, E. and Bivand, R.S. 2005. Classes and Methods for Spatial data in R. R News, 5, 9–13.

Richard, E., Calenge, C., Said, S., Hamann, J.L. and Gaillard, J.M. (2012) A null model approach to study the spatial interactions between roe deer (*Capreolus capreolus*) and red deer (*Cervus elaphus*). Ecography, in press.

Root, R. and Kareiva, P. 1984. The search for resources by cabbage butterflies (Pieris Rapae): Ecological consequences and adaptive significance of markovian movements in a patchy environment. Ecology, 65, 147–165.

Turchin, P. 1998. Quantitative Analysis of Movement: measuring and modeling population redistribution in plants and animals. Sinauer Associates, Sunderland, MA.

Wald, A. and Wolfowitz, J. 1944. Statistical Tests Based on Permutations of the Observations The Annals of Mathematical Statistics, 15, 358–372.

Wiktorsson, M., Ryden, T., Nilsson, E. and Bengtsson, G. 2004. Modeling the movement of a soil insect. Journal of Theoretical Biology, 231, 497–513.