funLBM: univariate and multivariate functional co-clustering

Amandine Schmutz, Charles Bouveyron, Julien Jacques

28 octobre 2019

Description

The funLBM algorithm (Bouveyron et al, 2018; Schmutz et al., 2019) allows to cluster simultaneously rows and columns of a functional data table. Which means that at the intersection of one row and column there is at least one curve. For example one row corresponds to a household, one column corresponds to a day of the year and at the intersection we have the electric consumption monitored every 30 min.

Load package

```
library(funLBM)
```

Data

We are going to work on simulated data as if we are in the user case described in the Description section for the univariate case and where we have the electric consumption and the temperature monitored for 90 days every 30 min and for 100 households in the multivariate case.

```
library(funLBM)
set.seed(1)
data.univariate<-simulateData(n=100,p=90,t=48)
data.multivar<-simulateData2(n=100,p=90,t=48)
```

Example of co-clustering univariate functional data

In this first part we are going to cluster data according to one functional variable.

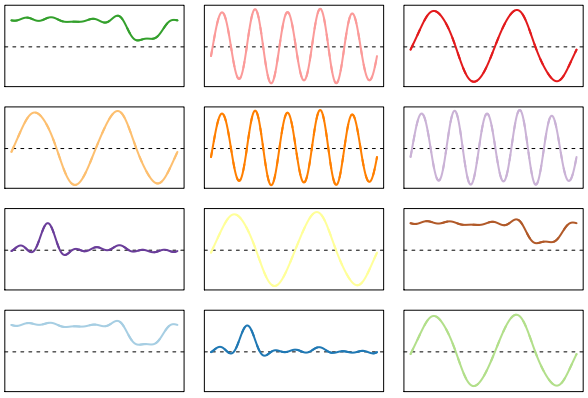
Basic code example

First you have to decide the type of smoothing you want: fourier or spline and the number of basis you want to use. There isn't straigth rules about how to choose those 2 parameters, we can nevertheless recommend the use of a Fourier basis in case of a repetitive pattern and spline otherwise.

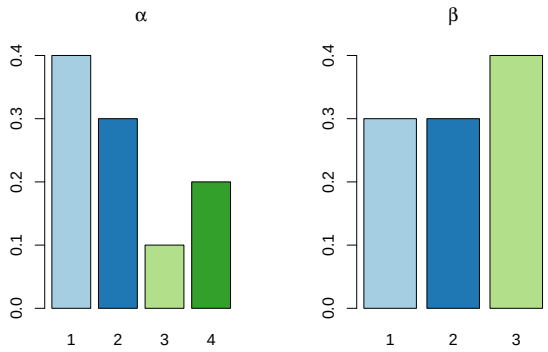
```
library(funLBM)
res.uni<-funLBM(data.univariate$data,K=4,L=3,
                basis.name = "fourier",nbasis=15,init="funFEM")
```

You can visualize the results, the first graph shows the block means functions. The second plot shows the group proportions, for row clusters (alpha) the first group contains 40% of the households, the second 30%, the third 10% and the last 20%. The column cluster (beta) is read in the same way with the first and second groups that contain 30% of the households and the third one 40%. The third plot overlays the block mean functions for each column cluster.

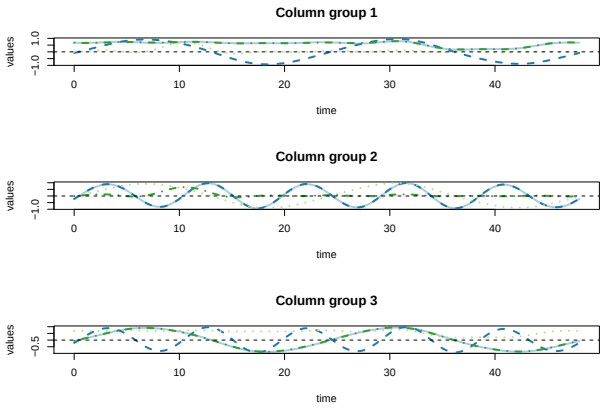
```
plot(res.uni,type='blocks')
```



```
plot(res.uni,type='proportions')
```




```
plot(res.uni,type='means')
```



Then you can evaluate the co-clustering results, checking if the algorithm gets back to the true row and column partitions.

```
ari(res.uni$col_clust,data.univariate$col_clust)
#> [1] 1
ari(res.uni$row_clust,data.univariate$row_clust)
#> [1] 1
```

Both ARI is equal to 1, thus we can conclude that with funFEM initialization the funLBM algorithm gets back to the correct partitions.

Looking for best partition

Most of the time we do not know how many clusters we have in the data and or aim is to estimate the best partition. In order to do that we use the ICL criterion as defined in (Bouveyron et al, 2018). Thus we will ask funLBM algorithm to look from 2 to 6 clusters for both column and row clusters and to give us the best partition according to the ICL criterion (depending on you computer features it may take a few minutes, you can speed up the computation with mc.cores parameter).

```
res.uni<-funLBM(data.univariate$data,K=2:6,L=2:6,basis.name = "fourier",nbasis=15,init="funFEM",display=
#>    K L       icl
#> 8  4 3 -69365.60
#> 7  3 3 -69545.38
#> 12 3 4 -70132.09
#> 13 4 4 -70152.75
#> 9  5 3 -70531.99
#> 10 6 3 -70876.60
#> 17 3 5 -71199.46
#> 18 4 5 -71672.34
#> 22 3 6 -71951.39
#> 6  2 3 -72334.63
#> 11 2 4 -72666.20
#> 16 2 5 -72749.40
#> 23 4 6 -72938.81
#> 15 6 4 -73052.61
#> 21 2 6 -73299.90
#> 19 5 5 -73656.80
#> 24 5 6 -76120.94
```

```
#> 2  3 2 -77219.61
#> 3  4 2 -77992.55
#> 4  5 2 -78424.11
#> 5  6 2 -78969.08
#> 1  2 2 -79002.63
#> 14 5 4        NA
#> 20 6 5        NA
#> 25 6 6        NA
```

Results show that the best partition is the one that maximizes the ICL criterion with K=4 and L=3, the column with no label correspond to the order in which each combination has been tested by the algorithm. It is those results that are stored in res.uni object, you can as previously directly apply plot functions on this object to obtain summary graphics.

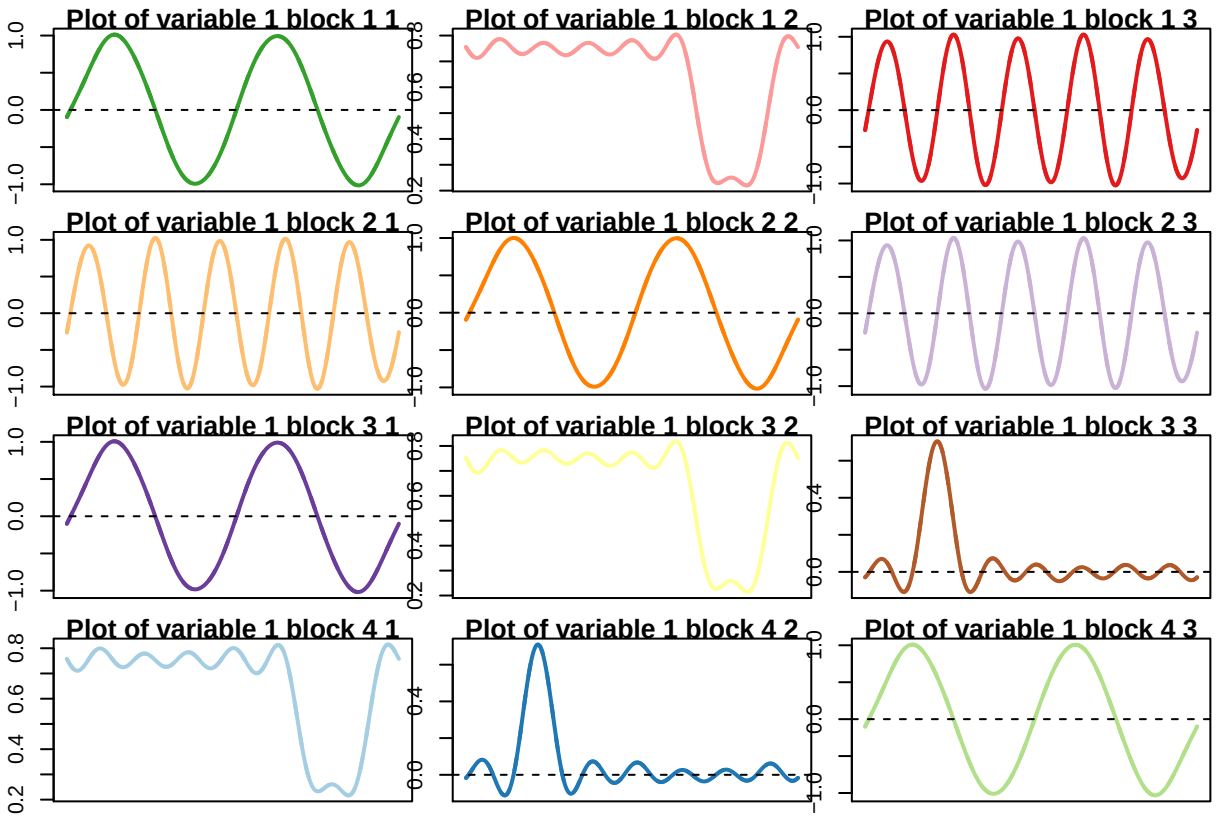
Example of co-clustering multivariate functional data

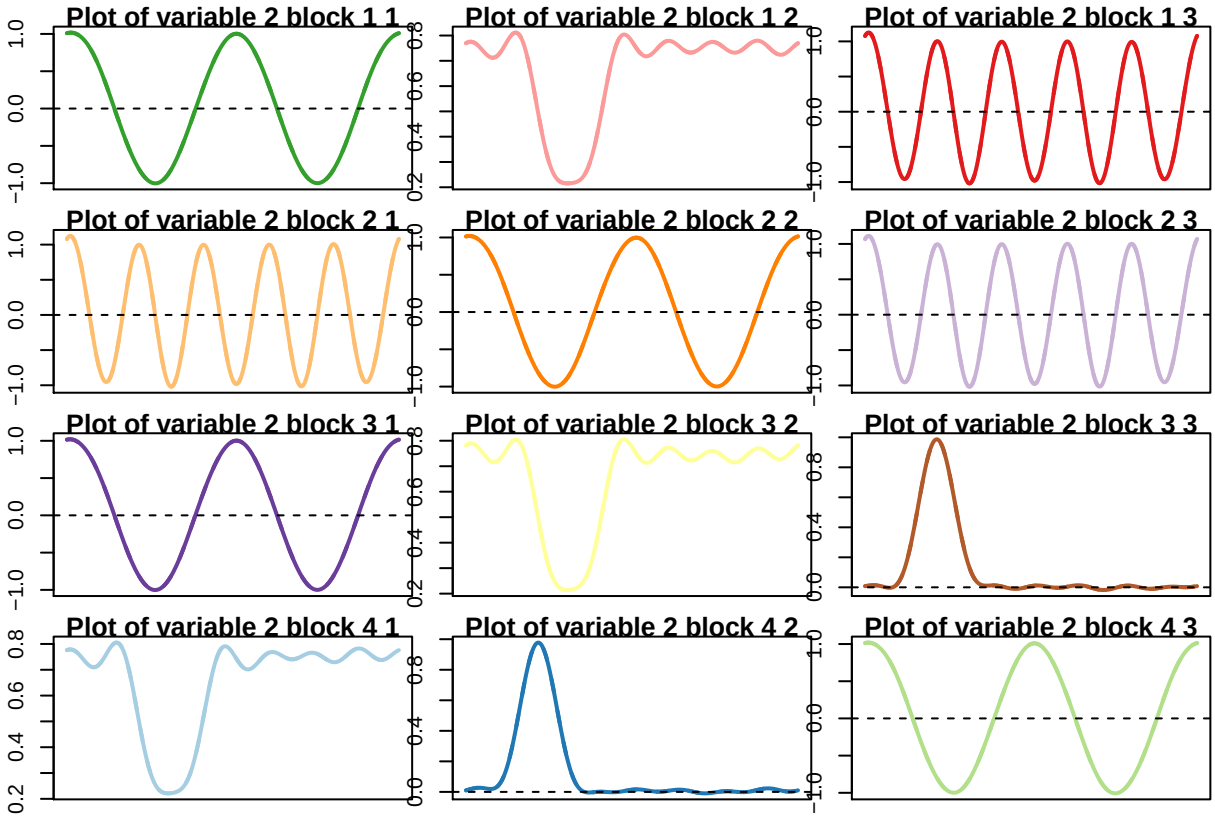
Now, we will apply funLBM algorithm to the case of two functional variables, for example the electric consumption and temperature monitored every 30min for 90 days. ## Basic code example

```
library(funLBM)
res.multi<-funLBM(list(data.multivar$data1,data.multivar$data2),K=4,L=3,
                  basis.name = "fourier",nbasis=15,init="funFEM")
```

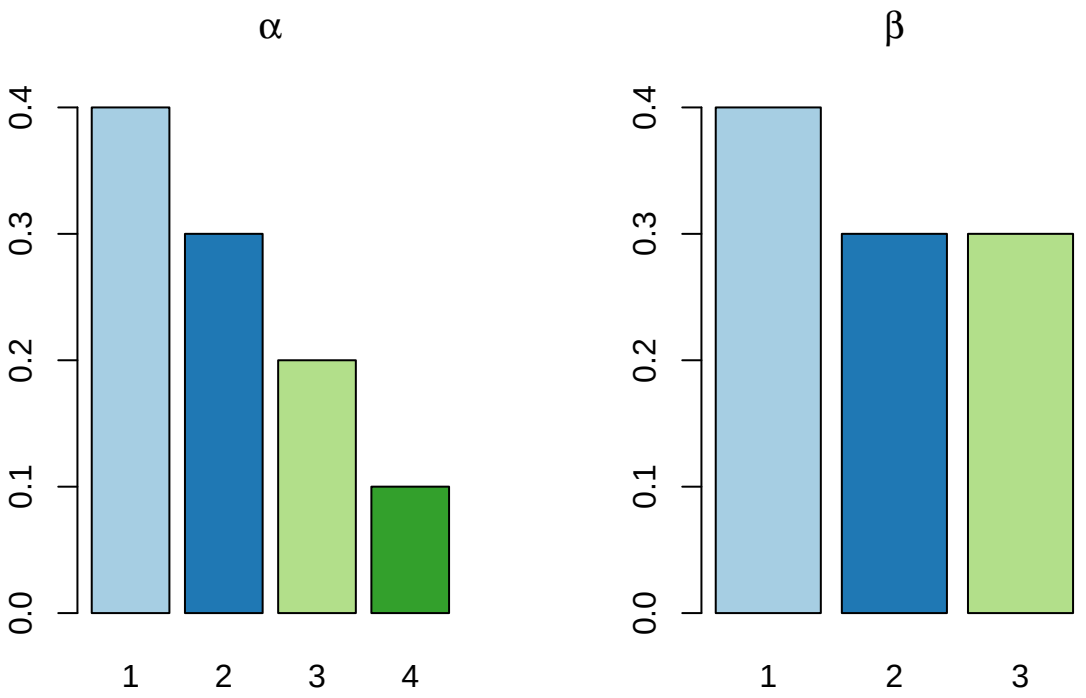
Then results can be viewed graphically with:

```
plot(res.multi,type='blocks')
```

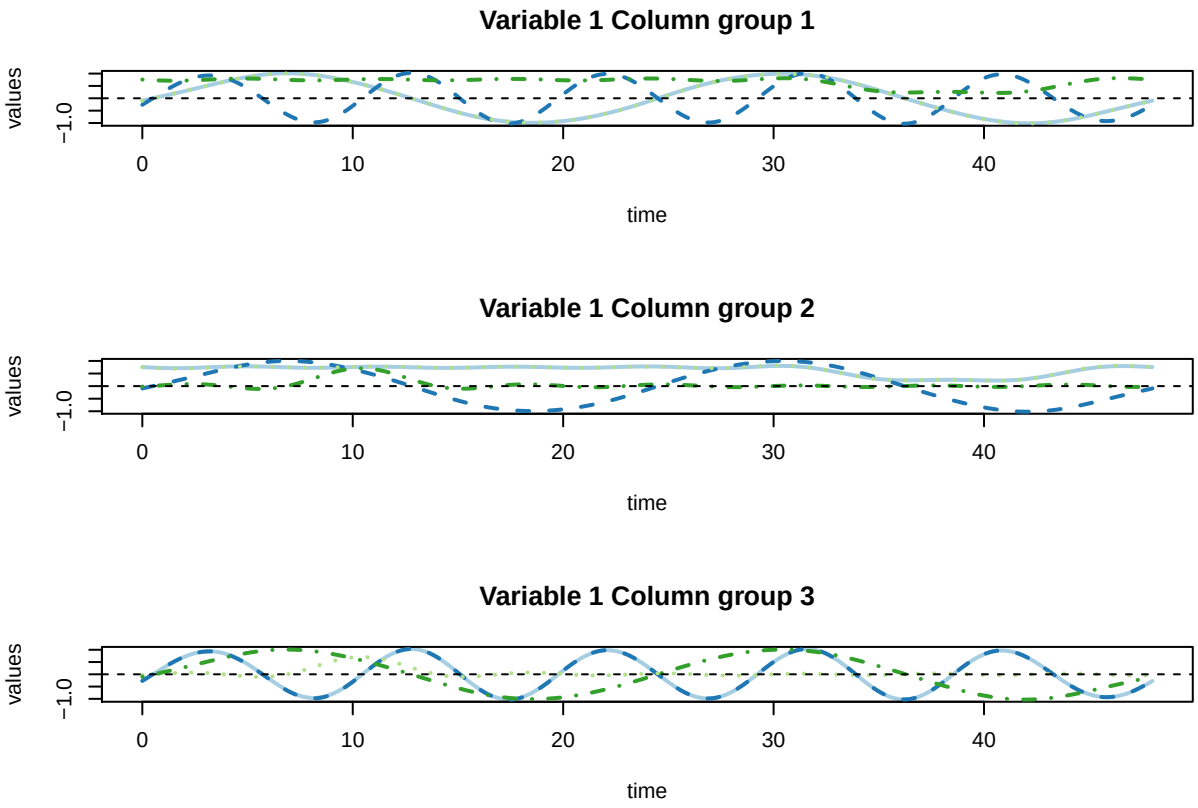

Plot of variable 1 block 1 1
Plot of variable 1 block 1 2
Plot of variable 1 block 1 3
Plot of variable 1 block 2 1
Plot of variable 1 block 2 2
Plot of variable 1 block 2 3
Plot of variable 1 block 3 1
Plot of variable 1 block 3 2
Plot of variable 1 block 3 3
Plot of variable 1 block 4 1
Plot of variable 1 block 4 2
Plot of variable 1 block 4 3

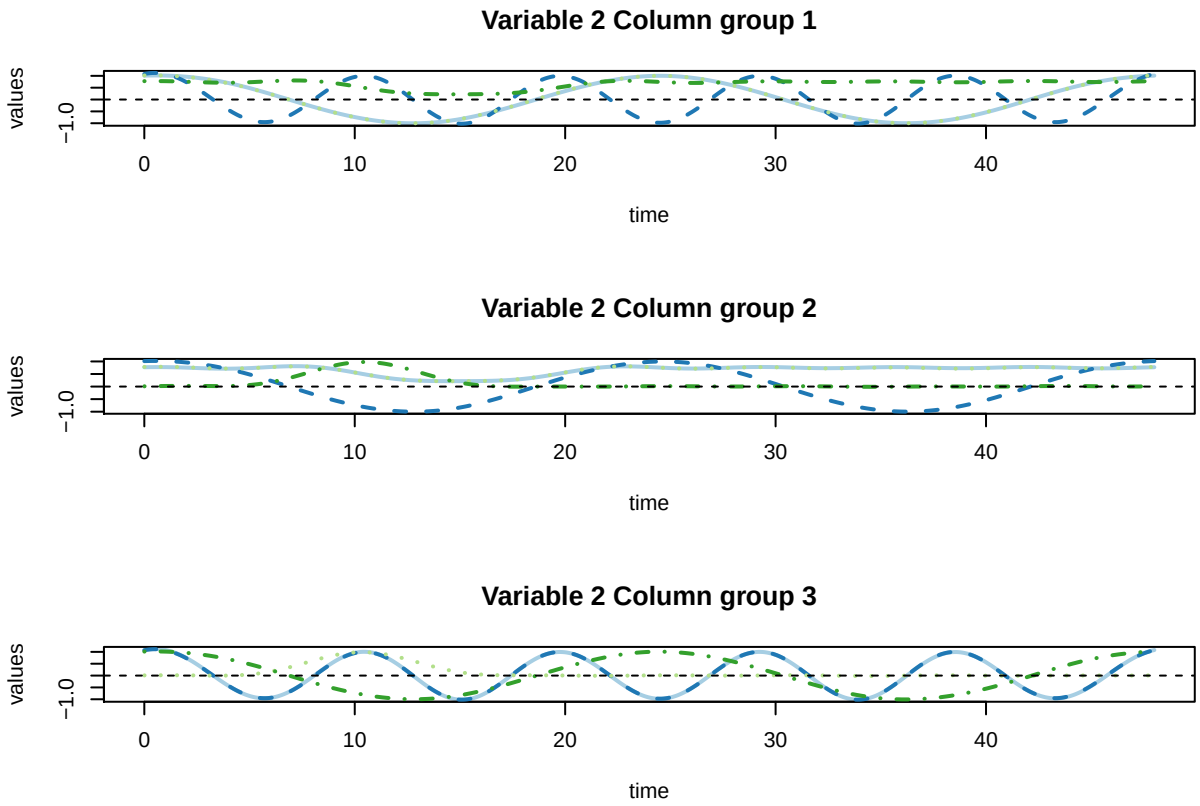



```
plot(res.multi,type='proportions')
```




```
plot(res.multi,type='means')
```


Variable 1 Column group 1
values
−1.0
0
10
20
30
40
time
Variable 1 Column group 2
values
−1.0
0
10
20
30
40
time
Variable 1 Column group 3
values
−1.0
0
10
20
30
40
time



The first line produces two graphics, one for each variable, with the blocks means. The second line produces the groups proportion, in our example for row clusters (alpha) the first group contains 40% of the households, the second 30%, the third 20% and the last 20%. The column cluster (beta) is read in the same way with the second and third groups that contain 30% of the households and the first one 40%. The third line produces two graphics, one per functional variable, where the block mean functions for each column cluster are overlaid.

Then you can evaluate the co-clustering results, checking if the algorithm gets back to the true row and column partitions.

```
ari(res.multi$col_clust,data.multivar$col_clust)
#> [1] 1
ari(res.multi$row_clust,data.multivar$row_clust)
#> [1] 1
```

Both ARI is equal to 1, thus we can conclude that with funFEM initialization the funLBM algorithm gets back to the correct partitions.

Looking for best partition

Most of the time we do not know how many clusters we have in the data and or aim is to estimate the best partition. In order to do that we use the ICL criterion as defined in (Bouveyron et al, 2018). Thus we will ask funLBM algorithm to look from 2 to 6 clusters for both column and row clusters and to give us the best partition according to the ICL criterion (depending on you computer features it may take a few minutes, you can speed up the computation with mc.cores parameter).

```
res.multi<-funLBM(list(data.multivar$data1,data.multivar$data2),K=2:6,L=2:6,basis.name = "fourier",nbas
#>    K L        icl
#> 7  3 3  -77902.49
#> 12 3 4  -82310.59
#> 8  4 3  -82932.39
#> 6  2 3  -83469.92
#> 3  4 2  -84109.36
#> 11 2 4  -85790.19
#> 2  3 2  -86100.42
#> 4  5 2  -86432.66
#> 17 3 5  -86685.90
#> 16 2 5  -88207.21
#> 5  6 2  -88747.74
#> 9  5 3  -89158.67
#> 21 2 6  -90554.69
#> 22 3 6  -91117.20
#> 13 4 4  -91250.22
#> 10 6 3  -95388.46
#> 1  2 2  -95803.70
#> 18 4 5  -99525.72
#> 14 5 4  -99555.65
#> 23 4 6 -107834.15
#> 15 6 4 -107863.35
#> 19 5 5 -109906.07
#> 24 5 6 -120285.83
#> 20 6 5 -120286.26
#> 25 6 6 -132666.18
```

This time, funLBM has some difficulties to get back to true partition, it suggests K=3 and L=3 has the best partition and the "true" one shows up in second position. This can be due to the small number of curves that makes this task difficult.

Indeed when the number of curves is greater, funLBM algorithm find easily the true partition:

```
data.multivar2<-simulateData2(n=300,p=90,t=45)
res.multi<-funLBM(list(data.multivar2$data1,data.multivar2$data2),K=2:6,L=2:6,basis.name="fourier",nbas
#>    K L       icl
#> 8  4 3 -203510.1
#> 7  3 3 -210188.4
#> 9  5 3 -210492.4
#> 13 4 4 -212868.0
#> 12 3 4 -215146.9
#> 10 6 3 -217484.6
#> 17 3 5 -220093.3
#> 18 4 5 -222186.5
#> 14 5 4 -222205.3
#> 22 3 6 -225086.0
#> 15 6 4 -231558.6
#> 23 4 6 -231565.7
#> 19 5 5 -233884.9
#> 3  4 2 -236011.1
#> 6  2 3 -237553.9
#> 4  5 2 -238615.7
#> 11 2 4 -240163.3
```

```
#> 5  6 2 -241202.2
#> 16 2 5 -242783.6
#> 21 2 6 -245421.2
#> 20 6 5 -245550.4
#> 24 5 6 -245575.7
#> 2  3 2 -249417.5
#> 25 6 6 -259546.6
#> 1  2 2 -281674.6
```