The EMbC R-package: quick reference

J.Garriga; F.Bartumeus
ICREA Movement Ecology Laboratory (CEAB-CSIC)
jgarriga@ceab.csic.es

Version 2.0.1, dic 2019

Abstract

In this document we give a brief overview of the EMbC R-package with special emphasis on its use for behavioural annotation of animal's movement trajectories. For details about the EMbC algorithm please refer to (Garriga et. al 2016) and for further details about the package please refer to the package reference manual.

1 The EMbC Algorithm

The Expectation-maximization binary clustering (EMbC) is a general purpose, unsupervised, multi-variate, clustering algorithm (Garriga et al. 2016), driven by two main motivations: (i) it looks for a good compromise between statistical soundness and ease and generality of use - by minimizing prior assumptions and favouring the semantic interpretation of the final clustering- and, (ii) it allows taking into account the uncertainty in the data. These two features make it specially suitable for the behavioural annotation of animal's movement trajectories.

The method is a variant of the well sounded Expectation-Maximization Clustering (EMC) algorithm - i.e. under the assumption of an underlying Gaussian Mixture Model (GMM) describing the distribution of the data-set - but constrained to generate a binary partition of the input space. This is achieved by means of the *delimiters*, a set of parameters that discretize the input features into high and low values and define the binary regions of the input space. As a result, each final cluster includes a unique combination of either low or high values of the input variables. Splitting the input features into low and high values is what favours the semantic interpretation of the final clustering.

The initial assumptions implemented in the EMbC algorithm aim at minimizing biases and sensitivity to initial conditions: (i) each data point is assigned a uniform probability of belonging to each cluster, (ii) the prior mixture distribution is uniform (each cluster starts with the same number of data points), (iii) the starting partition, (*i.e.* initial delimiters position), is selected based on a global maximum variance criterion, thus conveying the minimum information possible.

The number of output clusters is 2^m determined by the number of input features m. This number is only an upper bound as some of the clusters can vanish along the likelihood optimization process. The EMbC algorithm is intended to be used with not more than 5 or 6 input features, yielding a maximum of 32 or 64 clusters. This limitation in the number of clusters is consistent with the main motivation of the algorithm of favouring the semantic interpretation of the results.

The algorithm deals very intuitively with data reliability: the larger the uncertainty associated with a data point, the smaller the leverage of that data point in the clustering.

With respect to close related methods like EMC and Hidden Markov Models (HMM), the EMbC is specially useful when: (i) we can expect bi-modality, to some extent, in the conditional distribution of the input features or, at least, we can assume that a binary partition of the input space can provide useful information, and (ii) a first order temporal dependence assumption, a necessary condition in HMM, can not be guaranteed.

2 The EMbC R-package

The EMbC algorithm is of general purpose and can deal with any type of data sets or time series. However, the EMbC R-package is mainly intended for the behavioural annotation of animals' movement trajectories where an easy interpretation of the final clustering and the reliability of the data constitute two key issues, and the conditions of bi-modality and unfair temporal dependence usually hold. In particular, the temporal dependence condition is easily violated in animal's movement trajectories because of the heterogeneity in empirical time series due to large gaps, or prefixed sampling scheduling.

Input movement trajectories are given either as a *data.frame* or a *Move* object from the **move** R-package. The package deals also with stacks of trajectories for population level analysis. Segmentation is based on local estimates of velocity and turning angle, eventually including a solar position covariate as a daytime indicator.

The core clustering method is complemented with a set of functions to easily visualize and analyse the output:

- clustering statistics,
- clustering scatter-plot (2D and 3D),
- temporal labelling profile (ethogram),
- plotting of intermediate variables,
- confusion matrix (numerical validation with respect to an expert's labelling),
- visual validation versus external information (e.g. environmental data),
- generation of kml or web-map documents for detailed inspection of the output.

Also, some functions are provided to further refine the output, either by pre-processing (smoothing) the input data or by post-processing (smoothing, relabelling, merging) the output labelling.

The results obtained for different empirical datasets suggest that the EMbC algorithm behaves reasonably well for a wide range of tracking technologies, species, and ecological contexts (e.g. migration, foraging).

2.1 Working Environment

The EMbC package has dependencies with the following packages:

- **methods**, *formal methods and classes* (R Core Team 2015);
- **sp**, *classes and methods for spatial data* (Pebesma and Bivand 2005; Bivand, Pebesma, and Gomez-Rubio 2013);
- **maptools**, *tools for reading and handling spatial objects* (Bivand and Lewin-Koh 2015);
- **mnormt**, *the multivariate normal and t distributions* (Azzalini and Genz 2015);
- **RColorBrewer**, *ColorBrewer palettes* (Neuwirth 2014);

We also suggest the package **rgl**, *3D visualization device system (OpenGL)* (Adler, Murdoch, and others 2015), to allow for dynamic 3D scatter-plots in multivariate analyses.

For researchers who are familiar with the MoveBank framework, we include a special link with the package **move**, *Visualizing and analizing animal track data* (Kranstauber and Smolla 2015), to allow users to make use of *Move* objects as input trajectories.

2.2 Basic structure

Basically, the package consists of a hierarchy of classes:

- *binClst*, the main class, representing the binary clustering of a multivariate data set;
- *binClstPath*, a child class of the former, representing the binary clustering of a trajectory;
- *binClstStck*, basically a list of *binClstPath* objects resulting from the global clustering of a stack of trajectories.

Instances of these classes are build by means of two main constructors:

- *embc()*, the main core of the package, implementing the EMbC algorithm itself; this constructor takes as input a matrix of data-points and returns an object of class *binClst* with the **multivariate** binary clustering of the input data;
- *stbc()*, a specific constructor for the behavioural annotation of movement trajectories; the input to this constructor is a trajectory (given either as a data.frame, a *Move* object or a list of them) and returns an object of class *binClstPath* (or any of its child classes) with the **bivariate** (velocity/turn) clustering of the trajectory; eventually it can compute a **trivariate** clustering by including a parameter indicating a solar covariate (either *height* or *azimuth*) to be used as a daytime indicator.

The behaviour of the constructors can be modified by means of different parameters (*e.g.* maximum number of iterations, information shown at each step, pre-smoothing of the data).

The output objects have several slots containing all information related to the binary clustering (input data, intermediate computations and output data). All slots are accessible and can be used with any R function external to the package or even modified. However, **we recommend not to manually change the values in the slots in order to keep the internal consistency**.

Let's load the package;

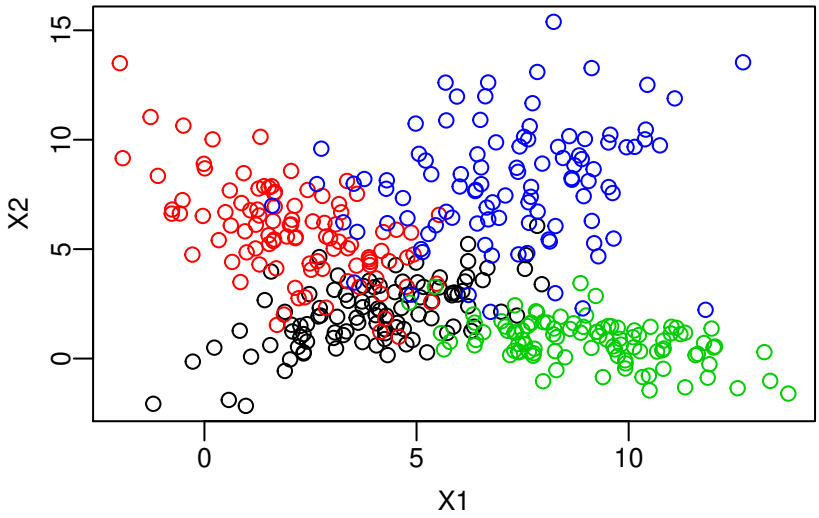
```
library(EMbC)
```

3 Class: binClst

This is the core class that implements the multivariate binary clustering algorithm. The input data-set is given as a matrix with data points given as rows and input features as columns. No more than 5 (6 at most) variables should be used in order to get a human interpretable set of clusters.

Let's use the object *x2d* included in the package. This object contains a set of data points generated from a bivariate GMM with four components (slot x2d@D), and a labelling indicating which component generated each data point (slot x2d@L);

```
par(mgp=c(1.5, 0.4, 0), cex.lab=0.8, cex.axis=0.8)
plot(x2d@D, col=x2d@L, xlab='X1', ylab='X2')
```




```
# x2d@D is a matrix with the input data
# x2d@L is a numeric vector with the reference labelling
```

3.1 Binary clustering

To perform the clustering of a data-set we simply call the *embc()* constructor passing in a matrix with the input data and storing the result in a variable (*e.g. mybc*);

```
mybc <- embc(x2d@D)
```

```
## ... computing starting delimiters
## [1]   0  -0.0000e+00       4       400
## [1]   1  -5.4847e+00       4       297
## [1]   2  -5.3257e+00       4        14
## [1]   3  -5.1640e+00       4        12
## [1]   4  -5.0542e+00       4        10
## [1]   5  -4.9906e+00       4         7
## [1]   6  -4.9597e+00       4         9
## [1]   7  -4.9448e+00       4         5
## [1]   8  -4.9360e+00       4         5
## [1]   9  -4.9308e+00       4         6
## [1]  10  -4.9282e+00       4         3
## [1]  11  -4.9271e+00       4         3
## [1]  12  -4.9265e+00       4         3
## [1]  13  -4.9270e+00       4         2
## [1]  14  -4.9281e+00       4         4
## [1]  15  -4.9282e+00       4         1
## [1]  16  -4.9278e+00       4         4
## [1]  17  -4.9279e+00       4         3
## [1]  18  -4.9274e+00       4         4
## [1]  19  -4.9300e+00       4         6
## [1]  20  -4.9291e+00       4         3
## [1]  21  -4.9292e+00       4         0
## [1]  22  -4.9295e+00       4         0
## [1]  23  -4.9293e+00       4         2
## [1]  24  -4.9291e+00       4         0
## [1]  25  -4.9288e+00       4         2
## [1]  26  -4.9287e+00       4         1
## [1]  27  -4.9293e+00       4         0
## [1]  28  -4.9289e+00       4         2
## [1]  29  -4.9296e+00       4         0
## [1]  30  -4.9296e+00       4         0
## [1]  31  -4.9294e+00       4         0
## [1]  32  -4.9294e+00       4         0
## [1]  33  -4.9294e+00       4         0
## [1] ... Stable clustering
```

At each iteration, the algorithm shows the iteration number, the current likelihood value, the number of effective clusters and the number of labels that have changed with respect to the previous iteration.

3.2 Slots

mybc is an instance of class *binClst*. Any slot of a *binClst* object is accessible and can be used by (passed to) any R function. The most basic slots of a *binClst* object are:

```
slotNames(mybc)
```

```
##  [1] "X"    "U"    "stdv" "m"    "k"    "n"    "R"    "P"    "W"    "A"
```

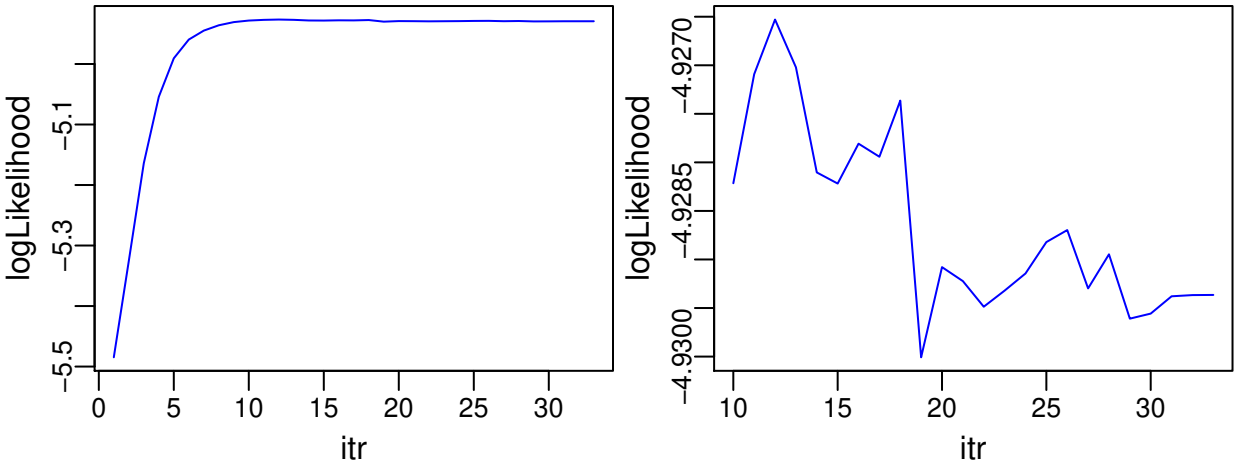
```
## [11] "L"    "C"
```

- mybc@X, a matrix with the input data points;
- mybc@U, a matrix of the same dimension as the input data matrix, with a reliability value (ranging from 0 to 1) for each input value, (by default is a matrix of ones);
- mybc@R, a matrix with the values of the delimiters for each binary region;
- mybc@P, a list where each element is a named list with the Gaussian parameters of each output cluster;
- mybc@W, a matrix with the likelihood weights of each data-point with respect to each output cluster;
- mybc@A, a numeric vector with the output labelling of each location, (the number of the cluster with the highest likelihood weight, coded as 1:LL, 2:LH, 3:HL, and 4:HH).

3.3 Likelihood Plot

The likelihood plot allows a visual assessment of the convergence of the algorithm;

```
# the lkhp() function allows an offset parameter;
lkhp(mybc)       # left panel
lkhp(mybc, 10)   # right panel
```



The last iterations may show some decrease in likelihood. This is due to a slight discrepancy between binary and optimal likelihood clusterings that can appear at the last steps of the algorithm, normally involving just a few data-points at the boundaries of the binary regions (note the low number of data-points changing their labels beyond iteration number 12 where the likelihood starts decreasing).

3.4 Clustering parameters

The function *stts()* shows the statistics of the clustering. The columns $X_i.mn$ and $X_i.sd$ show the mean and standard deviation of the input features. The last two columns show the marginal distribution of the clustering in absolute (number of data-points) and relative (percentage) values;

```
stts(mybc)
```

```
##          X1.mn    X1.sd    X2.mn    X2.sd     kn  kn(%)
##  1 LL     2.60     1.72     0.94     1.51     69  17.25
##  2 LH     1.78     1.76     6.19     2.30    107  26.75
##  3 HL     8.76     2.37     0.94     1.11    121  30.25
##  4 HH     7.41     1.96     8.02     2.92    103  25.75
```

The complete set of parameters of the Gaussian mixture is accessible through the slot mybc@P. This slot is a list of inner named-lists (M for mean and S for the covariance matrix) for each cluster. For instance, the parameters for cluster 1 (LL) are;

```
mybc@P[[1]]
```

```
## $M
## [1] 2.6000747 0.9403607
##
## $S
##          [,1]     [,2]
## [1,] 2.963896 1.924727
## [2,] 1.924727 2.265230
```

The *delimiters* are accessible through the slot mybc@R where we have the *min()* and *max()* values that delimit each binary region;

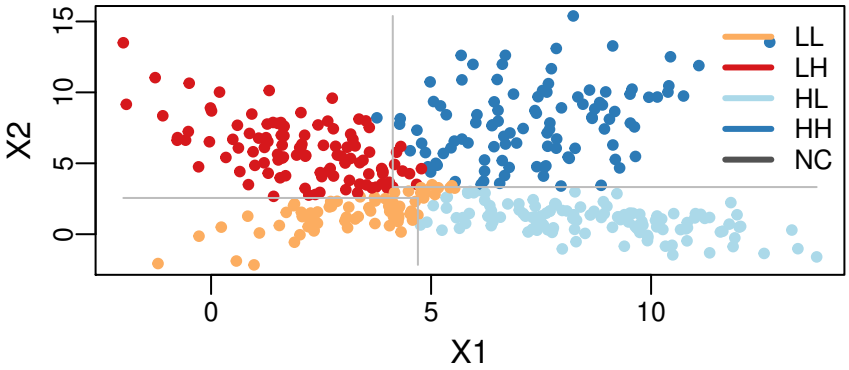
```
mybc@R
```

```
##         X1.min    X2.min    X1.max    X2.max
## 1.LL -1.994262 -2.162228  4.699946  2.553833
## 2.LH -1.994262  2.553833  4.129379 15.386923
## 3.HL  4.699946 -2.162228 13.762928  3.323976
## 4.HH  4.129379  3.323976 13.762928 15.386923
```

3.5 Clustering scatter-plot

The function *sctr()* makes a scatter-plot of the data-points, showing the clusters in different colours, and depicting the binary delimiters (light grey lines) to show the binary regions;

```
sctr(mybc)
```



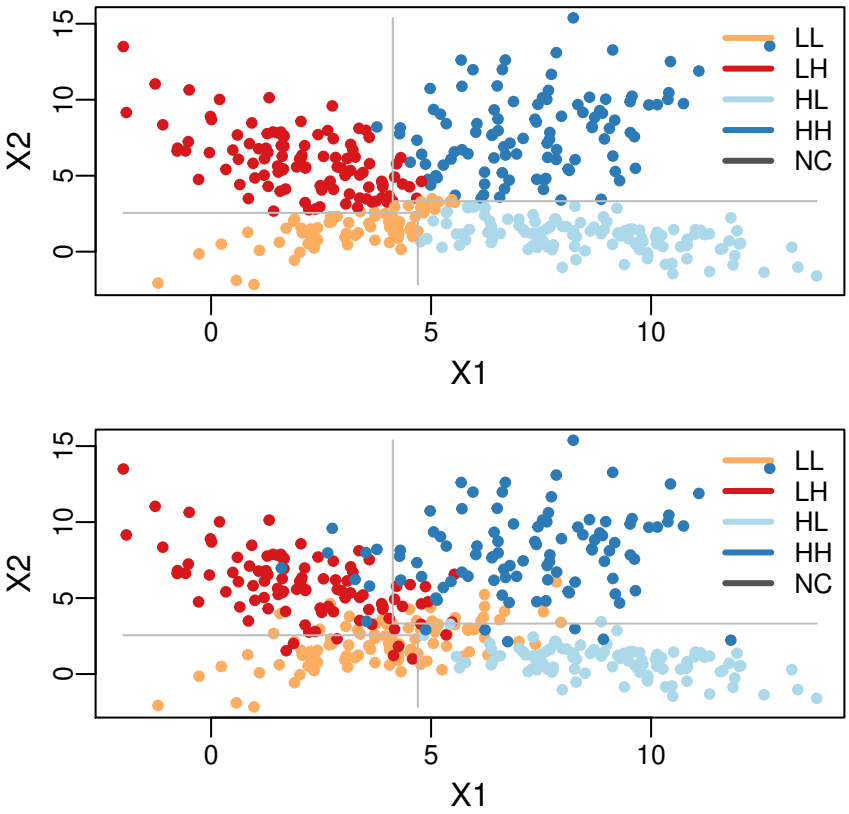
The *NC* in the legend stands for not classified points. Not classified points may appear only when performing the behavioural annotation of movement trajectories (explained later in this document) and correspond to outliers due to errors or gaps in the trajectory or, typically, to the last track of the trajectory.

3.6 Clustering validation

In a supervised case, that is in case that an expert's labelling is available, we can use this labelling to validate the results of the clustering. The expert labelling must be numerically coded and translated to the range of the number of clusters, and must be given as a numeric vector with one numeric label for each location.

We can make a visual validation of the clustering versus the expert labelling by means of the *sctr()* function passing in the expert labelling vector as a second parameter;

```
sctr(mybc, x2d@L)
```




```
# the top plot shows the clustering result;
# the bottom plot shows the reference labelling;
```

We can perform a numeric assessment of the clustering in terms of a confusion matrix by means of the function *cnfm()*;

```
cnfm(mybc, x2d@L)
```

```
##       cls.01 cls.02 cls.03 cls.04   mrg.   Prc.
## 1.LL      57      9      2      1   0.17   0.83
## 2.LH      13     86      0      8   0.27   0.80
## 3.HL      18      1     97      5   0.30   0.80
## 4.HH      12      4      1     86   0.26   0.83
##       ------ ------ ------ ------ ------ ------
## mrg     0.25   0.25   0.25   0.25   0.82   0.82
## Rcl     0.57   0.86   0.97   0.86   0.82    NaN
## Fms     0.68   0.83   0.88   0.84    NaN   0.81
```

The confusion matrix shows values of row *precision* and row *F-measure*, and values of column *recall* and column *F-measure*. The 3x2 subset of cells at the bottom-right show respectively: the *overall accuracy*, the *average recall*, the *average precision*, NaN, NaN and the *Macro-F-measure*.

4 Class: binClstPath

The *binClstPath* is a *binClst* subclass intended to automatically perform the bivariate clustering of a movement trajectory, based on estimated local values of velocity and turn. It can also perform a trivariate clustering by

incorporating a daytime covariate (i.e. solar height or solar azimuth).

The input data-set is a trajectory given as a data.frame with timestamps, longitudes and latitudes in columns 1:3 respectively (column headers are user free). Timestamps must be given as.POSIXct() with the specific format "%Y-%m-%d %H:%M:%S".

As an example, the package includes an object named *expth*. This is a synthetically generated trajectory stored as a data.frame;

```
head(expth)
```

```
##                   dTm      lon       lat lbl
## 1 2014-11-14 17:05:57 70.25000 -49.26000   1
## 2 2014-11-14 17:07:40 70.24925 -49.25862   1
## 3 2014-11-14 17:21:34 70.22671 -49.25500   1
## 4 2014-11-14 17:36:13 70.21570 -49.25763   1
## 5 2014-11-14 18:44:05 70.12802 -49.26653   3
## 6 2014-11-14 19:02:20 70.01982 -49.25872   3
```

expth is a synthetically generated trajectory with expert labelling (note the column expth$lbl). Further columns of data can be included in the input *data.frame* as long as the first three columns respect the required format.

Tip: by including an expert labelling with a column labelled *lbl* all validation functions will make use of it by default.

4.1 Bivariate velocity-turn clustering

To perform the bivariate velocity/turn clustering of this trajectory we simply call the *stbc()* constructor passing in the data.frame with the time/space coordinates of the trajectory and storing the output *binClstPath* object in a variable (*e.g. mybcp*);

```
mybcp <- stbc(expth, info=-1)
```

```
## [1]   0  -0.0000e+00       4       600
## [1] ... Stable clustering
```

```
# info=-1 supresses any step wise output information
```

The output object *mybcp* is a *binClstPath* instance with the following slots;

```
slotNames(mybcp)
```

```
##  [1] "pth"       "spn"       "dst"       "hdg"       "bursted"   "tracks"
##  [7] "midPoints" "X"         "U"         "stdv"      "m"         "k"
## [13] "n"         "R"         "P"         "W"         "A"         "L"
## [19] "C"
```

As a child class, a *binClstPath* object inherits and extends the set of slots of the *binClst* class. The basic slot differences with respect to the *binClst* class are:

- mybcp@pth, a data.frame with three first columns named as *dTm*, *lon*, *lat* plus all additional columns of data included in the input data.frame;
- mybcp@spn, a numeric vector with the computed time span between locations;
- mybcp@dst, a numeric vector with the estimated distances between locations, computed as loxodromic lines;
- mybcp@hdg, a numeric vector with local heading directions, given in clockwise radians from North (a value of $2\,\pi$ is used to distinguish no movement from movement heading North);

- mybcp@X, is the matrix of input data that in this case is automatically generated with the estimated local values of velocity and turn;
- mybcp@U, is the matrix of uncertainties that is also automatically generated based on the time-spans between locations.

Slots *tracks*, *midpoints* and *bursted* are related to the bursted visualization of the trajectory (covered later in this document) and should not be manipulated.

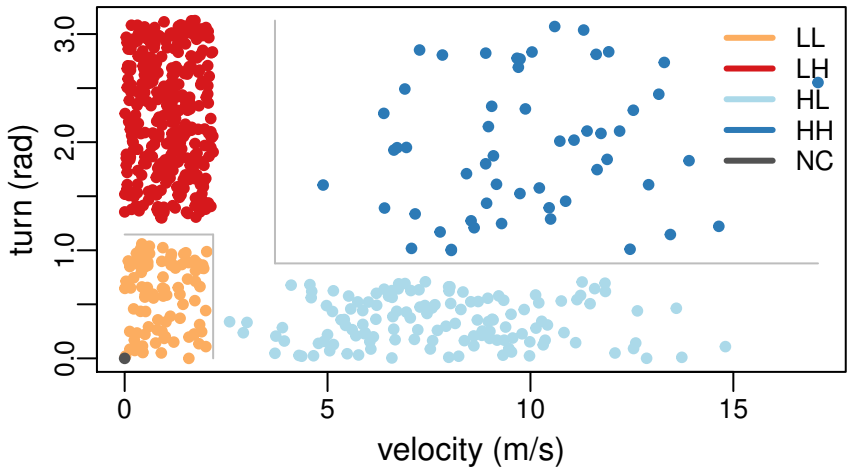
4.2 Basic functionality

Because of class inheritance, all functionality described for the *binClst* class (*e.g.* likelihood plot, clustering parameters, scatter-plot, validation) holds for a *binClstPath* instance.

```
stts(mybcp)
```

```
##          X1.mn    X1.sd    X2.mn    X2.sd     kn  kn(%)
##  1 LL     0.91     0.59     0.57     0.35     95  15.83
##  2 LH     1.07     0.61     2.25     0.56    316  52.67
##  3 HL     7.65     2.61     0.34     0.21    133  22.17
##  4 HH     9.99     2.39     1.93     0.67     55   9.17
```

```
sctr(mybcp)
```




```
cnfm(mybcp)
```

```
##       cls.01 cls.02 cls.03 cls.04   mrg.   Prc.
## 1.LL      95      0      0      0   0.16      1
## 2.LH       0    316      0      0   0.53      1
## 3.HL       0      0    133      0   0.22      1
## 4.HH       0      0      0     55   0.09      1
##       ------ ------ ------ ------ ------ ------
## mrg     0.16   0.53   0.22   0.09      1      1
## Rcl     1.00   1.00   1.00   1.00      1    NaN
## Fms     1.00   1.00   1.00   1.00    NaN      1
```

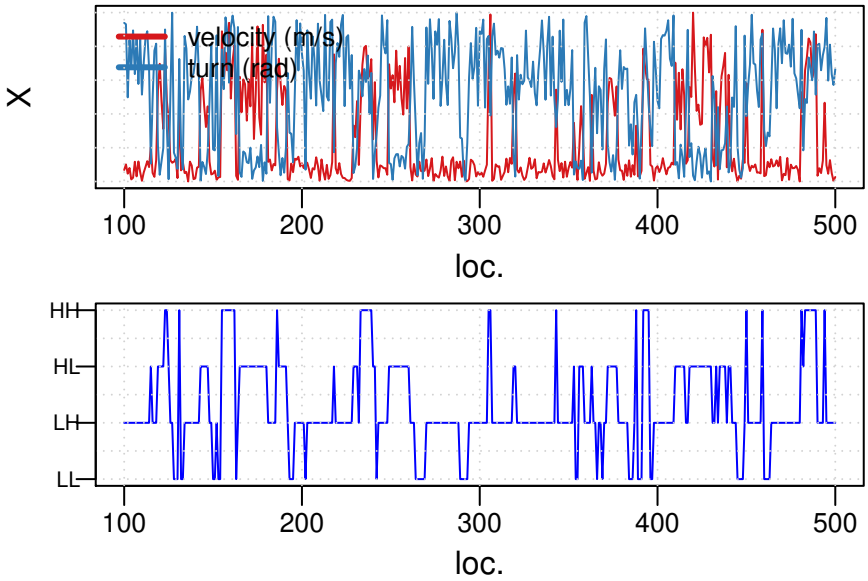
```
# the expert labelling given in expth$lbl is used by default
```

Nonetheless, the *binClstPath* class has some particular functionalities of special interest for the case of behavioural annotation of movement trajectories. These functionalities are described in the following.

4.2.1 Labeling profile

The function *lblp()* plots the temporal series of data and the temporal profile of the behavioural labelling;

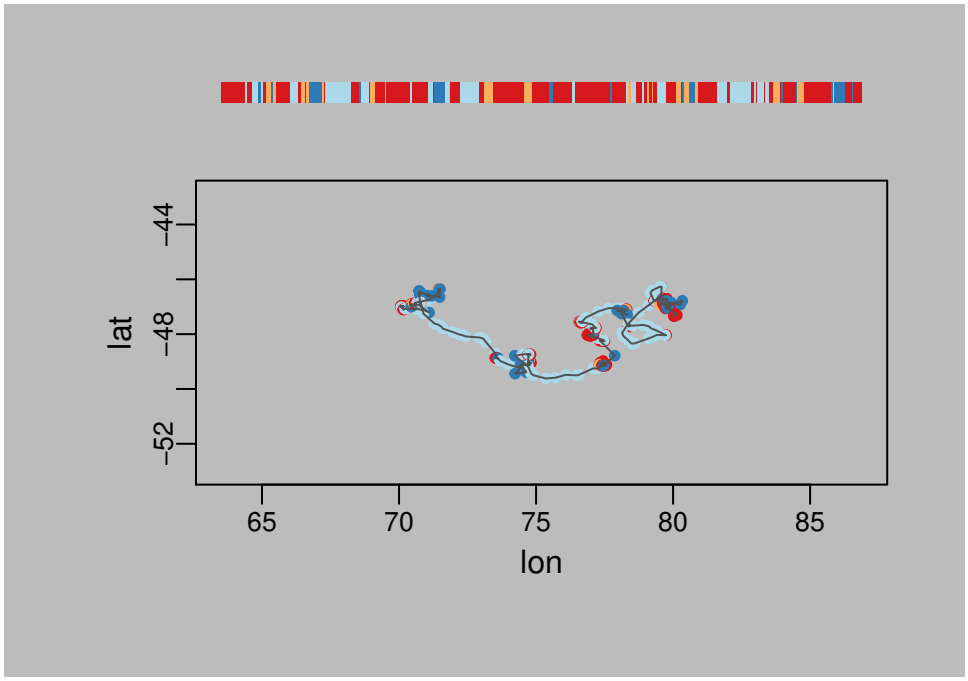
```
# lims=c(a, b) limits the plot to a chunk of the trajectory
lblp(mybcp, lims=c(100, 500))
```



4.2.2 Fast visualization of the annotated trajectory

The function *view()* shows the annotated trajectory and a top panel with the temporal sequence of behaviours;

```
# this function allows a parameter lims=c(a,b) as well
view(mybcp, lims=c(100, 500))
```



4.2.3 Detailed inspection of the annotated trajectory

We can generate *kml* or *html* documents for a detailed inspection of the output by means of *google-earth* or the user's system browser. The package allows two types of visualization of the annotated trajectory: a point-wise visualization (functions *pkml()* or *pmap()*) or a burst-wise visualization (functions *bkml()* or *bmap()*) (Garriga et al. 2016);

```
# point-wise kml doc generation;
# display=TRUE launches google-earth from within R;
pkml(bc, display=TRUE)
```

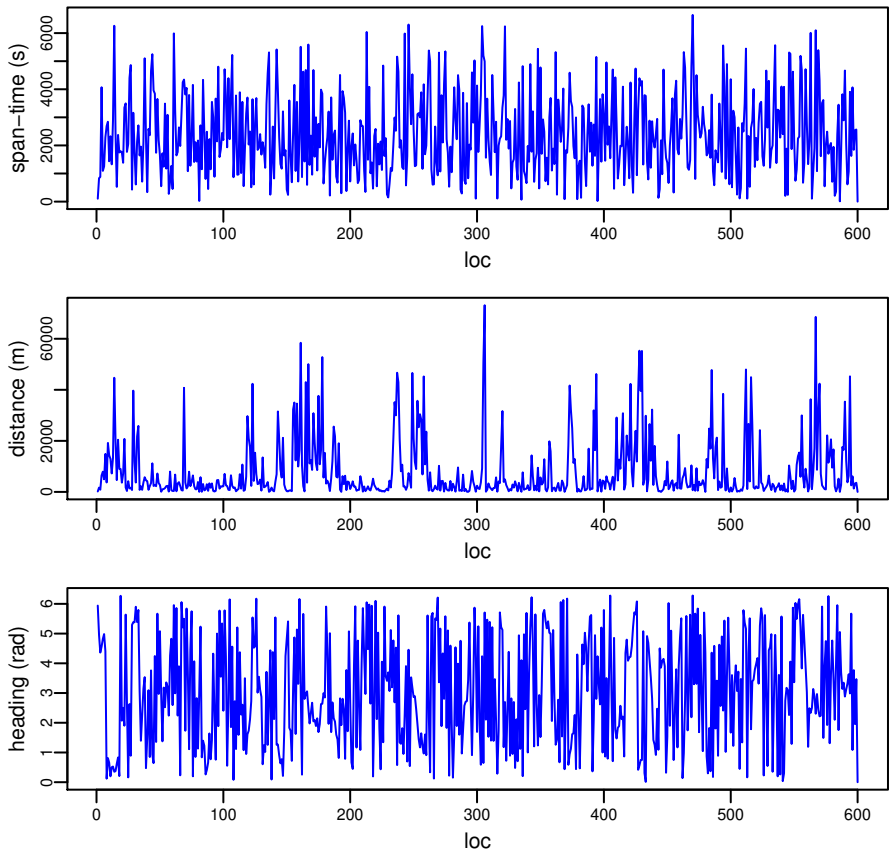
By default, the *kml* or *html* documents are named with a *Sys.time()* based name and saved in a folder *embcdocs* automatically generated in the user's home directory. This can be modified by means of the corresponding parameters.

The burst-wise visualization requires the computation of burst segments and midpoints. This is computed only the first time that a burst-visualization of a trajectory is requested. In case of long trajectories, this process can take some time.

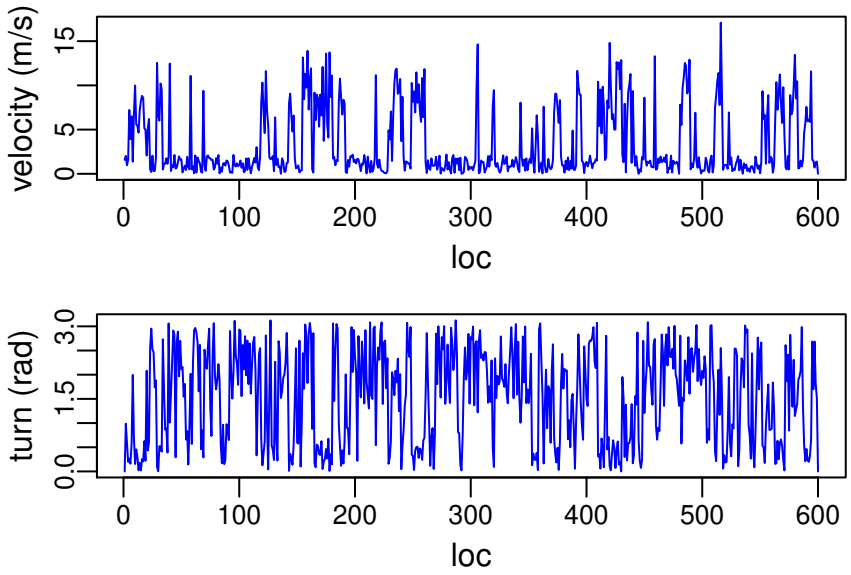
4.2.4 Plot intermediate variables

Intermediate data computed by the *stbc()* constructor and stored in the *binClstPath* object can be easily plotted with automatic formatting and labelling of axes;

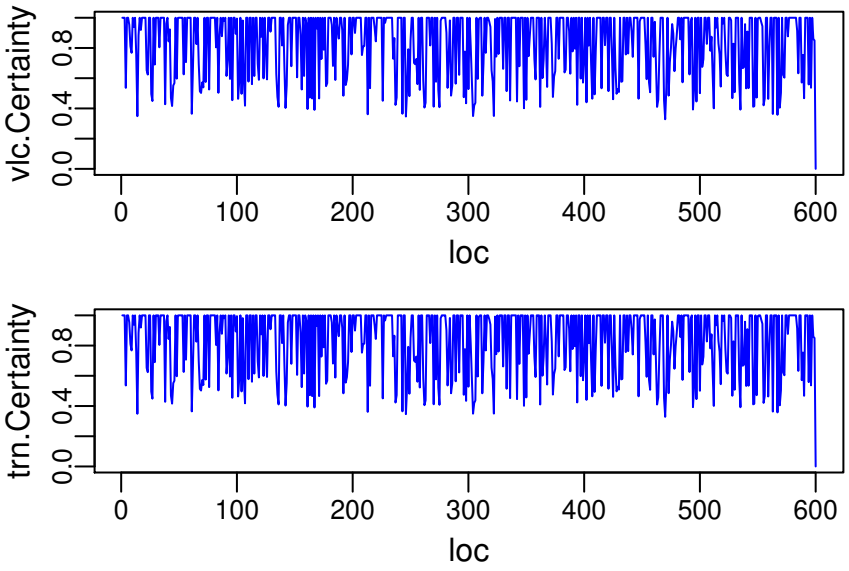
```
# plotting time-spans, distances and heading directions;
# this is the default behavior when we just pass the binClstPath instance;
varp(mybcp)
```



```r
# plotting input data (estimated local values of velocity and turn);
varp(mybcp@X)
```




```r
# plotting certainties associated to each data-point (and input feature)
varp(mybcp@U)
```



Indeed, the function *varp()* is a wrapper for the R *plot()* function. The purpose of this function is simply to ease the visualization of intermediate variables by formatting and labelling the axes accordingly to each one.

4.3 Using *Move* objects from the R-package *move*

Note: the dependency with respect to the *move* R-package has been dropped, and the use of the old *binClstMove* objects is now deprecated. Nonetheless *Move* objects can still be passed directly to the *stbc()* function.

This is intended for users having trajectories in *Movebank* (https://www.movebank.org/) and familiarized with the *move* R-package. Let's use the *leroy* data in the *Move R-package*.

```
library(move)
data(leroy)
```

leroy is a GPS trajectory of an urban Fisher (*Martes pennati*) with 919 tracks, spanning from 2009-02-11 12:16:45 to 2009-03-04 09:16:59, with a mean time interval between tracks of 32.7 minutes. *Move* objects can be passed directly to the *stbc()* constructor;

```
# leroy is passed directly to the constructor
leroybc <- stbc(leroy, info=-1)
```

```
## [1]   0  -0.0000e+00       4       919
## [1] ... Stable clustering
```

4.4 Trivariate clustering: including a daytime covariate

Daytime covariates refer to the solar position. This can be given as solar height in degrees above the horizon (night/day distinction), or by solar azimuth in degrees from north (sunrise/sunset distinction).

Including daytime covariates is the natural way of incorporating time information in the clustering of an animal's movement trajectory, with the potential advantage of increasing the maximum number of output clusters to $2^3 = 8$, *i.e.* the number of movement behaviours that can potentially be distinguish.

A trivariate clustering including a daytime covariate is done by means of the parameter *scv* with possible values 'height' or 'azimuth';

```
leroybc3 <- stbc(leroy, scv='height', info=-1)
```

```
## [1]   0  -0.0000e+00       8       919
## [1] ... Stable clustering
```

The output of the *stbc()* constructor is still a *binClstPath* (the *binClstMove* object of previous versions is deprecated). As we included a covariate, *leroybc3* corresponds now to a trivariate binary clustering and therefore its functionality presents some particularities.

Let's see the statistics of the clustering;

```
stts(leroybc3)
```

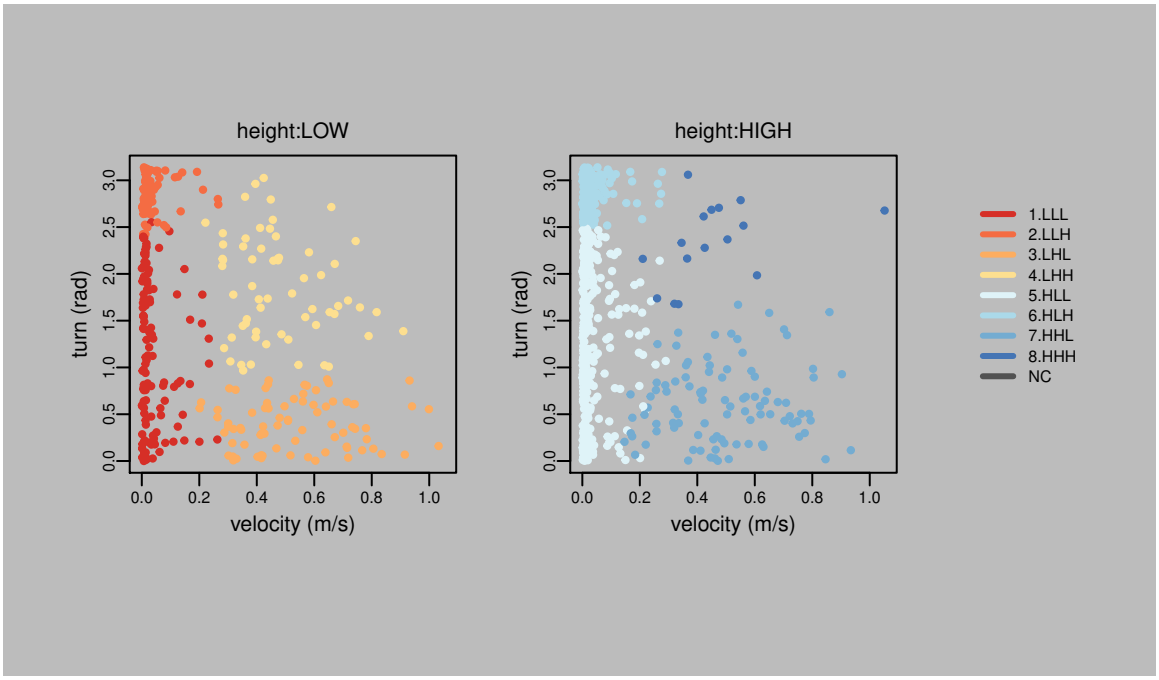
```
##          X1.mn    X1.sd    X2.mn    X2.sd    X3.mn    X3.sd     kn  kn(%)
##  1 LLL   -54.05     5.00     0.04     0.10     1.09     0.80    117  12.73
##  2 LLH   -51.57     5.44     0.03     0.10     2.89     0.20     85   9.25
##  3 LHL   -27.80    18.93     0.56     0.21     0.38     0.27     79   8.60
##  4 LHH   -33.94    19.46     0.49     0.17     1.89     0.61     62   6.75
##  5 HLL    -0.28    25.43     0.04     0.10     1.26     0.79    281  30.58
##  6 HLH     2.24    23.99     0.03     0.10     2.87     0.19    182  19.80
##  7 HHL    29.03     6.36     0.55     0.22     0.65     0.42     96  10.45
##  8 HHH    32.89     5.00     0.52     0.26     2.53     0.50     16   1.74
```

Features are ordered as X1:daytime, X2:velocity and X3:turn. Note that highs and lows for daytime (the solar height in degrees above the horizon) do not necessarily correspond to daytime or night-time clusters (note the negative mean for X1 in HXX clusters). This is so because almost all of the activity of this animal happens during the night and it is more likely to discern different behaviours along night-time.

4.5 Trivariate clustering scatter plot

By default, the *sctr()* function of a trivariate clustering depicts a double scatter-plot corresponding to low and high values of the covariate respectively. This can be changed by means of the parameter *showVars=c()*.

```
sctr(leroybc3, showVars=c(1, 2, 3))
```




```
# showVars=c(1,2,3) is the default option and it is only shown for illustrative purposes
# by default the background colour is set to light-grey to enhance visibility
# the "bg"" parameter allows changing this default behaviour
```

If the R-package *rgl* is installed one can use the function *sct3()* to get a dynamic 3D (i.e. can be zoomed and rotated) plot, more useful for a visual understanding of the clusters.

```
sct3(leroybc3, showClst=c(5, 6, 7, 8))
# with showClst=c() we can restrict the plot to a particular subset of clusters
```

The *sct3()* function is defined for and inherited from the *binClst* class, and therefore intended for a general multivariate clustering. If the number of input features is greater than 3 and *showVars=c()* is not specified, the first three variables are used by default.

4.6 Smoothing

When clustering a time series the EMbC disregards the temporal information. As a result, the output labelling may reveal small (possibly irrelevant) changes in behaviour framed in a broader temporal context (*e.g.* a long-term predominant behavioural mode).

The package includes two possibilities to account for the temporal information in the time series and smooth out the fine grain locality of the output labelling.

The *smth()* function applies a post-smoothing procedure (Garriga et al. 2016) to the output labelling and returns a smoothed copy of the input instance;

```
# dlta is the maximum likelihood difference to accept a relabelling
# dlta=1 (accept all changes) is the default behaviour
postbc3 <- smth(leroybc3, dlta=0.9)
```

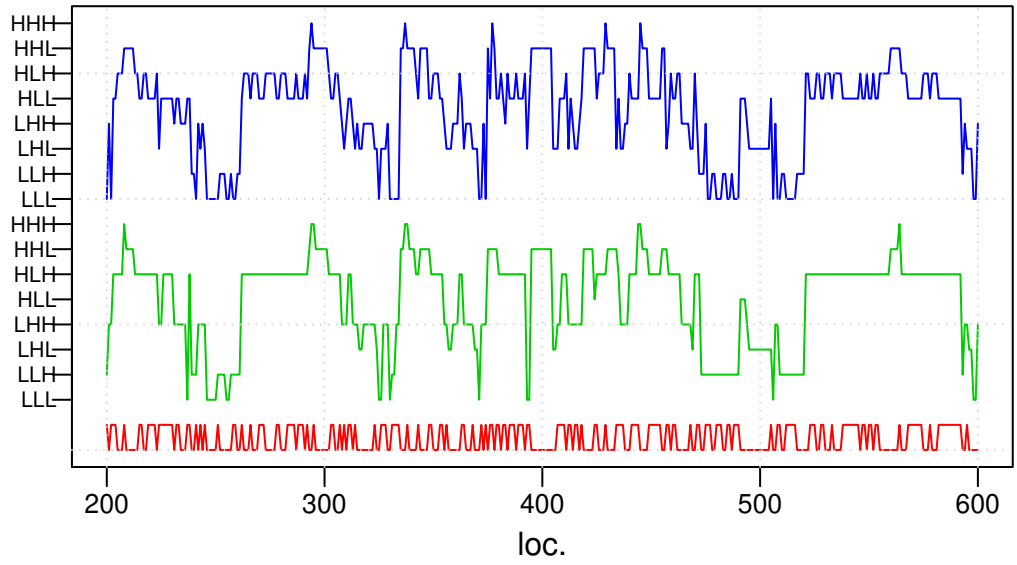
Alternatively, a pre-smoothing of the input data is also possible by means of the parameter *smth* of the *stbc()* constructor.

```
# smth sets the smoothing time window length in hours
prebc3 <- stbc(leroy, smth=1, scv='height', info=-1)
```

```
## [1]   0  -0.0000e+00       8       919
## [1] ... break: optimization cycle
```

The *lblp()* function allows comparing two output labellings, adding a bottom line indicating the differences;

```
lblp(postbc3, smth(prebc3), lims=c(200, 600))
```




```
# of note:
# although performing a pre-smoothing, we can still aply a post-smoothing;
# there is no real need to instantiate the smoothed copy of prebc3;
# this is useful for saving memory in case of long trajectories;
```

4.7 Relabelling

Note that by pre-smoothing the input data, cluster 5 (HLL) has been merged into cluster 6 (HLH) and we get a final clustering with only 7 different behaviours. When merging occurs, the semantics of the final labelling is somewhat misleading because the final labelling is only a result of how the algorithm evolved until reaching the merging point. In any case, the label should be read as HLX, that is, by taking into account that the last feature (in this case the turn) is meaningless given the values of the rest (*i.e* turn can be either H or L given H values of daytime and L values of velocity).

Using the *pkml()* function we can visualize which locations correspond to cluster HLH;

```
pkml(smth(prebc3), showClst=6, display=TRUE)
```

By combining the spatially clustered distribution of locations HLX (Figure 1) with the semantics of the cluster (high daytime, low velocity), we could tell that these locations are most probably indicating the nests.

Obviously, the package does not deal with labels like HLX. However, one can change labels as desired (even to manually force the merging of two clusters). In this case, we would probably feel more comfortable by relabelling the cluster HLH (cluster number 6) as HLL (cluster number 5) to suggest a more clear semantics of *resting* behaviour;

```
rlbl(prebc3, 6, 5)
```

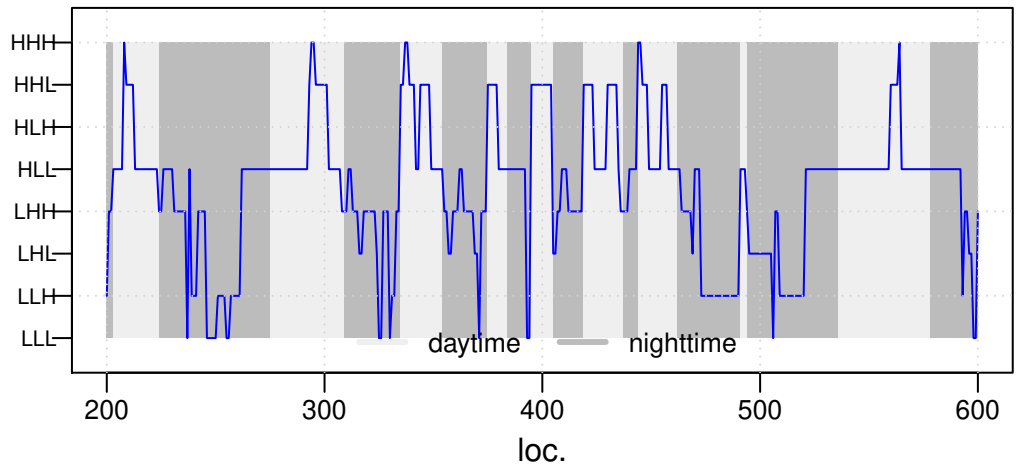
Figure 1: Fisher (*Martes pennati*) foraging trajectory. A kml point-wise view of the annotated trajectory showing only the HLH locations. The blob shows the spatial clustering of these locations most likely indicating the nest.

Note that the function *rlbl()* does not return a relabelled copy of the input instance, instead it relabels the self instance. Nonetheless, the parameters of the clustering remain unchanged. The relabelling is effective only for visualization purposes and can be easily reversed by means of the parameter "reset".

4.8 Validation versus external information

The *chkp()* function is similar to *lblp()* but plots the labelling profile versus a control variable (*e.g.* environmental information). The control variable must be given as a numeric vector that is depicted as coloured background bars (with specific parameters to control the colouring and legend labels);

```
chkp(smth(prebc3), lims=c(200, 600))
```




```
# the solar height is the control variable used by default;
# note the relabelling we did before;
```

5 Class: binClstStck

The *binClstStck* is an extension (not a child class) of the *binClstPath* class particularly designed to work with multiple trajectories. This is intended for population level analysis from trajectories of several individuals, or

period level analysis by splitting long trajectories of several years.

To illustrate this let's figure out two trajectories from our example path, simulating two different individuals;

```
tmp <- runif(nrow(expth))
# simulated trajectory of individual 1
expth1 <- expth[which(tmp<=0.5), ]
# simulated trajectory of individual 2
expth2 <- expth[which(tmp>=0.5), ]
```

To perform the clustering of a stack of trajectories we pass the individual trajectories to the *stbc()* constructor as a list (either of data.frame trajectories, *Move* objects, or a mixture of them);

```
# we can combine data.fame trajectories and move objects
# only for illustrative purposes !!!
mystck <- stbc(list(expth1, expth2, leroy), info=-1)
```

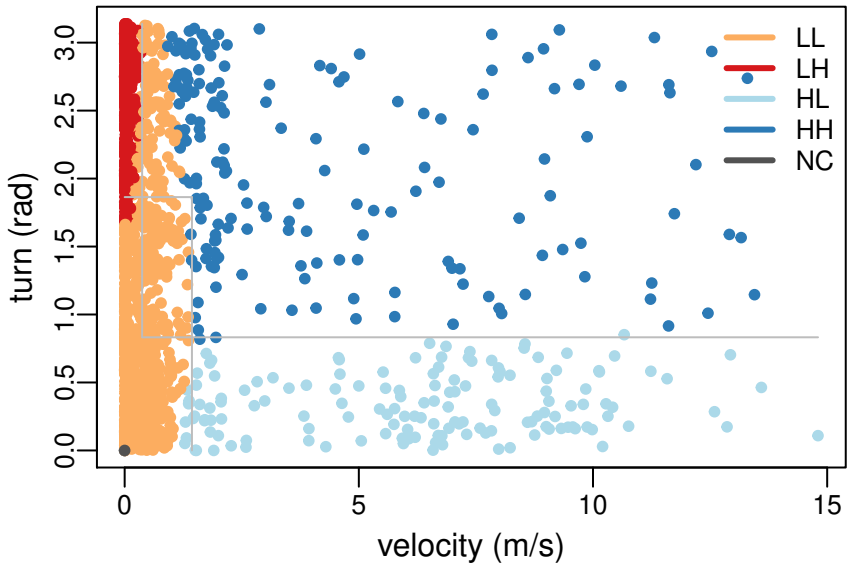
```
## [1]   0  -0.0000e+00       4      1519
## [1] ... Stable clustering
```

In this case, the *stbc()* constructor returns an instance of the *binClstStck* class. In general, all the functionality described for a *binClst* class will work for a *binClstStck* instance;

```
stts(mystck)
```

```
##          X1.mn    X1.sd    X2.mn    X2.sd     kn  kn(%)
##  1 LL     0.34     0.35     0.78     0.79    761  50.10
##  2 LH     0.04     0.10     2.67     0.43    434  28.57
##  3 HL     6.58     3.42     0.33     0.22    147   9.68
##  4 HH     3.96     3.66     2.15     0.70    174  11.45
```

```
sctr(mystck)
```



The exception is the *cnfm()* function. This function will work only if expert's labelling is supplied for all trajectories in the stack (in our example, *leroy* does not have expert's labelling);

```
cnfm(mystck)
```

```
## Error: no reference labels for obj
```

```
# this will only work when expert labelling is given for all trajectories in the stack
```

5.1 binClstStck slots

It is worth noting that a *binClstStck* instance is not a binary clustering object itself. Instead, it is an object with two slots:

```
slotNames(mystck)
```

```
## [1] "bCS" "bC"
```

- slot mystck@bC is a *binClst* object with the population level clustering, thus it has no path associated, and functions like *view()*, *pkml()* or *bkml()* will not work;

```
class(mystck@bC)
```

```
## [1] "binClst"
## attr(,"package")
## [1] "EMbC"
```

- slot mystck@bCS is a list of *binClstPath* objects, which are the results of the population level clustering upon each individual;

```
class(mystck@bCS)
```

```
## [1] "list"
```

Each element in mystck@bCS is a *binClstPath* instance corresponding to each individual path given in the input data list;

```
lapply(mystck@bCS, class)
```

```
## [[1]]
## [1] "binClstPath"
## attr(,"package")
## [1] "EMbC"
##
## [[2]]
## [1] "binClstPath"
## attr(,"package")
## [1] "EMbC"
##
## [[3]]
## [1] "binClstPath"
## attr(,"package")
## [1] "EMbC"
```

It is important to keep this in mind when applying the above functions to either the population (mystck@bC, a *binClst* instance) or the individual (mystck@bCS[[i]], a *binClstPath* instance) levels.

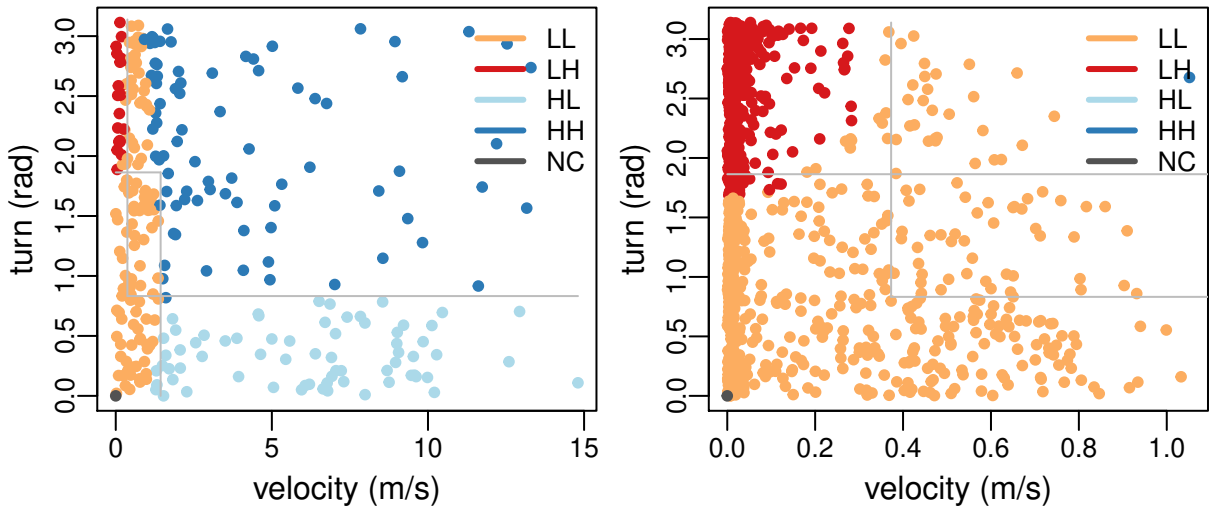
5.2 Select an individual out of the stack

For ease of use, the function *slct()* allows selecting an individual's clustering out of the population level;

```
bcInd1 <- slct(mystck,1)
```

As usual, it is not necessary to instantiate each individual;

```
sctr(slct(mystck, 1))  # left panel
sctr(slct(mystck, 3))  # right panel
```




```
# sctr(slct(mystck,1)) yields the same output as sctr(bcInd1) or sctr(mystck@bCS[[1]]);
```

5.3 Comparing individual's behaviour with population's average behaviour

We can use all the functionality of a *binClstPath* object that allows comparisons (*e.g. sctr()*, *lblp()*, *cnfm()*) to make numeric assessments or visualizations of diferences among individuals or among individuals and population:

- we can compare individuals with their correspondent out of the population clustering;

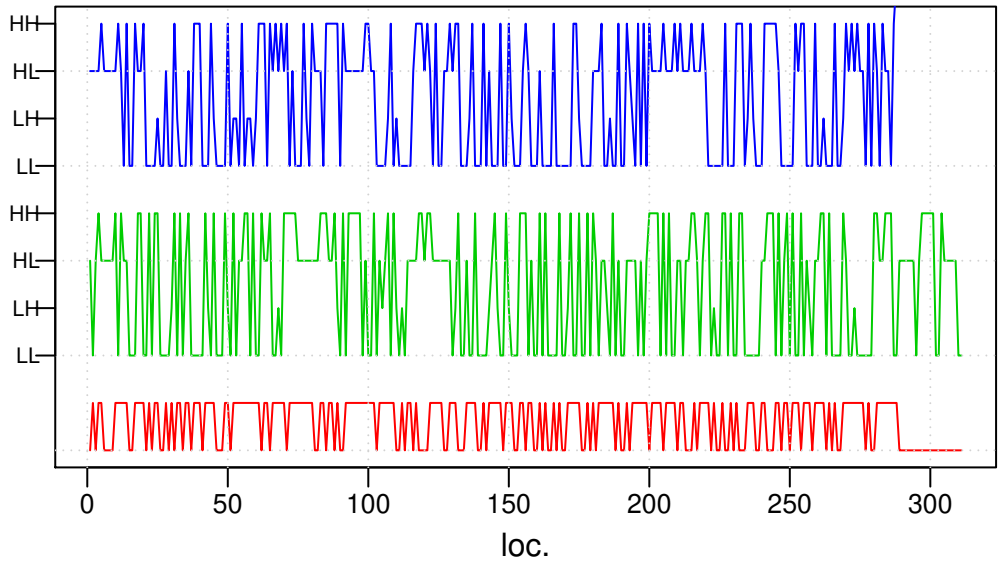
```
cnfm(stbc(expth1, info=-1), slct(mystck, 1))
```

```
## [1]   0  -0.0000e+00       4       288
## [1] ... Stable clustering
##      cls.01 cls.02 cls.03 cls.04   mrg.   Prc.
## 1.LL     52      0     14      2   0.24   0.76
## 2.LH     68     20      0     35   0.43   0.16
## 3.HL      0      0     54      2   0.20   0.96
## 4.HH      0      0      0     40   0.14   1.00
##      ------ ------ ------ ------ ------ ------
## mrg    0.42   0.07   0.24   0.28   0.58   0.72
## Rcl    0.43   1.00   0.79   0.51   0.68    NaN
## Fms    0.55   0.28   0.87   0.68    NaN   0.60
```

```
# stbc(expth1, info=-1) is the individual level clustering corresponding to individual 1;
# slct(mystck, 1) is the population level clustering corresponding to individual 1;
```

- or we can compare individuals within the population;

```
lblp(slct(mystck, 1), slct(mystck, 2))
```



References

Adler, Daniel, Duncan Murdoch, and others. 2015. *Rgl: 3D Visualization Using Opengl*. http://CRAN.R-project.org/package=rgl.

Azzalini, Adelchi, and Alan Genz. 2015. *The R Package `mnormt`: The Multivariate Normal and t Distributions (Version 1.5-3)*. http://azzalini.stat.unipd.it/SW/Pkg-mnormt.

Bivand, Roger, and Nicholas Lewin-Koh. 2015. *Maptools: Tools for Reading and Handling Spatial Objects*. http://CRAN.R-project.org/package=maptools.

Bivand, Roger S., Edzer Pebesma, and Virgilio Gomez-Rubio. 2013. *Applied Spatial Data Analysis with R, Second Edition*. Springer, NY. http://www.asdar-book.org/.

Garriga, Joan, John R. B. Palmer, Aitana Oltra, and Frederic Bartumeus. 2016. "Expectation-Maximization Binary Clustering for Behavioural Annotation." *PLoS ONE* 11 (3): 1–26. https://doi.org/10.1371/journal.pone.0151984.

Kranstauber, Bart, and Marco Smolla. 2015. *Move: Visualizing and Analyzing Animal Track Data*. http://CRAN.R-project.org/package=move.

Neuwirth, Erich. 2014. *RColorBrewer: ColorBrewer Palettes*. http://CRAN.R-project.org/package=RColorBrewer.

Pebesma, Edzer J., and Roger S. Bivand. 2005. "Classes and Methods for Spatial Data in R." *R News* 5 (2): 9–13. http://CRAN.R-project.org/doc/Rnews/.

R Core Team. 2015. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. http://www.R-project.org/.