fdata.usc Reference Card

March 27, 2019

Only main arguments are shown

fdata class objects

`fdata(mdata,argvals=NULL,...)` (class fdata) uses the evaluations at the discretization points and converts object of class: fd, fds, fts, sfts, vector, matrix, data.frame to an object of class fdata

`plot.fdata(x, type, main,...)` plots functional data class object

Basic operations

Group Math: `abs()`, `sqrt()`, `floor()`, `ceiling()`, `trunc()`, `round()`, `signif()`, `exp()`, `log()`, `cos()`, `sin()`, `tan()`

Group Summary: `all()`, `any()`, `sum()`, `prod()`, `min()`, `max()`, `range()`

Other S3 methods for class 'fdata': `[]`, `==`, `!=`, `+`, `-`, `*`, `/`, `^`, `is.fdata()`, `c()`, `dim()`, `ncol()`, `nrow()`, `NCOL()`, `NROW()`, `anyNA`, `count.na()`, `order.fdata(y, fdataobj, ...)`

Create Basis Set for Functional Data

`create.fdata.basis(fdataobj,l=1:5,maxl,type.basis,...)` computes fixed basis for FD

`create.pc.basis(fdataobj,l=1:5,...)` computes PCA basis for FD `create.pls.basis(fdataobj, y, l = 1:5,...)` computes PCA basis for FD

PCA and PLS for Functional Data

`fdata2pc(fdataobj, ncomp = 2,...)` computes (penalized) principal components (PC) for functional data

`fdata2pls(fdataobj, y, ncomp = 2,...)` computes penalized partial least squares (PLS) components for functional data

Tools for Functional Data

`fdata2fd(fdataobj, type.basis=NULL, nbasis=NULL,...)` converts fdata object to fd object (using the basis representation)

`fdata.deriv(fdataobj, nderiv=1, method="bspline",...)` computes the derivative of functional data.

`fdata.bootstrap(fdataobj,statistic=func.mean,...)` bootstrap samples for functional data.

Functional Smoothing

`S.basis(tt, basis, lambda=0,...)` provides the smoothing matrix with roughness penalties by fixed basis representation

`S.LLR(tt, h, Ker = Ker.norm,,...)` provides the smoothing matrix for the discretization points by Local Linear Regression

`S.NW(tt, h, Ker = Ker.norm,...)` provides the smoothing matrix for the discretization points by Nadaraya-Watson kernel estimator

`S.KNN(tt, h=NULL, Ker = Ker.unif,...)` provides the smoothing matrix for the discretization points by K nearest neighbors estimator

`min.basis()` computes a smooth functional data with a roughness penalty and number of basis representation selection by validation criteria

`min.np()` computes a smooth functional data with a kernel bandwidth selection by validation criteria for bandwidth selection

`CV.S()` computes the leave-one-out cross-validation (CV) score

`GCV.S()` computes the generalized correlated cross-validation (GCV) score

Inner product and norm for FD

`inprod.fdata(fdata1, fdata2=NULL, w = 1,...)` computes a inner products of functional data (fdata object)

`norm.fdata(fdataobj, metric=metric.lp,...)` approximates L_p-norm for functional data (fdata object)

`norm.fd(fdobj)` approximates L_2-norm for fd class object

Distance between functional elements

`metric.lp(fdata1,fdata2=NULL,lp=2,...)` for L_p-metric

`metric.hausdorff(fdata1, fdata2 = fdata1)` for Hausdorff distances between two sets of curves

`metric.kl(fdata1, fdata2 = NULL, symm=TRUE,...)` for Kullback Leibler distance between two groups of densities

`metric.dist(x, y = NULL, method = "euclidean",...)` for distances between the rows of a data matrix (wrapper of `dist()`)

`semimetric.basis(fdata1, fdata2 = fdata1, nderiv=0,...)` for distances based on fixed basis

`semimetric.deriv(fdata1, fdata2=fdata1, nderiv=1,...)` for distances between derivatives of the curves based on B-spline basis

`semimetric.fourier(fdata1, fdata2=fdata1, nderiv=0,...)` distance between the curves based on Fourier basis

`semimetric.hshift(fdata1, fdata2, ...)` computes distance between curves taking into account an horizontal shift effect

`semimetric.mplsr(fdata1, fdata2=fdata1, q=2, class1,...)` computes distance between curves based on the PLS

`semimetric.pca(fdata1, fdata2=fdata1, q=1, ...)` computes distance between curves based on PCA

Functional Univariate Data Depth

`depth.FM(fdataobj, fdataori = fdataobj, trim=0.25,...)` computes Fraiman and Muniz (FM) depth

`depth.mode(fdataobj, fdataori = fdataobj, trim=0.25,...)` computes modal depth

`depth.RT(fdataobj, fdataori = fdataobj, trim = 0.25,...)` computes random tukey (RT) depth

`depth.RP(fdataobj, fdataori = fdataobj, trim=0.25,...)` computes random project (RP) depth

`depth.RPD(fdataobj, fdataori = fdataobj, nproj=50,...)` double random project depth (RPD) depth

`depth.FSD(fdataobj, fdataori = fdataobj, trim = 0.25,...)` computes Functional Spatial depth

`depth.KFSD(fdataobj, fdataori = fdataobj, trim = 0.25,...)` computes Kernelized Functional Spatial depth

Functional Multivariate Data Depth

`depth.FMp(lfdata, lfdataref = lfdata, trim = 0.25,...)` computes Fraiman-Muniz depth

`depth.RPp(lfdata, lfdataref = lfdata, nproj = 50,...)` computes Random Projections depth

`depth.modep(lfdata, lfdataref = lfdata, h = NULL,...)` computes Modal depth

Multivariate Data Depth

`mdepth.MhD(x, xx=x,...)` computes Mahalanobis depth (Mean)

`mdepth.HS(x, xx=x, ...)` computes Halfspace depth, also known as Tukey Depth (Median)

`mdepth.SD(x, xx = NULL, ...)` computes Simplicial depth

`mdepth.LD(x,xx=x, ...)` computes Likelihood depth (Mode)

Functional Outlier Detection

`outliers.thres.lrt(fdataobj, nb=200, smo=0.05,...)` procedure for detecting functional outliers by likelihood ratio test

`outliers.depth.trim(fdataobj, nb=200, smo=0.05,...)` procedure for detecting functional outliers using trimmed data according to depth

`outliers.depth.pond(fdataobj, nb=200, smo=0.05,...)` procedure for detecting functional outliers using weights the data according to depth

Functional Data Generation

`rproc2fdata(n, t=NULL , ...)` generates curves from different processes: Ornstein Uhlenbeck, Brownian, Fractional Brownian, Gaussian or Exponential variogram

`gridfdata(coef, fdataobj, mu)` generates curves as lineal combination of the original curves plus a functional trend

`rcombfdata(n = 10, fdataobj, mu, ...)` generates random combinations of the curves plus a functional trend

Functional Linear Models

`fregre.lm(formula, data, basis.x=NULL,...)` computes LM model between functional (and non functional) explanatory variables and scalar response using (fixed or data-driven) basis representation

`fregre.basis(fdataobj, y, basis.x=NULL,...)` computes LM model between functional predictor and scalar response using fixed basis representation

`fregre.basis.cv(fdataobj, y, basis.x=NULL,...)` cross-validation functional regression with scalar response using basis representation

`fregre.pc(fdataobj, y, l =NULL,...)` computes functional (ridge or penalized) regression using PCA

`fregre.pc.cv(fdataobj, y, kmax=8,...)` computes functional (ridge or penalized) regression using selection of number of PC components

`fregre.pls(fdataobj, y=NULL, l = NULL,...)` computes functional linear regression using penalized PLS

`fregre.pls.cv(fdataobj, y, kmax=8,...)` computes functional linear regression using selection of number of PLS components

Goodness-of-fit test for the FLM

`flm.Ftest (X.fdata, Y, B=5000, verbose=TRUE)` tests the null hypothesis of no interaction between a functional covariate and a scalar response inside the Functional Linear Model

`flm.test(X.fdata, Y, beta0.fdata = NULL,...)` tests the composite null hypothesis of a FLM with scalar response

Model fitting

`summary(object,...)` summarizes information from fitted models such as: `fregre.pc`, `fregre.basis`, `fregre.pls`, `fregre.np`, and `fregre.plm`

`predict(object,...)` predictons from object fitted such as: `fregre.pc`, `fregre.basis`, `fregre.pls`, `fregre.np`, and `fregre.plm`

`influence.fdata(model,...)` computes influence measures from FLM

`fregre.bootstrap(model, nb = 500, wild = TRUE,...)` estimate the beta parameter by wild or smoothed bootstrap procedure

Functional Non-Linear Models

`fregre.np(fdataobj,y,h=NULL,...)` computes functional regression between functional explanatory variables and scalar response using kernel estimation

`fregre.np.cv(fdataobj, y, h=NULL,...)` computes functional regression between functional explanatory variables and scalar response using kernel estimation by cross-validation method

`fregre.plm(formula, data, h=NULL,...)` compute semi–functional partially linear model with scalar response using kernel estimation of functional predictor

Generalized Regression Models

`fregre.glm(formula,family = gaussian(), data,...)` fits functional GLM model

`fregre.gsam(formula, family = gaussian(), data,...)` fits functional GAM model with integrated Smoothness estimation and basis representation

`fregre.gkam(formula, family = gaussian(),data,...)` fits functional GAM model with Kernel estimation

Functional Response Model

`fregre.basis.fr(x,y, basis.s=NULL,...)` fits functional response models using basis representation

Models for Dependent Data

`fregre.gls(formula, data, correlation = NULL,...)` fits functional generalized least squares (GLS) model

`fregre.igls(formula, data, basis.x=NULL,...)` fits functional generalized least squares (GLS) model iteratively

`GCCV.S(y, S, criteria="GCCV1",...)` generalized correlated cross-validation (GCV) score

`predict.fregre.gls(object, newx = NULL, type = "response",...)` predictions from a functional gls object

`predict.fregre.igls(object, new.fdataobj = NULL, data,...)` predictions from a functional iteratative gls object

Variable and Impact Points Selection

`fregre.gsam.vs(data, y, x, alpha, type.basis = "pc", ...)` fits functional GAM model by selecting the model variables (between functional, scalar, factors,. . .)

`LMDC.select(y, covar, data,...)` selects impact points of functional predictor using local maxima distance correlation (LMDC) for a scalar response given

`LMDC.regre (y, covar, data, newdata,...)` fits a multivariate regression method using the selected impact points like covariates for a scalar response

Functional Classification

`classif.glm(formula, data, family = binomial(),...)` computes functional classification using functional (and non functional) explanatory variables by basis representation

`classif.knn(group, fdataobj, knn=NULL,...)` fits kNN Supervised Classification for Functional Data.

`classif.kernel(group, fdataobj,h=NULL,...)` fits Nonparametric (Kernel) Supervised Classification for Functional Data.

`classif.gsam(formula,data, family = binomial(),...)` computes functional classification using functional (and non functional) explanatory variables by basis representation

`classif.gkam(formula, family = binomial(), data,...)` computes functional classification using functional explanatory variables using backfitting algorithm

`classif.tree(formula, data, basis.x=NULL,...)` computes functional classification by Recursive Partitioning and Regression Trees (see `rpart`)

Functional Depth Classification

`classif.depth(group,fdataobj,newfdataobj,...)` classification of functional data using maximum depth

`classif.DD(group, fdataobj, depth="FM", classif="glm",...)` fits nonparametric classification procedure based on DD–plot (depth-versus-depth plot) for G dimensions

Functional Non-Supervised Classification

`kmeans.fd(fdataobj, ncl = 2, metric = metric.l,...)` allows the estimation of the groups in a functional data set fdata class by k–means method

Functional ANOVA

`anova.hetero(object = NULL, formula, pr = FALSE,...)` fits a univariate analysis of variance model for heteroscedastic data and allows calculate special contrasts defined by the user

`anova.onefactor(object, group, nboot=100,...)` contrasts the null hypothesis of equality of mean functions of functional data based on the an asymptotic version of the anova F–test

`anova.RPm(object, formula, data.fac,...)` tests ANOVA models for functional data with continuous covariates based on the analysis of randomly chosen one-dimensional projections

Conditional Distribution Function

`cond.F(fdata0, y0, fdataobj,...)` calculates the conditional distribution function of a scalar response with functional data.

Datasets

`aemet` contains geographic information of 73 Spanish weather station and the average for the period 1980-2009 of daily temperature, precipitation and wind speed. Meteorological State Agency of Spain (AEMET), http://www.aemet.es/

`MCO` The mithochondrial calcium overload (MCO) was measured in two groups (control -original cells - and treatment -permeabilized cells-) every 10 seconds during an hour in isolated mouse cardiac cells.

`phoneme` contains 250 learning curves and 250 test curves discretized in 150 log-periodograms points corresponding to 5 class membership (five phonemes): "sh" 1, "iy" 2, "dcl" 3, "aa" 4 and "ao" 5 (50 by class level).

`poblenou` NOx levels measured every hour by a control station in Poblenou in Barcelona (Spain). The dataset starts on 23 February and ends on 26 June, in 2005.

`tecator` contains the Water, Fat and Protein content of 215 meat samples and the 215 absorbance curves recorded in the wavelength range 850 - 1050 nm.

Documentation

Created by Manuel Oviedo de la Fuente, website http://eio.usc.es/pub/moviedo and by Manuel Febrero Bande, website http://eio.usc.es/pub/febrero, feel free to contact us for contributions.

Links

fda.usc R package http://cran.r-project.org/web/packages/fda.usc

JSS paper http://www.jstatsoft.org/article/view/v051i04

fda.usc R Manual https://cran.r-project.org/web/packages/fda.usc/fda.usc.pdf

CRAN Task View https://cran.r-project.org/web/views/FunctionalData.html

RPubs document

Installation and Descriptive Statistics http://rpubs.com/moviedo/fda_usc_introduction

Functional Regression http://rpubs.com/moviedo/fda_usc_regression

Functional Classification and ANOVA http://rpubs.com/moviedo/fda_usc_classification

HTML reference card http://rpubs.com/moviedo/fda_usc_rcard