Package 'cSFM'

July 2, 2014

Type Package

Title Covariate-adjusted Skewed Functional Model (cSFM)

Version 1.1

Date 2014-01-20

Author Meng Li, Ana-Maria Staicu, and Howard D. Bondell

Maintainer Meng Li <mli9@ncsu.edu>

Depends R (>= 2.15.3), sn

Imports mgcv, mnormt, MASS, moments, splines

LazyLoad yes

Description

cSFM is a method to model skewed functional data when considering covariates via a copula-based approach.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2014-01-23 16:49:05

R topics documented:

`cSFM-package` *Covariate-adjusted Skewed Functional Model*

Description

This package provides functions for functional data analysis, when covariate effect is present and high-moment information is considerred. Skewed normal distributions are used for the pointwise distributions, and the dependence is modeled by a Gaussian copula.

Details

Package:	cSFM
Type:	Package
Version:	1.1
Date:	2014-01-20
License:	GPL-2

The main function `cSFM.est` applies one of the proposed method `cSFM` and its variates `2cSFM` and `cSFM0` to the observed data. The generic functions `print`, `fitted` and `predict` are applicable for summarizing the output, obtaining fitted values (including quantile estimation) and predicting at new data point. See `data.simulation` for an example of the simulated data where various methods including cSFM and other variantes can be applied to. Kronecker product basis is exploited for bivariate parameter functions. The smoothness is controlled by the number of knots which is selected by minimizing AIC; the function `cSFM.est.parallel` allows parallel computing for various combinations of knots when the facility is available to the users.

References

[1]. Meng Li, Ana-Maria Staicu and Howard D. Bondell (2013), Incorporating Covariates in Skewed Functional Data Models. http://www.stat.ncsu.edu/information/library/papers/mimeo2654_Li.pdf.

See Also

`cSFM.est`, `cSFM.est.parallel`, `data.simulation`.

`case2.b.initial` *Initial Estimates of Parameter Functions*

Description

Obtain the initial estimates of functional parameters (mean, var, shape and skewness) when both the mean and variance are covariate depedent but the skewness is covariate independent.

Usage

```
case2.b.initial(y, tp, cp, nbasis.mean = 10, gam.method = "REML",
                bin = 10, skew.method = "mle", cate = 1)
```

Arguments

`y`	observed data matrix
`tp`	timepoint vector with length = `ncol(y)`
`cp`	covariate vector with length = `nrow{y}`
`nbasis.mean`	number of bases when smoothing the mean
`gam.method`	smoothing method for the mean
`bin`	the length of bin to estimate the variance
`skew.method`	estimation method for skewness; See 'Details'
`cate`	method category, taking values as `1,2,3`; See 'Details'

Details

The variance is estimated using binning method with length = `bin`. The skewness can be estimated by method of moment `mome` or maximum likelihood `mle`. Stepwise estimates are used here: the variance is based on the residuals after removing the mean effect; the skewness is based on the scaled residuals after removing both the mean and variance effect.

`cate` indiates the method cateory as follows:

- `cate = 1` all three parameter functions are to be estimated
- `cate = 2` the skewnewss parameter is fixed as 0
- `cate = 3` the mean to be fixed at 0; typically used when a stepwise estimation procedure is used, e.g. `2cSFM` in `cSFM.est`

Value

A list of initial estimate of parmeters (length 4: mean, variance, shape and skewness).

See Also

`cSFM.est`

Examples

```
data(data.simulation)
## Not run:
cp.hat <- case2.b.initial(y = DST$obs, tp= DST$tp, cp=DST$cp)
# visulize the parameter function and the estimated function
par(mfrow = c(1,2))
persp(DST$cp, DST$tp, exp(DST$pars$logvar), theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="variance surface")
persp(DST$cp, DST$tp, cp.hat$var, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="initial estiamtes")

## End(Not run)
```

---

cp.beta *Transformation between Parameters and B-spline Coefficients*

Description

Given a B-spline basis, transfer parameters (mean, variance, skewness) to the corresponding coefficients, and vice versa.

Usage

```
cp2beta(cp.list, Basis.list)
beta2cp(vec.beta, Basis.list)
```

Arguments

`cp.list`	list of parameters with names to be (`mean`, `var`, `skew`) corresponding to (mean, variance, skew)
`Basis.list`	list of basis matrices for the mean, variance and skewness
`vec.beta`	vector of coefficients

Value

`cp2beta` returns a vector of coefficients with the same form of `vec.beta`; `beta2cp` returns a list of parameters with the same form of `cp.list`

References

[1]. Meng Li, Ana-Maria Staicu and Howard D. Bondell (2013), Incorporating Covariates in Skewed Functional Data Models. http://www.stat.ncsu.edu/information/library/papers/mimeo2654_Li.pdf.

Examples

```
data(data.simulation)
# bivariate for mean and variance; univariate for shape parameter
cases = c(2,2,1)
# 2 knots at time direction for each parameter
nknots.tp = c(2,2,2)
# 2 knots at covariate direction for mean and variance
nknots.cp = c(2,2)
basis.list <- lapply(1:3, function(k)
  kpbb(DST$tp, DST$cp, nknots.tp = nknots.tp[k],
       nknots.cp= nknots.cp[k], sub.case=cases[k]))
cp.hat <- DST$pars # true parameters
cp.hat$var <- exp(cp.hat$logvar) # follow the fomart stricely: (mean, var, skew)
beta <- cp2beta(cp.hat, basis.list)
cp.recover <- beta2cp(beta, basis.list)
norm(cp.hat$mean - cp.recover$mean)
```

`cSFM object` *Generic Method for 'cSFM' Objects*

Description

Print, extract fitted values and predict for cSFM object.

Methods of the generic function `print`, `fitted` and `predict` for objects inheriting from class cSFM.

Usage

```
## S3 method for class 'cSFM'
print(x, ...)
## S3 method for class 'cSFM'
fitted(object, quantile = TRUE,
                     quantile.level = c(0.5, 0.8, 0.9, 0.95, 0.99), ...)
## S3 method for class 'cSFM'
predict(object, newdata, cp.valid, tp.valid = NULL, ...)
```

Arguments

`x`	A cSFM object; returned by the function `cSFM.est`
`object`	A cSFM object; returned by the function `cSFM.est`
`quantile`	logical; if TRUE(default), estimates of quantiles are returned
`quantile.level`	quantile vector; quantile levels to be estimated when `quantile = TRUE`
`newdata`	the partically observed new data set (missing data allowed) to be predicated
`cp.valid`	observed covariate vector for `newdata` with length `nrow{newdata}`
`tp.valid`	observed timepoint vector for `newdata` with length `ncol{newdata}`; See "Details".
`...`	other arguments passed to the generic functions

Details

When use the function `predict`, each row of `newdata` corresponds to covariate information `cp.valid`, while the column is for the time points `tp.valid`. When `tp.valid` is `null`, then we assume the validation data set has the same time points as the training data set, which is used to obtain `object`.

See Also

`cSFM.est`, `print`, `fitted`, `predict`

`cSFM.est` *Model Estimation with Bivariate Regression B-Splines*

Description

Function used for model estimation with bivariate (or univariate) B-splines, when the number of knots are given for each parameters (mean, variance and skewness). Kronecker product is used for bivariate case. It returns cSFM objects.

Usage

```
cSFM.est(data, tp, cp, nknots.tp, nknots.cp, degree.poly = c(3, 3, 3),
        method = c("cSFM", "cSFM0", "2cSFM"), bi.level = 2,
        nbasis.mean = 10, gam.method = "REML")
```

Arguments

`data`	The fully observed data matrix n by m. The number of row n is the number of subjects; the number of columns m is the number of time points for each subject.
`tp`	The timepoint vector of length m, shared by each subject
`cp`	The covariate vector of length n corresponding to n subjects
`nknots.tp`	The vector (k1,k2,k3) for the number of knots (mean, variance, skewness) in the time direction
`nknots.cp`	The vector (s1,s2,s3) for the number of knots (mean, variance, skewness) in the covariant direction; the effective length is `bi.level`. For example, if `bi.level = 2`, only the first two numbers (s1, s2) are used
`degree.poly`	The vector (d1,d2,d3) for the degree of B-splines (mean, variance, skewness) in both the time and covariant direction; the cubic splines are the default splines.
`method`	Estimation method for the model. See 'Details'.
`bi.level`	Bivariate level taking values at 0, 1, 2, 3. See 'Details'.
`nbasis.mean`	Number of basis functions to smooth the mean; only applicable when `method` is "2cSFM".
`gam.method`	Method to smooth the mean; only applicable when `method` is "2cSFM".

Details

The three methods for the argument `method` are all based on skewed normal and corpula, but exploits different approaches to conduct the estimation. Method "`cSFM`" assumes the full model and estimate all the parameter functions (mean, variance, skewness) jointly; Method "`cSFM0`" enforces the skewness to be zero, and estimate the other two parameter functions jointly; Method "`2cSFM`" first estimate the mean function via penalized bivariate spline and then estimate the variance and skewness for the residulas, thus it is a Stepwise approach.

The argument `bi.level` indicates the bivariate dependence for all three parameters. For example, Bi.level = 2 means the parameters are bivariate up to the 2nd momdnet, i.e. the mean and variance are bivariate while the skewness is univariate.

Value

A cSFM object as a list with components:

`beta.hat`	Estimated coefficients
`cp.hat`	Estimated parameter functions including the mean, variance and skewness
`copula`	Estimated copula for the data
`corr.est`	Estimated correlation matrix for the copula
`AIC`	AIC for the model
`num.pars`	Number of parameters in the model
`logL1`	Marginal loglikelihood value
`logL2`	Loglikelihood value for the corpula
`sm.basis`	Smoothing Matrices used as the basis
`gam.mean`	Results when smoothing the mean, returned by `gam` function and is only applicable for method `2cSFM`

References

[1]. Meng Li, Ana-Maria Staicu and Howard D. Bondell (2013), Incorporating Covariates in Skewed Functional Data Models. http://www.stat.ncsu.edu/information/library/papers/mimeo2654_Li.pdf.

See Also

`print.cSFM`, `predict.cSFM`, `fitted.cSFM`, `gam`, `data.simulation`

Examples

```
data(data.simulation) # load benchmark data
y <- DST$obs # matrix of observation
tp <- DST$tp; cp <- DST$cp

# fixed number of knots
# bivariate for mean and variance, univrariate for shape
nknots.tp <- c(2,2,2)
nknots.cp <- c(2,2)
```

```
## Not run:
fit.cSFM <- cSFM.est(y, tp, cp, nknots.tp, nknots.cp, method = "cSFM")
fit.cSFM #print out the results briefly
# fitted mean, logvar and skewness; fitted quantiles
fitted.value <- fitted(fit.cSFM)

# visualize the parameter function and the estimated function
par(mfrow = c(1,2))
persp(DST$cp, DST$tp, DST$pars$mean, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="parameter surface")
persp(DST$cp, DST$tp, fit.cSFM$cp.hat$mean, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="estimated surface via 'cSFM' method")

# predication for missing data
y.valid <- DSV$obs # matrix of training data; include missing values
tp.valid <- DSV$tp; cp.valid <- DSV$cp
#predication for training data
yhat.predict <- predict(fit.cSFM, y.valid, cp.valid, tp.valid)

# visualize the data and predicted surface
par(mfrow = c(1,2))
persp(DSV$cp, DSV$tp, y.valid, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="data surface (partically observed)")

persp(DSV$cp, DSV$tp, yhat.predict, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="predicted surface via 'cSFM' method")

# predication error
mean(((yhat.predict - DSV$obs.full)[!is.na(DSV$obs)])^2)

## End(Not run)
```

`cSFM.est.parallel` *Knots Selection by AIC*

Description

Select the best number of knots to minimize AIC. This is a convinient function to implement `cSFM.est` for different combination of knots by allowing parallel computing.

Usage

```
cSFM.est.parallel(data, tp, cp, nknots.tp = NULL, nknots.cp = NULL,
                  max.knots = NULL, parallel = TRUE, num.core = NULL,
                  method = c("cSFM", "cSFM0", "2cSFM"), bi.level = 2,
                degree.poly = c(3, 3, 3), nbasis.mean = 10, gam.method = "REML")
```

Arguments

`data`	The fully observed data matrix n by m. The number of row n is the number of subjects; the number of columns m is the number of time points for each subject.
`tp`	The timepoint vector of length m, shared by each subject
`cp`	The covariate vector of length n corresponding to n subjects
`nknots.tp`	knots matrix with each row to be a vector (k1,k2,k3) for the number of knots (mean, variance, skewness) in the time direction
`nknots.cp`	knots matrix with each row to be a vector (s1,s2,s3) for the number of knots (mean, variance, skewness) in the covariant direction; the effective length is `bi.level`. For example, if `bi.level = 2`, only the first two numbers (s1, s2) are used
`max.knots`	the maximum number of knots to be considerred; `nknots.tp` and `nknots.cp` are not used if `max.knots` is provided; See "Details"
`parallel`	logical indicator; whether parallel computing will be used or not
`num.core`	number of cores to be used when `parallel = TRUE`; default to be the total available cores
`method`	Estimation method for the model. See `cSFM.est`
`bi.level`	Bivariate level taking values at 0, 1, 2, 3. See `cSFM.est`.
`degree.poly`	The vector (d1,d2,d3) for the degree of B-splines (mean, variance, skewness) in both the time and covariant direction; the cubic splines are the default splines.
`nbasis.mean`	Number of basis functions to smooth the mean; only applicable when `method` is "2cSFM".
`gam.method`	Method to smooth the mean; only applicable when `method` is "2cSFM".

Details

This function is mainly based on `cSFM.est`, except that different sets of knots are considerred here. The parallel computing is based on the package `parallel`.

When supplied, `max.kntos` will determine `(knots.tp, knots.cp)` as follows. For each direction (time and covariate) and each parameter (mean, variance and skewness), the number of knots is from 1 to `max.knots`; we enforce the lower moments have more knots than higher moments (for example, mean has more knots than variance), therefore delete those exceptions to give `(knots.tp, knots.cp)`.

Value

best.cSFM	cSFM object with the minimal AIC value; see cSFM.est for a complete explanation of HAC object
knots.mat	matrix for knots to be considerred; each row is a vector with length 6: first three are knots for time direction, while the last three are knots for covariate direction
AIC	AIC vector for all sets of knots

Note

This function is the parallel version of cSFM.est. The major difference is that this function allows for a pool of knots combination and apply the parallel computing to save time. Most of the arguments are the same between the two functions.

See Also

cSFM.est, clusterMap

Examples

```
## Not run:
data(data.simulation)
# Example 1: use the convinient default to generate knots
ret <- cSFM.est.parallel(DST$obs, DST$tp, DST$cp, max.knots = 4)
# AIC vector
ret$AIC
# best number of knots
ret$knots.mat[which.min(ret$AIC), ]
# Example 2: assign combinations of knots subjectively
nknots.tp = rbind(c(3,2,1), c(6,5,4))
nknots.cp = nknots.tp
ret2 <- cSFM.est.parallel(DST$obs, DST$tp, DST$cp,
                          nknots.tp = nknots.tp, nknots.cp = nknots.cp)

## End(Not run)
```

D.SN *Derivatives of Normalized Skewed Normal Parameterized by Shape*

Description

Calculate the derivatives of Skewed Normal density with location = 0 and scale = 1 when parameterized by shape parameter. The probability density function (pdf) is a bivariate function, wrt. y and a. The first and second derivatives are calculated.

Usage

```
D.SN(y,a)
```

Arguments

y	vector or matrix of quantiles, taking values in the real line; Missing values (NA's) and Inf's are allowed
a	vector or matrix of shape parameters, taking values in the real line; should have the same dimension as y

Details

In general, a skewed normal is parameterized by three parameters (location, scale, shape). It is referred as normalized skewed normal distribution in some cases when location = 0 and scale = 1. A normalized skewed normal distribution with location = 0 and scale = 1 has the density

$$f(x, a) = 2\phi(x)\Phi(x * a),$$

where a is the shape parameter; here ϕ is the pdf of the standard normal distribution, and Φ is the corresponding cdf. The function D.SN(x,a) calculates the function value of $f(x, a)$, along with the first and second order derivatives.

Value

A list with the following components

sn	function value of $f(x, a)$
sn1	the first derivative $f'(x, a)$ wrt. x
sn11	the second derivative $f''(x, a)$ wrt. x
sn12	the cross-partial $f''(x, a)$ wrt. x and a
sn2	the first derivative $f'(x, a)$ wrt. a
sn22	the second derivative $f''(x, a)$ wrt. a

References

[1]. Azzalini, A. (1985). A class of distributions which includes the normal ones. Scandinavian Journal of Statistics, 171-178.

[2]. A very brief introduction to the skew-normal distribution

See Also

skewness.cp, shape.dp, dsn

Examples

```
ret1 <- D.SN(0.1, 10)   # y = 0.1, a = 10
ret2 <- D.SN(-0.1, -10) # y = -0.1, a = 10
ret3 <- D.SN(rnorm(10), rnorm(10))   # y and a are a vector
# y and a are matrices
ret4 <- D.SN(matrix(rnorm(10), 2, 5), matrix(rnorm(10), 2, 5))
```

`Data Simulation` *Generate Data using Skewed Pointwise Distributions and Gaussian copulas*

Description

Generate function data with skewed pointwise distributions using a Gaussian copula. The Gaussian copula is generated by a eigen-decomposition with the eigenfunction specified by a basis system. This function can be used for simulation to check the performance of various approaches.

Usage

```
data.generator.y.F(n.subject, n.timepoints, s, D, csi,lambdas,
                   basis.system=legendre.polynomials, var.noise=0.10)
```

Arguments

`n.subject`	number of subjects in the data denoted by n
`n.timepoints`	number of timepoints, which are equally spaced in [0,1] and denoted by m
`s`	vector with length m or matrix with dimension m by n for the mean parameter; See "Details"
`D`	vector with length m or matrix with dimension m by n for the log varaince parameter; See "Details"
`csi`	vector with length m or matrix with dimension m by n for the shape parameter; See "Details"
`lambdas`	vector of eigenvalue for the latent Gaussian process
`basis.system`	basis system for the latent Gaussian process; `legendre.polynomial` uses the Legendre polynomials and `DFT.basis` uses Fourier basis.
`var.noise`	variance of white noises to corrupte the latent Gaussian Process

Details

The generated data is determined by both the marginal distributions and dependence structure. The marginal distributions require three parameters (mean, variance, shape), which corresponds to `(s, D, csi)`. Each of them can be a vector or a matrix, indicating the distributions to be covariate indepedent (univariate) or covariant dependent (bivariate). When the parameter is a vecotor, it must have the length of `n.timepoints`.

The latent Gaussian process with 0 mean and unit variance is determined by a correlation matrix. The correlation is given by `lambdas, basis.system`, which specify the eigenvalues and eigenfunctions correspondingly. We allow measurement error for the considerred Gaussian process, to be corrupted by independent Gaussian noises with mean 0 and variance `var.noise`.

Value

A list with two components:

`data`	generated data as a matrix with dimension m by n
`corr.true`	correlation matrix of the latent Gaussian process (Gaussian copula is used)

Note

When the shape parameter is equal to 0, then the generated data is a regular functional data with covariate-adjusted mean and variances and Gaussian errors.

References

[1]. Meng Li, Ana-Maria Staicu and Howard D. Bondell (2013), Incorporating Covariates in Skewed Functional Data Models. http://www.stat.ncsu.edu/information/library/papers/mimeo2654_Li.pdf.

Examples

```
###### generate the data set ####################################
set.seed(2013)
n.sub <- 100    # number of subjects
n.tim <- 80     # number of timepoints
# true population-level functions (mean, standard deviation and shape)
true.fun <- function(par){
  if (par == "mean") f =function(x,y) sin(pi*x)* cos(pi*y)
  if (par == "logvar"){
    f=function(x,y)   2*dnorm(x, mean=0.5, sd = 0.2, log=TRUE) +
      2*dnorm(y, mean=0.5, sd = 0.2, log =TRUE) - 8
  }
  if (par == "shape") f=function(x,y)  10*sin(2*pi*y)
  return(f)
}
# covriate: Sort at the very beginning
point.cov <- sort(runif(n.sub));
# timepoints
point.tim <- seq(from=0, to=1, length=n.tim);
# calculate and collect all the true population-level parameters
col.true <- list(mean = outer(point.cov, point.tim, true.fun("mean")),
                 logvar = outer(point.cov, point.tim, true.fun("logvar")),
                 shape = outer(point.cov, point.tim, true.fun("shape")))

# generate data
my.data <- data.generator.y.F(n.subject = n.sub, n.timepoints = n.tim,
                              s = col.true$mean, D = col.true$logvar,
                              csi=col.true$shape,
                              lambdas = c(1/2, 1/4, 1/8),
                              basis.system = legendre.polynomials,
                              var.noise = 0.10)
# here shape is covariate independent
# so it's sufficient to keep 1st row of shape to use csi = col.true$shape[1,]
```

```
#####################Visualize the data ####################################
par(mfrow = c(2,2))
# plot the data surface
persp(point.cov, point.tim, my.data$data, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="data surface")
# plot the mean surface
persp(point.cov, point.tim, col.true$mean, theta=45, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="mean", main="mean surface")
# the logvar surface
persp(point.cov, point.tim, col.true$logvar, theta=45, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="logvar", main="logvar surface")
# the shape surface
persp(point.cov, point.tim, col.true$shape, theta=45, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="shape", main="shape surface")
```

`DFT.basis` *Discrete Fourier Transformation (DFT) Basis System*

Description

Generate an orthogonal Fourier basis system in the interval [0, 1]. The set of basis functions are used to be the eigenfunctions to generate the covariance matrix of a latent process.

Usage

```
DFT.basis(t, degree = 0, normalized = TRUE)
```

Arguments

`t`	the set of values to be evaluated, taking values from [0, 1]
`degree`	the degree of Fourier basis functions, taking values as $0, 1, 2, \ldots$; See 'Details'
`normalized`	logical value; If TRUE (default) then the values are normalized such that the $L2$ norm of the function values is 1

Details

The Fourier basis functions considered here are

$$1, \sqrt{2}cos(2\pi t), \sqrt{2}sin(2\pi t), \sqrt{2}cos(4\pi t), \sqrt{2}sin(4\pi t), \ldots,$$

which corresponding to `degree` = 0, 1, 2, 3, 4 .... Typically the `degree` is even.

Value

A vector which are the evaluations of the Fourier basis function at `t`.

See Also

`legendre.polynomials`

Examples

```
# time points
t <- seq(from = 0, to = 1, length = 100)
# basis matrix evaluated at the time points t
# an intercept column is included
Phi <- cbind(DFT.basis(t, degree = 0), DFT.basis(t, degree = 1),
             DFT.basis(t, degree = 2))
# check the orthogonality
crossprod(Phi) # is equal to I_3 up to rounding errors
# plot the basis system
matplot(t, Phi, type = "l", lwd = 2, lty = 1:3,
        ylab = "basis function", main = "Fourier Basis (normalized)")
legend("top", c("degree = 0", "degree = 1", "degree = 2"),
       col = 1:3, lwd = 2, lty = 1:3)
```

---

kpbb *Kronecker Product Bspline Basis*

Description

Generate the B-spline basis matrix for a polynomial spline, applicable for both univariate basis and bivariate basis systems.

Usage

```
kpbb(tp, cp = NULL, nknots.tp, nknots.cp = NULL,
     sub.case = 2, degree.poly = 3)
```

Arguments

`tp`	timepoint vector in the time direction; must be sorted
`cp`	covariate vector in the covariate direction; must be sorted
`nknots.tp`	number of knots for the time direction; See 'Details'
`nknots.cp`	number of knots for the covariate direction; See 'Details'
`sub.case`	indicator for univariate (`sub.case = 1`) or bivariate (`sub.case = 2`)
`degree.poly`	degree of polynomials for B-spline basis system; default is `3`, i.e. cubis splines

Details

kpbb is built based on the function bs. For the time direction, the generated basis matrix has the dimension length(tp) by nknots.tp + degree.poly + 1, regardless whehter the number of knots is negative or not. When sub.case = 2, the generated basis matrix is the kronecker product of the two basis matrices.

Value

A basis matrix, with attributes which are for the future use of predict.kpbb.

References

[1]. Meng Li, Ana-Maria Staicu and Howard D. Bondell (2013), Incorporating Covariates in Skewed Functional Data Models. http://www.stat.ncsu.edu/information/library/papers/mimeo2654_Li.pdf.

See Also

bs, predict.kpbb, data.simulation

Examples

```
data(data.simulation) # load benchmark data
y <- DST$obs # matrix of observation
# generate bivariate basis using 3 knots for each direction
basis <- kpbb(DST$tp, DST$cp, nknots.tp = 3, nknots.cp = 3)
# linear regression
lm.fit <- lm(as.vector(y) ~ basis - 1)
y.fit <- matrix(fitted(lm.fit), nrow = 100, ncol = 80)

# visualize the data and fitted surface
par(mfrow = c(1,2))
persp(DST$cp, DST$tp, y, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="data surface")
persp(DST$cp, DST$tp, y.fit, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="fitted surface via lm")
```

---

legendre.polynomials *Orthogonal Legendre Polynomials Basis System*

Description

Generate Legendre polynomials as an orthogonal basis system in the interval [0, 1]. The set of polynomial functions are used to be the eigenfunctions to generate the covariance matrix of a latent process.

Usage

```
legendre.polynomials(t, degree, normalized = TRUE)
```

Arguments

t	the set of values to be evaluated, taking values from [0, 1]
degree	the polynomial degree of the function, taking values as 0 (constant), 1 (linear), 2 (quadratic) and 3 (cubic)
normalized	logical value; If TRUE (default) then the values are normalized such that the $L2$ norm of the function values is 1

Details

Legendre polynomials can take any integers as degree, however, it is sufficent for most applications up to degree = 3. It is also messy to obtain the exploit expression for a degree greater than 3.

The logical value normalized take TRUE as the default, which means the generated eigenfunction basis matrix will be an orthogonal matrix.

Value

A vector which are the evaluations of the Legendre polynomial function at t.

See Also

DFT.basis

Examples

```
# time points
t <- seq(from = 0, to = 1, length = 100)
# basis matrix evaluated at the time points t
# an intercept column is included
Phi <- cbind(legendre.polynomials(t, degree = 0), legendre.polynomials(t, degree = 1),
             legendre.polynomials(t, degree = 2), legendre.polynomials(t, degree = 3))
# check the orthogonality
crossprod(Phi) # is equal to I_4 up to rounding errors
# plot the basis system excluding the constant column
matplot(t, Phi[,2:4], type = "l", lwd = 2, lty = 1:3,
        ylab = "basis function", main = "Legendre Polynomials (normalized)")
legend("top", c("linear", "quadratic", "cubic"), col = 1:3, lwd = 2, lty = 1:3)
```

predict.kpbb *Evaluate a predefined Kronecker product B-spline basis at provided values*

Description

Evaluate a predefined Kronecker product B-spline basis at provided values.

Usage

```
## S3 method for class 'kpbb'
predict(object, tp.valid, cp.valid = NULL, ...)
```

Arguments

object	the result returned by kpbb with corresponding attributes
tp.valid	the new time points at which the evaluations are requested; mainly for validation purpose
cp.valid	the new covariate points at which the evaluations are requested; mainly for validation purpose
...	other arguments passed to the generic function

Details

When cp.valid is allowed to be null, when only an univariate basis is considerred at object.

Value

An object just like the inputed object, except evaluated at the new values of tp.valid and cp.valid.

See Also

predict.kpbb, data.simulation

Examples

```
data(data.simulation) # load benchmark data
y <- DST$obs # matrix of observation
y.valid <- DSV$obs # matrix of training data; include missing values
# generate bivariate basis using 3 knots for each direction
basis <- kpbb(DST$tp, DST$cp, nknots.tp = 3, nknots.cp = 3)
# linear regression
lm.fit <- lm(as.vector(y) ~ basis - 1)
# prediction
new.basis <- predict(basis, tp.valid = DSV$tp, cp.valid = DSV$cp)
y.predict <- matrix(crossprod(t(new.basis),coef(lm.fit)), nrow = length(DSV$cp))

# visualize the data and predicted surface
```

```
par(mfrow = c(1,2))
persp(DSV$cp, DSV$tp, y.valid, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="data surface (partically observed)")

persp(DSV$cp, DSV$tp, y.predict, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="predicted surface via lm")

# predication error
mean(((y.predict - DSV$obs.full)[!is.na(DSV$obs)])^2)
```

`Reparameterization` *Reparameterize Skewed Normal Parameterized using Shape and Skewness.*

Description

The transformation functions used to reparameterize skewned normal from shape (direct parameter) to skewness (central parameter), and vice visa.

Usage

```
skewness.cp(alpha)
shape.dp(gamma)
D.gamma(alpha)
```

Arguments

`alpha`	shape parameters
`gamma`	skewness parameters

Details

For skewed normal distributions, there is a one-to-one mapping from the shape to the skewness, regardless of the other parameters such as mean and variances. The parameters (mean, variance, skewness) are called centeral parameters (cp), while (location, scale, shape) are called direct parameters (dp). When estimating model parameters, skewness (central parameter) is more stable than the shape parameter. Note that the skewness for a skewed normal is bounded.

Value

`skewness.cp(alpha)` gives the skewness corresponding to the shape alpha; `shape.dp(gamma)` gives the shape value corresponding to the skewness gamma.

`D.gamma(alpha)` gives the first and second derivative of skewness wrt. the shape alpha.

References

[1]. Azzalini, A. (1985). A class of distributions which includes the normal ones. Scandinavian journal of statistics, 171-178.

See Also

D.SN, cp2dp, dp2cp

Examples

```
gamma1 <- skewness.cp(10) # the skewness when the shape is 10
alpha1 <- shape.dp(gamma1) # the shape when the skewenss is gamma1; should be 10
ret <- D.gamma(10) # the derivatives of the skewness as a function of the shape
```

Simulation *Data with Skewed Marginal Distributions and Gaussian Copula (Simulated)*

Description

This simulated benchmark data follows the model with skewed normal distribution as the marginal distribution and Gaussian copula as the dependence structure. It can be used to demonstrate the usage of methods and validation. DST is the Data Simulated for Training purpose; DSV is the Data Simulated for Validation (prediction) purpose.

Usage

```
data(data.simulation) #load data sets: DST and DSV
DST  # data simulated for training
DSV  # data simulated for validation
```

Format

Each of DST and DSV is a list containing the following components:

- obs matrix of 100 observations (as row), with 80 timepoints for each
- tp vector of time points, with length 80
- cp vector of covariates for each subject, with length 100
- pars parameter list with mean, logvar (matrices of 100 by 80) and shape, skew (vectors of length 80)
- corr correlation matrix to determine the Gaussian corpula

In addition, each data set contains one specific component:

quantile 3-dimensional array with dimension c(5, 100, 80) for the true quantiles of the quantile levels c(.50, .80, .90, .95, .99); only available for the training data set DST

obs.full fully observed observation matrix with diemsnion c(100, 80), in the validation data set DSV;

obs in DSV contains missing values, and obs.full can be used to measure the performance of predication at those points where missing values are observed.

SSN *Standard Skewed Normal Parameterized using Skewness.*

Description

Calculate the density function, log density function, and derivatives of the standard skewed normal (SSN) distribution parameterized using skewness.

Usage

```
g(y, gamma, log = FALSE)
D.lg(y, gamma)
```

Arguments

y	function arigument, taking values in the real line
gamma	skewness parameter; should have the same dimension as y
log	logical; if TRUE, the log of g(y, gamma) is given

Details

Calculate the pdf (probability density function) and derivatives of standard skewed normal when parameterized by skewness.

Value

g(y, gamma) gives the pdf value at y and gamma as the skewness; g(y, gamma, log = TRUE) gives the log of the pdf value at y and gamma as the skewness; D.lg(y, gamma) gives the list of derivatives of the log pdf at y and gamma with the following components:

D1	1st derivative of log.g(y,gamma) wrt. y
D2	1st derivative of log.g(y,gamma) wrt. gamma
D12	2nd cross-partial derivative of log.g(y,gamma) wrt. y and gamma
D11	2nd derivative of log.g(y,gamma) wrt. y
D22	2nd derivative of log.g(y,gamma) wrt. gamma

See Also

D.SN, shape.dp, skewness.cp, D.gamma

Examples

```
# pdf of SSN
ret1 <- g(seq(-3, 3, length = 100), 0.9)
# plot the pdf
plot(seq(-3, 3, length = 100), ret1, type = "l",
     xlab = "x", ylab = "pdf", main = "Plot of Stardard Skewed Normal Density")
# derivatives of pdf
ret2 <- D.lg(10, 0.5)
# y and a are a vector
ret3 <- D.lg(rnorm(10), seq(0.1,0.5,length = 10))
# y and a are matrices
ret4 <- D.lg(matrix(rnorm(10), 2, 5), matrix(seq(0.1,0.5,length = 10), 2, 5))
```

`uni.fpca` *Functional Principle Component Analysis with Corpula*

Description

`uni.fpca` is used to do regular FPCA for univariate data with given covariance matrix. The sample covariance matrix will be used if covariance matrix is not given.

Usage

```
uni.fpca(Y, Kendall = NULL, Y.pred = NULL,
         nbasis.mean = 10, nbasis = 10, pve = 0.99, npc = NULL,
         center = TRUE, gam.method = "REML")
```

Arguments

`Y`	matrix for the fully observed curves; each row corresponds to one subject
`Kendall`	given covariance matrix for Y (raw matrix up to smoothing); typically obtained using Kendall's Tau approach. See 'Details'
`Y.pred`	partically observed curves to be predicted; default is `null` indicating no predication on new data set
`nbasis.mean`	number of basis functions to smooth the mean
`nbasis`	number of basis functions for each direction to smooth the covariance matrix
`pve`	threshold of the percentage of variance explained to select number of principal components
`npc`	prespecified value for the number of principal components to be included in the expansion (if given, this overrides 'pve').
`center`	if `TRUE`, we center the curves; otherwise, do not center the curves and assume the curves have mean 0 already
`gam.method`	estimation method when using `gam`

Details

This function is build in the spirit of the function `fpca.sc` in the package `refund`, but here the corpula plays a role. The covariance matrix `Kendall` is typically estimated before the application of this function, for instance, via Kendall's Tau approach. It is highly recommended that the Knedall's Tau covariance matris should be provided here since it is more stable.

Value

A list of the following components:

`Yhat`	matrix whose rows are the estimates through Functional FPCA of the curves in `Y`
`scores`	matrix of estimated principal component scores
`mu`	estimated mean function
`efunctions`	matrix of estimated eigenfunctions of the functional covariance operator, i.e., the FPC basis functions
`evalues`	estimated eigenvalues of the covariance operator, i.e., variances of FPC scores
`npc`	number of FPCs: either the supplied npc, or the minimum number of basis functions needed to explain proportion pve of the variance in the observed curves
`K.tilde`	Smoothed covariance matrix of `Y` after removing white noises
`K.hat`	Smoothed covariance matrix of `Y`
`sigma2`	estimated measurement error variance

References

[1]. Yao, F., Mueller, H.-G., and Wang, J.-L. (2005). Functional data analysis for sparse longitudinal data. Journal of the American Statistical Association, 100(470):577-590.

[2]. Goldsmith, J., Greven, S., and Crainiceanu, C. (2013). Corrected confidence bands for functional data using principal components. Biometrics, 69(1), 41-51.

Examples

```
data(data.simulation)
fpca <- uni.fpca(Y = DST$obs, Y.pred=DSV$obs)

# visualize the data and predicted surface
par(mfrow = c(1,2))
persp(DSV$cp, DSV$tp, DSV$obs, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="data surface (partically observed)")
persp(DSV$cp, DSV$tp, fpca$Yhat, theta=60, phi=15,
      ticktype = "detailed", col="lightblue",
      xlab = "covariate", ylab = "time",
      zlab="data", main="predicted surface via univariate FPCA")

# predication error
mean(((fpca$Yhat - DSV$obs.full)[!is.na(DSV$obs)])^2)
```

unmll *Negative loglikelihood function and the Gradient*

Description

Calculate the negative loglikelihood function as the objective function to be minimize in terms of coefficients β's. The mean and variance parameters are bivariate, while the skewness parameter is univariate. The gradient is calculated by a closed form.

Usage

```
case2.unmll.optim(beta, dataone, Basis.list, cate = 1)
case2.gr(beta, dataone, Basis.list, cate = 1)
```

Arguments

beta	smoothing coefficients as a vector
dataone	observation as a matrix n by m (n: number of subjects; m: number of timepoints)
Basis.list	a list of 3 components corresponding to smooth matrices for (mu, logvar, and skewness), typically generated by kpbb
cate	category of model to be considered; 1 for full model, 2 for the model when the skewness is fixed at 0 (no skewness)

Details

The coefficient beta is a vector by combining all coefficients for the mean, variance and skewness.

Value

case2.unmll.optim	negative loglikehood at beta when data = dataone and the Basis.list is used
case2.gr	gradient vector of case2.unmll.optim at beta when data = dataone and the Basis.list is used

Note

This is the negative log Marginal likelihood function of the data assuming independence.

References

[1]. Meng Li, Ana-Maria Staicu and Howard D. Bondell (2013), Incorporating Covariates in Skewed Functional Data Models. http://www.stat.ncsu.edu/information/library/papers/mimeo2654_Li.pdf.

See Also

kpbb

Examples

```
data(data.simulation)
y <- DST$obs
tp <- DST$tp
cp <- DST$cp
# generate basis
cases = c(2,2,1)   # bivariate for mean and variance; univariate for shape
nknots.tp = c(2,2,2) # 2 knots at time direction for each parameter
nknots.cp = c(2,2)   # 2 knots at covariate direction for mean and variance
basis.list <- lapply(1:3, function(k)
  kpbb(tp, cp, nknots.tp = nknots.tp[k],
       nknots.cp= nknots.cp[k], sub.case=cases[k]))
# obtain coefficients randomly
length.beta <- sum(sapply(basis.list, ncol))
beta <- runif(length.beta)
unmll <- case2.unmll.optim(beta, y, basis.list)
gradient <- case2.gr(beta, y, basis.list)
```

Index