Vignette moveNT

Guillaume Bastille-Rousseau

February 26, 2020

Simulating movement strategies - *sim_mov*

The function *sim_mov* generates movement trajectories including patches and movement between patches. Movement within patches can follow an Ornstein-Uhlenbeck process (based on *simm.mou* function from package *adehabitatLT*) or two-states movement model (based on *simmData* function from package *moveHMM*). Movement between patches is following a brownian bridge movement model (based on *simm.bb* function from package *adehabitatLT*). Generated outputs are of the class *ltraj* from package *adehabitatlt*.

```
# Simulating migration with two-states model
mig<-sim_mov(type="2states", npatches=2, ratio=2, nswitch=25, ncore=150, grph=F)
mig
```

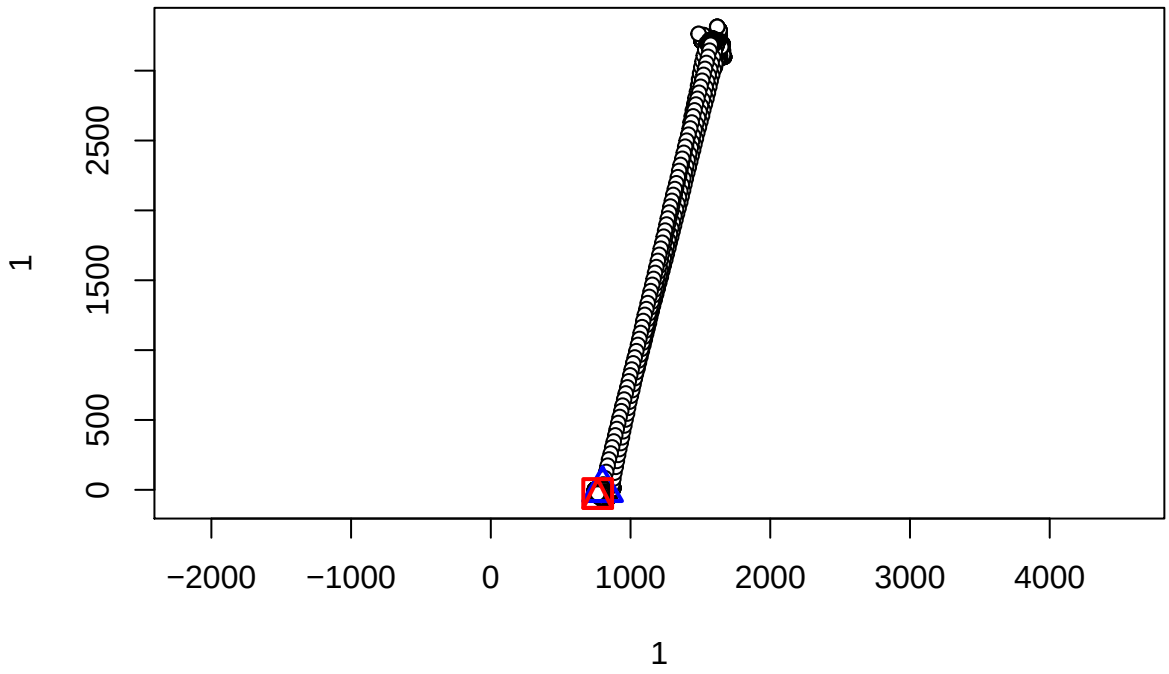
```
##
## *********** List of class ltraj ***********
##
## Type of the traject: Type II (time recorded)
## * Time zone: GMT *
## Regular traject. Time lag between two locs: 1 seconds
##
## Characteristics of the bursts:
##   id burst nb.reloc NAs          date.begin            date.end
## 1 id    id     4350   0 1960-01-01 00:00:01 1960-01-01 01:12:30
##
##
##  infolocs provided. The following variables are available:
## [1] "out.Corri"
```

```
head(ld(mig))
```

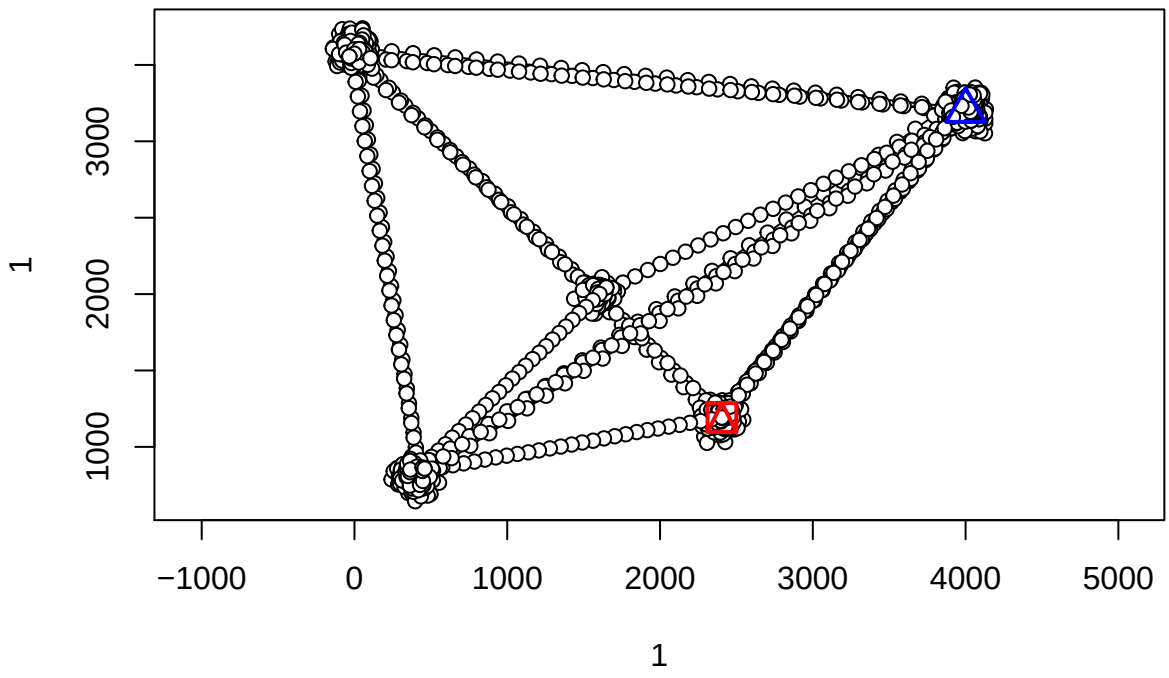
```
##          x           y                date           dx          dy       dist
## 1 800.0000  0.00000000 1960-01-01 00:00:01 -0.001947019 -0.01681934 0.01693166
## 2 799.9981 -0.01681934 1960-01-01 00:00:02  0.863845829 -7.31222824 7.36307758
## 3 800.8619 -7.32904758 1960-01-01 00:00:03  2.166620286 -2.12967349 3.03805079
## 4 803.0285 -9.45872107 1960-01-01 00:00:04 -0.125838917  0.01311456 0.12652045
## 5 802.9027 -9.44560651 1960-01-01 00:00:05  0.388805077  2.69764791 2.72552263
## 6 803.2915 -6.74795861 1960-01-01 00:00:06 -0.682436426  0.12227990 0.69330502
##   dt          R2n  abs.angle  rel.angle id burst out.Corri
## 1  1 0.000000e+00 -1.6860441         NA id    id         2
## 2  1 2.866811e-04 -1.4532042  0.2328399 id    id         2
## 3  1 5.445781e+01 -0.7767987  0.6764055 id    id         2
## 4  1 9.863933e+01  3.0377505 -2.4686361 id    id         2
## 5  1 9.764503e+01  1.4276546 -1.6100959 id    id         2
```

```
## 6   1 5.636882e+01  2.9642928  1.5366381 id    id         2
```

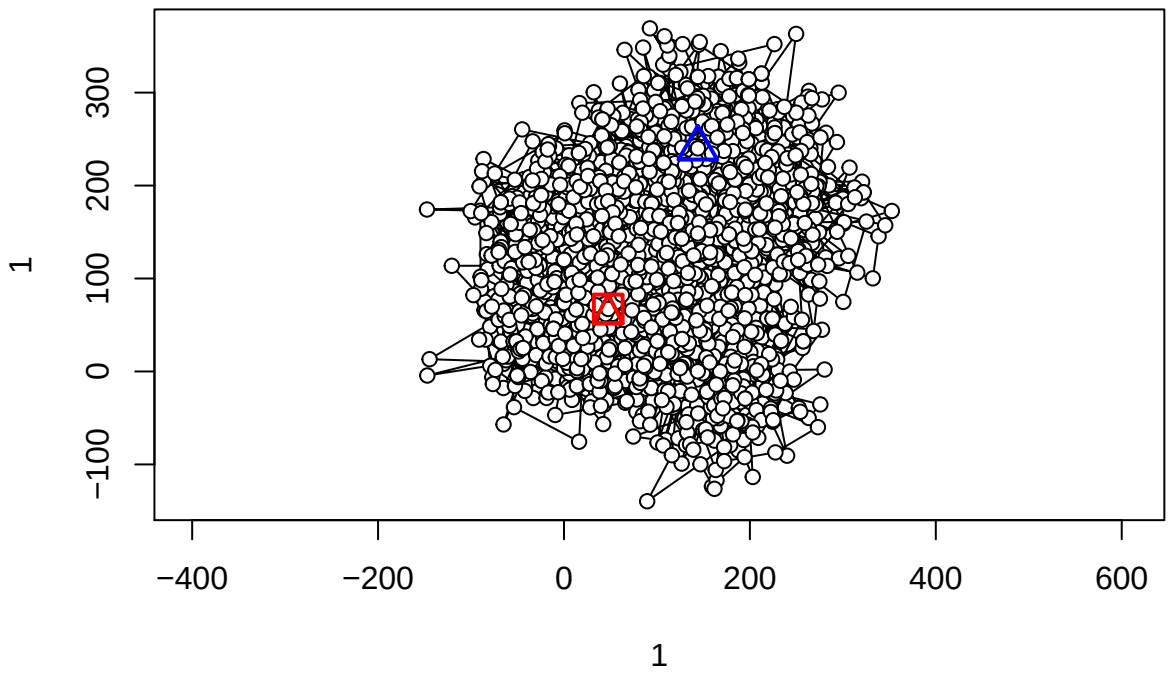
```
plot(mig)
```




```
# Simulating multi-patches movement with Ornstein-Uhlenbeck process
patches<-sim_mov(nswitch=25, ncore=150, ratio=5, type="OU", npatches=5, grph=T)
```




```
# Simulating sedentary movement
seden<-sim_mov(type="OU", npatches=10, spacecore=12, ratio=3, nswitch=150, ncore=20, grph=T)
```



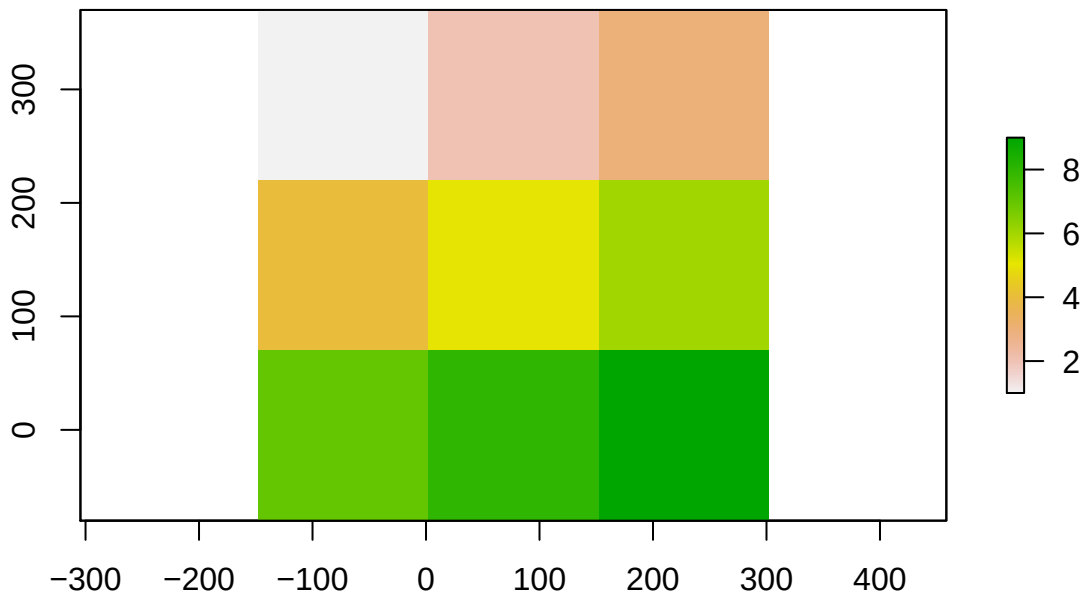
Converting movement to adjacency matrix - *traj2adj*

The function *traj2adj* converts a trajectory object of class *ltraj* to an adjacency matrix. This is done by overlapping a grid over the relocation data and tallying the number of transitions among each pixel. Users need to specify the grid size, which can be based on distance travelled. The function *quant* is a wrapper that allows to estimate quantiles of step length distribution from a *ltraj* object. Output produced by *traj2adj* is a list containing the adjacency matrix, the grid used (raster format), and a raster indicating pixel numbers that are occupied. These rasters are used by other functions such as *adj2stack* and *clustnet*.

```
# Using sedentary movement and user specific grid-size
adj_seden<-traj2adj(seden, res=150) #Pixel size of 150m
adj_seden[[1]] # Adjency matrix
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    2    0    0    8    5    0    0    0    0
##  [2,]    5  124   41    5   60   17    0    0    0
##  [3,]    0   43  161    0   19   71    0    0    0
##  [4,]    5    4    0  133   83    0   29   18    0
##  [5,]    3   62   25   83  899   82   11  135    9
##  [6,]    0   18   67    0   83  395    0   11   15
##  [7,]    0    0    0   21   16    0   69   44    0
##  [8,]    0    0    0   22  133    8   41  327   56
##  [9,]    0    0    0    0   10   17    0   52  191
```

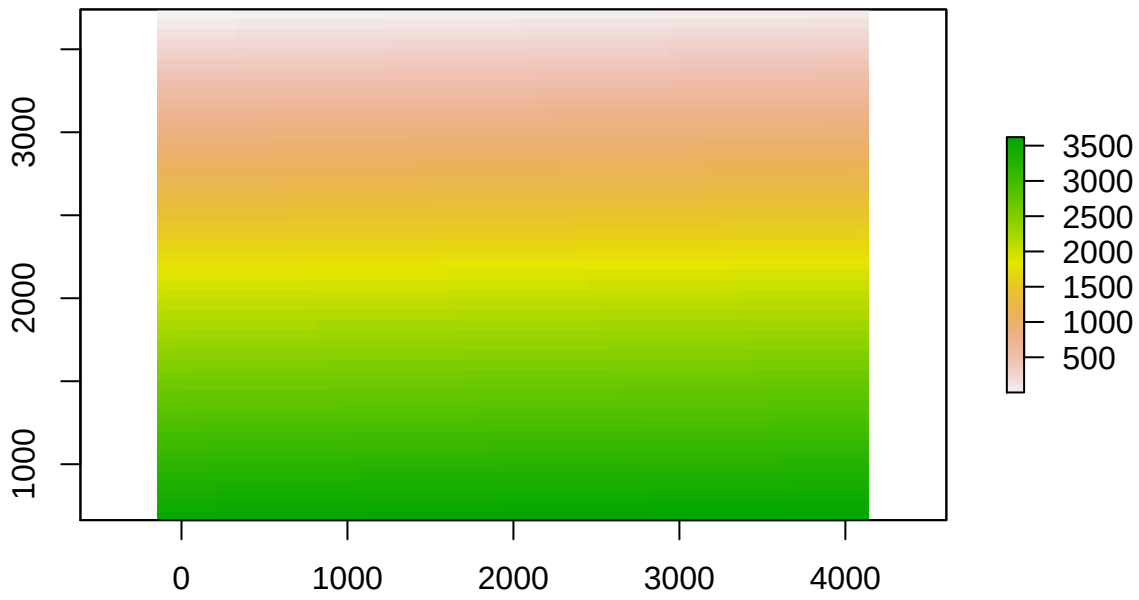
```r
plot(adj_seden[[2]]) #Plot grid used
```



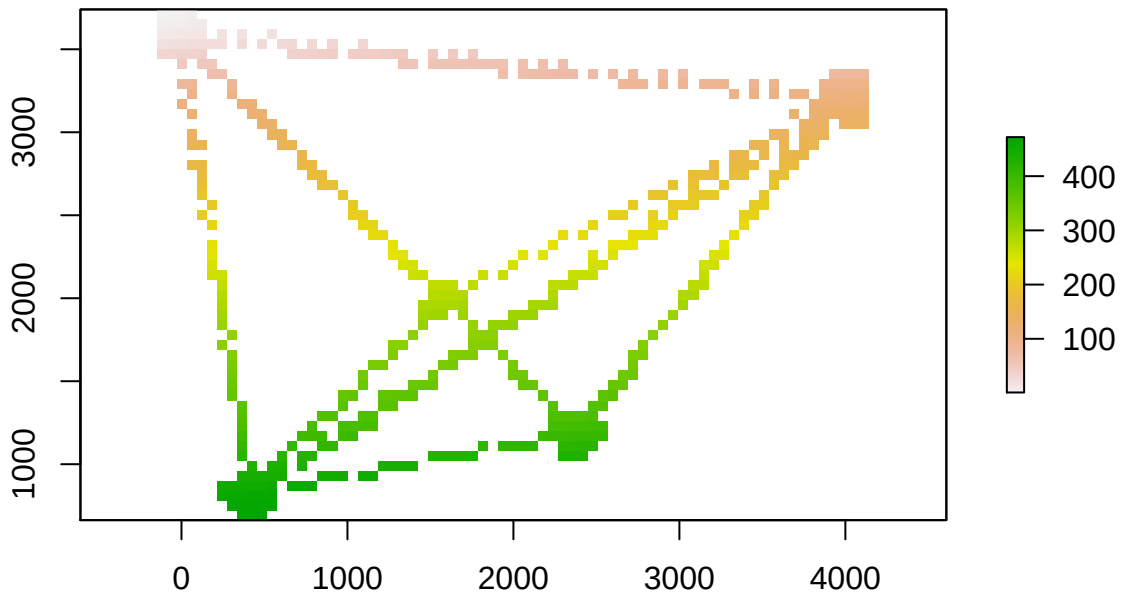

```r
# Using multi-patches movement and median distance travelled
adj_patches<-traj2adj(patches, res=quant(patches, p=0.5)) #Grid size based on median
dim(adj_patches[[1]]) # Size of the adjacency matrix
```

```
## [1] 472 472
```

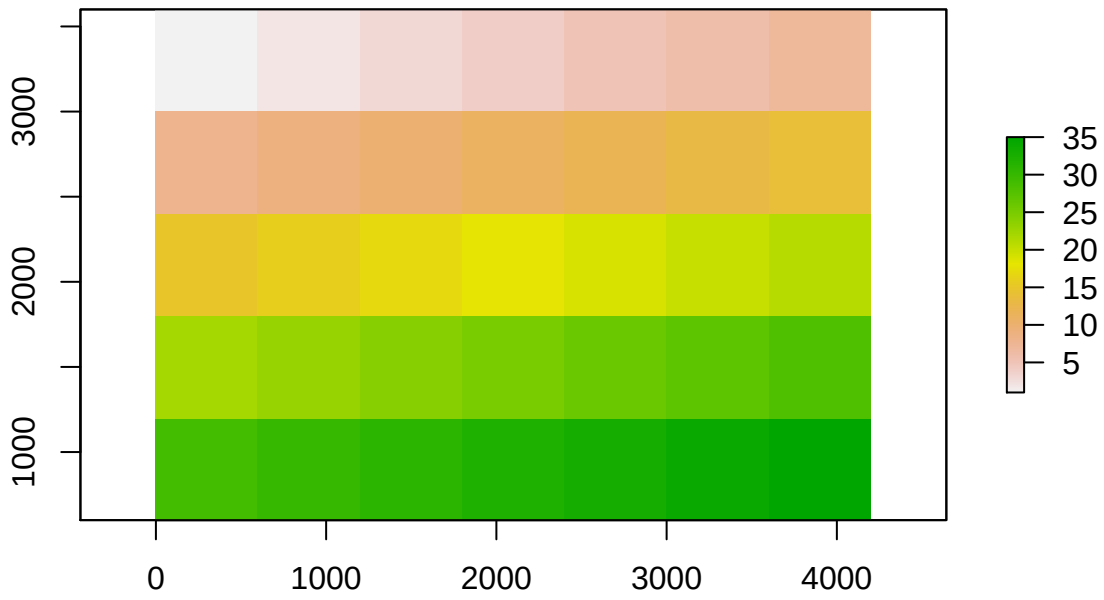
```r
plot(adj_patches[[2]]) #Plot grid used
```




```
plot(adj_patches[[3]]) #Plot occupied pixels
```



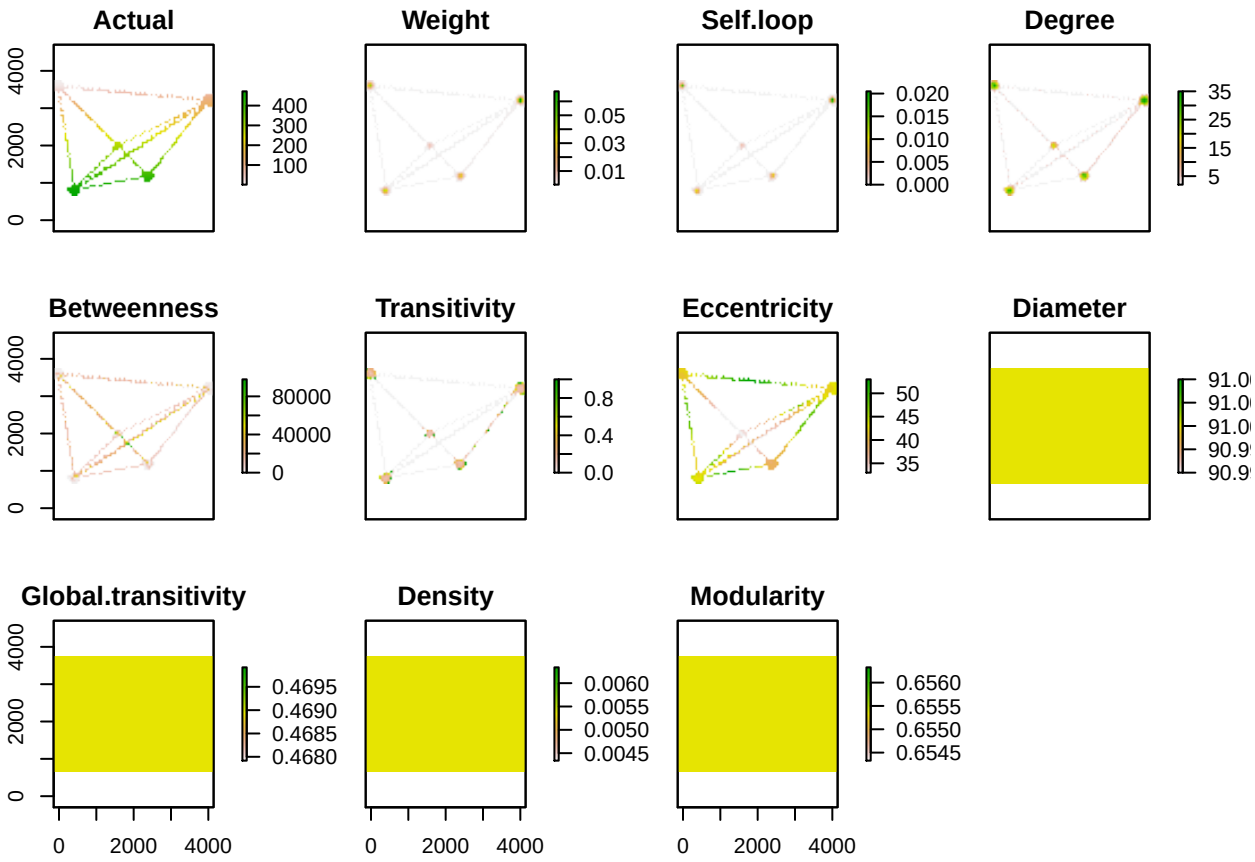

```
# Using user defined grid
ras<-raster(nrows=10, ncols=10, xmn=0, ymn=0, xmx=6000, ymx=6000)
adj_patches2<-traj2adj(patches, res=quant(patches, p=0.5), grid=ras) #Grid size based on median
plot(adj_patches2[[2]]) #Crop version of the grid created
```



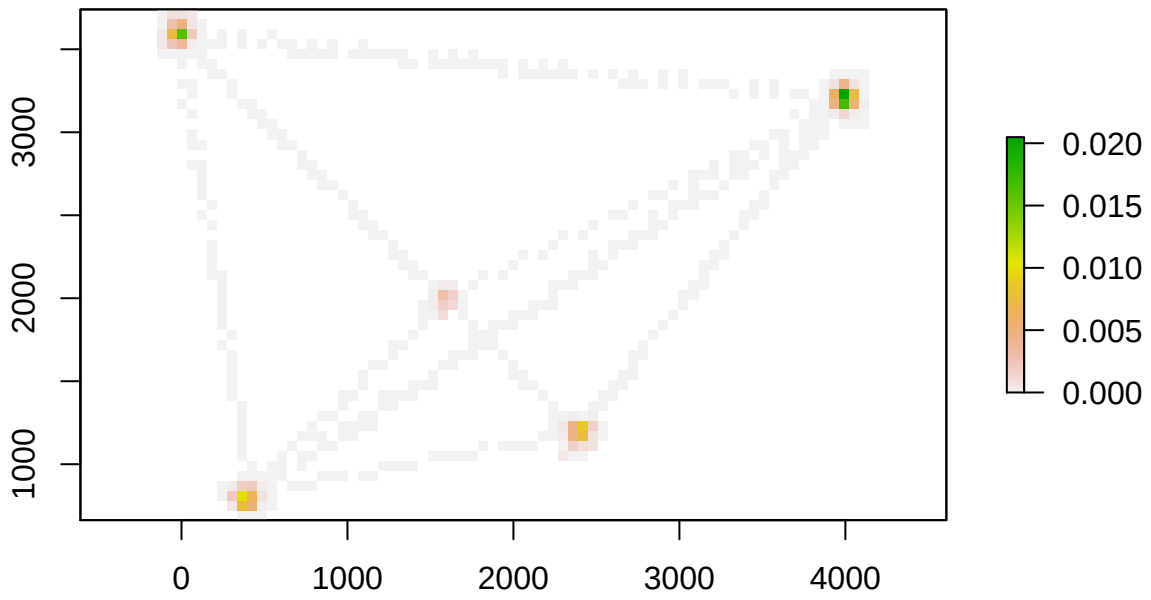
Calculation of network metrics - *adj2stack*

The function *adj2stack* takes the output of function *traj2adj* and calculates a series of node- and graph-level metrics. Each metric is stored as a individual raster and the output is a raster stack combining each metric. Graph-level metrics are also stored as a raster, each containing an unique value. The function *graphmet* extracts graph-level metrics. The function *val* extracts only the occupied cells (remove NA) in a raster and allows the calculation of statistics from node-level metrics.

```
# Using multi-patches movement and median distance travelled
stck<-adj2stack(adj_patches, grph=F) #Plot the node-level metrics at the same time
plot(stck) #Plot also the graph-level metrics (not really useful)
```




```
plot(stck[[3]]) #Plot only one metric (degree)
```




```
graphmet(stck) # Extract graph-level metrics
```

```
##            Diameter Global.transitivity             Density          Modularity
##        91.000000000          0.468917577          0.005339343          0.655327559
```

```
cv(val(stck, 4)) #Extract coefficient of variation of node-level betweenness.
```

```
## [1] 127.263
```

Clustering of node level metrics - *clustnet*

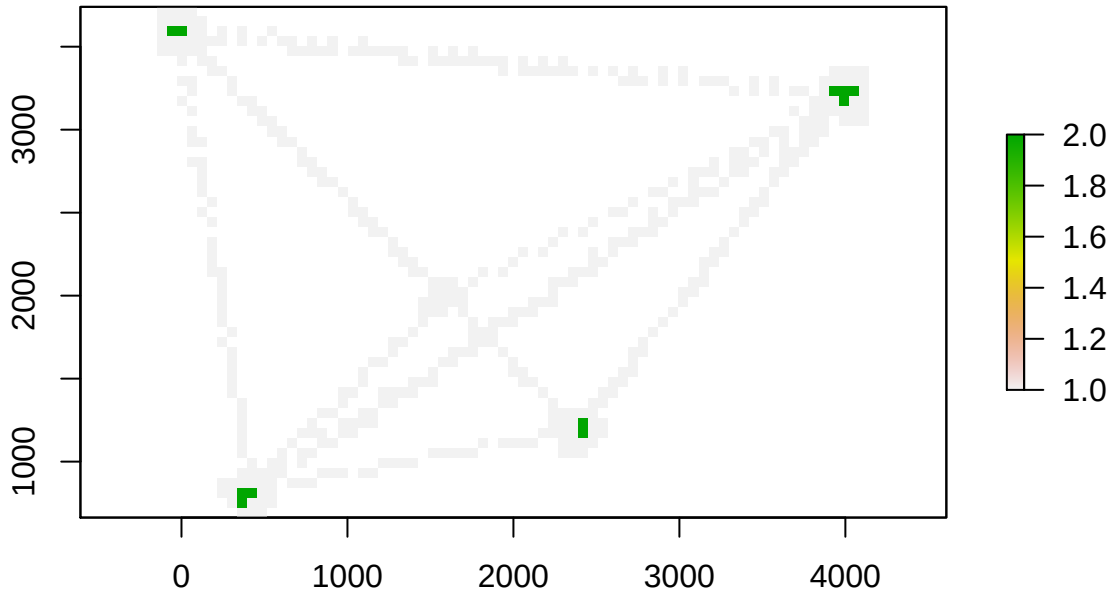
The function *clustnet* applies a normal mixture model to node-level metrics in order to cluster them into separate groups (default = 2). The function takes the output of function *adj2stack* with the user specifying the metric to cluster and the number of groups. Return a list containing output of function *Mclust* from package *mclust* and a raster displaying classification.

```
# Using multi-patches movement and median distance travelled
clust2<-clustnet(stck, id=3, nclust=2, grph=F) # Clustering of degree in two groups
clust3<-clustnet(stck, id=4, nclust=3, grph=F) #Clustering of betweenness in three groups
summary(clust2[[1]])
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
```

```
## Mclust E (univariate, equal variance) model with 2 components:
##
##  log-likelihood   n df      BIC      ICL
##        2540.225 472  4 5055.822 5055.337
##
## Clustering table:
##   1   2
## 461  11
```

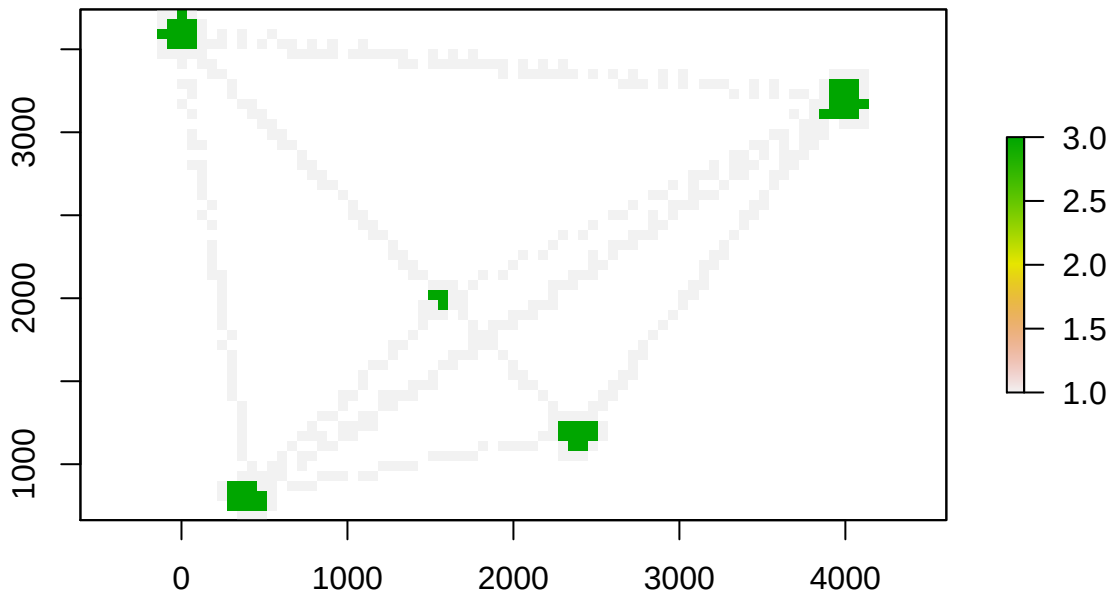
```
plot(clust2[[2]])
```




```
summary(clust3[[1]])
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust E (univariate, equal variance) model with 3 components:
##
##  log-likelihood   n df       BIC       ICL
##       -1294.02 472  6 -2624.981 -2859.393
##
## Clustering table:
##   1   2   3
## 423   0  49
```

```
plot(clust3[[2]])
```



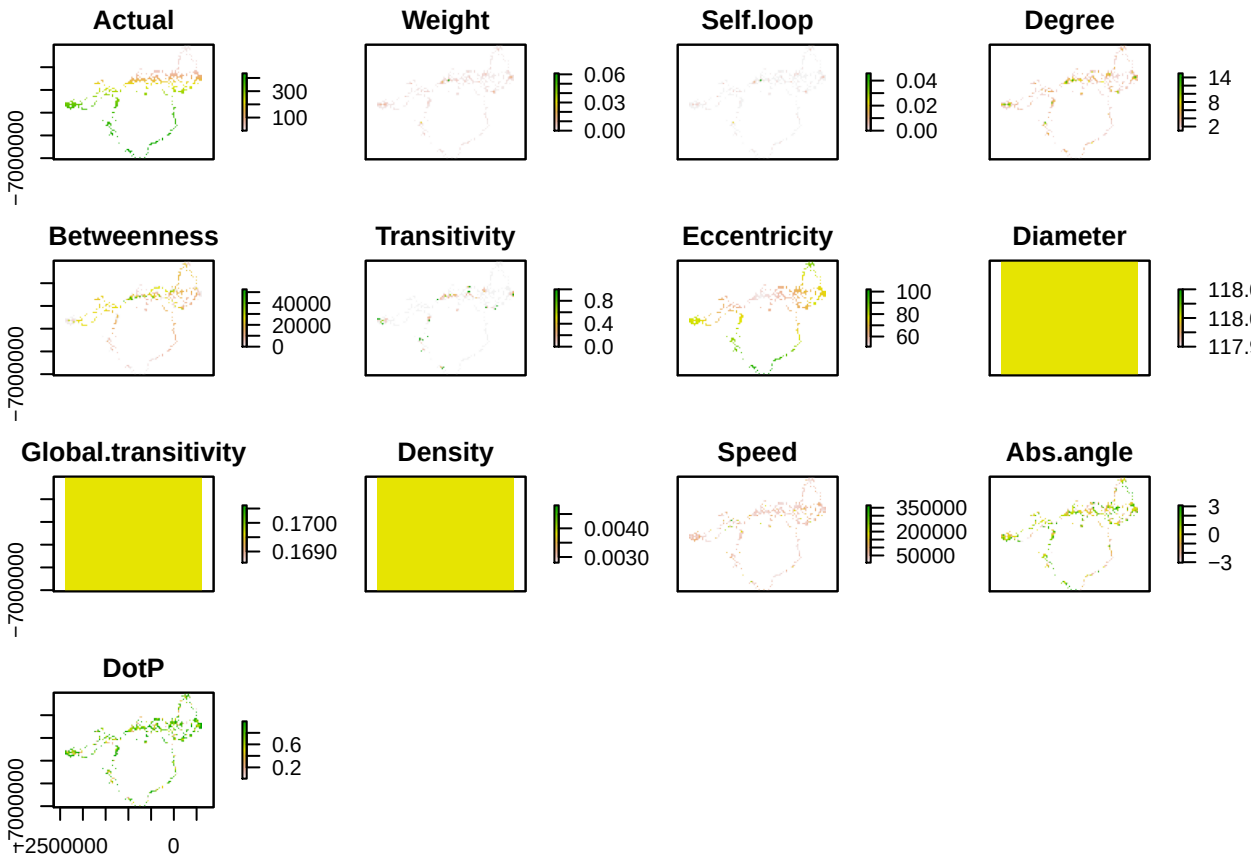
Looping over all individuals *loop*

The function *loop* is a wrapper of *traj2adj* and *adj2stack* applied to all individuals within a trajectory. The function will keep the same grid for all individuals. The user simply need to specify the trajectory object and the grid size. The loop function also adds additional movement properties regarding speed, absolute angle, and turning angle.

```
data(albatross) #Load a traj object from adehabitatLT
out1<-loop(albatross, res=35000)
```

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

```
plot(out1[[1]]) #Plot the first individual
```



Mosaic individual

Even if the the function *loop* perform the analysis forevery individuals the outputs produced are at the individual-level. The function *mosaic_network* can combine the different individual levels into a single raster representation. When multiple individuals overlap, the function apply a function (mean or max) to calculate a population-level value for that pixel. To use the function, the user needs to specify which variable to mosaic (using index), whether to scale the individual layers (recommended) and the function to apply. We recommend to use mean for degree and weight and max for the betweenness.

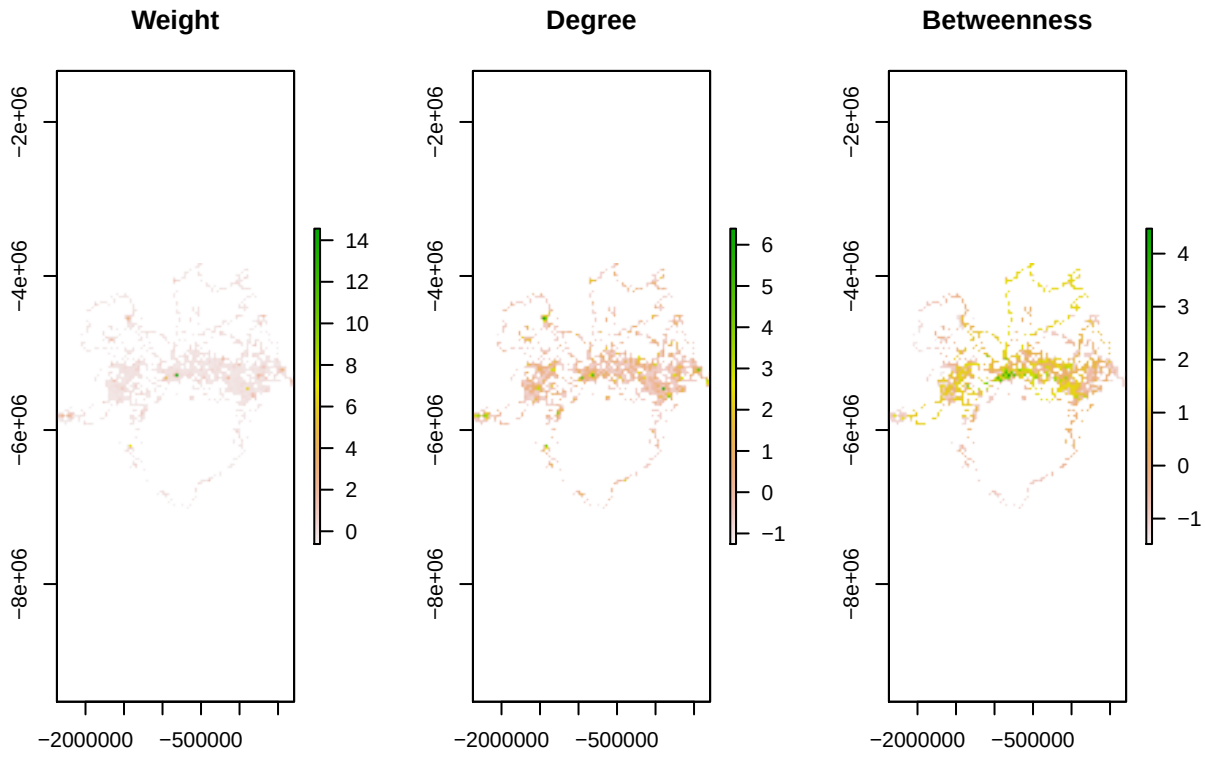
```
mean_weight<-mosaic_network(out1, index=2, sc=T, fun=mean) #Perform mean weight (not-interpolated)

#writeRaster(...

mean_degree<-mosaic_network(out1, index=4, sc=T, fun=mean) #Perform mean weight (not-interpolated)


max_between<-mosaic_network(out1, index=5, sc=T, fun=max) #Perform mean weight (not-interpolated)

par(mfrow=c(1,3))
plot(mean_weight, main="Weight")
plot(mean_degree, main="Degree")
plot(max_between, main="Betweenness")
```



Linear interpolation

As can be seen in the last plot produced, one of the limitation of the current approach is that it creates gaps in areas where no locations are observed (only pixels with gps locations in them have values). This can sometime limit interpretability or the visual appeal of the maps produced. To assist with this, we created a linear interpolation approach that can be applied at the individual level network calculation (i.e. after *loop*). The interpolation linearly interpolate each step (i.e. straight line) and assign the network metric of each starting location to the whole step. When multiples overlap in a pixel, a function is applied to summarize these steps (e.g. mean or max). This function will take an output from *loop* and performed the interpolation for five metrics (weight, degree, betweenness, speed, and turning angles). We recommend to take the mean for weight, degree, betweenness, and speed, the max for betweenness, and the dot-product for the turning angles (default).

```
data(albatross) #Load a traj object from adehabitatLT
out1<-loop(albatross, res=35000)
```

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

```
out2<-interpolation(albatross, out1) #This is very slow, more than 5 minutes
```

```
##
```

```
  |
  |                                                                      |   0%
  |
  |==============                                                        |  20%
  |
  |============================                                          |  40%
  |
  |==========================================                            |  60%
  |
  |========================================================              |  80%
  |
  |======================================================================| 100%
```

```
mean_mean_degree<-mosaic_network(out2, index=2, sc=T, fun=mean)
max_max_between<-mosaic_network(out2, index=3, sc=T, fun=max)
mean_mean_speed<-mosaic_network(out2, index=4, sc=T, fun=mean)
mean_dot_TA<-mosaic_network(out2, index=5, sc=T, fun=mean)
par(mfrow=c(2,2))
plot(mean_mean_degree, main= "Degree")
plot(max_max_between, main="Betweenness")
plot(mean_mean_speed, main="Speed")
plot(mean_dot_TA, main="Directionality")
```