CONNECTOR - Instruction Manual

Simone Pernice, Roberta Sirovich and Francesca Cordero

Contents

Introduction

The transition from the evaluation of a single time point to the examination of the entire dynamic evolution of a system is possible only in the presence of the proper framework. Here we introduce CONNECTOR, a data-driven framework able to analyze and inspect longitudinal high-dimensional data in a straightforward and revealing way. CONNECTOR is a tool for the unsupervised analysis of longitudinal data, that is, it can process any sample consisting of measurements collected sequentially over time. CONNECTOR is built on the model-based approach for clustering functional data presented in (James and Sugar 2003), which is particularly effective when observations are sparse and irregularly spaced, as growth curves usually are.

The CONNECTOR framework is based on the following steps:

(1) The **pre-processing step** consists of a visualization of the longitudinal data by a line plot and a time grid heat map helping in the inspection of the sparsity of the time points.

(2) Then, by means of the FDA analysis, the sampled curves are processed by CONNECTOR with a functional clustering algorithm based on a mixed effect model. This step requires a **model selection phase**, in which the dimension of the spline basis vector measures and number of clusters are computed that help the user to properly set the two free parameters of the model. The selection of the optimal set of parameters is supported by the generation of several plots.

(3) Once the model selection phase is completed, the **output** of CONNECTOR is composed of several graphical visualizations to easily mine the results.

Installation

The `connector` package is hosted on the GitHub platform. The simplest way to obtain `connector` is to install it using `devtools`. Type the following commands in R console:

```
# Install
install.packages("devtools",repos = "http://cran.us.r-project.org")
library(devtools)
install_github("qBioTurin/connector", ref="master",dependencies=TRUE)
# Load
library(connector)
```

Users may change the `repos` options depending on their locations and preferences. For more details, see `help(install.packages)`.

The following packages are required before installing *CONNECTOR*:

```
install.packages(c('cowplot', 'fda', 'flexclust','ggplot2',
                   'MASS', 'Matrix', 'plyr','ggplotify',
                   'RColorBrewer', 'readxl',
                   'reshape2', 'splines', 'statmod',
                   'sfsmisc','shinyWidgets', 'viridis',
                   'dashboardthemes','shinybusy',
                   'shinydashboard','shinyjs','tidyr'))
```

Quick Start

A demo is available to provide the list of commands to perform the analysis of the example treated in this tutorial. To run the example just type:

```
demo("MainCommandsList", package = "connector")
```

Docker

To use Docker it is necessary install Docker on your machine. For more info see this document https://docs.docker.com/engine/installation/. Ensure your user has the rights to run docker (no sudo). To create the docker group and add your user:

Create the docker group.

```
$ sudo groupadd docker
```

Add your user to the docker group.

```
$ sudo usermod -aG docker $USER
```

Log out and log back in, your group membership is re-evaluated.

To run CONNECTOR from docker Permalink without installing CONNECTOR it is necessary download the image from dockerhub

```
$ docker pull qbioturin/connector:latest
```

Run the RStudio Docker image with CONNECTOR installed and work with RStudio in the browser

```
$ docker run --cidfile=dockerID -e DISABLE_AUTH=true -p 8787:8787 -d qbioturin/connector:latest
$ open http://localhost:8787/
```

From CONNECTOR, it is possible to download and run the RStudio Docker image by using the CONNECTOR functions called *downloadContainers* and *docker.run* as follows:

```
$ downloadContainers()
$ docker.run()
```

Case study on tumor growth dataset.

To illustrate how `connector` works we use patient-derived xenografts (PDXs) lines from 21 models generated from chemotherapy-naive high-grade serous epithelial ovarian cancer.

Data Importing by files

The analysis starts from two distinct files.

1. An excel file reporting the discretely sampled curve data. The longitudinal data must be reported in terms of time t and y-values y. Each sample is described by two columns: the first column, named *time*, includes the lags; while the second column, named with the *ID sample*, contains the list of y values. Note that, each sample must have different *ID sample*. Hence, the 21 tumor growth curves are collected in a file composed of 48 columns. See Figure 1.
2. A csv (or txt) file contains the annotations associated with the sampled curves. The first column reports the **IDSample**, the other columns report the features relevant for the analysis, one per column. Note that the column *ID sample* must contain the same ID names which appear in the excel file of the sampled curves. See Figure 2.

The two files are imported by the `DataImport` function, which arguments are the file names. In this example `TimeSeriesFile` and `AnnotationFile`.

	A	B	C	D	E	F	G	H	I	J	K	L
1	time	475_P1b	time	475_P1bP2a	time	475_P1bP2b	time	475_P1bP2aP3a	time	475_P1bP2aP3b	time	475_P1bP2aP3bP4a
2	9	63.500364	6	0	6	0	5	163.24589	5	155.6897385	8	166.184865
3	16	0	13	95.05932	13	86.046797	12	201.7201105	12	143.31273	16	98.039808
4	23	78.082284	20	161.872992	20	83.6289155	15	152.6497715	15	158.468778	23	211.310442
5	31	137.9122875	27	117.22128	27	98.1059625	19	180.1715625	19	147.841551	29	367.4509825
6	36	188.4384	34	221.493368	34	197.3022925	22	202.069848	22	182.780862	36	692.8898625
7	44	206.87733	40	318.922848	40	302.830785	25	254.380926	25	289.27054	42	883.658919
8	51	227.2856	47	302.19948	47	304.706709	29	234.353944	29	373.100092	52	1618.681856
9	57	297.041256	60	505.1016	60	210.039786	34	317.25834	34	407.379968		
10	64	356.4912915	68	545.456296	68	147.4506	42	348.9675	42	589.055558		
11	72	428.757616	76	692.499456	76	164.2974	46	388.8712125	46	613.992704		
12	79	337.8964256	83	633.245184	83	306.1658655	50	500.5334375	50	770.646725		
13	87	354.6382	89	822.15675	89	394.5477205	54	471.919608	54	791.796568		
14	92	361.942784	96	605.362048	96	256.130749	57	618.094184	57	849.535074		
15	99	654.0703275	103	759.5610715	103	388.4685805	62	548.856	62	945.561428		
16	107	514.2261105	106	828.655	106	460.291624						
17												
18												

Figure 1: First lines of the excel file of the sampled curve data.

```
IDSample,  Progeny,  Source,  Real.Progeny
475_P1b,  P1,  P0,  P3
475_P1bP2a,  P2,  475_P1b,  P4
475_P1bP2b,  P2,  475_P1b,  P4
475_P1bP2aP3a,  P3,  475_P1bP2a,  P5
475_P1bP2aP3b,  P3,  475_P1bP2a,  P5
475_P1bP2aP3bP4a,  P4,  475_P1bP2aP3b,  P6
475_P1bP2aP3bP4b,  P4,  475_P1bP2aP3b,  P6
475_P1bP2aP3bP4d,  P4,  475_P1bP2aP3b,  P6
475_P1bP2aP3bP4c,  P4,  475_P1bP2aP3b,  P6
475_P1bP2aP3bP4e,  P4,  475_P1bP2aP3b,  P6
475_P1bP2aP3bP4aP5a,  P5,  475_P1bP2aP3bP4a,  P7
475_P1bP2aP3bP4aP5b,  P5,  475_P1bP2aP3bP4a,  P7
475_P1bP2aP3bP4aP5c,  P5,  475_P1bP2aP3bP4a,  P7
475_P1bP2aP3bP4aP5d,  P5,  475_P1bP2aP3bP4a,  P7
475_P1bP2aP3bP4aP5e,  P5,  475_P1bP2aP3bP4a,  P7
475_P1bP2aP3bP4cP5f,  P5,  475_P1bP2aP3bP4c,  P7
475_P1bP2aP3bP4dP5g,  P5,  475_P1bP2aP3bP4d,  P7
475_P1bP2aP3bP4dP5h,  P5,  475_P1bP2aP3bP4d,  P7
475_P1bP2aP3bP4dP5i,  P5,  475_P1bP2aP3bP4d,  P7
475_P1bP2aP3bP4eP5j,  P5,  475_P1bP2aP3bP4e,  P7
```

Figure 2: First lines of the csv file of the annotated features.

```
# find the full path names of the example files
TimeSeriesFile<-system.file("data", "475dataset.xlsx", package = "connector")
AnnotationFile <-system.file("data", "475info.txt", package = "connector")
# import the samples
CONNECTORList<-DataImport(TimeSeriesFile = TimeSeriesFile,
                          AnnotationFile = AnnotationFile)
## ################################
## ######## Summary ##############
##
##  Number of curves: 21 ;
##  Min curve length:  7 ; Max curve length:  18 .
## ################################
```

A list of four objects is created:

```
# show the CONNECTORList structure
str(CONNECTORList)
## List of 4
##  $ Dataset :'data.frame':    265 obs. of  3 variables:
##   ..$ ID         : int [1:265] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ Observation: num [1:265] 63.5 0 78.1 137.9 188.4 ...
##   ..$ Time       : num [1:265] 9 16 23 31 36 44 51 57 64 72 ...
##  $ LenCurv : int [1:21] 15 15 15 14 14 7 7 11 7 11 ...
##  $ LabCurv :'data.frame':    21 obs. of  5 variables:
##   ..$ IDSample    : chr [1:21] "475_P1b" "475_P1bP2a" "475_P1bP2b" "475_P1bP2aP3a" ...
##   ..$ Progeny     : chr [1:21] "  P1" "  P2" "  P2" "  P3" ...
##   ..$ Source      : chr [1:21] "  P0" "  475_P1b" "  475_P1b" "  475_P1bP2a" ...
##   ..$ Real.Progeny: chr [1:21] "  P3" "  P4" "  P4" "  P5" ...
##   ..$ ID          : int [1:21] 1 2 3 4 5 6 7 8 9 10 ...
##  $ TimeGrid: num [1:66] 3 5 6 8 9 10 12 13 14 15 ...
```

The components of the `CONNECTORList` are:

1. `$Dataset`, a data frame with three variables: `ID` of the curve, `Observation` the y values and `Time` the time lags;
2. `$LenCurv`, the vector reporting the number of observations per sample;
3. `$LabCurv`, a data frame matching the sample with the corresponding annotated features. The variables are extracted and named from the `AnnotationFile`;
4. `$TimeGrid`, the vector containing the time grid points.

Data importing by data frame

The data can be imported by two data frames used to generate `CONNECTORList`:

1. *TimeSeriesFrame*: dataframe of three columns storing the time series. The first columns, called "*ID*", stores the sample identification. The second column, labeled "*Observation*", contains the observations over the time, and the third column, called "*Time*", reports the time point.
2. *AnnotationFrame*: dataframe with a number of rows equal to the number of samples in the *TimeSeriesFrame*. The first column must be called "*ID*" reporting the sample identifiers. The other columns correspond to the features associate with the respective sample (one column per sample). If the dataframe is composed of one column, in the `CONNECTORList` only the feature *ID* will be reported.

Hence, `CONNECTORList` can be generated by exploiting the *DataFrameImport* function.

```
CONNECTORList <- DataFrameImport(TimeSeriesDataFrame = GrowDataFrame,
                                 AnnotationFrame = AnnotationFrame)
```

Data Visualization

The `PlotTimeSeries` function generates the plot of the sampled curves, coloured by the selected feature from the `AnnotationFile`. Figure 3 reports the line plot generated by the following code:

```
CurvesPlot<-PlotTimeSeries(data = CONNECTORList,
                           feature = "Progeny")
CurvesPlot
```

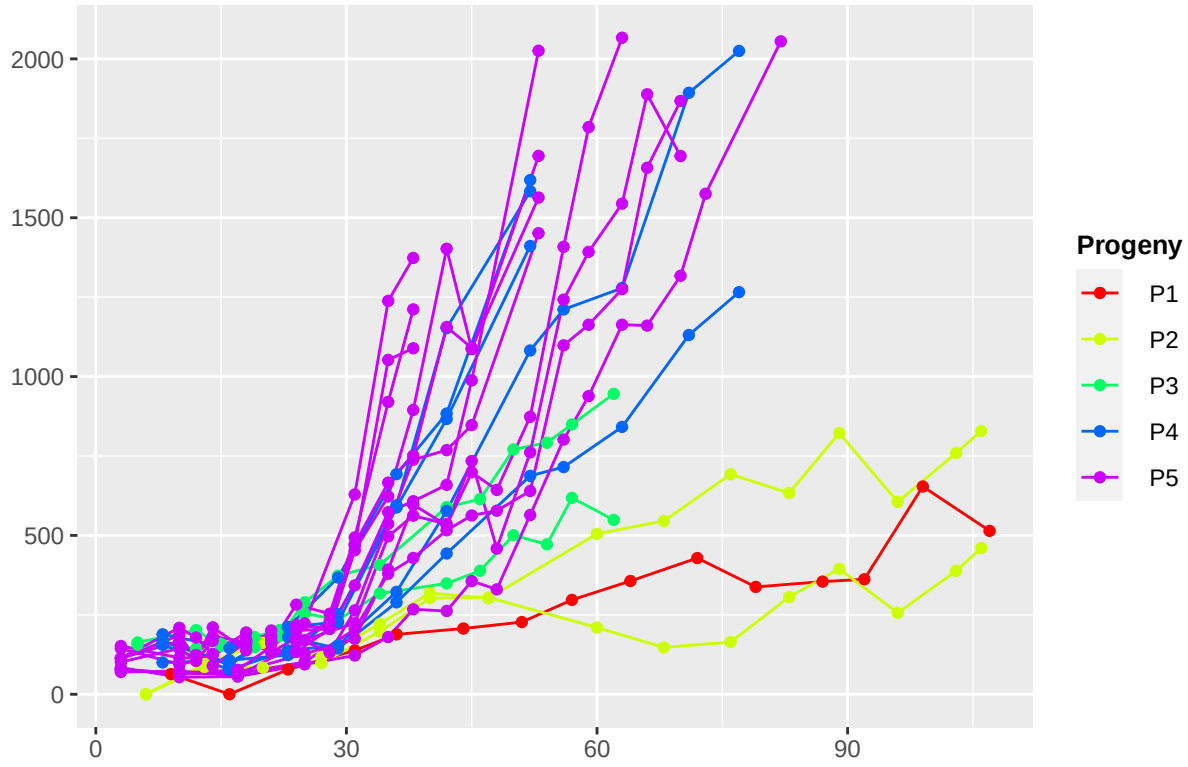

Figure 3: Sampled curves, coloured by progeny feature.

As we can see from 3, only few samples have observations at the last time points. The `DataVisualization` function plots the time grid heat map helping in the inspection of the sparsity of the time points. The `DataTruncation` function have been developed to truncate the time series at specific time value.

```
# Growth curves and time grid visualization
Datavisual<-DataVisualization(data = CONNECTORList,
                              feature = "Progeny",
                              labels = c("Time","Volume","Tumor Growth"))
Datavisual
```

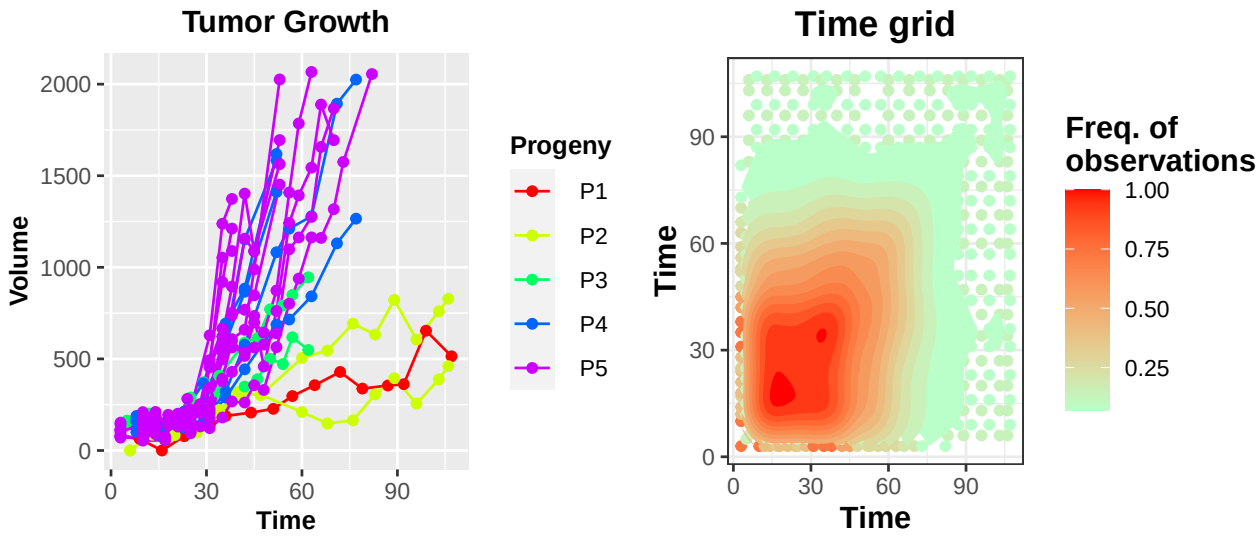

Figure 4: Sampled curves, coloured by progeny feature (left panel) and time grid density (right panel).

In Figure 4 the time grid density is showed. Each point $p_{x,y}$ is defined by a pair of coordinates $p_{x,y} = (x, y)$ and by a colour. $p_{x,y}$ is defined if only if exists at least one sample with two observations at time x and y. The colour encodes the number of samples in which $p_{x,y}$ is reported. In our example, according to Figure 4 we decide to truncate the observations at 70 days.

```
# data truncation
trCONNECTORList<-DataTruncation(data = CONNECTORList,
                                feature="Progeny",
                                truncTime = 70,
                                labels = c("Time","Volume","Tumor Growth"))
## ################################################################
## ######## Summary of the trunc. data ############
##
##  Number of curves: 21 ;
##  Min curve length:  7 ; Max curve length:  16 ;
##
##  Number of truncated curves: 6 ;
##  Min points deleted:  2 ; Max points deleted:  6 ;
## ################################################################
# the trCONNECTORList structure
str(trCONNECTORList, max.level = 1)
## List of 6
##  $ Dataset            :'data.frame': 241 obs. of  3 variables:
##  $ LenCurv            : num [1:21] 9 9 9 14 14 7 7 9 7 9 ...
##  $ LabCurv            :'data.frame': 21 obs. of  5 variables:
##  $ TimeGrid           : num [1:50] 3 5 6 8 9 10 12 13 14 15 ...
##  $ ColFeature         : chr [1:5] "#FF0000" "#CCFF00" "#00FF66" "#0066FF" ...
##  $ PlotTimeSeries_plot:List of 9
##   ..- attr(*, "class")= chr [1:2] "gg" "ggplot"
```

The output of the function `DataTruncation` is a list of six objects reporting th truncated versions of the `DataImport` function. Moreover, the plot stored in `$PlotTimeSeries_plot` shows the sampled curves plot with a vertical line indicating the truncation time, see Figure 5.

```
# plot
trCONNECTORList$PlotTimeSeries_plot
```

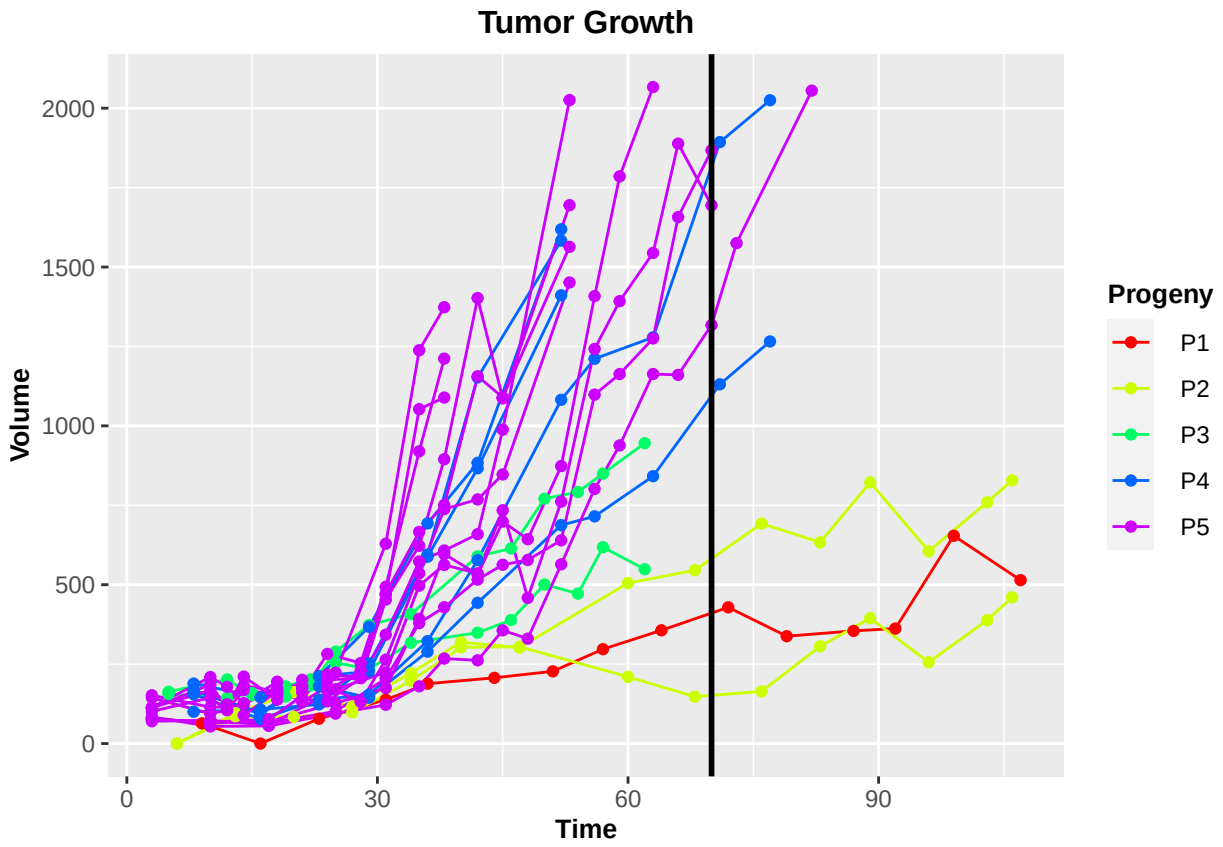

Figure 5: Sampled curves, coloured by progeny feature and truncation lag (vertical solid black line).

Model Selection Tools

Before running the fitting and clustering approach, we have to properly chose the two free parameters:

1. the spline basis dimension, p;
2. the number of clusters, G.

We developed several functions to enable the user to properly set the free parameters.

The spline basis dimension - p

The dimension of the spline basis can be chose by exploiting the `BasisDimension.Choice` function, by taking the $p \in [p_{min}, p_{max}]$ value corresponding to the largest cross-validated likelihood, as proposed in (James, Hastie, and Sugar 2000), where the interval of values $[p_{min}, p_{max}]$ is given by the user. In particular, a ten-fold cross-validation approach is implemented: firstly the data are split into 10 equal-sized parts, secondly the model is fitted considering 9 parts and the computation of the log-likelihood on the excluded part is performed.

In details two plots are returned:

1. The *$CrossLogLikePlot* returns the plot of the mean tested log-likelihoods versus the dimension of the basis, see Figure 6. Each gray dashed line corresponds to the cross-log-likelihood values obtained on different test/learning sets and the solid black line is their mean value. The resulting plot could be used as a guide to choose the largest cross-validated likelihood. Specifically, the optimal value of p is generally the smallest ensuring the larger values of the mean cross-log-likelihood function.

2. The *$KnotsPlot* shows the time series curves (top) together with the knots distribution over the time grid (bottom), see Figure 7. The knots divide the time domain into contiguous intervals, and the curves are fitted with separate polynomials in each interval. Hence, this plot allows the user to visualize whether the knots properly split the time domain considering the distribution of the sampled observations. The user should choose p such that the number of cubic polynomials (i.e., $p-1$) defined in each interval is able to follow the curves dynamics.

```
# ten-fold cross-validation
CrossLogLike<-BasisDimension.Choice(data = trCONNECTORList,
                                    p = 2:6 )
CrossLogLike$CrossLogLikePlot
```

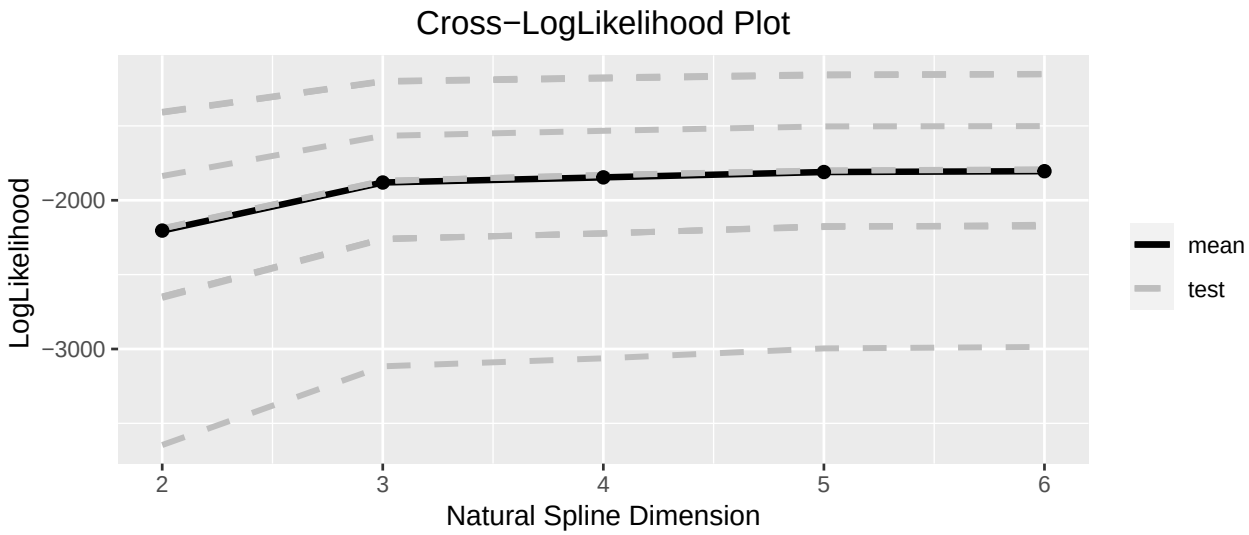

Figure 6: Cross-validated loglikelihood functions.

```
CrossLogLike$KnotsPlot
```

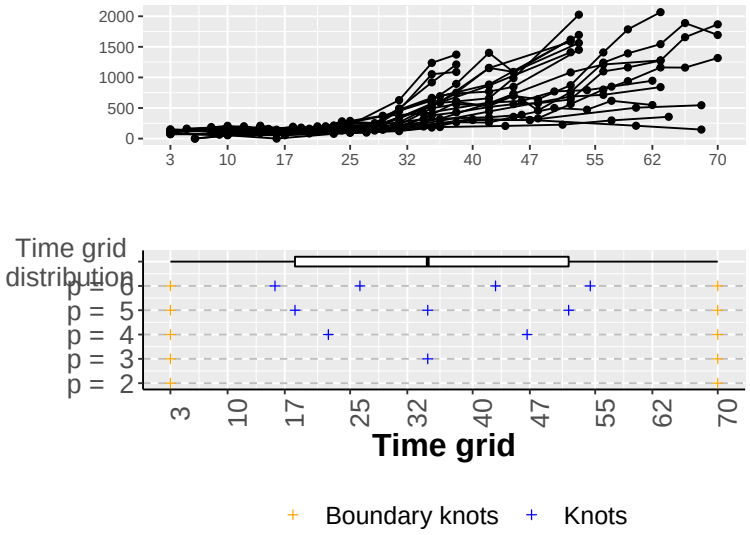

Figure 7: Knots distribution.

```
# set p
p <- 3
```

In our example, the optimal value of p is 3.

The number of clusters - G

`connector` provides two different plots to properly guide to set the number of clusters. Two measures of proximity are introduced: the *total tightness* T and the *functional Davied-Bouldin index* fDB. Both measures rely on the family of semi-metrics between curves defined as

$$D_q(f,g) = \sqrt{\int \left|f^{(q)}(s) - g^{(q)}(s)\right|^2 ds}, \qquad q = 0, 1, 2, \tag{1}$$

where f and g are two curves and $f^{(q)}$ and $g^{(q)}$ are their qth derivatives. Note that for $q = 0$, eq. (1) is the distance induced by the classical L^2-norm. It turns out that D_q can be reliably calculated in our setting, since we are interested in proximity measures between curves and center-curves for each cluster (tightness of the cluster), as well as center-curve and center-curve of different clusters (separateness of clusters). D_q is calculated taking advantage of the spline representation of the estimated curves and mean curves, see eq. (1).

The *total tightness* T is the dispersion measure defined as

$$T = \sum_{k=1}^{G} \sum_{i=1}^{n} D_0(\hat{g}_i, \bar{g}^k), \tag{2}$$

where $\hat{g}_i$ is the estimated i–th curve given in eq. (**??**) and $\bar{g}^k$ is the center of k–th cluster given in eq. (**??**).

As the number of clusters increases, the total tightness decreases to zero, the value which is attained when the number of fitted clusters is equal to the number of sampled curves. In this case, any kth cluster mean curve coincides with an estimated curve and $D_0(\hat{g}_i, \bar{g}^k) = 0$ for any i and k.

A proper number of clusters can be inferred as large enough to let the total tightness drop down to little values over which it does not decrease substantially. Hence, we look for the location of an *elbow* in the plot of

the total tightness against the number of clusters. The second index, which is a cluster separation measure, is called *functional David Bouldin* (fDB) index. Specifically, we defined it as follows

$$\text{fDB}_q = \frac{1}{G}\sum_{k_1=1}^{G} \max_{k_2 \neq k_1} \{R_{k_2 k_1}\}, \tag{3}$$

where, for each cluster k_1 and k_2,

$$R_{k_2 k_1} = \frac{S_{k_1} + S_{k_2}}{M_{k_2 k_1}},$$

$$S_k = \sqrt{\frac{1}{G_k}\sum_{i=1}^{G_k} D_q^2(\hat{g}_i, \bar{g}^k)} \qquad \text{and} \qquad M_{k_2 k_1} = D_q(\bar{g}^{k_2}, \bar{g}^{k_1}),$$

with G_k the number of curves in the kth cluster. The significance of eq. (3) can be understood as the average of the blend measures of each cluster from its most overlapping cluster. The optimal choice of clusters, then, will be that which minimizes this average blend. Furthermore, the random initialization of the k-means algorithm to get initial cluster memberships into the FCM algorithm and the stochasticity characterizing the method lead to some variability among different runs. For these reasons, multiple runs are necessary to identify the most frequent clustering fixed a number of clusters (G).

To effectively take advantage of those two measures, `connector` supplies the function `ClusterAnalysis` which repeats the clustering procedure a number of times equal to the parameter `runs` and for each of the number of clusters given in the parameter `G`. The output of the function is a list of three objects (i) `$Clusters.List`: the list of all the clustering divisions obtained varying among the input G values, (ii) `$seed`: the seed sets before running the method, and (iii) `$runs`: the number of runs.

Specifically, the object storing the clustering obtained with $G = 2$ (i.e., `ClusteringList$Clusters.List$G2`) is a list of three elements:

1. `$ClusterAll`: the list of all the possible FCM parameters values that can be obtained through each run. In details, the item `$ParamConfig.Freq` reports the number of times that the respectively parameters configuration (stored in `$FCM`) is found. Let us note, that the sum might be not equal to the number of runs since errors may occur;
2. `$ErrorConfigurationFit`: list of errors that could be obtained by running the FCM method with extreme parameters configurations;
3. `$h.selected`: the dimension of the mean space, denoted as h, selected to perform the analysis. In details, this parameter gives a further parametrization of the mean curves allowing a lower-dimensional representation of the curves with means in a restricted subspace. Its value is choose such that the inequality $h \leq \min(G-1,\ p)$ holds. Better clusterings are obtained with higher values of h, but this may lead to many failing runs. In this cases h values is decreased until the number of successful runs are higher than a specific constraint which can be defined by the user (by default is 100% of successful runs).

```
ClusteringList <-ClusterAnalysis(data = trCONNECTORList,
                                 G = 2:5,
                                 p = p,
                                 runs = 100)
```

```
# the output structure
str(ClusteringList, max.level = 2, vec.len=1)
## List of 4
##  $ Clusters.List:List of 4
##   ..$ G2:List of 2
```

```
##   ..$ G3:List of 2
##   ..$ G4:List of 2
##   ..$ G5:List of 2
##  $ CONNECTORList:List of 6
##   ..$ Dataset            :'data.frame':  241 obs. of  3 variables:
##   ..$ LenCurv            : num [1:21] 9 9 ...
##   ..$ LabCurv            :'data.frame':  21 obs. of  5 variables:
##   ..$ TimeGrid           : num [1:50] 3 5 ...
##   ..$ ColFeature         : chr [1:5] "#FF0000" ...
##   ..$ PlotTimeSeries_plot:List of 9
##   .. ..- attr(*, "class")= chr [1:2] "gg" ...
##  $ seed         : num 2404
##  $ runs         : num 100
str(ClusteringList$Clusters.List$G2, max.level = 3, vec.len=1)
## List of 2
##  $ ClusterAll:List of 2
##   ..$ :List of 2
##   .. ..$ FCM             :List of 4
##   .. ..$ ParamConfig.Freq: int 43
##   ..$ :List of 2
##   .. ..$ FCM             :List of 4
##   .. ..$ ParamConfig.Freq: int 57
##  $ h.selected: num 1
```

The function `IndexesPlot.Extrapolation` generated two plots reported in Figure 8. In details, the tightness and fDB indexes are plotted for each value of G. Each value of G is associated with a violin plot created by the distribution of the index values collected from the runs performed. The blue line shows the most probable configuration which will be exploited hereafter.

```
IndexesPlot.Extrapolation(ClusteringList)-> indexes
indexes$Plot
```

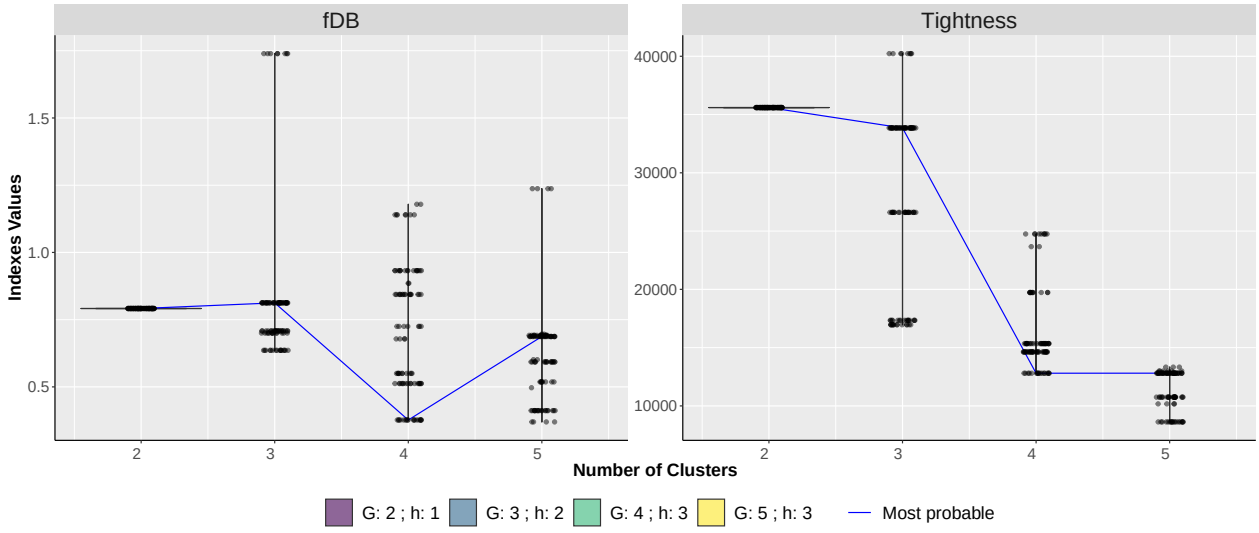

Figure 8: Violin Plots of the *total tightness* T calculated on each run and for different number of clusters G (right panel). Violin Plots of the *functional DB index* fDB calculated on each run and for different number of clusters G (left panel).

$G = 4$ corresponds to the optimal choice for our example, since it represents the elbow for the tightness indexes (right panel) and explicitly minimizes the fDB indexes (left panel).

The variability of the two measures among runs, exhibited in Figure 8, is related to the random initialization of the k-means algorithm to get initial cluster memberships from points. The stability of the clustering procedure can be visualized through the stability matrix extrapolated by the function `ConsMatrix.Extrapolation`, as shown in Figure 9. The plot informs about the stability of the final clustering across different runs. Indeed, each cell of the matrix is coloured proportionally to the frequency of the two corresponding curves belonging to the same cluster across different runs. Hence the larger the frequencies are (which corresponds to warmer colours of the cells in the plot), the more stable is the final clustering. For each cluster the average stability value is reported.

```
ConsMatrix<-ConsMatrix.Extrapolation(stability.list = ClusteringList)
str(ConsMatrix, max.level = 2, vec.len=1)
## List of 4
##  $ G2:List of 2
##   ..$ ConsensusMatrix: num [1:21, 1:21] 1 1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   ..$ ConsensusPlot  :List of 9
##   .. ..- attr(*, "class")= chr [1:2] "gg" ...
##  $ G3:List of 2
##   ..$ ConsensusMatrix: num [1:21, 1:21] 1 1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   ..$ ConsensusPlot  :List of 9
##   .. ..- attr(*, "class")= chr [1:2] "gg" ...
##  $ G4:List of 2
##   ..$ ConsensusMatrix: num [1:21, 1:21] 1 1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   ..$ ConsensusPlot  :List of 9
##   .. ..- attr(*, "class")= chr [1:2] "gg" ...
##  $ G5:List of 2
##   ..$ ConsensusMatrix: num [1:21, 1:21] 1 0.81 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   ..$ ConsensusPlot  :List of 9
##   .. ..- attr(*, "class")= chr [1:2] "gg" ...
ConsMatrix$G4$ConsensusPlot
```

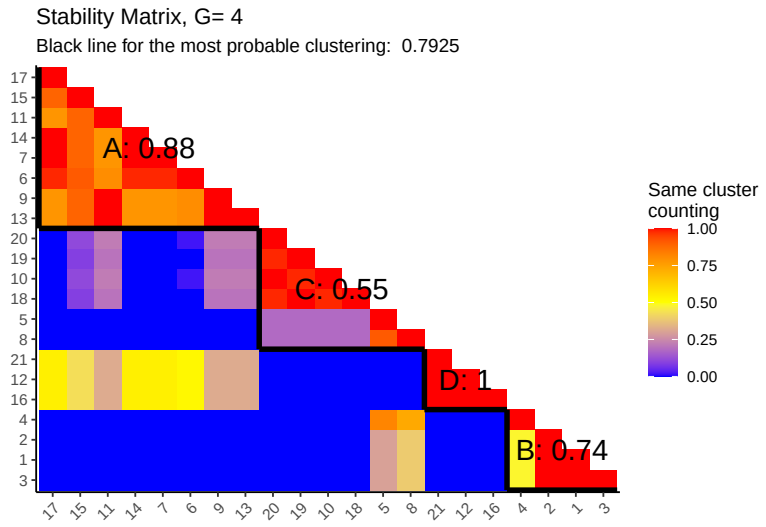

Figure 9: Stability Matrix for G = 4.

Once the free parameters are all set, the function `MostProbableClustering.Extrapolation` can be used to fix the most probable clustering with given dimension of the spline basis p, and number of clusters G and save the result in a dedicated object.

```
CONNECTORList.FCM.opt <- MostProbableClustering.Extrapolation(
  stability.list = ClusteringList,
  G = G )
```

Output

The output of `connector` consists of the line plot of the samples grouped by the `connector` cluster, the discriminant plot, the discriminant function, and the estimation of the entire curve for each single subject.

CONNECTOR clusters

`connector` provides the function `ClusterWithMeanCurve` which plots the sampled curves grouped by cluster membership, together with the mean curve for each cluster, see Figure 10. The function prints as well the values for S_h, M_{hk}, R_{hk} and fDB given in equation (3). The _1 and _2 denote the first and second derivatives respectively of the corresponding index.

```
FCMplots<- ClusterWithMeanCurve(clusterdata = CONNECTORList.FCM.opt,
                                feature = "Progeny",
                                labels = c("Time","Volume"),
                                title = "FCM model")
##
## ######## S indexes ############
##
##
## |          |        S|       S_1|      S_2|
## |:---------|--------:|---------:|--------:|
## |Cluster B | 1702.753| 114.57314| 5.128564|
## |Cluster A |  654.418|  29.34886| 1.021555|
```

```
## |Cluster C | 2791.051| 136.25605| 6.430096|
##
## ################################################################
## ################################################################
##
##         ######## M indexes ############
##
##
## |          | Cluster B| Cluster A| Cluster C|
## |:---------|---------:|---------:|---------:|
## |Cluster B |     0.000|  2851.265|  5731.600|
## |Cluster A |  2851.265|     0.000|  8831.041|
## |Cluster C |  5731.600|  8831.041|     0.000|
##
## ################################################################
## ######## R indexes ############
##
##
## |          |         R|       R_1|      R_2|
## |:---------|---------:|---------:|--------:|
## |Cluster B | 0.8267106| 1.0914505| 1.530825|
## |Cluster A | 0.8267106| 1.0914505| 1.530825|
## |Cluster C | 0.7840400| 0.8803222| 1.255647|
##
## ################################################################
## ######## fDB indexes ############
##
##
## |       fDB|    fDB_1|    fDB_2|
## |---------:|--------:|--------:|
## | 0.8124871| 1.021074| 1.439099|
##
## ################################################################
```

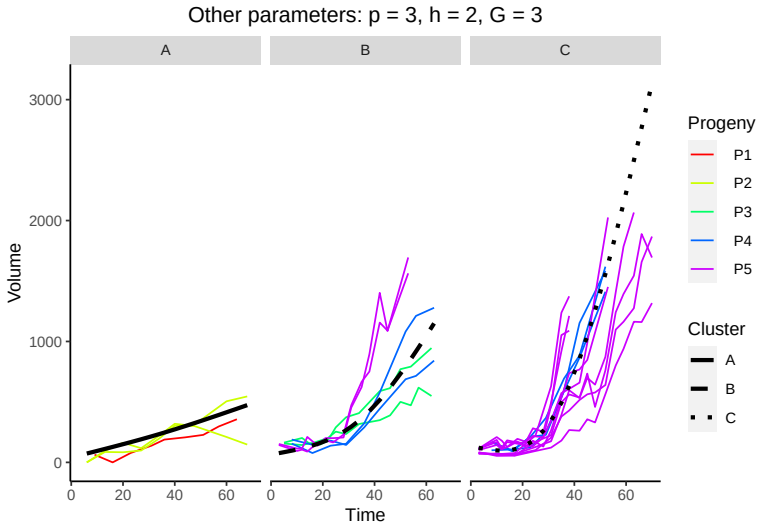

Figure 10: Sampled curves grouped by cluster membership.

Finally, to inspect the composition of the clusters, the function `CountingSamples` reports the number and the name of samples in each cluster according to the feature selected by the user.

```
NumberSamples<-CountingSamples(clusterdata = CONNECTORList.FCM.opt,
                               feature = "Progeny")
str(NumberSamples, max.level = 2)
## List of 2
##  $ Counting    :'data.frame':    15 obs. of  3 variables:
##   ..$ Cluster: chr [1:15] "A" "A" "A" "A" ...
##   ..$ Progeny: Factor w/ 5 levels "  P1","  P2",..: 1 2 3 4 5 1 2 3 4 5 ...
##   ..$ n      : int [1:15] 1 2 0 0 0 0 0 2 2 2 ...
##  $ ClusterNames:'data.frame':    21 obs. of  2 variables:
##   ..$ ID     : int [1:21] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ Cluster: chr [1:21] "A" "A" "A" "B" ...
```

Discrimination Plot

A detailed visualization of the clusterization of the sampled curves can be obtained by means of the `DiscriminantPlot` function.
The low-dimensional plots of curve datasets, enabling a visual assessment of clustering. The curves can be projected the into the lower dimensional space of the mean space, in this manner the curve can be plot as points (with coordinates the functional linear discriminant components). In details, when h is equal to 1 or 2, than the `DiscriminantPlot` function return the plots of the curves projected onto the h dimensional mean space:

1. when $h = 1$, the functional linear discriminant α_1 is plotted versus its variability;
2. when $h = 2$, the functional linear discriminant α_1 is plotted versus the functional linear discriminant α_2.

If $h > 2$, the principal component analysis is exploited to extrapolate the components of the h-space with more explained variability (which is reported in brackets), that are then plotted together.

In the case study here described, we get the plots in Figure 11 which is colored by cluster membership and in Figure 12 which is colored by the `"Progeny"` feature.

```
DiscrPlt<-DiscriminantPlot(clusterdata = CONNECTORList.FCM.opt,
                           feature = "Progeny")
DiscrPlt$ColCluster
```

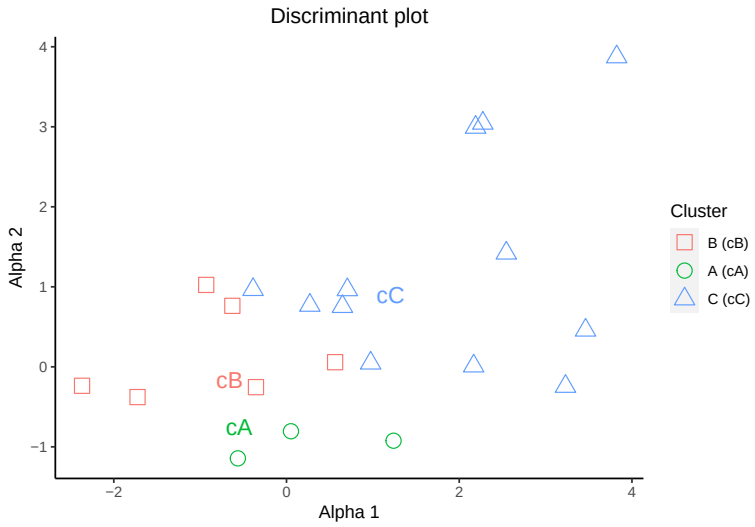

Figure 11: Curves projected onto the 2-dimensional mean space: symbols are coloured by cluster membership.

```
DiscrPlt$ColFeature
```

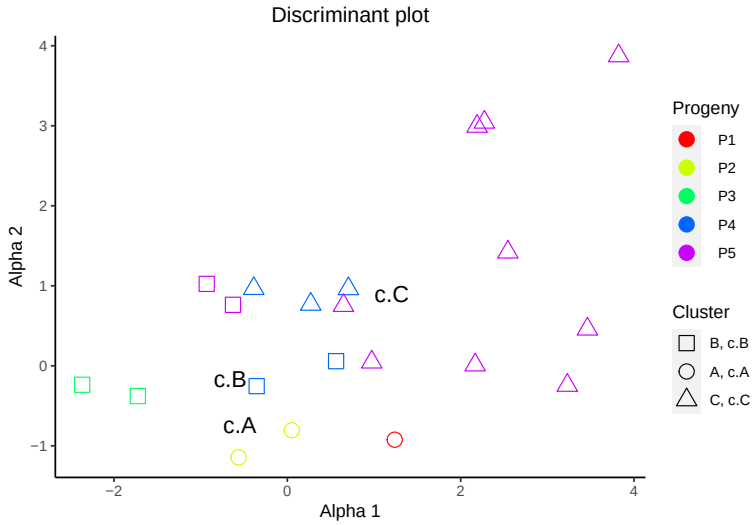

Figure 12: Curves projected onto the 2-dimensional mean space: symbols are coloured by progeny.

Discrimination Function

There will be h discriminant functions and each curve shows the times with higher discriminatory power, which are the times corresponding to largest absolute (positive or negative) values on the y-axis, see Figure 13.

```
MaxDiscrPlots<-MaximumDiscriminationFunction(clusterdata = CONNECTORList.FCM.opt)
MaxDiscrPlots[[1]]
```

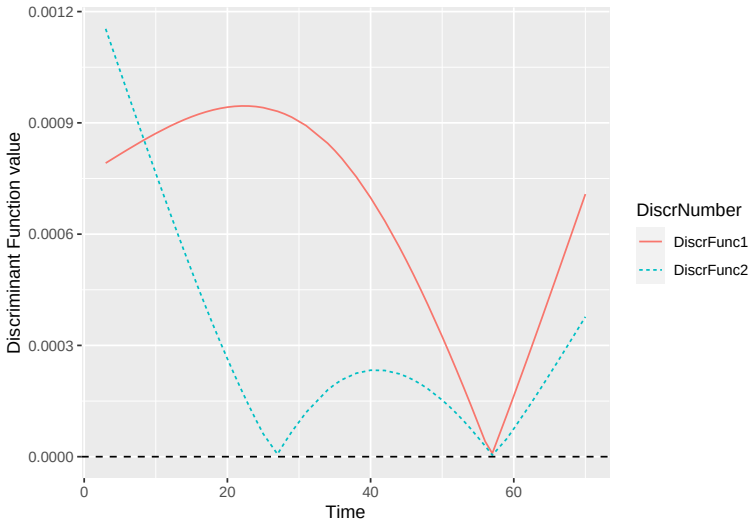

Figure 13: Discriminant curve.

Large absolute values correspond to large weights and hence large discrimination between clusters. From the Figure 13 it is possible assess that earlier measurements are crucial in determining cluster assignment as well as latter ones.

Estimated Curve

The functional clustering procedure predict unobserved portions of the true curves for each subject. The estimated curves are returned by `connector` and plotted with confidence intervals as well that is possible to visualize using the `Spline.plots` function.

```
PlotSpline = Spline.plots(FCMplots)
PlotSpline$`1`
```

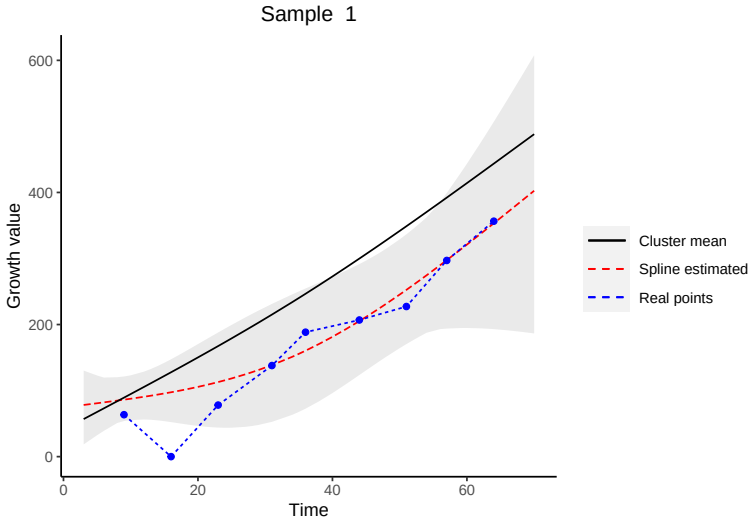

Figure 14: Fitting of the Sample with ID = 1: in blue the observations characterinzing the curve, in red the estimated spline from the fitting, and in black the mean curve of the associated cluster. The grey area represents the confidence interval.

References

James, Gareth M, Trevor J Hastie, and Catherine A Sugar. 2000. "Principal Component Models for Sparse Functional Data." *Biometrika* 87 (3): 587–602.

James, Gareth M, and Catherine A Sugar. 2003. "Clustering for Sparsely Sampled Functional Data." *Journal of the American Statistical Association* 98 (462): 397–408.