An introduction to `connector`

Simone Pernice, Roberta Sirovich and Francesca Cordero

Contents

Introduction

The time series data continues to be of interest to modelers and experimentalists, since are valuable data from which it is possible to extrapolate useful insights and hypothesis. Thus, Connector is an R-package for clustering time series (e.g., functional data), built on the model-based approach described by James and Sugar in (James and Sugar 2003). Their technique is particularly effective when the data are characterized with sparse and irregularly spaced observations. Several statistical approaches and graphical visualizations are implemented in order to help the user to select the right model and to run the unsupervised clustering algorithm. For example, let us consider cancer growth time series, a collection of data which increased constantly in the last years. These data collected starting from different type of biological material, i.e. cancer cell line, patient xenograph models, organoids are used as pre-clinical models to both elucidate the cancer progression and to select the treatments having a good overcome. Then become urgent the identification of an approach able to fit and to cluster the temporal data in order to highlight differences in the dynamics reflecting a divergence in the mechanisms at the basis of the system studied.

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*: `devtools`, `dashboardthemes`, `dplyr`, `sfsmisc`, `shinyWidgets`, `viridis`, `ggplotify`.

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")
```

Case study: tumor growth dataset

We illustrate the functionalities and the usage of the `connector` package on a tumor growth curves dataset, which is included in the package. The general idea is to identify similar patterns of growth in a data set of n samples (tumor growths). Each sample is a curve (i.e., time series - functional data) characterized by a number of observations taken at different times and irregularly.

Data description

Data Importing

Data Importing by files

The analysis starts from two distinct files.

1. An excel file reporting the discretely sampled curve data. As functional data are longitudinal, we respect the convention of parametrizing the models in terms of time t and y-values y. Each sample is represented as two columns of the excel file, the first one named *time* includes the lags list and the second one named with the *ID sample* (each sample has different ID name, for example 475_P1b is the ID name of the first sample in the following figure) includes the list of observed y values. Hence, if we record 24 tumor growth curves, the file have 48 columns. See Figure 1.

2. A csv (or txt) file containing the annotations associated to the sampled curves. It is composed by a number of rows equal to the number of samples. The first row must be the columns names, in which the first column must collect the *ID sample*, and the remaining ones codify for the annotated features. Notice that the column *ID sample* must contain the same ID names which appear in the excel file of the sampled curves. See Figure 2.

	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.

Once data have been prepared in the two files above described, they are imported by the `DataImport` function. Two arguments, with the file names respectively, should be specified. In this example the full path names have been saved in the `TimeSeriesFile` and `AnnotationFile` strings.

```
# 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 object 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" ...
```

```
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.

```
##   ..$ 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 of the number of observations per sample;
3. `$LabCurv`, a data frame matching the sample with the corresponding annotated features. Hence the variables are extracted and named from the `AnnotationFile`;
4. `$TimeGrid`, the vector of the complete time grid points.

Data importing by data.frame

Instead of using two files, two data frames can be exploited to generate `CONNECTORList`. Specifically

1. *TimeSeriesFrame*: dataframe of three columns storing the time series. The first columns, called "*ID*" stores the identification number of each sample. The second column, labeled "*Observation*" contains the observations over the time and the respective time point is reported in the third column, labeled "*Time*";
2. *AnnotationFrame*: dataframe with a number of rows equal to the number of samples in the *TimeSeriesFrame*. The first column must be named "*ID*" and it stores the identification numbers exploited for the matching with the samples saved in the *TimeSeriesFrame*. Then it is possible to add a column per feature to consider and to associate with the respective sample (depending on the identification number in the first column). If NULL, then in the `CONNECTORList` only the feature *ID* will be reported.

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

```
# show the CONNECTORList structure
CONNECTORList <- DataFrameImport(GrowDataFrame = GrowDataFrame,
                                 AnnotationFrame = AnnotationFrame)
```

Data Visualization

The `PlotTimeSeries` function provides a plot of the sampled curves, coloured by a user selected feature out of the ones given in the `AnnotationFile`. In Figure 3 an example is illustrated, that has been produced by the following code:

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

As we can see from 3, only few samples have observations at the last time points. Since data may be irregularly and sparsely sampled like this, as a preliminary step, it may be useful to check whether the data are sufficiently dense in the domain plane. In this context, the `DataVisualization` function plots the sampled curves together with the density grid to eventually decide for truncation, since low-density subregions may be excluded from the analysis, In this contest, the function `DataTruncation` have been developed to truncate the time series into a specific time interval.

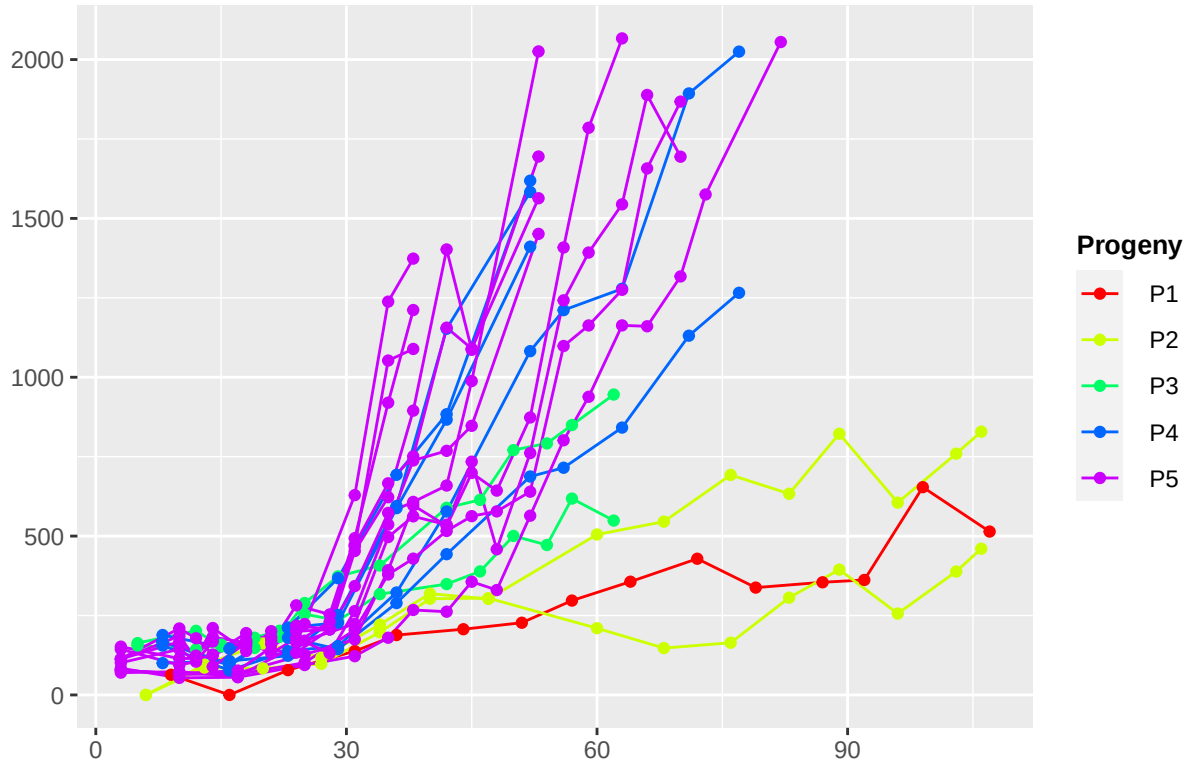

Figure 3: Sampled curves, coloured by progeny feature.

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

Specifically, in Figure 4 the time grid density is showed, in which a 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 associates with it encodes the frequency of samples in which $p_{x,y}$ is present. Therefore, 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 ;
## ################################################################
```

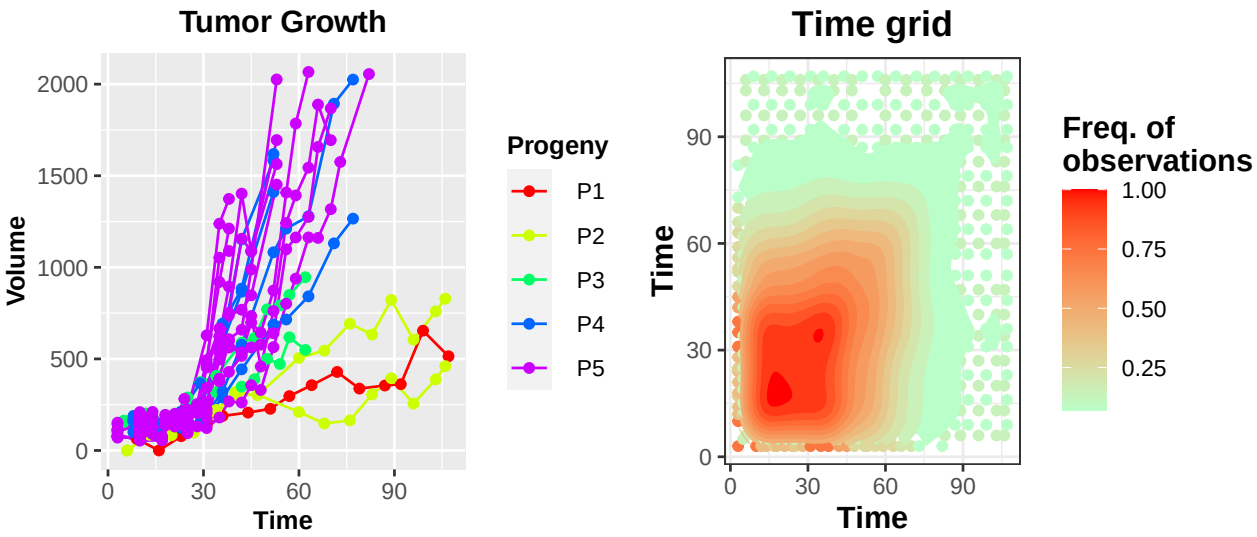

Figure 4: Sampled curves, coloured by progeny feature (left panel) and time grid density (right panel).

```
# 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. The first four of which are the truncated versions of the `DataImport` function. The plot stored in `$PlotTimeSeries_plot` shows the sampled curves plot with a vertical line at the truncation time, see Figure 5.

```
# plot
trCONNECTORList$PlotTimeSeries_plot
```

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 a tool set of functions to guide the user through the identification of each of the above parameters. Let us stress out that rough choices for the free parameters may compromise the full analysis.

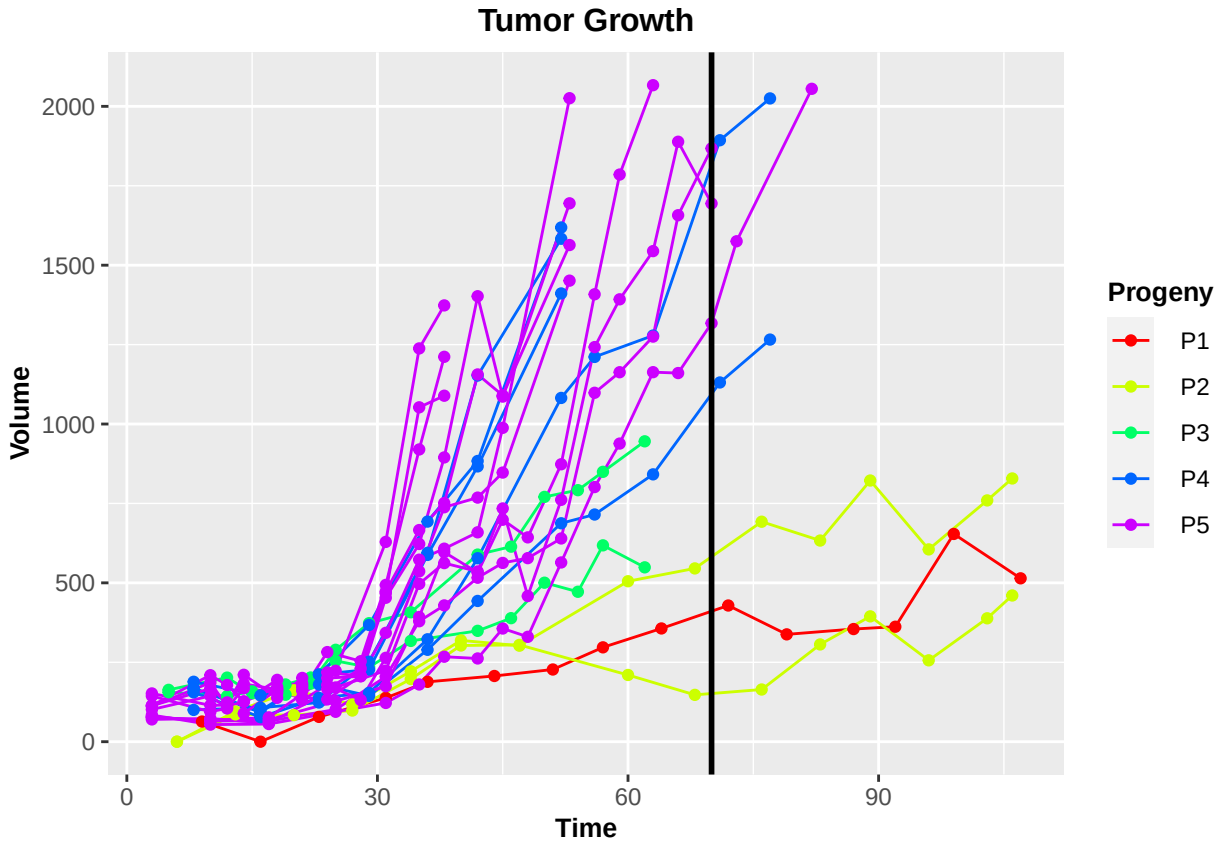

Figure 5: Sampled curves, coloured by progeny feature and truncation lag (vertical solid black line).

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, we implemented a ten-fold cross-validation, which involves splitting data into 10 roughly equal-sized parts, fitting the model to 9 parts and calculating the log-likelihood on the excluded part, repeat 10 times and combine log-likelihoods. Finally, the function returns the plot of the mean tested log-likelihoods versus the dimension of the basis. Notice that the resulting plot should be treated as a guide for choosing the largest cross-validated likelihood rather than an absolute rule, keeping in mind that working with sparse data pushes to spare parameters.

In details two plots are returned:

1. The `$CrossLogLikePlot` gives a visual representation of the results, 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 user should choose the smallest value of `p` that ensures larger values of the mean cross-log-likelihood function.

2. The `$KnotsPlot` shows the time series curves (upper plot) together with the knots (the values of where the pieces of polynomial defining the spline meet) distribution over the time grid (lower plot). Specifically, the lower plot is characterized by one horizontal line for each p value passed in input to the function, and following this line it is showed how the knots, defining the spline of dimension p (i.e., in how many intervals th time grid will be divided), are distributed over the time grid defined by the input curves. 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
```

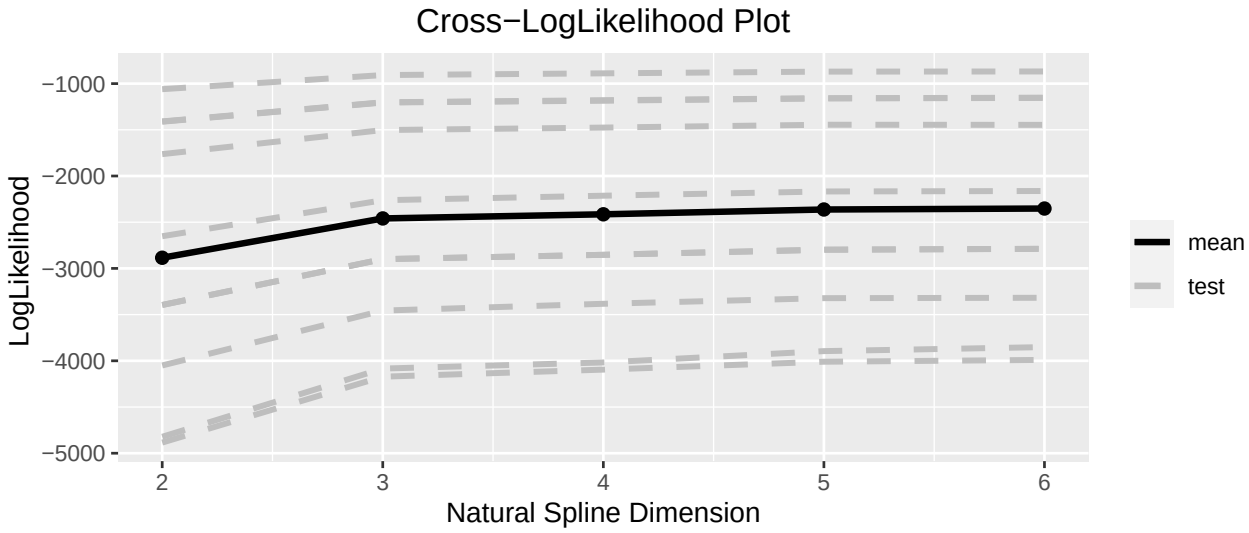

Figure 6: Cross-validated loglikelihood functions.

```
CrossLogLike$KnotsPlot
```

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

Hence, in the current example, we set $p = 3$, as it is the smallest value that ensures a large log-likelihood.

The number of clusters - G

'connector' provides two different plots to properly guide in one of the most difficult problems in cluster analysis: setting properly the number of clusters. As in the finite-dimensional case, where data are points in place of curves, we need some proximities measures to validate and compare results of a clustering procedure. In [qui rif al nostro paper] we introduced two measures of proximity, the *total tightness* T and the *functional Davied-Bouldin index* fDB. Both measures rely on 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 where 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). Hence we may have f being the estimated ith curve and g being the estimated mean curve of cluster k, or f and g being both mean curves. In any case D_q can be calculated taking advantage of the spline representation of the estimated curves and mean curves, see eq. (**??**).

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 equals the number of sampled curves. In this limiting 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 relatively little values but as the smallest over which the total tightness 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

$$\text{fDB}_q = \frac{1}{G} \sum_{k_1=1}^{G} \max_{k_2 \neq k_1} \left\{ \frac{S_{k_2} + S_{k_2}}{M_{k_2 k_1}} \right\}, \tag{3}$$

where, for each cluster k_1 and k_2

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

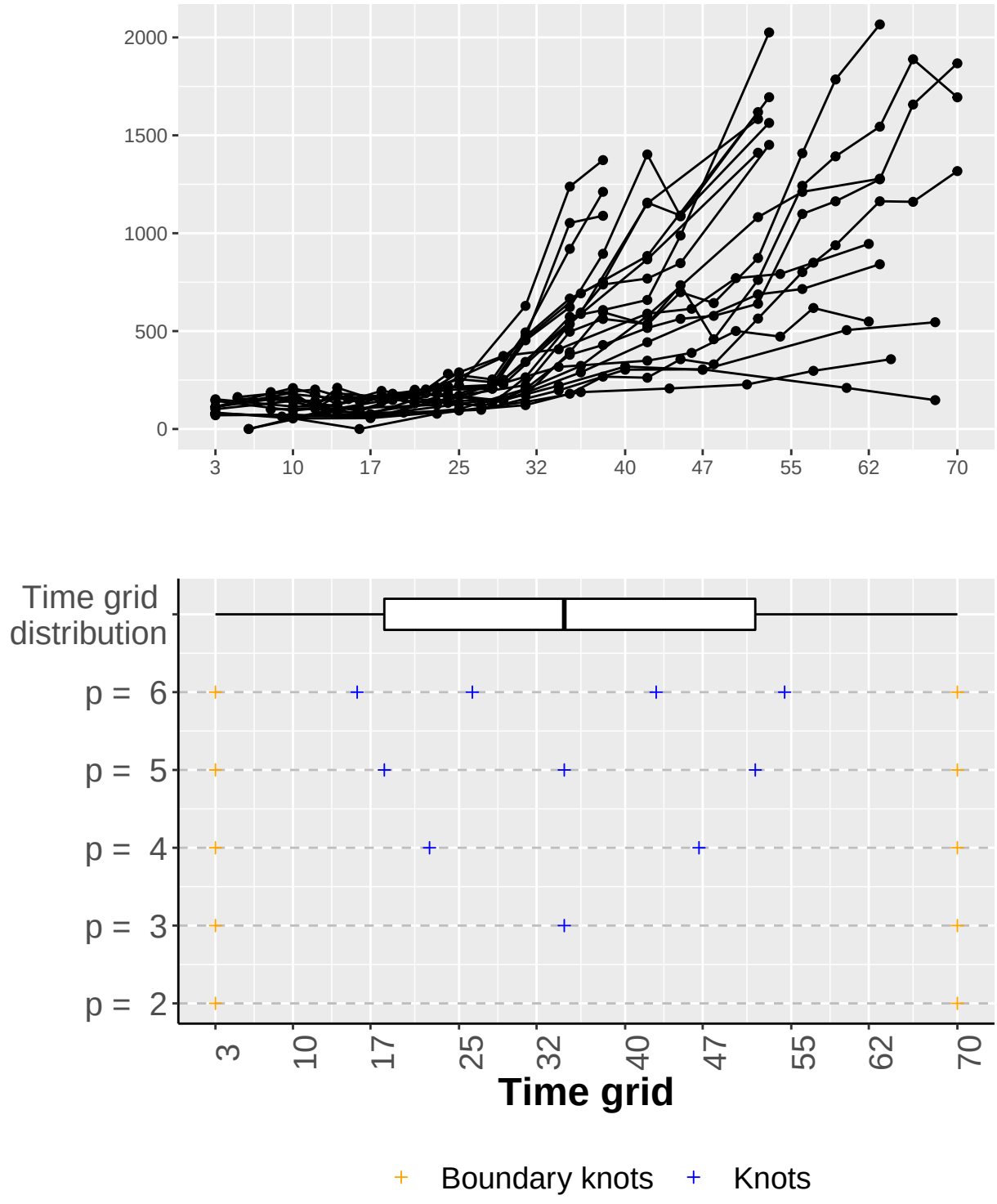

Figure 7: Knots ditribution.

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 (we suggest greater than 50, the default value, but it depends on the size and the variability of the dataset under analysis) 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,

1. `$Clusters.List`: the list of all the clustering divisions obtained varying among the input G values;
2. `$seed`: the seed sets before running the method;
3. `$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. `$ClusterAll`: the list of all the possible FCM parameters values (see Sec. *Details on the functional clustering model*) that can be obtained through each run. In details, the item `$ParamConfig.Freq` reports the number of times that the respectively parameters configuration (stored in `$FCM`) is found. Let us note thaty the sum might be not equal to the number of runs since errors may occur;
2. `$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
```

By means of the function `IndexesPlot.Extrapolation` the two plots represented in Figure 8 can be generated from `ClusteringList`. In details, the tightness and fDB indexes are plotted for each value of G. The vertical lines associated with each G are violin plots in which the points represent the index values for the respectively runs. The blu line shows the most probable configuration which will be exploited hereafter.

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

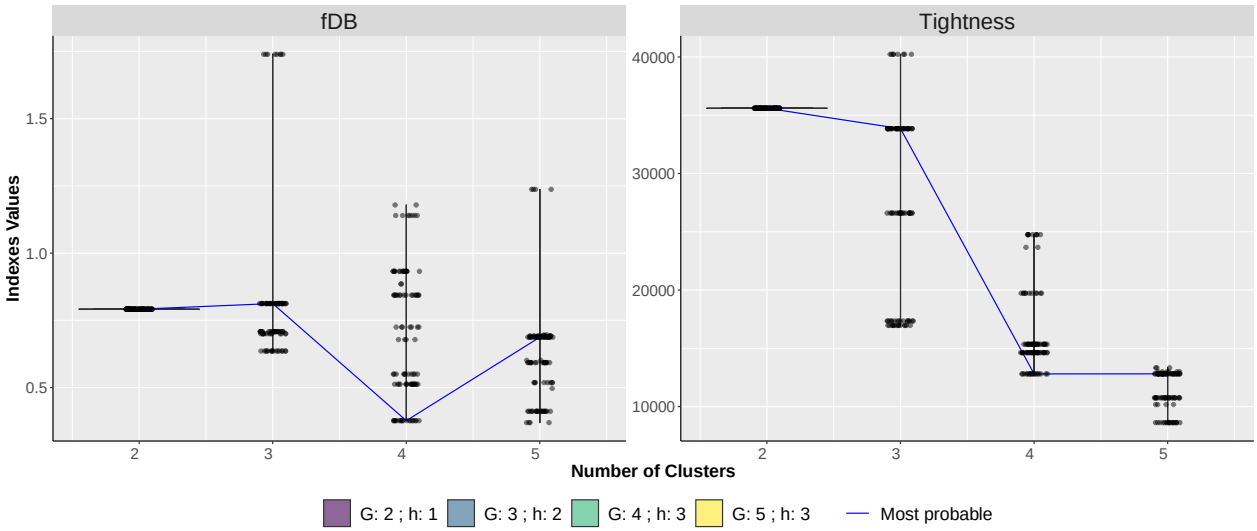

Figure 8: Violin Plots of the *total tightness* T calculated on each run and for different number of clusters G (right panel). Violin Plots of the *functional DB index* fDB calculated on each run and for different number of clusters G (left panel).

The indexes show that $G = 4$ is a good choices, since it is the number of clusters that represents the elbow for the tightness indexes (plotted in the right panel) and explicitly minimizes the fDB indexes (plotted in the 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 consensus matrix extrapolated by the function `ConsMatrix.Extrapolation`, as shown in Figure 9. Let us note that the number associated with each cluster in the consensus matrix represents the average number of time that the curves belonging to the same cluster are counted in the most probable cluster; while the average of these indexes is reported in the title.

```
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
```

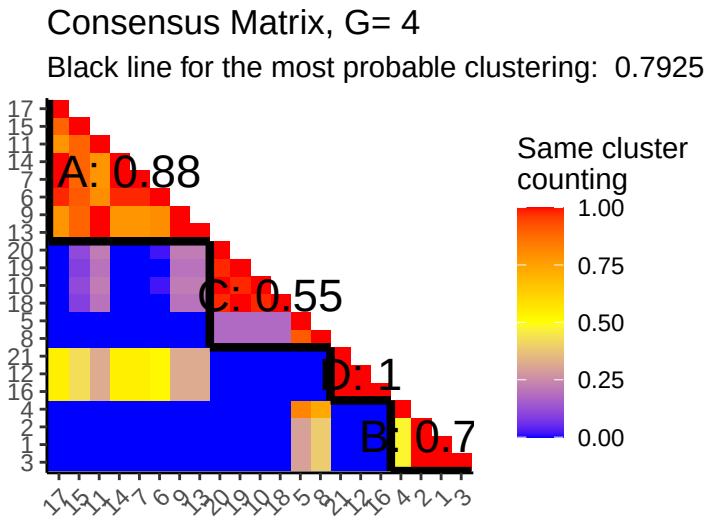

Figure 9: Consensus Matrices for G = 4.

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 )
```

Results Visualization and Inspection

`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} and fDB given in equation (3).

```
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|
##
## ################################################################
```

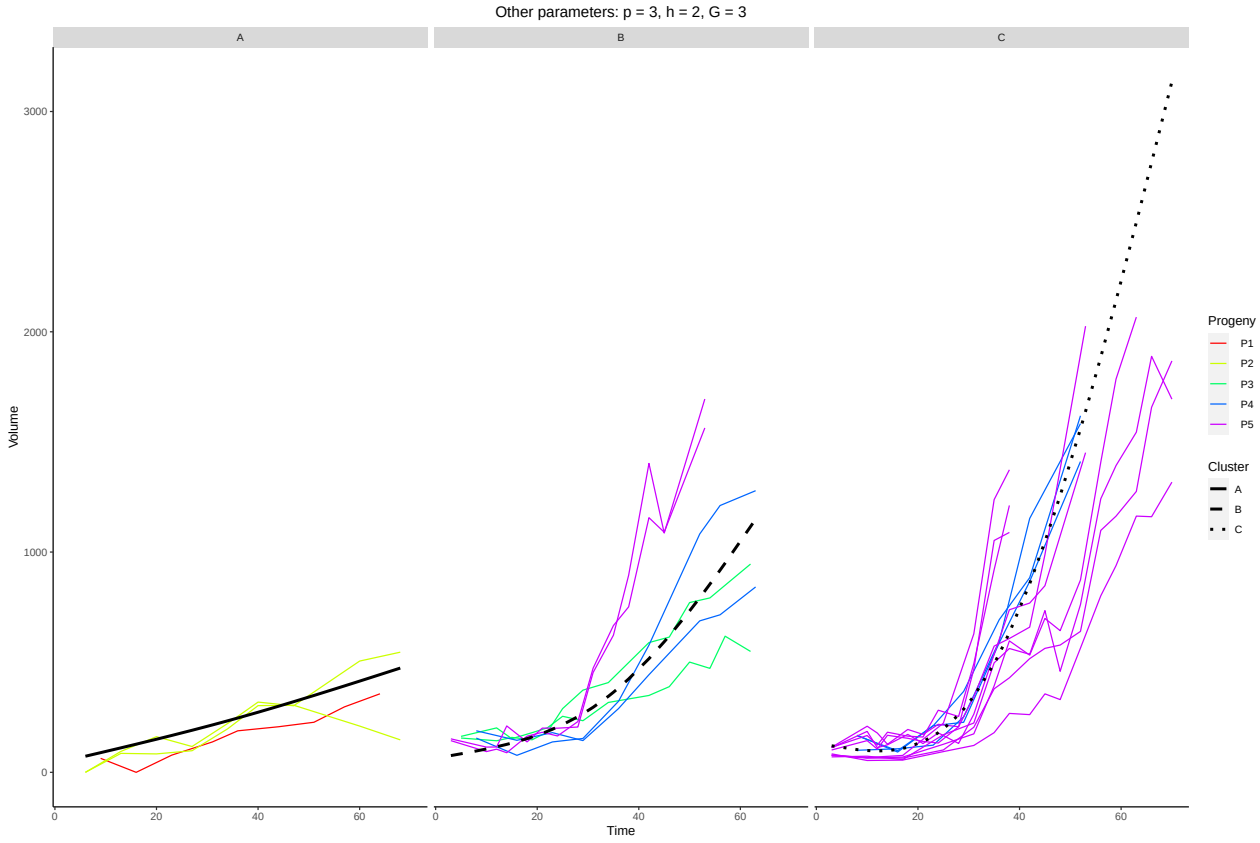

Figure 10: Sampled curves grouped by cluster membership.

A detailed visualization of the clusterization of the sample curves can be obtained by means of the `DiscriminantPlot` function. In (James and Sugar 2003), the authors describe how to obtain low-dimensional plots of curve datasets, enabling a visual assessment of clustering. They propose to project the curves into the lower dimensional space of the mean space, so that they can be plotted as points (with coordinates the functional linear discriminant components), making it much easier to detect the presence of clusters. 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 $alpha_1$ is plotted versus its variability;
2. when $h = 2$, the functional linear discriminant $alpha_1$ is plotted versus the functional linear discriminant $alpha_2$.

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 user selected feature called `"Progeny"`.

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

DiscrPlt$ColCluster
```

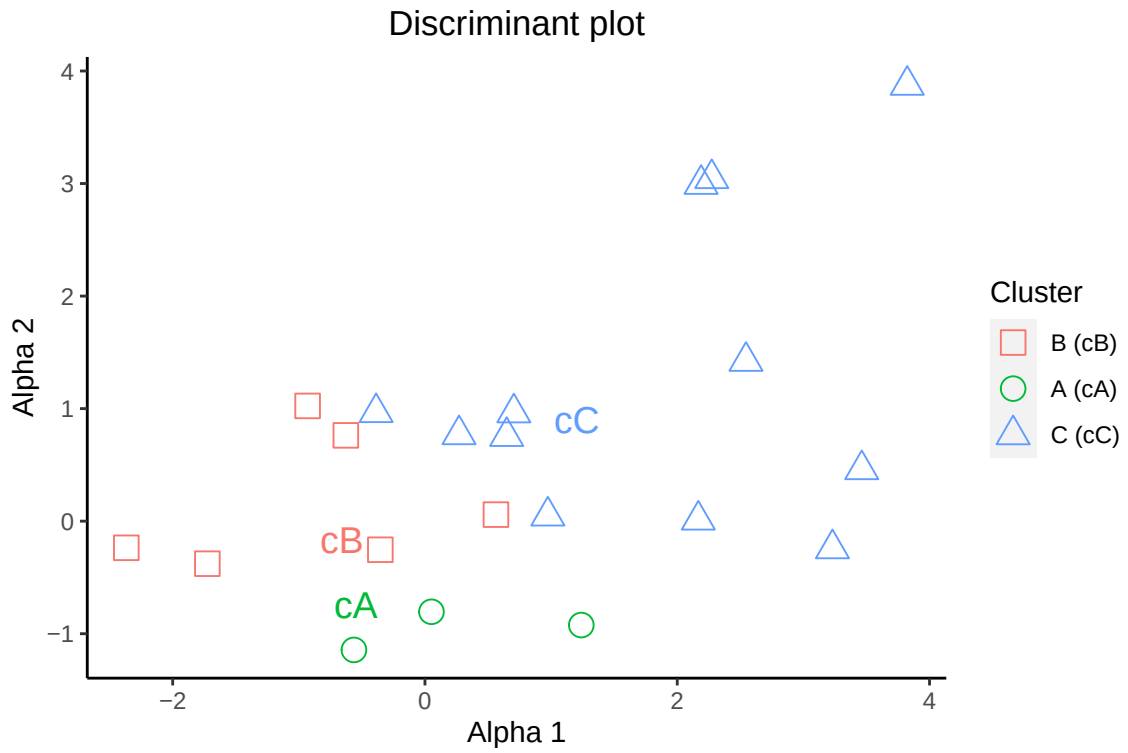

Figure 11: Curves projected onto the 3-dimensional mean space: symbols are coloured by cluster membership.

```
DiscrPlt$ColFeature
```

Furthermore, it is possible to visualize and save the fitting obtained exploiting the algorithm for each curve in the dataset, using the `Spline.plots` function.

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

In the end, 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" ...
```

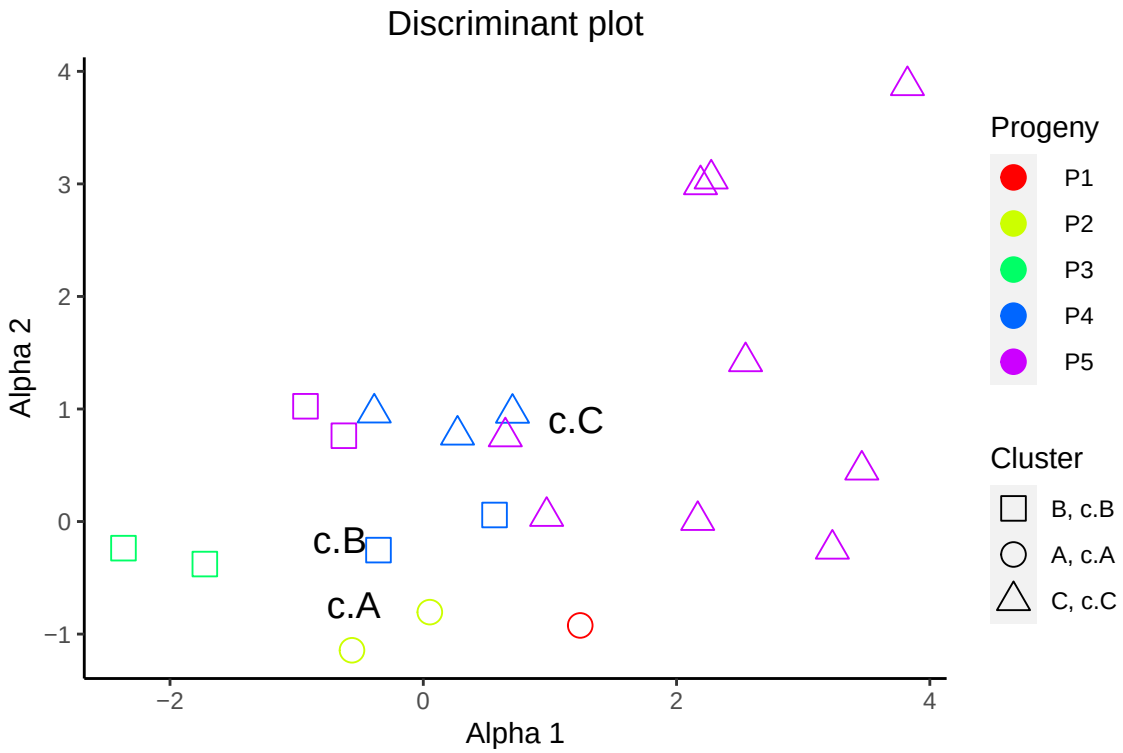

Figure 12: Curves projected onto the 3-dimensional mean space: symbols are coloured by progeny.

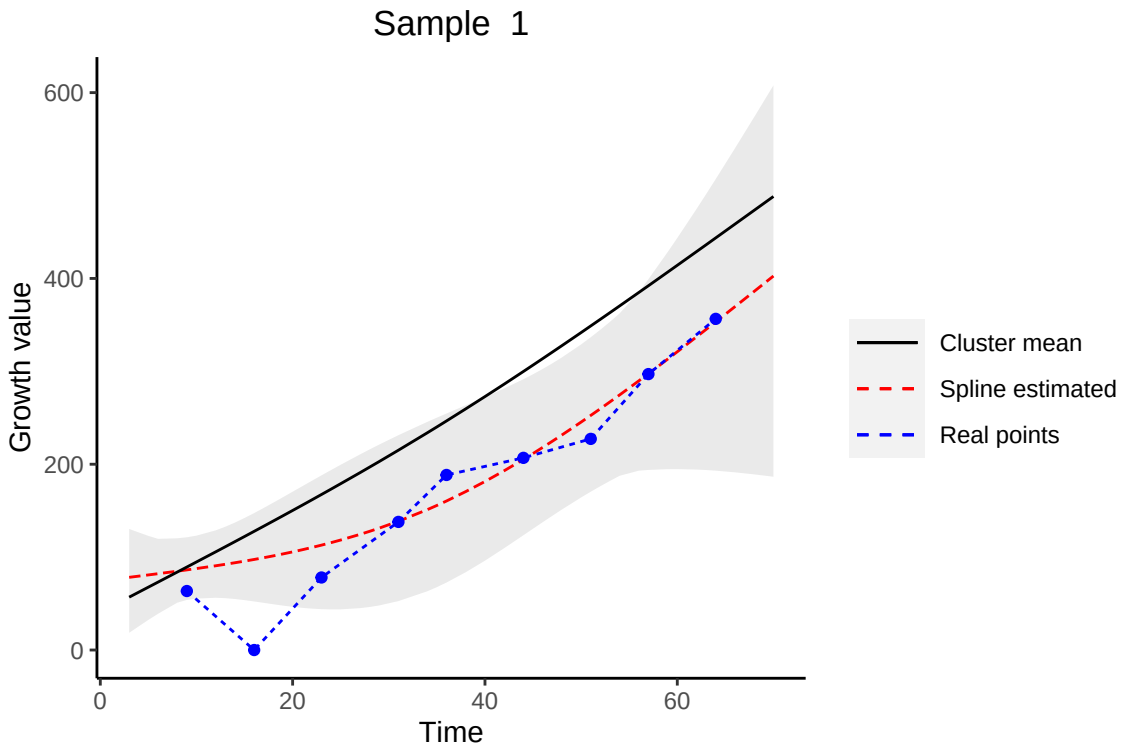

Figure 13: Fitting of the Sample with ID = 1: in blue the observations characterinzing the curve, in red the estimated spline from the fitting, and in black the mean curve of the associated cluster. The grey area represents the confidence interval.

```
##   ..$ Progeny: chr [1:15] "  P1" "  P2" "  P3" "  P4" ...
##   ..$ freq   : int [1:15] 1 5 0 0 0 0 0 9 18 26 ...
##  $ ClusterNames:'data.frame':    21 obs. of  2 variables:
##   ..$ Cluster: chr [1:21] "A" "A" "A" "B" ...
##   ..$ ID     : int [1:21] 1 2 3 4 5 6 7 8 9 10 ...
```

Maximum Discrimination Function

In (James and Sugar 2003) the authors suggest to consider all the information about the traits that distinguish one cluster from another. In particular, they calculate the optimal weights to apply to each dimension for determining cluster membership, as shown in Figure 14.

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

MaxDiscrPlots[[1]]
```

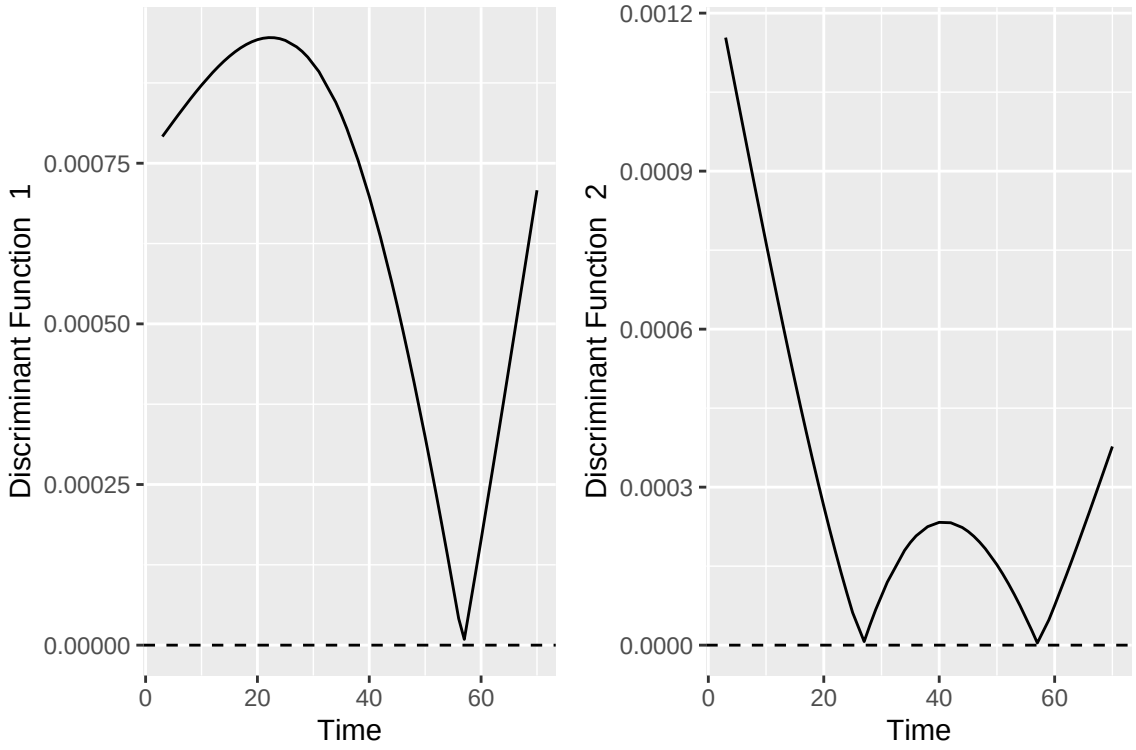

Figure 14: Discriminant curve.

Large absolute values correspond to large weights and hence large discrimination between clusters. Roughly speaking Figure 14 tells us that earlier measurements are important in determining cluster assignment as well as latter ones.

References

nocite: | Benzekry (2014) ...

Benzekry, Clare AND Beheshti, Sébastien AND Lamont. 2014. "Classical Mathematical Models for Description and Prediction of Experimental Tumor Growth." *PLOS Computational Biology* 10 (8): 1–19. https://doi.org/10.1371/journal.pcbi.1003800.

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.