Some useful R `fda` commands and data types

This is a quick reference for the most used commands and data types. The complete reference can be found by typing `?your-command` at the R console.

- `create.bspline.basis(rangeval, nbasis, norder)`
  Creates a B-splines function basis object that is used to describe your curves. A basis is defined on a time interval, specified by the numeric vector of length 2 `rangeval`, it consists of `nbasis` elementary basis components, which are adjacent hill-shaped curves, each curve being a polynomial of the same order `norder`. Usually `norder = 4`, which corresponds to a cubic curve, but in case you need to compute derivatives you have to add 1 for each derivative order, e.g. `norder = 5` if you need first derivative, i.e. velocity.

- `fd(coef, basis)`
  Creates a functional data object that describes a curve or a set of curves. The argument `basis` is the basis that is used to describe the curve(s), e.g. a B-spline obtained from `create.bspline.basis`. The `coef` argument specifies the coefficients associated to each elementary basis component, which for a B-spline roughly correspond to the height of each of the `basis$nbasis` adjacent hill-shaped curves. To represent only one curve, `coef` is a vector of length `basis$nbasis`; to represent N curves, `coef` is a matrix of dimension `basis$nbasis` × N; if the curves are D-dimensional trajectories, `coef` is a `basis$nbasis` × N × D-dimensional array. Normally `coef` is the result of a computation carried out in your code rather than manually designed, except for trivial cases, e.g. to produce a flat curve you need a `coef` vector of `basis$nbasis` identical values. The output of the command is an object of type `fd`, which is a list containing the `coef` and the `basis` components.

- `fdPar(basis, Lfdobj, lambda)`
  Creates an object (of type `fdPar`) containing all the information necessary to carry out smoothing. The argument `basis` is the basis that will be used to describe the curve(s); `Lfdobj` specifies the roughness penalty, typically `Lfdobj = 2`; `lambda` specifies the trade-off parameter between interpolation error and roughness penalty. Note: `Lfdobj + 2 = norder` should hold, where `norder` is the B-splines order (see `create.bspline.basis`).

- `smooth.basis(argvals, y, fdParobj)`
  Carries out smoothing from a set of curves described as sample values at specified points in time. `argvals` are the time points where all curves have been sampled. `y` is a vector or matrix or array containing the sample values at `argvals` time points, following the same conventions as in `fd$coef`. `fdParobj` is a `fdPar` object that specifies the common basis and the smoothing parameters. This command produces an object of type `fdSmooth` that is a list containing several components. You typically need only its `$fd` component, which is the produced functional data object.

- `eval.fd(evalarg, fdobj)`
  Returns the values of the curves represented by the `fd` object `fdobj` evaluated at the time points listed in the vector `evalarg`.

- `landmarkreg.nocurve(ximarks, x0marks, WfdPar=NULL, nhknots = 30, hlambda=1e-7, wlambda =1e-7)`
  This is a modified version of the `fda` function `landmarkreg`. This function computes the time warping required for landmark registration of curves. Contrary to the original `landmarkreg` function, this function does not carry out the curve registration itself, which has to be computed later on. Contrary to `landmarkreg`, it allows to change the total duration of curves. Finally, it provides not only the time warping information, but also the logarithm of the first derivative of time warping curves, which express the logarithm of the relative speech rate.
  The input is the N × L matrix `ximarks`, which specifies where landmarks are on each input curve (note: curves are *not* part of the input). N is the number of curves that have to be registered and L is the number of landmarks that are provided, which, contrary to `landmarkreg`, *always include the beginning and the end of the time interval where all curves are defined*. The vector `x0marks` has length L and specifies the desired landmark positions, i.e. where all the input landmarks should be moved to; if not specified, it is set to the mean of the input landmarks. The other parameters are needed in order to control the smoothing operations involved. `WfdPar` may be left unspecified, `wlambda` is the λ parameter involved in time warping, while `nhknots` and `hlambda` are the number of knots and the λ parameter for the representation of the warping function $h(t)$, which is not the same as the

one used internally to carry out the warping. Setting those values to the same used for smoothing curves is a good starting point, but tuning has to be carried out, especially when landmark positions vary a lot. The output is a list of the following components. `warpfd` is a `fd` object that describes the N time warping functions $h(t)$. `hfunmat` is a matrix containing the values of `warpfd` on a dense sequence of time points, `x` is the vector listing those time points. `logvelfd` is the transformation of $h(t)$ according to the formula:

$$-\log\frac{dh(t)}{dt}.$$

`land` is the desired position of landmarks (same as the input `x0marks` when specified). `landerr` shows the difference between the desired and the actual position of landmarks after registration, accounting for approximation errors and difficulties into accommodating large variations in landmark positions. This command may require a long execution time when many curves and many landmarks are used.

- `pca.fd(fdobj, nharm, harmfdPar)`
  Carries out FPCA on the set of curves represented by the functional data object `fdobj`. The first `nharm` PCs are computed. Since FPCA involves a smoothing operation, the `fdPar` object `harmfdPar` has to be specified. In general, PC curves may be expressed on a basis (included in `harmfdPar`) different from the basis used for the input curves `fdobj`, but usually the same basis is used for both. Setting `harmfdPar` equal to the `fdPar` object used for smoothing the input curves `fdobj` is usually a good solution. The output is a `pca.fd` object, which is a list containing the following components. `meanfd` is a `fd` object giving the mean function, `harmonics` is a `fd` object giving the PCs, `scores` is an N × `nharm` matrix providing the PC scores, `varprop` is a vector giving the proportion of variance explained by each PC.
  Note: the behavior of this function has changed after version 2.2.5, in particular the structure of `scores` is different in more recent versions. Use `pca.fd` version from package `fda_2.2.5.tar.gz` from CRAN or earlier.

- `plot.pca.fd.corr(pcafd, nx = 128, height = 240, pointplot = TRUE, harm = 0, expand = 0, pcweight = NULL, cycle = FALSE,`

```
land = NA, xlab = NULL, ylab = NULL, landlab = NULL, png = FALSE,
plots_dir = NULL, basename = NULL, ylog = NULL,...)
```

A modified version of the command `plot.pca.fd` that plots the FPCA object `pcafd` by showing mean curves and the effect of adding/subtracting PC curves multiplied by the standard deviation of the respective PC scores. If `png = TRUE` it uses the `png` command to produce a number of plots in the `plots_dir` directory, otherwise prints on the screen. Plot file names are concatenated as `basename` + component number + dimension number, the latter only when multidimensional FPCA was computed. `nx` is the number of points shown in the +/- graphical representation, `height` is a parameter passed to the `png` command, `land` provides the position of landmarks, `landlab` the labels associated to the intervals determined by the landmarks. `pcweight` is multiplicative factor in order to represent PC functions in their original scale. Useful if multidimensional signals were used and they were previously scaled to e.g. similar st. dev. `ylog = NULL` or `ylog = 'n'` means no log scale on the y axis, `ylog = 'y'` means that y values are already logs and are first transformed back by applying exponentiation and then represented in log scale. This is useful when relative speech rate is represented, so that values in terms of multiplicative rates are read on the y axis. In case of multidimensional plots, `ylog` is a vector, e.g. = `c('n','y')` to have log scale only on the second dimension. For the other parameters, see `?plot.pca.fd`.