Package 'KFPCA'

October 12, 2022

Title Kendall Functional Principal Component Analysis

Version 2.0

Description Implementation for Kendall functional principal component analysis. Kendall functional principal component analysis is a robust functional principal component analysis technique for non-Gaussian functional/longitudinal data. The crucial function of this package is KFPCA() and KFPCA_reg(). Moreover, least square estimates of functional principal component scores are also provided. Refer to Rou Zhong, Shishi Liu, Haocheng Li, Jingxiao Zhang. (2021) <arXiv:2102.01286>. Rou Zhong, Shishi Liu, Haocheng Li, Jingxiao Zhang. (2021) <doi:10.1016/j.jmva.2021.104864>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports kader, utils, pracma, fdapace, fda, stats, graphics

Depends R (>= 2.10)

NeedsCompilation no

Author Rou Zhong [aut, cre],
Jingxiao Zhang [aut]

Maintainer Rou Zhong <zhong_rou@163.com>

Repository CRAN

Date/Publication 2022-02-04 11:30:16 UTC

R topics documented:

`CD4` *CD4 cell counts*

Description

A dataset containing the logarithm of CD4 cell counts for 190 patients with AIDS from June 1997 to January 2002. The data come from a human immunodeficiency virus (HIV) study by Wohl et al. (2005) and can be obtained from Cao et al. (2015).

Usage

```
CD4
```

Format

A data frame with 741 rows and 3 variables:

PATIENT Patient ID.

CD4OBS Logarithm of CD4 cell counts.

CD4DATE Day of measurement.

References

David A. Wohl, Donglin Zeng, Paul Stewart, Nicolas Glomb, Timothy Alcorn, Suzanne Jones, Jean Handy, Susan Fiscus, Adriana Weinberg, Deepthiman Gowda, and Charles van der Horst (2005). "Cytomegalovirus viremia, mortality, and end-organ disease among patients with aids receiving potent antiretroviral therapies." Journal of Acquired Immune Deficiency Syndromes, 38(5):538-544.

Hongyuan Cao, Donglin Zeng, and Jason P. Fine (2015). "Regression analysis of sparse asynchronous longitudinal data." Journal of The Royal Statistical Society Series B-statistical Methodology, 77(4):755-776.

FPCscoreLSE *Least square estimates of functional principal component scores*

Description

Least square estimates (LSE) of functional principal component scores.

Usage

```
FPCscoreLSE(Lt, Ly, kern, bw, FPC_dis, RegGrid, more = FALSE)
```

Arguments

`Lt`	A `list` of n vectors, where n is the sample size. Each entry contains the observation time in ascending order for each subject.
`Ly`	A `list` of n vectors, where n is the sample size. Each entry contains the measurements of each subject at the observation time correspond to `Lt`.
`kern`	A `character` denoting the kernel type; 'epan'(Epanechnikov), 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).
`bw`	A scalar denoting the bandwidth for mean function estimate.
`FPC_dis`	A `nRegGrid` by `nK` `matrix` containing the eigenfunction estimates at `RegGrid`, where `nRegGrid` is the length of `RegGrid` and `nK` is the number of FPCs.
`RegGrid`	A `vector` of the equally spaced time points in the support interval.
`more`	Logical; If `FALSE`, only the estimates of FPC scores are returned. If `TRUE`, the mean function estimates and the eigenfunction estimates at all observation time points are also returned.

Value

If `more = FALSE`, a n by `nK` `matrix` containing the estimates of the FPC scores is returned, where n is the sample size. If `more = TRUE`, a `list` containing the following components is returned:

`score`	a n by `nK` `matrix` containing the estimates of the FPC scores.
`meanest_fine`	Mean function estimates at all observation time points.
`FPC_dis_fine`	Eigenfunction estimates at all observation time points.

Examples

```
# Generate data
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
```

```
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 3:5, regular = FALSE,
                     meanfun = function(x){0}, score = score,
                     eigfun = eigfun, sd = sqrt(0.1))
basis <- fda::create.bspline.basis(interval, nbasis = 13, norder = 4,
                              breaks = seq(0, 10, length.out = 11))
Klist <- KFPCA(DataNew$Lt, DataNew$Ly, interval, nK = 2, bw = 1,
               nRegGrid = 51, fdParobj = basis)
# Just an example to explain the use of FPCscoreLSE().
# One can obtain FPC scores estimates for KFPCA method
# by KFPCA() directly. Note that FPCscoreLSE() can also be used
# to estimate FPC scores for methods except KFPCA.
scoreKFPCA <- FPCscoreLSE(DataNew$Lt, DataNew$Ly, kern = "epan",
                          bw = Klist$bwmean, FPC_dis = Klist$FPC_dis,
                          RegGrid = seq(interval[1], interval[2], length.out = 51))
```

GenDataKL *Generate functional/longitudinal data via KL expansion*

Description

Generate functional/longitudinal data via Karhunen–Loève expansion.

Usage

```
GenDataKL(n, interval, sparse, regular, meanfun, score, eigfun, sd)
```

Arguments

n	number of sample size.
interval	A `vector` of length two denoting the supporting interval.
sparse	A `vector` denoting the possible numbers of observation size. The elements are chosen with equal chance. The length of `sparse` must be one if `regular = TRUE`.
regular	Logical; If `TRUE`, the observation grids are equally-spaced.
meanfun	A function for the mean.
score	A n by `nK` `matrix` containing the estimates of the FPC scores, where `nK` is the number of FPCs.
eigfun	A `list` containing the eigenfunctions.
sd	A scalar denoting the standard deviation of measurement errors.

Value

A `list` containing the following components:

Lt	A `list` of n vectors, where n is the sample size. Each entry contains the observation time in ascending order for each subject.
Ly	A `list` of n vectors, where n is the sample size. Each entry contains the measurements of each subject at the observation time correspond to `Lt`.

Examples

```
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 6:8, regular = FALSE,
                     meanfun = function(x){0}, score = score,
                     eigfun = eigfun, sd = sqrt(0.1))
```

GetGCVbw1D *Bandwidth selection through GCV for one-dimension cases*

Description

Bandwidth selection through generalized cross-validation (GCV) for one-dimension cases.

Usage

```
GetGCVbw1D(Lt, Ly, kern, dataType = "Sparse")
```

Arguments

`Lt`	A `list` of *n* vectors, where *n* is the sample size. Each entry contains the observation time in ascending order for each subject.
`Ly`	A `list` of *n* vectors, where *n* is the sample size. Each entry contains the measurements of each subject at the observation time correspond to `Lt`.
`kern`	A `character` denoting the kernel type; 'epan'(Epanechnikov), 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).
`dataType`	A `character` denoting the data type; 'Sparse'-default, 'Dense'.

Value

A scalar denoting the optimal bandwidth.

Examples

```
# Generate data
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
```

```
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 6:8, regular = FALSE,
                     meanfun = function(x){0}, score = score,
                     eigfun = eigfun, sd = sqrt(0.1))
# Optimal bandwidth for mean function estimate
bwOpt <- GetGCVbw1D(DataNew$Lt, DataNew$Ly, kern = "epan")
```

GetGCVbw2D *Bandwidth selection through GCV for two-dimension cases*

Description

Bandwidth selection through generalized cross-validation (GCV) for two-dimension cases.

Usage

```
GetGCVbw2D(tPairs, yin, Lt, kern, ObsGrid, RegGrid, dataType = "Sparse")
```

Arguments

`tPairs`	A `matrix` with two columns containing the pairs of time points.
`yin`	A `vector` denoting the corresponding values.
`Lt`	A `list` of *n* vectors, where *n* is the sample size. Each entry contains the observation time in ascending order for each subject.
`kern`	A `character` denoting the kernel type; 'epan'(Epanechnikov), 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).
`ObsGrid`	A `vector` containing all observation grids in ascending order.
`RegGrid`	A `vector` of the equally spaced time points in the support interval.
`dataType`	A `character` denoting the data type; 'Sparse'-default, 'Dense'.

Value

A scalar denoting the optimal bandwidth.

Examples

```
# Generate data
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 6:8, regular = FALSE,
                     meanfun = function(x){0}, score = score,
```

```
                     eigfun = eigfun, sd = sqrt(0.1))
# Optimal bandwidth for the estimate of
# E{X(s)X(t)} = cov(X(s), X(t)) + mu(s) * mu(t)
xin2D <- NULL
yin2D <- NULL
for(i in 1:n){
  xin2D <- rbind(xin2D, t(utils::combn(DataNew$Lt[[i]], 2)))
  yin2D <- rbind(yin2D, t(utils::combn(DataNew$Ly[[i]], 2)))
}
tPairs <- xin2D
yin <- yin2D[,1] * yin2D[, 2]
bwOpt <- GetGCVbw2D(tPairs = tPairs, yin = yin, Lt = DataNew$Lt,
                    kern = "epan", ObsGrid = sort(unique(unlist(DataNew$Lt))),
                    RegGrid = seq(interval[1], interval[2], length.out = 51))
```

`kernfun` *Kernel Functions*

Description

Some common-used kernel functions.

Usage

```
kernfun(type)
```

Arguments

type	A `character` denoting the kernel type; 'epan'(Epanechnikov), 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).

Value

The corresponding kernel function.

Examples

```
x <- seq(-2, 2, 0.01)
par(mfrow = c(2,2))
plot(x, kernfun("epan")(x), type = "l", main = "Epanechnikov")
plot(x, kernfun("unif")(x), type = "l", main = "Uniform")
plot(x, kernfun("quar")(x), type = "l", main = "Quartic")
plot(x, kernfun("gauss")(x), type = "l", main = "Gaussian")
par(mfrow = c(1,1))
```

KFPCA *Kendall Functional Principal Component Analysis (KFPCA) for sparse design*

Description

KFPCA for non-Gaussian functional data with sparse design or longitudinal data.

Usage

```
KFPCA(
  Lt,
  Ly,
  interval,
  dataType = "Sparse",
  nK,
  kern = "epan",
  bw,
  kernK = "epan",
  bwK = "GCV",
  kernmean = "epan",
  bwmean = "GCV",
  nRegGrid,
  fdParobj,
  more = TRUE
)
```

Arguments

`Lt`	A `list` of *n* vectors, where *n* is the sample size. Each entry contains the observation time in ascending order for each subject.
`Ly`	A `list` of *n* vectors, where *n* is the sample size. Each entry contains the measurements of each subject at the observation time correspond to `Lt`.
`interval`	A `vector` of length two denoting the supporting interval.
`dataType`	A `character` denoting the data type; 'Sparse'-default, 'Dense'.
`nK`	An integer denoting the number of FPCs.
`kern`	A `character` denoting the kernel type for the Nadaraya-Watson estimators; 'epan'(Epanechnikov)-default, 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).
`bw`	A scalar denoting the bandwidth for the Nadaraya-Watson estimators.
`kernK`	A `character` denoting the kernel type for the estimation of the Kendall's tau function; 'epan'(Epanechnikov)-default, 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).
`bwK`	The bandwidth for the estimation of the Kendall's tau function. If `is.numeric(bwK) == T`, `bwK` is exactly the bandwidth. If `bwK == "GCV"`, the bandwidth is chosen by GCV. (default: "GCV")

kernmean — A character denoting the kernel type for the estimation of the mean function; 'epan'(Epanechnikov)-default, 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).

bwmean — The bandwidth for the estimation of the mean function. If is.numeric(bwmean) == T, bwmean is exactly the bandwidth. If bwmean == "GCV", the bandwidth is chosen by GCV. (default: "GCV")

nRegGrid — An integer denoting the number of equally spaced time points in the supporting interval. The eigenfunctions and mean function are estimated at these equally spaced time points.

fdParobj — A functional parameter object for the smoothing of the eigenfunctions. For more detail, see smooth.basis.

more — Logical; If FALSE, estimates of FPC scores and predictions of trajectories are not returned.

Value

A list containing the following components:

ObsGrid — A vector containing all observation time points in ascending order.

RegGrid — A vector of the equally spaced time points in the support interval.

bwmean — A scalar denoting the bandwidth for the mean function estimate.

kernmean — A character denoting the kernel type for the estimation of the mean function

bwK — A scalar denoting the bandwidth for the Kendall's tau function estimate.

kernK — A character denoting the kernel type for the estimation of the Kendall's tau function

mean — A vector of length nRegGrid denoting the mean function estimate.

KendFun — A nRegGrid by nRegGrid matrix denoting the Kendall's tau function estimate.

FPC_dis — A nRegGrid by nK matrix containing the eigenfunction estimates at RegGrid.

FPC_smooth — A functional data object for the eigenfunction estimates.

score — A n by nK matrix containing the estimates of the FPC scores, where n is the sample size. The results are returned when more = TRUE.

X_fd — A functional data object for the prediction of trajectories. The results are returned when more = TRUE.

Xest_ind — A list containing the prediction of each trajectory at their own observation time points. The results are returned when more = TRUE.

Lt — The input 'Lt'.

Ly — The input 'Ly'.

CompTime — A scalar denoting the computation time.

References

Rou Zhong, Shishi Liu, Haocheng Li, Jingxiao Zhang (2021). "Robust Functional Principal Component Analysis for Non-Gaussian Longitudinal Data." Journal of Multivariate Analysis, https://doi.org/10.1016/j.jmva.2021.

Examples

```
# Generate data
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 6:8, regular = FALSE,
                     meanfun = function(x){0}, score = score,
                     eigfun = eigfun, sd = sqrt(0.1))
basis <- fda::create.bspline.basis(interval, nbasis = 13, norder = 4,
                              breaks = seq(0, 10, length.out = 11))
# KFPCA
Klist <- KFPCA(DataNew$Lt, DataNew$Ly, interval, nK = 2, bw = 1,
               nRegGrid = 51, fdParobj = basis)
plot(Klist$FPC_smooth)
```

`KFPCA_reg` *Kendall Functional Principal Component Analysis (KFPCA) for dense and regular design*

Description

KFPCA for non-Gaussian functional data with dense and regular design.

Usage

```
KFPCA_reg(Lt, Ly, nGrid, nK, fdParobj)
```

Arguments

`Lt`	A `list` of n vectors, where n is the sample size. Each entry contains the observation time in ascending order for each subject. The observation times are the same for each subject.
`Ly`	A `list` of n vectors, where n is the sample size. Each entry contains the measurements of each subject at the observation time correspond to `Lt`.
`nGrid`	An integer denoting the number of observation time for each subject.
`nK`	An integer denoting the number of FPCs.
`fdParobj`	A functional parameter object for the smoothing of mean function and eigenfunctions. For more detail, see `smooth.basis`.

Value

A `list` containing the following components:

- `meanfd` A functional data object for the mean function estimates.
- `FPC_list` A `list` containing `nK` functional data objects, which are the eigenfunction estimates.
- `score` A n by `nK` `matrix` containing the estimates of the FPC scores, where n is the sample size.
- `CompTime` A scalar denoting the computation time.

References

Rou Zhong, Shishi Liu, Haocheng Li, Jingxiao Zhang (2021). "Functional principal component analysis estimator for non-Gaussian data." <arXiv: https://arxiv.org/abs/2102.01286>.

Examples

```
# Generate data
n <- 100
interval <- c(0, 10)
lambda_1 <- 16 #the first eigenvalue
lambda_2 <- 9 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 51, regular = TRUE,
                     meanfun = function(x){0}, score = score,
                     eigfun = eigfun, sd = sqrt(0.25))
basis <- fda::create.bspline.basis(interval, nbasis = 13, norder = 4,
                              breaks = seq(0, 10, length.out = 11))
#KFPCA
Klist <- KFPCA_reg(DataNew$Lt, DataNew$Ly, nGrid = 51, nK = 2, fdParobj = basis)
plot(Klist$FPC_list[[1]])
plot(Klist$FPC_list[[2]])
```

`MeanEst` *Local linear estimates of mean function*

Description

Local linear estimates of mean function.

Usage

```
MeanEst(Lt, Ly, kern, bw, gridout)
```

Arguments

Lt	A `list` of n vectors, where n is the sample size. Each entry contains the observation time in ascending order for each subject.
Ly	A `list` of n vectors, where n is the sample size. Each entry contains the measurements of each subject at the observation time correspond to `Lt`.
kern	A `character` denoting the kernel type; 'epan'(Epanechnikov), 'unif'(Uniform), 'quar'(Quartic), 'gauss'(Gaussian).
bw	A scalar denoting the bandwidth.
gridout	A `vector` denoting the time points that the mean function need to be estimated.

Value

A `list` containing the following components:

Grid	A `vector` denoting the time points that the mean function need to be estimated.
mean	A `vector` containing the mean function estimates.

Examples

```
# Generate data
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 6:8, regular = FALSE,
                     meanfun = function(x){x}, score = score,
                     eigfun = eigfun, sd = sqrt(0.1))
# Mean function estimate at all observation time points
bwOpt <- GetGCVbw1D(DataNew$Lt, DataNew$Ly, kern = "epan")
meanest <- MeanEst(DataNew$Lt, DataNew$Ly, kern = "epan", bw = bwOpt,
                   gridout = sort(unique(unlist(DataNew$Lt))))
plot(meanest$Grid, meanest$mean)
```

---

predict.KFPCA   *Predict FPC scores*

Description

Predict FPC scores using least square estimate (LSE) for a new sample.

Usage

```
## S3 method for class 'KFPCA'
predict(object, newLt, newLy, nK, more = FALSE, ...)
```

Arguments

object	A KFPCA object obtained from KFPCA.
newLt	A list of n vectors, where n is the new sample size. Each entry contains the observation time in ascending order for each new subject.
newLy	A list of n vectors, where n is the new sample size. Each entry contains the measurements of each new subject at the observation time correspond to newLt.
nK	An integer denoting the number of FPCs.
more	Logical; If FALSE, only the predictions of FPC scores are returned. If TRUE, the mean function estimates and the eigenfunction estimates at the new observation time points are also returned.
...	Not used.

Value

If more = FALSE, a n by nK matrix containing the predictions of the FPC scores is returned, where n is the new sample size. If more = TRUE, a list containing the following components is returned:

score_new	a n by nK matrix containing the predictions of the FPC scores.
meanest_new	Mean function estimates at the new observation time points.
FPC_dis_new	Eigenfunction estimates at the new observation time points.

Examples

```
# Generate training data
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
DataNew <- GenDataKL(n, interval = interval, sparse = 6:8, regular = FALSE,
                     meanfun = function(x){0}, score = score,
                     eigfun = eigfun, sd = sqrt(0.1))
basis <- fda::create.bspline.basis(interval, nbasis = 13, norder = 4,
                              breaks = seq(0, 10, length.out = 11))
Klist <- KFPCA(DataNew$Lt, DataNew$Ly, interval, nK = 2, bw = 1,
               nRegGrid = 51, fdParobj = basis)
# Generate test data
n_test <- 20
score_test <- cbind(rnorm(n_test, 0, sqrt(lambda_1)),
                    rnorm(n_test, 0, sqrt(lambda_2)))
Data_test <- GenDataKL(n_test, interval = interval, sparse = 6:8, regular = FALSE,
                       meanfun = function(x){0}, score = score_test,
                       eigfun = eigfun, sd = sqrt(0.1))
# Prediction
score_pre <- predict(Klist, Data_test$Lt, Data_test$Ly, nK = 2)
plot(score_test[,1], score_pre[,1])
```

SparsePlot *Sparse plot*

Description

Create sparse plot to see the sparsity of the data.

Usage

```
SparsePlot(Lt, interval, ...)
```

Arguments

Lt	A list of n vectors, where n is the sample size. Each entry contains the observation time in ascending order for each subject.
interval	A vector of length two denoting the supporting interval.
...	Other arguments passed into plot.

Details

For the sparse plot, x-axis is the observation time while y-axis represents various subjects.

Value

Create the corresponding sparse plot.

Examples

```
# Generate data
n <- 100
interval <- c(0, 10)
lambda_1 <- 9 #the first eigenvalue
lambda_2 <- 1.5 #the second eigenvalue
eigfun <- list()
eigfun[[1]] <- function(x){cos(pi * x/10)/sqrt(5)}
eigfun[[2]] <- function(x){sin(pi * x/10)/sqrt(5)}
score <- cbind(rnorm(n, 0, sqrt(lambda_1)), rnorm(n, 0, sqrt(lambda_2)))
# DataNew1 and DataNew2 have different sparsity
DataNew1 <- GenDataKL(n, interval = interval, sparse = 6:8, regular = FALSE,
                      meanfun = function(x){0}, score = score,
                      eigfun = eigfun, sd = sqrt(0.1))
DataNew2 <- GenDataKL(n, interval = interval, sparse = 2:4, regular = FALSE,
                      meanfun = function(x){0}, score = score,
                      eigfun = eigfun, sd = sqrt(0.1))
# Create sparse plots
par(mfrow = c(1, 2))
SparsePlot(DataNew1$Lt, interval = interval)
SparsePlot(DataNew2$Lt, interval = interval)
par(mfrow = c(1, 1))
```

Index