Vignette UseScape

Guillaume Bastille-Rousseau

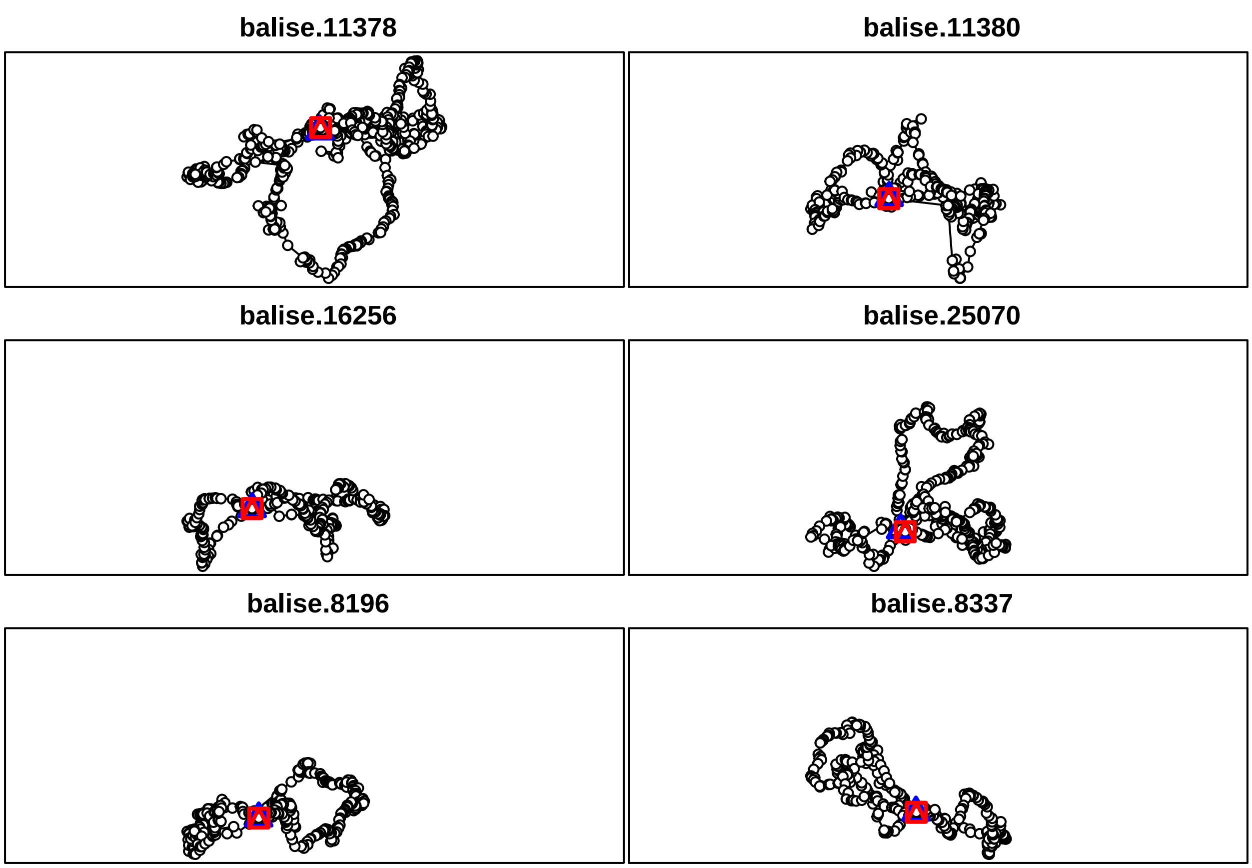
November 1, 20223

This vignette presents a simple workflow to extract the UseScape of GPS tracked individuals. We recommend interest readers read the documentation associated to the *UseScape* package for more examples.

A- Data preparation

For simplification, we initiate the analysis with a simple trajectory object of class *ltraj*. This trajectory object is freely available in the *adehabitatLT* package and contains the GPS locations of 6 albatrosses.

```
#library(devtools)
#install_github("BastilleRousseau/UseScape")
#library(UseScape)
data(albatross)
plot(albatross)
```



B- UseScape of a single albatross

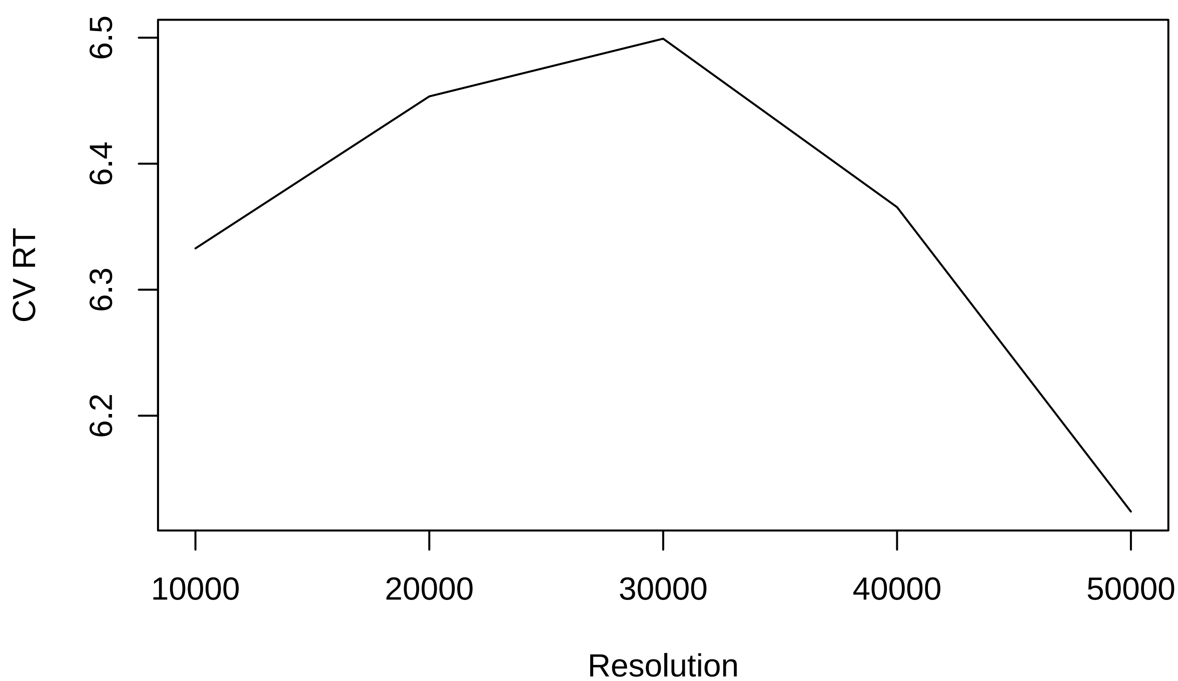
1- Selection of a grid size

We first investigate potential grid size to extract the timing history. We test the two approaches suggested in the manuscript; median step length, and a grid size that maximizes the variance in residency time. Luckily, both approaches converge at around 30,000m, which we will use in the next step.

```
#Median step length
quantile(ld(albatross[1])$dist, na.rm=T) #Extract quartile of step length
```

```
##          0%         25%         50%         75%        100%
##    235.8079  13924.2171  29424.2803  57777.3258 365043.4312
```

```
#Variance in residency time calculation
res_test(na.omit(albatross[1]), res_seq=seq(10000, 50000, 10000))
```



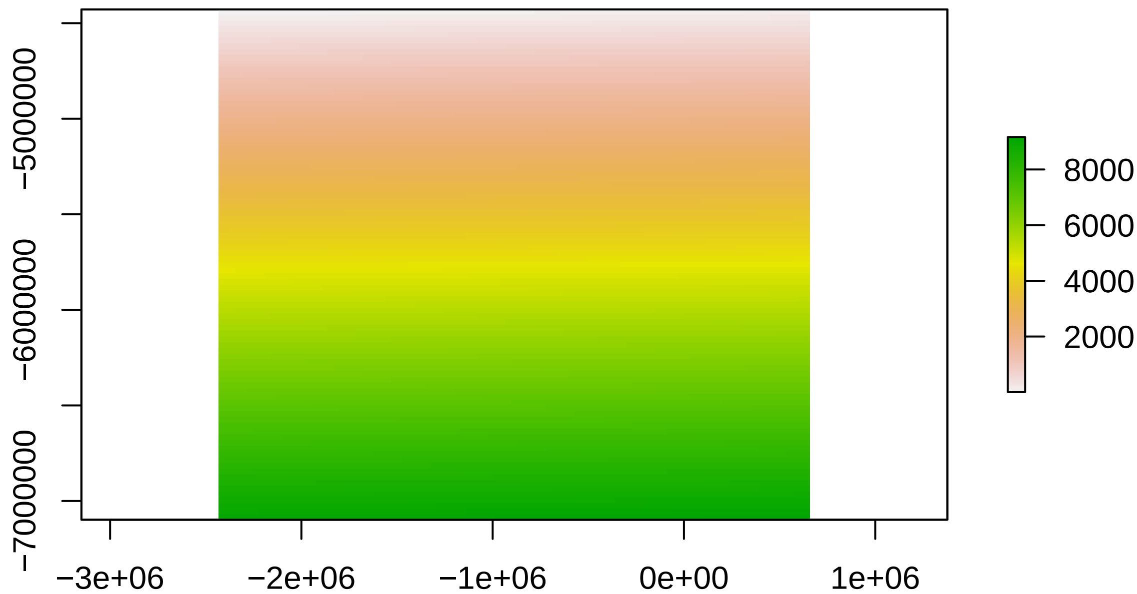
2- Evaluating timing history

We use the function *traj2timing* to extract a timing history which corresponds of the entrance and exit times of an individual in each pixel for each visit. The function return a list of two elements, the first is a list of timing history of every pixel (note that the majority were never visited), the second element of the list is the reference grid.

```r
timing_ls<-traj2timing(na.omit(albatross[1]), res=30000, grid=NULL)
timing_ls[[1]][2334:2339] #Example of the timing history of a few pixels
```

```
## [[1]]
##                time_in            time_out
## 2 2003-01-25 10:32:27 2003-01-25 11:57:48
## 3 2003-01-25 13:54:31 2003-01-25 14:54:04
## 4 2003-01-25 15:12:53 2003-01-25 16:29:31
##
## [[2]]
##                time_in            time_out
## 2 2003-01-25 09:44:05 2003-01-25 10:32:27
## 3 2003-01-25 11:57:48 2003-01-25 13:54:31
## 4 2003-01-25 17:15:46 2003-01-25 18:56:58
##
## [[3]]
##                time_in            time_out
## 2 2003-01-25 07:01:50 2003-01-25 08:32:17
## 3 2003-01-25 08:56:17 2003-01-25 09:44:05
##
## [[4]]
##                time_in            time_out
## 2 2003-01-25 03:00:11 2003-01-25 06:55:55
##
## [[5]]
## [1] time_in  time_out
## <0 rows> (or 0-length row.names)
##
## [[6]]
##                time_in            time_out
## 2 2003-01-24 20:36:49 2003-01-24 22:01:52
## 3 2003-01-25 00:03:24 2003-01-25 01:51:05
```

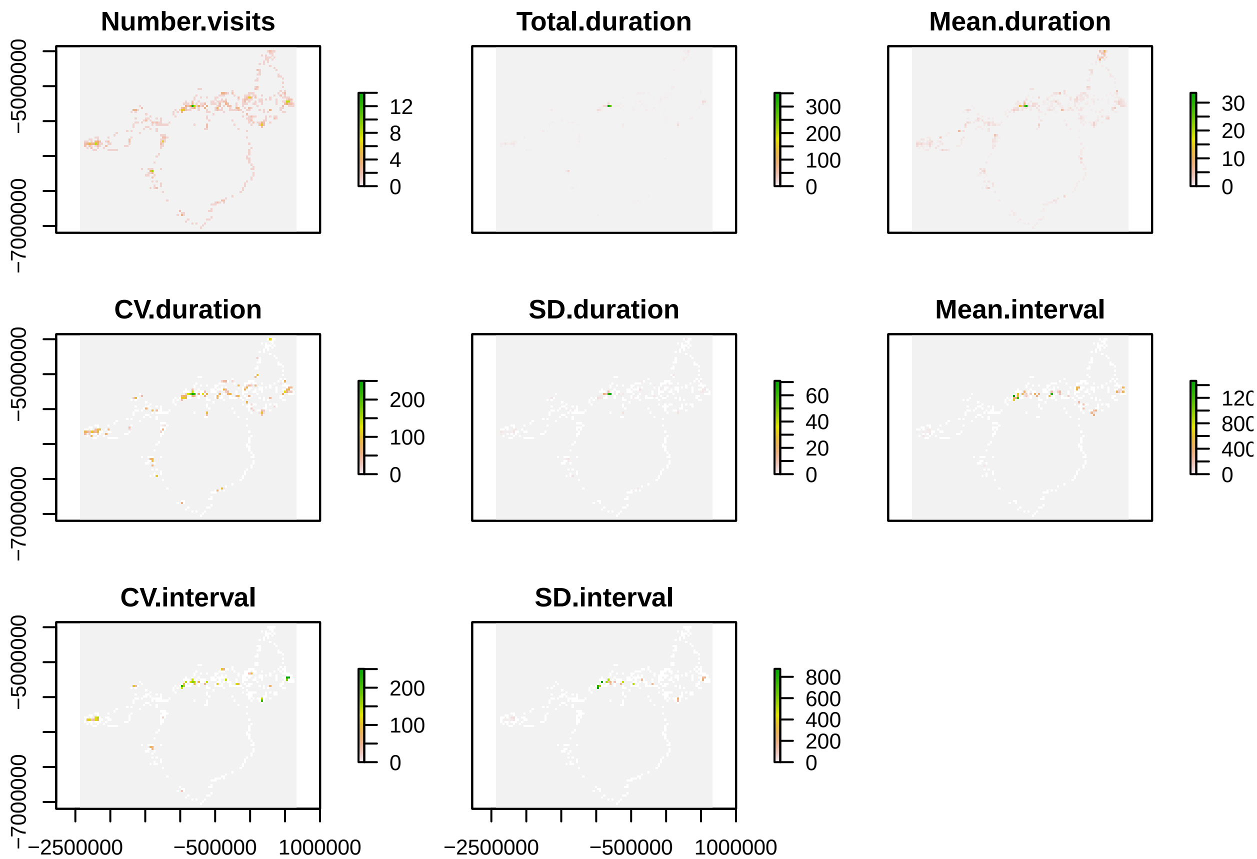
```r
plot(timing_ls[[2]]) #The reference grid
```



3 - Extracting timing metrics

We then use *timing2stack* to convert the timing history into a series of metrics that can be displayed spatially. The object returned by the function is a raster stack. The function offers the option of selecting the unit of the time we want displayed. Note that selection here will be carried forward in other steps.

```
stck<-timing2stack(timing_ls, unit_time = "hours")
plot(stck)
```



4- Clustering

Once the stack object is created, *clust_use* can be use to perform an unsupervised classification. The argument *col* allows the selection of the metrics to be integrated (based on the order of the stack object), *nb_clust* provides a range of cluster numbers to be tested, and *min_fix* set a threshold for how many times a pixel need to be visited to be integrated in the analysis. A minimum of three visits is required for the coefficient of variation interval metric to be calculated. Values outputted represent the center of each cluster according to each metric and the last row present the proportion of each cluster. Values are scaled and centered around zero.In the case of this albatross, cluster 1 represent a cluster that is rarely visited and use very little relative to the other cluster based on the center of each metric.

```
cluster<-clust_use(stck, col=c(1,2,3,4,6,7), nb_clust=1:9, min_fix=3)
```

```
## [1] "mclust is loaded correctly"
##                     [,1]        [,2]       [,3]       [,4]         [,5]
## Number.visits  -0.61489245 -0.49313514  0.3475334  0.8314805 -0.139007994
## Total.duration -0.26496290 -0.25563892 -0.2108365 -0.1436575 -0.148807189
## Mean.duration  -0.26600148 -0.25298451 -0.2848751 -0.1785671  0.004474786
## CV.duration    -0.59917387  0.01329371 -0.3547665 -0.1155257  0.521964990
## Mean.interval  -0.40762194 -0.46387188 -0.7015080 -0.4936313  1.268995814
## CV.interval     0.03353579 -0.86026452 -1.1173147  1.2978247  0.499051289
##                     [,6]       [,7]
## Number.visits  -0.33218822  2.7801599
## Total.duration -0.22305770  4.3573437
```

```
## Mean.duration  -0.21786799  4.2868528
## CV.duration     0.05119477  3.2299397
## Mean.interval   2.33349767 -0.1537907
## CV.interval     0.25723985  0.9090266
## [1] 0.26032710 0.19043289 0.14285523 0.14448003 0.11904762 0.09523810 0.04761905
```

5- Backtransformation

The output of *clust_use* are standardized (1 SD) and centered (around zero). At time, it may be easier to see the values on their original scale (according to argument set with *timing2stack*). The function *backscaling_clust_use* can be used for this. Note that arguments most parallel arguments in *clust_use*.

```
backscaled_cluster<-backscaling_clust_use(stck, cluster, col=c(1,2,3,4,6,7), min_fix=3)
```

```
##                      [,1]      [,2]      [,3]       [,4]       [,5]       [,6]
## Number.visits    3.091460  3.375085  5.333365   6.460686   4.200000   3.750000
## Total.duration   4.342744  4.959127  7.920889  12.361909  12.021479   7.112982
## Mean.duration    1.372822  1.452721  1.256973   1.909505   3.033039   1.668271
## CV.duration     44.770443 74.554007 56.655679  68.289675  99.290080  76.397090
## Mean.interval   54.788217 44.661111  1.877672  39.303302 356.642489 548.292871
## CV.interval    113.203396 61.365526 46.457347 186.528550 140.201968 126.177594
##                     [,7]
## Number.visits   11.00000
## Total.duration 309.91073
## Mean.duration   29.31881
## CV.duration    230.97564
## Mean.interval  100.48739
## CV.interval    163.97937
```

C- Population-level clustering

1- Looping over all individuals

The previous example focuses on a single individual. A two-steps clustering approach similar to what is presented in Bastille-Rousseau et al. (2021) Con. Bio can also be used to generate population results. The *loop_id* function loops the *traj2timing* and *timing2stack* over all individuals of a *traj* object using the same grid size.

```
traj<-na.omit(albatross)
ls1<-loop_id(traj, res=30000)
```

```
##  balise.11378
##  balise.11380
##  balise.16256
##  balise.25070
##  balise.8196
##  balise.8337
```

```r
table<-table_cluster(traj, ls1)

#Showing the first few rows of the table created.
head(table)
```

```
##   Number.visits Total.duration Mean.duration CV.duration SD.duration
## 1             0              0             0           0           0
## 2             0              0             0           0           0
## 3             0              0             0           0           0
## 4             0              0             0           0           0
## 5             0              0             0           0           0
## 6             0              0             0           0           0
##   Mean.interval CV.interval SD.interval            ID
## 1             0           0           0  balise.11378
## 2             0           0           0  balise.11378
## 3             0           0           0  balise.11378
## 4             0           0           0  balise.11378
## 5             0           0           0  balise.11378
## 6             0           0           0  balise.11378
```

2- Individual-level clustering

The first step of the analysis is to apply the clustering to each individual. *ind_clust* applies a mixture model to each individual. The same arguments can be passed as in *clust_use*. *ind_clust* simply return a list object with each element representing a single individual.

```r
ind<-ind_clust(table)
```

```
## [1] "mclust is loaded correctly"
## [1] " balise.11378"
## [1] " balise.11380"
## [1] " balise.16256"
## [1] " balise.25070"
## [1] " balise.8196"
## [1] " balise.8337"
```

3- Population-level clustering

After performing the individual clustering, a second clustering is applied via *pop_clust*. This second clustering takes the output of *ind_clust* and will identify which individual clusters could be considered as one population-level clusters. The function automatically selects the optimal number of clusters (based on BIC). It is possible for two clusters from the same individual to be in the same population-level cluster. Likewise, it is possible that a population level cluster does not have all individuals. Here, five different population clusters were calculated. The center (mean) of each cluster and proportion of each cluster is output by default.

```r
pop<-pop_clust(traj, ind)
```

```
##                         [,1]       [,2]        [,3]         [,4]         [,5]
## Number.visits  -0.39820312 1.1173323 -0.1103665  3.09776747 -0.09179529
```

```
## Total.duration -0.28270103 1.7064212 -0.3331008  3.34815139 -0.30829812
## Mean.duration  -0.17147540 1.6783498 -0.3726472  3.19458317 -0.36200486
## CV.duration    -0.10430795 0.9644704 -0.4118141  2.79734892 -0.17868476
## Mean.interval  -0.06454229 0.5858700 -0.4846216 -0.09286929  2.88410111
## SD.interval    -0.11887844 0.6538889 -0.4509593  0.10260608  2.69443405
## [1] 0.3199727 0.1600000 0.2400000 0.1600000 0.1200273
```

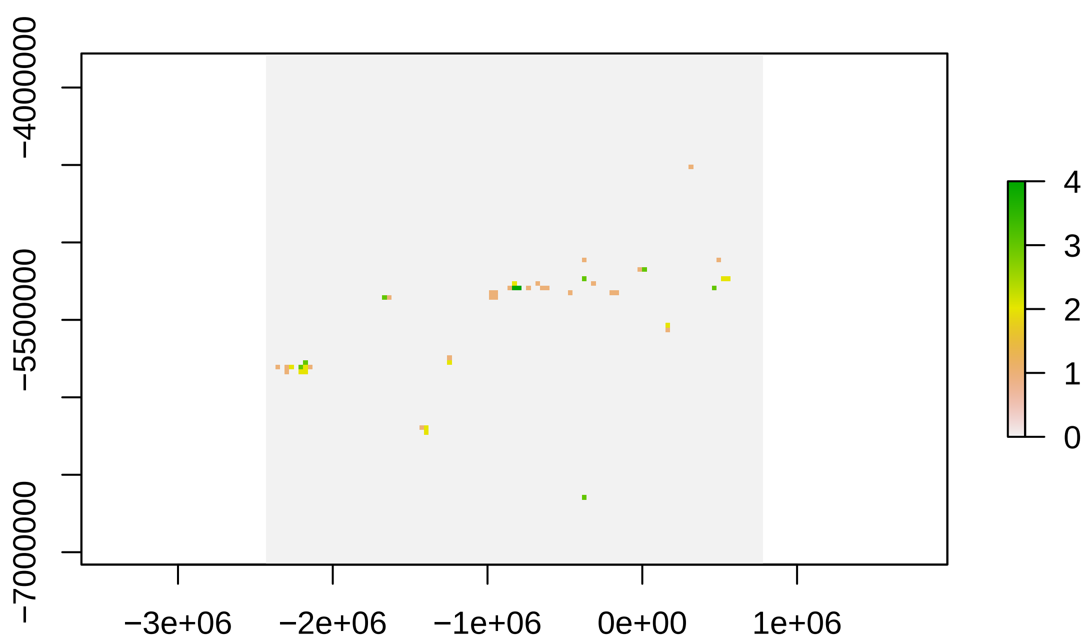
4- Mapping results

After performing the population level cluster, the function *clust_stack* recombines the individual and population level clustering and produce a *stack* object for each individual albatross showing the most likely cluster, and also the probability of observing each cluster (uncertainty) in any given pixel. These object can be exported to be used in other software using the *writeRaster* function.

```
stack<-clust_stack(ls1, pop, ind, table, min_fix = 3)
```

```
## [1] " balise.11378"
## [1] " balise.11380"
## [1] " balise.16256"
## [1] " balise.25070"
## [1] " balise.8196"
## [1] " balise.8337"
```

```
plot(stack[[1]][[1]]) #Plot first individuals
```



```r
plot(stack[[2]][[1]]) #Plot second individuals
```