Package ‘fdasrvf’

March 7, 2020

Type Package
Title Elastic Functional Data Analysis
Version 1.9.3
Date 2020-02-11
Author J. Derek Tucker <jdtuck@sandia.gov>
Maintainer J. Derek Tucker <jdtuck@sandia.gov>
License GPL-3
LazyData TRUE
SystemRequirements C++11
Imports Rcpp (>= 0.12.1), coda, foreach, mvtnorm, matrixcalc, splines, parallel, fields, doParallel, viridisLite, tolerance, testthat
Suggests akima, plot3D, plot3Drgl, rgl
LinkingTo Rcpp, RcppArmadillo
Depends R (>= 3.5.0),
RoxygenNote 7.0.2
Encoding UTF-8
NeedsCompilation yes
Repository CRAN
Date/Publication 2020-03-07 11:30:02 UTC

R topics documented:

align_fPCA .......................................................... 3
AmplitudeBoxplot .................................................. 5
R topics documented:

beta ................................................................. 6
bootTB ............................................................... 6
calc_shape_dist .................................................. 7
curve_geodesic .................................................... 8
curve_karcher_cov ............................................... 8
curve_karcher_mean .............................................. 9
curve_pair_align .................................................. 10
curve_principal_directions ..................................... 11
curve_srf_align .................................................... 12
curve_to_q .......................................................... 13
elastic.distance ................................................... 13
elastic.logistic .................................................... 14
elastic.lpcr.regression ......................................... 15
elastic.mllogistic ................................................ 16
elastic.mlpcr.regression ........................................ 18
elastic.pcr.regression ........................................... 19
elastic.prediction ............................................... 20
elastic.regression ............................................... 21
fdasrvf ............................................................. 22
function_group_warp_bayes .................................... 23
function_mean_bayes ............................................ 24
f_to_srf ............................................................ 26
gauss_model ....................................................... 26
gradient ........................................................... 27
growth_vel ........................................................ 28
horizFPCA ........................................................ 28
im ................................................................. 29
invertGamma ....................................................... 29
jointFPCA ........................................................ 30
joint_gauss_model ............................................... 31
kmeans_align ..................................................... 32
multiple_align_functions ...................................... 34
optimum.reparam ............................................... 35
outlier.detection ............................................... 36
pair_align_functions .......................................... 37
pair_align_functions_bayes ................................... 38
pair_align_functions_expomap ................................ 40
pair_align_image ................................................. 42
pcaTB ............................................................... 43
PhaseBoxplot ..................................................... 44
predict.lpcr ....................................................... 45
predict.mlpcr ..................................................... 45
predict.pcr ........................................................ 46
q_to_curve ........................................................ 47
reparam_curve .................................................... 48
reparam_image .................................................... 49
resamplecurve .................................................... 50
rgam ............................................................... 50
align_fPCA

Description

This function aligns a collection of functions while extracting principal components.

Usage

align_fPCA(
  f,
  time,
  num_comp = 3,
  showplot = T,
  smooth_data = FALSE,
  sparam = 25,
  parallel = FALSE,
  cores = 8,
  MaxItr = 51
)

Arguments

f             matrix \((N \times M)\) of \(M\) functions with \(N\) samples
time          vector of size \(N\) describing the sample points
num_comp      number of principal components to extract (default = 3)
showplot      shows plots of functions (default = T)
smooth_data   smooth data using box filter (default = F)
sparam        number of times to apply box filter (default = 25)
parallel enable parallel mode using `foreach` and `doParallel` package
cores set number of cores to use with `doParallel` (default = 2)
MaxItr maximum number of iterations

Value

Returns a list containing

- f0 original functions
- fn aligned functions - matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- qn aligned srvfs - similar structure to fn
- q0 original srvfs - similar structure to fn
- mqn srvf mean - vector of length \(N\)
- gam warping functions - vector of length \(N\)
- Dx cost function
- vf pca list containing
- q pca srvf principal directions
- f pca f principal directions
- latent latent values
- coef coefficients
- U eigenvectors

References


Examples

```r
## Not run:
data("simu_data")
out = align_fPCA(simu_data$f, simu_data$time)
## End(Not run)
```
AmplitudeBoxplot

Description
This function constructs the amplitude boxplot

Usage
AmplitudeBoxplot(warp_median, alpha = 0.05, ka = 1, showplot = TRUE)

Arguments
- `warp_median`: fdawarp object from time_warping of aligned data using the median
- `alpha`: quantile value (default=.05, i.e., 95%)
- `ka`: scalar for outlier cutoff (default=1)
- `showplot`: shows plots of functions (default = T)

Value
Returns a ampbox object containing
- `median_y`: median function
- `Q1`: First quartile
- `Q3`: Second quartile
- `Q1a`: First quantile based on alpha
- `Q3a`: Second quantile based on alpha
- `minn`: minimum extreme function
- `maxx`: maximum extreme function
- `outlier_index`: indexes of outlier functions
- `fmedian`: median function

References

Examples
data("simu_warp_median")
out <- AmplitudeBoxplot(simu_warp_median, showplot=FALSE)
**MPEG7 Curve Dataset**

**Description**

Contains the MPEG7 curve data set which is 20 curves in 65 classes. The array is structured with dimension (2,100,65,20)

**Usage**

```r
data("mpeg7")
```

**Format**

an array of shape (2,100,65,20)

---

**bootTB**

*Tolerance Bound Calculation using Bootstrap Sampling*

**Description**

This function computes tolerance bounds for functional data containing phase and amplitude variation using bootstrap sampling

**Usage**

```r
bootTB(f, time, a = 0.05, p = 0.99, B = 500, no = 5, parallel = T)
```

**Arguments**

- `f`: matrix of functions
- `time`: vector describing time sampling
- `a`: confidence level of tolerance bound (default = 0.05)
- `p`: coverage level of tolerance bound (default = 0.99)
- `B`: number of bootstrap samples (default = 500)
- `no`: number of principal components (default = 5)
- `parallel`: enable parallel processing (default = T)

**Value**

Returns a list containing

- `amp`: amplitude tolerance bounds
- `ph`: phase tolerance bounds
References


Examples

```r
## Not run:
data("simu_data")
out1 = bootTB(simu_data$f,simu_data$time)
## End(Not run)
```

calc_shape_dist(Elastic Shape Distance)

Description

Calculate elastic shape distance between two curves beta1 and beta2

Usage

calc_shape_dist(beta1, beta2, mode = "O")

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta1</td>
<td>array describing curve1 (n,T)</td>
</tr>
<tr>
<td>beta2</td>
<td>array describing curve</td>
</tr>
<tr>
<td>mode</td>
<td>Open (&quot;O&quot;) or Closed (&quot;C&quot;) curves</td>
</tr>
</tbody>
</table>

Value

d geodesic distance

References


Examples

data("mpeg7")
d = calc_shape_dist(beta[,1,1],beta[,1,4])
curve_geodesic  
*Form geodesic between two curves*

**Description**
Form geodesic between two curves using Elastic Method

**Usage**
```r
curve_geodesic(beta1, beta2, k = 5)
```

**Arguments**
- `beta1` array describing curve 1 (n,T)
- `beta2` array describing curve 2 (n,T)
- `k` number of curves along geodesic (default 5)

**Value**
a list containing
- `geod` curves along geodesic (n,T,k)
- `geod_q` srvf's along geodesic

**References**

**Examples**
```r
data("mpeg7")
out = curve_geodesic(beta[,1,1], beta[,1,5])
```

curve_karcher_cov  
*Curve Karcher Covariance*

**Description**
Calculate Karcher Covariance of a set of curves

**Usage**
```r
curve_karcher_cov(betamean, beta, mode = "O")
```
curve_karcher_mean

**Arguments**

- `betamean` array (n,T) of mean curve
- `beta` array (n,T,N) for N number of curves
- `mode` Open ("O") or Closed ("C") curves

**Value**

K covariance matrix

**References**


**Examples**

```r
data("mpeg7")
out = curve_srvf_align(beta[,,1:2],maxit=2) # note: use more shapes, small for speed
K = curve_karcher_cov(out$betamean, beta[,,1:2])
```

---

**curve_karcher_mean**  
*Karcher Mean of Curves*

**Description**

Calculates Karcher mean of a collection of curves using the elastic square-root velocity (srvf) framework.

**Usage**

curve_karcher_mean(beta, mode = "O", rotated = T, maxit = 20)

**Arguments**

- `beta` array (n,T,N) for N number of curves
- `mode` Open ("O") or Closed ("C") curves
- `rotated` Optimize over rotation (default = T)
- `maxit` maximum number of iterations

**Value**

Returns a list containing

- `mu` mean srvf
- `betamean` mean curve
- `v` shooting vectors
- `q` array of srvfs
References

Examples
data("mpeg7")
out = curve_karcher_mean(bet[,1,1:2],maxit=2) # note: use more shapes, small for speed

curve_pair_align  Pairwise align two curves

Description
This function aligns to curves using Elastic Framework

Usage
curve_pair_align(beta1, beta2)

Arguments
beta1 array describing curve 1 (n,T)
beta2 array describing curve 2 (n,T)

Value
a list containing
beta2n aligned curve 2 to 1
q2n aligned srvf 2 to 1
gam warping function
q1 srvf of curve 1

References

Examples
data("mpeg7")
out = curve_pair_align(bet[,1,1], bet[,1,5])
curve_principal_directions

Curve PCA

Description

Calculate principal directions of a set of curves

Usage

curve_principal_directions(betamean, mu, K, mode = "O", no = 3, N = 5)

Arguments

betamean array (n,T) of mean curve
mu array (n,T) of mean srvf
K array (2*T,2*T) covariance matrix
mode Open ("O") or Closed ("C") curves
no number of components
N number of samples on each side of mean

Value

pd list describing principal directions

References


Examples

data("mpeg7")
out = curve_srvf_align(beta[,1:1:2],maxit=2) # note: use more shapes, small for speed
K = curve_karcher_cov(out$betamean, beta[,1:1:2])
pd = curve_principal_directions(out$betamean, out$q_mu, K)
curve_srvf_align

Align Curves

Description
Aligns a collection of curves using the elastic square-root velocity (srvf) framework.

Usage
```
curve_srvf_align(beta, mode = "O", rotated = T, maxit = 20)
```

Arguments
- **beta**: array (n,T,N) for N number of curves
- **mode**: Open ("O") or Closed ("C") curves
- **rotated**: Optimize over rotation (default = T)
- **maxit**: maximum number of iterations

Value
Returns a list containing
- **betan**: aligned curves
- **qn**: aligned srvfs
- **betamean**: mean curve
- **q_mu**: mean SRVFs

References

Examples
```
data("mpeg7")
out = curve_srvf_align(beta[,1,1:2],maxit=2) # note: use more shapes, small for speed
K = curve_karcher_cov(out$betamean, beta[,1,1:2])
```
curve_to_q

Convert to SRVF space

Description
This function converts curves to SRVF

Usage
curve_to_q(beta)

Arguments
beta array describing curve (n,T)

Value
q array describing srvf

References

Examples
data("mpeg7")
q = curve_to_q(beta[,,1,1])

elastic.distance
Calculates two elastic distance

Description
This functions calculates the distances between functions, $D_y$ and $D_x$, where function 1 is aligned to function 2

Usage
elastic.distance(f1, f2, time, lambda = 0)

Arguments
f1 sample function 1
f2 sample function 2
time sample points of functions
lambda controls amount of warping (default = 0)
elastic.logistic

Value

Returns a list containing

\( \text{Dy} \) \hspace{1cm} \text{amplitude distance} \\
\( \text{Dx} \) \hspace{1cm} \text{phase distance} \\

References


Examples

```r
data("simu_data")
distances <- elastic.distance(simu_data$f[,1],simu_data$f[,2],simu_data$time)
```

---

**elastic.logistic**  
*Elastic Logistic Regression*

**Description**

This function identifies a logistic regression model with phase-variability using elastic methods.

**Usage**

```r
elastic.logistic(
  f, 
  y, 
  time, 
  B = NULL, 
  df = 20, 
  max_itr = 20, 
  smooth_data = FALSE, 
  sparm = 25, 
  parallel = FALSE, 
  cores = 2
)
```

**Arguments**

- **f** \hspace{1cm} matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- **y** \hspace{1cm} vector of size \(M\) labels \((-1/1)\)
- **time** \hspace{1cm} vector of size \(N\) describing the sample points
- **B** \hspace{1cm} matrix defining basis functions (default = NULL)
elastic.lpcr.regression

df scalar controlling degrees of freedom if B=NULL (default=20)
max_itr scalar number of iterations (default=20)
smooth_data smooth data using box filter (default = F)
sparam number of times to apply box filter (default = 25)
parallel enable parallel mode using foreach and doParallel package
cores set number of cores to use with doParallel (default = 2)

Value
Returns a list containing
alpha model intercept
beta regressor function
fn aligned functions - matrix (N x M) of M functions with N samples
qn aligned srvfs - similar structure to fn
gamma warping functions - similar structure to fn
q original srvf - similar structure to fn
B basis matrix
b basis coefficients
Loss logistic loss
type model type ('logistic')

References

---

elastic.lpcr.regression

*Elastic Logistic Principal Component Regression*

Description
This function identifies a logistic regression model with phase-variability using elastic pca

Usage
```
elastic.lpcr.regression(
  f,
  y,
  time,
  pca.method = "combined",
  no = 5,
  smooth_data = FALSE,
  sparam = 25
)
```
Arguments

- **f**: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- **y**: vector of size \(M\) labels
- **time**: vector of size \(N\) describing the sample points
- **pca.method**: string specifying pca method (options = "combined", "vert", or "horiz", default = "combined")
- **no**: scalar specifying number of principal components (default = 5)
- **smooth_data**: smooth data using box filter (default = F)
- **sparam**: number of times to apply box filter (default = 25)

Value

Returns a lpcr object containing

- **alpha**: model intercept
- **b**: regressor vector
- **y**: label vector
- **warp_data**: fdawarp object of aligned data
- **pca**: pca object of principal components
- **Loss**: logistic loss
- **pca.method**: string specifying pca method used

References


---

elastic.mlogistic  
{

**Elastic Multinomial Logistic Regression**

Description

This function identifies a multinomial logistic regression model with phase-variability using elastic methods

Usage

```r
elastic.mlogistic(
  f,
  y,
  time,
  B = NULL,
  df = 20,
)```
max_itr = 20,
smooth_data = FALSE,
sparam = 25,
parallel = FALSE,
cores = 2
)

Arguments

f matrix (N x M) of M functions with N samples
y vector of size M labels 1,2,...,m for m classes
time vector of size N describing the sample points
B matrix defining basis functions (default = NULL)
newf scalar controlling degrees of freedom if B=NULL (default=20)
max_itr scalar number of iterations (default=20)
smooth_data smooth data using box filter (default = F)
sparam number of times to apply box filter (default = 25)
parallel enable parallel mode using foreach and doParallel package
cores set number of cores to use with doParallel (default = 2)

Value

Returns a list containing

alpha model intercept
beta regressor function
fn aligned functions - matrix (N x M) of M functions with N samples
qn aligned srvfs - similar structure to fn
gamma warping functions - similar structure to fn
q original srvf - similar structure to fn
B basis matrix
b basis coefficients
Loss logistic loss
type model type (‘mlogistic’)

References

elastic.mlpcr.regression

Elastic Multinomial Logistic Principal Component Regression

Description

This function identifies a multinomial logistic regression model with phase-variability using elastic pca

Usage

```r
elastic.mlpcr.regression(
  f,
  y,
  time,
  pca.method = "combined",
  no = 5,
  smooth_data = FALSE,
  sparam = 25
)
```

Arguments

- `f`: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- `y`: vector of size \(M\) labels
- `time`: vector of size \(N\) describing the sample points
- `pca.method`: string specifying pca method (options = "combined", "vert", or "horiz", default = "combined")
- `no`: scalar specifying number of principal components (default=5)
- `smooth_data`: smooth data using box filter (default = F)
- `sparam`: number of times to apply box filter (default = 25)

Value

Returns a mlpcr object containing

- `alpha`: model intercept
- `b`: regressor vector
- `y`: label vector
- `Y`: Coded labels
- `warp_data`: fdawarp object of aligned data
- `pca`: pca object of principal components
- `Loss`: logistic loss
- `pca.method`: string specifying pca method used
Elastic Linear Principal Component Regression

Description

This function identifies a regression model with phase-variability using elastic pca.

Usage

```
elastic.pcr.regression(
  f,
  y,
  time,
  pca.method = "combined",
  no = 5,
  smooth_data = FALSE,
  sparam = 25,
  parallel = F,
  C = NULL
)
```

Arguments

- **f**: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- **y**: vector of size \(M\) responses
- **time**: vector of size \(N\) describing the sample points
- **pca.method**: string specifying pca method (options = "combined", "vert", or "horiz", default = "combined")
- **no**: scalar specify number of principal components (default=5)
- **smooth_data**: smooth data using box filter (default = F)
- **sparam**: number of times to apply box filter (default = 25)
- **parallel**: run in parallel (default = F)
- **C**: scale balance parameter for combined method (default = NULL)

References

elastic.prediction

Value

Returns a pcr object containing

- alpha: model intercept
- b: regressor vector
- y: response vector
- warp_data: fdawarp object of aligned data
- pca: pca object of principal components
- SSE: sum of squared errors
- pca.method: string specifying pca method used

References


description

This function performs prediction from an elastic regression model with phase-variability

Usage

elastic.prediction(f, time, model, y = NULL, smooth_data = FALSE, sparam = 25)

Arguments

- f: matrix (N x M) of M functions with N samples
- time: vector of size N describing the sample points
- model: list describing model from elastic regression methods
- y: responses of test matrix f (default=NULL)
- smooth_data: smooth data using box filter (default = F)
- sparam: number of times to apply box filter (default = 25)

Value

Returns a list containing

- y_pred: predicted values of f or probabilities depending on model
- SSE: sum of squared errors if linear
- y_labels: labels if logistic model
- PC: probability of classification if logistic
References


---

**elastic.regression**  
*Elastic Linear Regression*

**Description**

This function identifies a regression model with phase-variability using elastic methods.

**Usage**

```r
elastic.regression(
  f,
  y,
  time,
  B = NULL,
  lam = 0,
  df = 20,
  max_itr = 20,
  smooth_data = FALSE,
  sparam = 25,
  parallel = FALSE,
  cores = 2
)
```

**Arguments**

- `f`: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- `y`: vector of size \(M\) responses
- `time`: vector of size \(N\) describing the sample points
- `B`: matrix defining basis functions (default = NULL)
- `lam`: scalar regularization parameter (default=0)
- `df`: scalar controlling degrees of freedom if \(B=\)NULL (default=20)
- `max_itr`: scalar number of iterations (default=20)
- `smooth_data`: smooth data using box filter (default = F)
- `sparam`: number of times to apply box filter (default = 25)
- `parallel`: enable parallel mode using `foreach` and `doParallel` package
- `cores`: set number of cores to use with `doParallel` (default = 2)
Value

Returns a list containing

- \texttt{alpha} model intercept
- \texttt{beta} regressor function
- \texttt{fn} aligned functions - matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- \texttt{qn} aligned srvfs - similar structure to \texttt{fn}
- \texttt{gamma} warping functions - similar structure to \texttt{fn}
- \texttt{q} original srvf - similar structure to \texttt{fn}
- \texttt{B} basis matrix
- \texttt{b} basis coefficients
- \texttt{SSE} sum of squared errors
- \texttt{type} model type (‘linear’)

References


Description

A library for functional data analysis using the square root velocity framework which performs pair-wise and group-wise alignment as well as modeling using functional component analysis

References


**function_group_warp_bayes**

Bayesian Group Warping

**Description**

This function aligns a set of functions using Bayesian SRSF framework

**Usage**

```r
function_group_warp_bayes(
  f,
  time,
  iter = 50000,
  powera = 1,
  times = 5,
  tau = ceiling(times * 0.04),
  gp = seq(dim(f)[2]),
  showplot = TRUE
)
```
Arguments

- `f` matrix $(N \times M)$ of $M$ functions with $N$ samples
- `time` sample points of functions
- `iter` number of iterations (default = 150000)
- `powera` Dirchelet prior parameter (default 1)
- `times` factor of length of subsample points to look at (default = 5)
- `tau` standard deviation of Normal prior for increment (default ceil(times*.4))
- `gp` number of colors in plots (default seq(dim(f)[2]))
- `showplot` shows plots of functions (default = T)

Value

Returns a list containing

- `f0` original functions
- `f_q` $f$ aligned quotient space
- `gam_q` warping functions quotient space
- `f_a` $f$ aligned ambient space
- `gam_a` warping ambient space
- `qmn` mean srsf

References


Examples

```r
## Not run:
data("simu_data")
out = function_group_warp_bayes(simu_data$f, simu_data$time)
## End(Not run)
```

---

**function_mean_bayes**

Bayesian Karcher Mean Calculation

Description

This function calculates karcher mean of functions using Bayesian method

Usage

```r
function_mean_bayes(f, time, times = 5, group = 1:dim(f)[2], showplot = TRUE)
```
function_mean_bayes

Arguments

- **f**: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- **time**: sample points of functions
- **times**: factor of length of subsample points to look at (default = 5)
- **group**: (default \(1:dim(f)[2]\))
- **showplot**: shows plots of functions (default = T)

Value

Returns a list containing:

- **distfamily**: dist matrix
- **match.matrix**: matrix of warping functions
- **position**: position
- **mu_5**: function mean
- **rtmatrix**: rtmatrix
- **sumdist**: sumdist
- **qt.fitted**: aligned srsf functions
- **estimator**: estimator
- **estimator2**: estimator2
- **regfuncs**: registered functions

References


Examples

```r
## Not run:
data("simu_data")
out = function_mean_bayes(simu_data$f, simu_data$time)

## End(Not run)
```
f_to_srvf  
Convert to SRSF

Description
This function converts functions to srsf

Usage
f_to_srvf(f, time)

Arguments
f          matrix of functions
time       time

Value
q          matrix of SRSFs

References
Tucker, J. D., Wu, W., Srivastava, A., Generative Models for Function Data using Phase and Ampli-

Examples
data("simu_data")
q <- f_to_srvf(simu_data$f, simu_data$time)

gauss_model  
Gaussian model of functional data

Description
This function models the functional data using a Gaussian model extracted from the principal com-
ponents of the srvfs

Usage
gauss_model(warp_data, n = 1, sort_samples = FALSE)
Arguments

- **warp_data**: fdawarp object from `time_warping` of aligned data
- **n**: number of random samples (n = 1)
- **sort_samples**: sort samples (default = F)

Value

Returns a fdawarp object containing

- **fs**: random aligned samples
- **gams**: random warping function samples
- **ft**: random function samples

References


Examples

```r
data("simu_warp")
out1 = gauss_model(simu_warp,n = 10)
```

---

**gradient**

Gradient using finite differences

Description

This function takes the gradient of f using finite differences

Usage

`gradient(f, binsize)`

Arguments

- **f**: vector with N samples
- **binsize**: scalar of time samples

Value

- **g**: vector with N samples which is the gradient of f

Examples

```r
data("simu_data")
out = gradient(simu_data$f[,1],mean(diff(simu_data$time)))
```
growth_vel  

**Berkley Growth Velocity Dataset**

**Description**
Combination of both boys and girls growth velocity from the Berkley Dataset

**Usage**
```r
data("growth_vel")
```

**Format**
A list which contains f and time

---

horizFPCA  

**Horizontal Functional Principal Component Analysis**

**Description**
This function calculates vertical functional principal component analysis on aligned data

**Usage**
```r
horizFPCA(warp_data, no, showplot = TRUE)
```

**Arguments**
- `warp_data`: fdawarp object from time_warping of aligned data
- `no`: number of principal components to extract
- `showplot`: show plots of principal directions (default = T)

**Value**
Returns a hfpca object containing

- `gam_pca`: warping functions principal directions
- `psi_pca`: srvf principal directions
- `latent`: latent values
- `U`: eigenvectors
- `vec`: shooting vectors
- `mu`: Karcher Mean
References

Examples
```r
data("simu_warp")
hfPCA = horizFPCA(simu_warp, no = 3)
```

---

### im: Example Image Data set

**Description**
Contains two simulated images for registration

**Usage**
```r
data("image")
```

**Format**
a list containing two images of dimension (64,64)

---

### invertGamma: Invert Warping Function

**Description**
This function calculates the inverse of gamma

**Usage**
```r
invertGamma(gam)
```

**Arguments**
gam vector of $N$ samples

**Value**
Returns gamI inverted vector
References


Examples

data("simu_warp")
out = invertGamma(simu_warp$gam[,1])

```
jointFPCA

Description

This function calculates amplitude and phase joint functional principal component analysis on aligned data

Usage

jointFPCA(
  warp_data,
  no,
  id = round(length(warp_data$time)/2),
  C = NULL,
  showplot = T
)

Arguments

  warp_data  fdawarp object from time_warping of aligned data
  no         number of principal components to extract
  id         integration point for f0 (default = midpoint)
  C          balance value (default = NULL)
  showplot   show plots of principal directions (default = T)

Value

Returns a list containing

  q_pca       srvf principal directions
  f_pca       f principal directions
  latent      latent values
`joint_gauss_model` 31

- **coef**: coefficients
- **U**: eigenvectors
- **mu_psi**: mean psi function
- **mu_g**: mean g function
- **id**: point use for f(0)
- **C**: optimized phase amplitude ratio

**References**


**Examples**

```r
data("simu_warp")
data("simu_data")
jfpca = jointFPCA(simu_warp, no = 3)
```

**Description**

This function models the functional data using a Gaussian model extracted from the principal components of the srvfs using the joint model

**Usage**

```r
joint_gauss_model(warp_data, n = 1, no = 5)
```

**Arguments**

- **warp_data**: fdawarp object from `time_warping` of aligned data
- **n**: number of random samples (n = 1)
- **no**: number of principal components (n=4)

**Value**

Returns a fdawarp object containing

- **fs**: random aligned samples
- **gams**: random warping function samples
- **ft**: random function samples
- **qs**: random srvf samples
References


Examples

data("simu_warp")
out1 = joint_gauss_model(simu_warp,n = 10)

---

**kmeans_align**

*K-Means Clustering and Alignment*

**Description**

This function clusters functions and aligns using the elastic square-root slope (srsf) framework.

**Usage**

```r
kmeans_align(f, time, K, seeds = NULL, lambda = 0, showplot = TRUE, smooth_data = FALSE, sparam = 25, parallel = FALSE, alignment = TRUE, omethod = "DP", MaxItr = 50, thresh = 0.01)
```

**Arguments**

- `f`: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- `time`: vector of size \(N\) describing the sample points
- `K`: number of clusters
- `seeds`: indexes of cluster center functions (default = NULL)
- `lambda`: controls the elasticity (default = 0)
- `showplot`: shows plots of functions (default = T)
- `smooth_data`: smooth data using box filter (default = F)
kmeans_align

sparam  number of times to apply box filter (default = 25)
parallel enable parallel mode using foreach and doParallel package (default=F)
alignment whether to perform alignment (default = T)
omethod  optimization method (DP,DP2,RBFGS)
MaxItr  maximum number of iterations
thresh  cost function threshold

Value

Returns a fdakma object containing

f0  original functions
fn  aligned functions - matrix (N x M) of M functions with N samples which is a list for each cluster
qn  aligned SRSFs - similar structure to fn
q0  original SRSFs
labels  cluster labels
templates  cluster center functions
templates.q  cluster center SRSFs
gam  warping functions - similar structure to fn
qun  Cost Function Value

References


Examples

## Not run:
data("growth_vel")
out <- kmeans_align(growth_vel$f,growth_vel$time, K=2)

## End(Not run)
multiple_align_functions

*Group-wise function alignment to specified mean*

**Description**
This function aligns a collection of functions using the elastic square-root slope (srsf) framework.

**Usage**
```r
multiple_align_functions(
  f,
  time,
  mu,
  lambda = 0,
  showplot = TRUE,
  smooth_data = FALSE,
  sparam = 25,
  parallel = FALSE,
  omethod = "DP",
  MaxItr = 20,
  iter = 2000
)
```

**Arguments**
- `f`: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- `time`: vector of size \(N\) describing the sample points
- `mu`: vector of size \(N\) that \(f\) is aligned to
- `lambda`: controls the elasticity (default = 0)
- `showplot`: shows plots of functions (default = T)
- `smooth_data`: smooth data using box filter (default = F)
- `sparam`: number of times to apply box filter (default = 25)
- `parallel`: enable parallel mode using `foreach` and `doParallel` package (default=F)
- `omethod`: optimization method (DP,DP2,RBFGS,dBayes,expBayes)
- `MaxItr`: maximum number of iterations
- `iter`: bayesian number of mcmc samples (default 2000)

**Value**
Returns a `fdawarp` object containing
- `f0`: original functions
- `fn`: aligned functions - matrix \((N \times M)\) of \(M\) functions with \(N\) samples
Align two functions

This function aligns two SRSF functions using Dynamic Programming

Usage

```r
optimum.reparam(
  Q1,
  T1,
  Q2,
  T2,
  lambda = 0,
  method = "DP",
  w = 0.01,
  f1o = 0,
  f2o = 0
)
```

Arguments

- `Q1`: srsf of function 1
- `T1`: sample points of function 1
- `Q2`: srsf of function 2
- `T2`: sample points of function 2

References


### outlier.detection

**Description**

This function calculates outlier's using geodesic distances of the SRVF's from the median.

**Usage**

```
outlier.detection(q, time, mq, k = 1.5)
```

**Arguments**

- `q`: matrix ($N \times M$) of $M$ SRVF functions with $N$ samples
- `time`: vector of size $N$ describing the sample points
- `mq`: median calculated using `time_warping`
- `k`: cutoff threshold (default = 1.5)

**Value**

- `q_outlier`: outlier functions

---

### References


### Examples

```r
data("simu_data")
q = f_to_srvf(simu_data$f,simu_data$time)
gam = optimum.reparam(q[,1],simu_data$time,q[,2],simu_data$time)
```

---

### Outlier Detection

**Value**

- `gam`: warping function

---

### Values

- `lambda`: controls amount of warping (default = 0)
- `method`: controls which optimization method (default="DP") options are Dynamic Programming ("DP"), Coordinate Descent ("DP2"), and Riemannian BFGS ("RBFGS")
- `w`: controls LRBFGS (default = 0.01)
- `f1o`: initial value of f1, vector or scalar depending on q1, defaults to zero
- `f2o`: initial value of f2, vector or scalar depending on q1, defaults to zero
pair_align_functions

References


Examples

```r
data("toy_data")
data("toy_warp")
q_outlier = outlier.detection(toy_warp$q0, toy_data$time, toy_warp$mqn, k=1)
```


data("toy_data")
data("toy_warp")
q_outlier = outlier.detection(toy_warp$q0, toy_data$time, toy_warp$mqn, k=1)

pair_align_functions Align two functions

Description

This function aligns two functions using SRSF framework. It will align f2 to f1

Usage

```r
pair_align_functions(
  f1, f2, time,
  lambda = 0, method = "DP", w = 0.01, iter = 2000
)
```

Arguments

- **f1**: function 1
- **f2**: function 2
- **time**: sample points of functions
- **lambda**: controls amount of warping (default = 0)
- **method**: controls which optimization method (default="DP") options are Dynamic Programming ("DP"), Coordinate Descent ("DP2"), Riemannian BFGS ("RBFGS"), Simultaneous Alignment ("SIMUL"), Dirchelet Bayesian ("dBayes"), and Expon-Map Bayesian ("expBayes")
- **w**: controls LRBFGS (default = 0.01)
- **iter**: number of mcmc iterations for mcmc method (default 2000)
Value

Returns a list containing

- \( \tilde{f}_2 \) aligned \( f_2 \)
- \( \gamma \) warping function

References


Examples

```r
data("simu_data")
out = pair_align_functions(simu_data$f[,1],simu_data$f[,2],simu_data$time)
```

pair_align_functions_bayes

Align two functions

Description

This function aligns two functions using Bayesian SRSF framework. It will align \( f_2 \) to \( f_1 \)

Usage

```r
pair_align_functions_bayes(
  f1,
  f2,
  timet,
  iter = 15000,
  times = 5,
  tau = ceiling(times * 0.4),
  powera = 1,
  showplot = TRUE,
  extrainfo = FALSE
)
```
pair_align_functions_bayes

Arguments

- f1: function 1
- f2: function 2
- timet: sample points of functions
- iter: number of iterations (default = 15000)
- times: factor of length of subsample points to look at (default = 5)
- tau: standard deviation of Normal prior for increment (default ceil(times*.4))
- powera: Dirchelet prior parameter (default 1)
- showplot: shows plots of functions (default = T)
- extrainfo: T/F whether additional information is returned

Value

Returns a list containing

- f1: function 1
- f2_q: registered function using quotient space
- gam_q: warping function quotient space
- f2_a: registered function using ambient space
- q2_a: warping function ambient space
- match_collect: posterior samples from warping function (returned if extrainfo=TRUE)
- dist_collect: posterior samples from the distances (returned if extrainfo=TRUE)
- kappa_collect: posterior samples from kappa (returned if extrainfo=TRUE)
- log_collect: log-likelihood of each sample (returned if extrainfo=TRUE)
- pct_accept: vector of acceptance ratios for the warping function (returned if extrainfo=TRUE)

References


Examples

data("simu_data")
out = pair_align_functions_bayes(simu_data$f[,1], simu_data$f[,2], simu_data$time)
pair_align_functions_expomap

Align two functions using geometric properties of warping functions

Description

This function aligns two functions using Bayesian framework. It will align f2 to f1. It is based on mapping warping functions to a hypersphere, and a subsequent exponential mapping to a tangent space. In the tangent space, the Z-mixture pCN algorithm is used to explore both local and global structure in the posterior distribution.

Usage

```r
pair_align_functions_expomap(
  f1,
  f2,
  timet,
  iter = 20000,
  burnin = min(5000, iter/2),
  alpha0 = 0.1,
  beta0 = 0.1,
  zpcn = list(betas = c(0.5, 0.05, 0.005, 1e-04), probs = c(0.1, 0.1, 0.7, 0.1)),
  propvar = 1,
  init.coef = rep(0, 2 * 10),
  npoints = 200,
  extrainfo = FALSE
)
```

Arguments

- **f1**: observed data, numeric vector
- **f2**: observed data, numeric vector
- **timet**: sample points of functions
- **iter**: length of the chain
- **burnin**: number of burnin MCMC iterations
- **alpha0, beta0**: IG parameters for the prior of sigma1
- **zpcn**: list of mixture coefficients and prior probabilities for Z-mixture pCN algorithm of the form list(betas, probs), where betas and probs are numeric vectors of equal length
- **propvar**: variance of proposal distribution
- **init.coef**: initial coefficients of warping function in exponential map; length must be even
- **npoints**: number of sample points to use during alignment
- **extrainfo**: T/F whether additional information is returned
Details

The Z-mixture pCN algorithm uses a mixture distribution for the proposal distribution, controlled by input parameter zpcn. The zpcn$betas must be between 0 and 1, and are the coefficients of the mixture components, with larger coefficients corresponding to larger shifts in parameter space. The zpcn$probs give the probability of each shift size.

Value

Returns a list containing

- `f2_warped` f2 aligned to f1
- `gamma` Posterior mean gamma function
- `g.coef` matrix with iter columns, posterior draws of g.coef
- `psi` Posterior mean psi function
- `sigma1` numeric vector of length iter, posterior draws of sigma1
- `accept` Boolean acceptance for each sample (if extrainfo=TRUE)
- `betas.ind` Index of zpcn mixture component for each sample (if extrainfo=TRUE)
- `logl` numeric vector of length iter, posterior loglikelihood (if extrainfo=TRUE)
- `gamma_mat` Matrix of all posterior draws of gamma (if extrainfo=TRUE)
- `gamma_q025` Lower 0.025 quantile of gamma (if extrainfo=TRUE)
- `gamma_q975` Upper 0.975 quantile of gamma (if extrainfo=TRUE)
- `sigma_eff_size` Effective sample size of sigma (if extrainfo=TRUE)
- `psi_eff_size` Vector of effective sample sizes of psi (if extrainfo=TRUE)
- `xdist` Vector of posterior draws from xdist between registered functions (if extrainfo=TRUE)
- `ydist` Vector of posterior draws from ydist between registered functions (if extrainfo=TRUE)

References


Examples

```r
## Not run:
# This is a mcmc algorithm and takes a long time to run
data("simu_data")
myzpcn <- list(betas = c(0.1, 0.01, 0.005, 0.0001),
          probs = c(0.2, 0.2, 0.4, 0.2))
out = pair_align_functions_expomap(simu_data$f[,1], simu_data$f[,2],
        timet = simu_data$time, zpcn = myzpcn, extrainfo = TRUE)
# overall acceptance ratio
mean(out$accept)
# acceptance ratio by zpcn coefficient
with(out, tapply(accept, myzpcn$betas[betas.ind], mean))
## End(Not run)
```
**pair_align_image**

Pairwise align two images This function aligns to images using the q-map framework

**Description**

Pairwise align two images This function aligns to images using the q-map framework

**Usage**

```r
pair_align_image(
  I1,
  I2,
  M = 5,
  ortho = TRUE,
  basis_type = "t",
  resizei = FALSE,
  N = 64,
  stepsize = 1e-05,
  itermax = 1000
)
```

**Arguments**

- **I1**: reference image
- **I2**: image to warp
- **M**: number of basis elements (default=5)
- **ortho**: orthonormalize basis (default=TRUE)
- **basis_type**: ("t","s","i","o"; default="t")
- **resizei**: resize image (default=TRUE)
- **N**: size of resized image (default=64)
- **stepsize**: gradient stepsize (default=1e-5)
- **itermax**: maximum number of iterations (default=1000)

**Value**

Returns a list containing

- **Inew**: aligned I2
- **gam**: warping function

**References**

Examples

```r
# Not run:
# This is a gradient descent algorithm and takes a long time to run
data("image")
out <- pair_align_image(im$I1, im$I2)
# End(Not run)
```

### pcaTB

Tolerance Bound Calculation using Elastic Functional PCA

#### Description

This function computes tolerance bounds for functional data containing phase and amplitude variation using principal component analysis.

#### Usage

```r
pcaTB(f, time, m = 4, B = 1e+05, a = 0.05, p = 0.99)
```

#### Arguments

- `f`: matrix of functions
- `time`: vector describing time sampling
- `m`: number of principal components (default = 4)
- `B`: number of monte carlo iterations
- `a`: confidence level of tolerance bound (default = 0.05)
- `p`: coverage level of tolerance bound (default = 0.99)

#### Value

Returns a list containing

- `pca`: pca output
- `tol`: tolerance factor

#### References


PhaseBoxplot

Description

This function constructs the amplitude boxplot.

Usage

PhaseBoxplot(warp_median, alpha = 0.05, kp = 1, showplot = TRUE)

Arguments

- `warp_median`: fdawarp object from `time_warping` of aligned data using the median
- `alpha`: quantile value (default=.05, i.e., 95%)
- `kp`: scalar for outlier cutoff (default=1)
- `showplot`: shows plots of functions (default = T)

Value

Returns a phbox object containing

- `median_x`: median warping function
- `Q1`: First quartile
- `Q3`: Second quartile
- `Q1a`: First quantile based on alpha
- `Q3a`: Second quantile based on alpha
- `minn`: minimum extreme function
- `maxx`: maximum extreme function
- `outlier_index`: indexes of outlier functions

References


Examples

data("simu_warp_median")
out <- PhaseBoxplot(simu_warp_median, showplot=FALSE)
**predict.lpcr**  
*Elastic Prediction for functional logistic PCR Model*

**Description**
This function performs prediction from an elastic logistic fPCR regression model with phase-variability.

**Usage**
```r
## S3 method for class 'lpcr'
predict(object, newdata = NULL, y = NULL, ...)
```

**Arguments**
- `object`: Object of class inheriting from "elastic.pcr.regression"
- `newdata`: An optional matrix in which to look for variables with which to predict. If omitted, the fitted values are used.
- `y`: An optional vector of labels to calculate PC. If omitted, PC is NULL
- `...`: additional arguments affecting the predictions produced

**Value**
Returns a list containing
- `y_pred`: predicted probabilities of the class of newdata
- `y_labels`: class labels of newdata
- `PC`: probability of classification

**References**

---

**predict.mlpcr**  
*Elastic Prediction for functional multinomial logistic PCR Model*

**Description**
This function performs prediction from an elastic multinomial logistic fPCR regression model with phase-variability.

**Usage**
```r
## S3 method for class 'mlpcr'
predict(object, newdata = NULL, y = NULL, ...)
```

**Arguments**
- `object`: Object of class inheriting from "elastic.pcr.regression"
- `newdata`: An optional matrix in which to look for variables with which to predict. If omitted, the fitted values are used.
- `y`: An optional vector of labels to calculate PC. If omitted, PC is NULL
- `...`: additional arguments affecting the predictions produced

**Value**
Returns a list containing
- `y_pred`: predicted probabilities of the class of newdata
- `y_labels`: class labels of newdata
- `PC`: probability of classification

**References**
predict.pcr

**Arguments**

- **object**: Object of class inheriting from "elastic.pcr.regression"
- **newdata**: An optional matrix in which to look for variables with which to predict. If omitted, the fitted values are used.
- **y**: An optional vector of labels to calculate PC. If omitted, PC is NULL.
- **...**: additional arguments affecting the predictions produced

**Value**

Returns a list containing

- **y_pred**: predicted probabilities of the class of newdata
- **y_labels**: class labels of newdata
- **PC**: probability of classification per class
- **PC.comb**: total probability of classification

**References**


---

**predict.pcr**

*Elastic Prediction for functional PCR Model*

**Description**

This function performs prediction from an elastic pcr regression model with phase-variability.

**Usage**

```r
## S3 method for class 'pcr'
predict(object, newdata = NULL, y = NULL, ...)
```

**Arguments**

- **object**: Object of class inheriting from "elastic.pcr.regression"
- **newdata**: An optional matrix in which to look for variables with which to predict. If omitted, the fitted values are used.
- **y**: An optional vector of responses to calculate SSE. If omitted, SSE is NULL.
- **...**: additional arguments affecting the predictions produced

**Value**

Returns a list containing

- **y_pred**: predicted values of newdata
- **SSE**: sum of squared errors
q_to_curve

References


q_to_curve  Convert to curve space

Description

This function converts SRVF to curves

Usage

q_to_curve(q)

Arguments

q array describing SRVF (n,T)

Value

beta array describing curve

References


Examples

data(“mpeg7”)  
q = curve_to_q(beta[,1,1])  
beta1 = q_to_curve(q)
**reparam_curve**

Align two curves

**Description**

This function aligns two SRVF functions using Dynamic Programming

**Usage**

```r
reparam_curve(
  beta1,
  beta2,
  lambda = 0,
  method = "DP",
  w = 0.01,
  rotated = T,
  isclosed = F,
  mode = "O"
)
```

**Arguments**

- **beta1**: array defining curve 1
- **beta2**: array defining curve 1
- **lambda**: controls amount of warping (default = 0)
- **method**: controls which optimization method (default="DP") options are Dynamic Programming ("DP"), Coordinate Descent ("DP2"), Riemannian BFGS ("RBFGS")
- **w**: controls LRBFGS (default = 0.01)
- **rotated**: boolean if rotation is desired
- **isclosed**: boolean if curve is closed
- **mode**: Open ("O") or Closed ("C") curves

**Value**

return a List containing

- **gam**: warping function
- **R**: rotation matrix
- **tau**: seed point

**References**

**Examples**

```r
data("mpeg7")
gam = reparam_curve(beta[,1,1],beta[,1,5])$gam
```

**Description**

Finds the optimal warping function between two images using the elastic framework

**Usage**

```r
reparam_image(It, Im, gam, b, stepsize = 1e-05, itermax = 1000, lmark = FALSE)
```

**Arguments**

- `It`: template image matrix
- `Im`: test image matrix
- `gam`: initial warping array
- `b`: basis matrix
- `stepsize`: gradient stepsize (default=1e-5)
- `itermax`: maximum number of iterations (default=1000)
- `lmark`: use landmarks (default=FALSE)

**Value**

Returns a list containing

- `gamnew`: final warping
- `Inew`: aligned image
- `H`: energy
- `stepsize`: final stepsize

**References**

resamplecurve  

Description

This function resamples a curve to a number of points

Usage

resamplecurve(x, N = 100, mode = "O")

Arguments

x  
matrix defining curve (n,T)

N  
Number of samples to re-sample curve, N usually is > T

mode  
Open ("O") or Closed ("C") curves

Value

xn matrix defining resampled curve

References


Examples

data("mpeg7")

xn = resamplecurve(beta[,1,1],200)

rgam  

Description

Generates random warping functions

Usage

rgam(N, sigma, num)
Arguments

N  length of warping function
sigma  variance of warping functions
num  number of warping functions

Value

gam warping functions

References


Examples

gam = rgam(N=101, sigma=.01, num=35)

---

Sample shapes from model

Description

Sample shapes from model

Usage

sample_shapes(mu, K, mode = "O", no = 3, numSamp = 10)

Arguments

mu  array (n,T) of meansrvf
K  array (2*T,2*T) covariance matrix
mode  Open ("O") or Closed ("C") curves
no  number of principal components
numSamp  number of samples

Value

samples list of sample curves
References


Examples
data("mpeg7")
out = curve_srvf_align(beta[,1:1:2], maxit=2)  # note: use more shapes, small for speed
K = curve_karcher_cov(out$betamean, beta[,1:1:2])
samples = sample_shapes(out$q_mu, K)

Description

A functional dataset where the individual functions are given by: 
\[ y_i(t) = z_{i,1}e^{-(t-1.5)^2/2} + z_{i,2}e^{-(t+1.5)^2/2}, \quad t \in [-3,3], \quad i = 1, 2, \ldots, 21 \]
where \( z_{i,1} \) and \( z_{i,2} \) are i.i.d. normal with mean one and standard deviation 0.25. Each of these functions is then warped according to: 
\[ \gamma_i(t) = 6(e^{a_i(t+3)/a_i^2/6} - 1) - 3 \quad \text{if} \quad a_i \neq 0, \quad \text{otherwise} \quad \gamma_i = \gamma_{id} \]
\( (\gamma_{id}(t) = t) \) is the identity warping). The variables are as follows: \( f \) containing the 21 functions of 101 samples and \( \text{time} \) which describes the sampling.

Usage
data("simu_data")

Format

A list which contains \( f \) and \( \text{time} \) which has been aligned.

Description

A functional dataset where the individual functions are given by: 
\[ y_i(t) = z_{i,1}e^{-(t-1.5)^2/2} + z_{i,2}e^{-(t+1.5)^2/2}, \quad t \in [-3,3], \quad i = 1, 2, \ldots, 21 \]
where \( z_{i,1} \) and \( z_{i,2} \) are i.i.d. normal with mean one and standard deviation 0.25. Each of these functions is then warped according to: 
\[ \gamma_i(t) = 6(e^{a_i(t+3)/a_i^2/6} - 1) - 3 \quad \text{if} \quad a_i \neq 0, \quad \text{otherwise} \quad \gamma_i = \gamma_{id} \]
\( (\gamma_{id}(t) = t) \) is the identity warping). The variables are as follows: \( f \) containing the 21 functions of 101 samples and \( \text{time} \) which describes the sampling which has been aligned.
**Usage**

```r
data("simu_warp")
```

**Format**

A list which contains the outputs of the `time_warping` function

---

**simu_warp_median**

*Aligned Simulated two Gaussian Dataset using Median*

**Description**

A functional dataset where the individual functions are given by:

\[
y_i(t) = z_{i,1} e^{-\left(t-1.5\right)^2/2} + z_{i,2} e^{-\left(t+1.5\right)^2/2}, \quad t \in [-3,3], \quad i = 1, 2, \ldots, 21,
\]

where \(z_{i,1}\) and \(z_{i,2}\) are i.i.d. normal with mean one and standard deviation 0.25. Each of these functions is then warped according to:

\[
\gamma_i(t) = 6\left(e^{a_i(t+3)/6} - e^{a_i-1}\right) - 3 \quad \text{if} \quad a_i \neq 0,
\]

otherwise \(\gamma_i = \gamma_{id}\) (\(\gamma_{id}(t) = t\)) is the identity warping). The variables are as follows: `f` containing the 21 functions of 101 samples and `time` which describes the sampling which has been aligned.

**Usage**

```r
data("simu_warp_median")
```

**Format**

A list which contains the outputs of the `time_warping` function finding the median

---

**smooth.data**

*Smooth Functions*

**Description**

This function smooths functions using standard box filter

**Usage**

```r
smooth.data(f, sparam)
```

**Arguments**

- `f`: matrix \((N \times M)\) of \(M\) functions with \(N\) samples
- `sparam`: number of times to run box filter

**Value**

- `f0` smoothed functions
References


Tucker, J. D., Wu, W., Srivastava, A., Generative Models for Function Data using Phase and Amplitu-

Examples

```r
data("simu_data")
fo = smooth.data(simu_data$f,25)
```

---

**SqrtMean**

**SRVF transform of warping functions**

Description

This function calculates the srvf of warping functions with corresponding shooting vectors and finds
the mean

Usage

`SqrtMean(gam)`

Arguments

- `gam` matrix ($N \times M$) of $M$ warping functions with $N$ samples

Value

Returns a list containing

- `mu` Karcher mean psi function
- `gam_mu` Karcher mean warping function
- `psi` srvf of warping functions
- `vec` shooting vectors

References


Tucker, J. D., Wu, W., Srivastava, A., Generative Models for Function Data using Phase and Amplitu-

Examples

```r
data("simu_warp")
out = SqrtMean(simu_warp$gam)
```
SqrtMedian

SqrtMedian \( \text{SRVF transform of warping functions} \)

Description

This function calculates the srvf of warping functions with corresponding shooting vectors and finds the median.

Usage

SqrtMedian(gam)

Arguments

gam \( \text{matrix (} N \times M \text{) of } M \text{ warping functions with } N \text{ samples} \)

Value

Returns a list containing

- median \( \text{Karcher median psi function} \)
- gam_median \( \text{Karcher mean warping function} \)
- psi \( \text{srvf of warping functions} \)
- vec \( \text{shooting vectors} \)

References


Examples

data("simu_warp_median")
out = SqrtMedian(simu_warp_median$gam)
srsf_to_f

Convert SRSF to f

Description

This function converts SRSFs to functions

Usage

srsf_to_f(q, time, f0 = 0)

Arguments

q matrix of srsf
time time
f0 initial value of f

Value

f matrix of functions

References


Examples

data("simu_data")
q = f_to_srvf(simu_data$f,simu_data$time)
f = srsf_to_f(q,simu_data$time,simu_data$f[1,])

time_warping

Group-wise function alignment

Description

This function aligns a collection of functions using the elastic square-root slope (srsf) framework.
time_warping

Usage

time_warping(
    f,
    time,
    lambda = 0,
    method = "mean",
    showplot = TRUE,
    smooth_data = FALSE,
    sparam = 25,
    parallel = FALSE,
    omethod = "DP",
    MaxItr = 20
)

Arguments

f      matrix \((N \times M)\) of \(M\) functions with \(N\) samples
time   vector of size \(N\) describing the sample points
lambda controls the elasticity (default = 0)
method warp and calculate to Karcher Mean or Median (options = "mean" or "median", default = "mean")
showplot shows plots of functions (default = T)
smooth_data smooth data using box filter (default = F)
sparam number of times to apply box filter (default = 25)
parallel enable parallel mode using foreach and doParallel package (default=F)
omethod optimization method (DP,DP2,RBFGS)
MaxItr maximum number of iterations

Value

Returns a fdawarp object containing

\(f_0\) original functions
\(f_n\) aligned functions - matrix \((N \times M)\) of \(M\) functions with \(N\) samples
\(q_n\) aligned SRSFs - similar structure to \(f_n\)
\(q_0\) original SRSF - similar structure to \(f_n\)
\(fmean\) function mean or median - vector of length \(N\)
\(mqn\) SRSF mean or median - vector of length \(N\)
gam warping functions - similar structure to \(f_n\)
orig.var Original Variance of Functions
amp.var Amplitude Variance
phase.var Phase Variance
qun Cost Function Value
References


Examples

```r
## Not run:
data("simu_data")
out = time_warping(simu_data$f, simu_data$time)
## End(Not run)
```

### toy_data

**Distributed Gaussian Peak Dataset**

Description

A functional dataset where the individual functions are given by a Gaussian peak with locations along the $x$-axis. The variables are as follows: f containing the 29 functions of 101 samples and time which describes the sampling

Usage

data("toy_data")

Format

A list which contains f and time

### toy_warp

**Aligned Distributed Gaussian Peak Dataset**

Description

A functional dataset where the individual functions are given by a Gaussian peak with locations along the $x$-axis. The variables are as follows: f containing the 29 functions of 101 samples and time which describes the sampling which as been aligned

Usage

data("toy_warp")

Format

A list which contains the outputs of the time_warping function
vertFPCA

**Vertical Functional Principal Component Analysis**

**Description**

This function calculates vertical functional principal component analysis on aligned data.

**Usage**

```r
vertFPCA(warp_data, no, id = round(length(warp_data$time)/2), showplot = TRUE)
```

**Arguments**

- `warp_data`: fdawarp object from `time_warping` of aligned data.
- `no`: number of principal components to extract.
- `id`: point to use for `f(0)` (default = midpoint).
- `showplot`: show plots of principal directions (default = T).

**Value**

Returns a vfpca object containing:

- `q_pca`: srvf principal directions.
- `f_pca`: f principal directions.
- `latent`: latent values.
- `coef`: coefficients.
- `U`: eigenvectors.
- `id`: point used for `f(0)`.

**References**


**Examples**

```r
data("simu_warp")
vf pca = vertFPCA(simu_warp, no = 3)
```
warp_f_gamma  
**Warp Function**

**Description**

This function warps function $f$ by $\gamma$.

**Usage**

```r
warp_f_gamma(f, time, gamma, spl.int = FALSE)
```

**Arguments**

- `f`: vector function
- `time`: time
- `gamma`: vector warping function
- `spl.int`: use spline interpolation (default F)

**Value**

fnew warped function

**References**


**Examples**

```r
data("simu_data")
fnew = warp_f_gamma(simu_data$f[,1], simu_data$time, seq(0,1,length.out=101))
```

warp_q_gamma  
**Warp SRSF**

**Description**

This function warps srsf $q$ by $\gamma$.

**Usage**

```r
warp_q_gamma(q, time, gamma, spl.int = FALSE)
```
warp_q_gamma

Arguments

q vector

Arguments

q vector

time time

gamma vector warping function

spl.int use spline interpolation (default F)

Value

qnew warped function

References


Examples

data("simu_data")
q = f_to_srvf(simu_data$f,simu_data$time)
qnew = warp_q_gamma(q[,1],simu_data$time,seq(0,1,length.out=101))
Index

*Topic alignment
  AmplitudeBoxplot, 5
  calc_shape_dist, 7
  curve_geodesic, 8
  curve_karcher_cov, 8
  curve_karcher_mean, 9
  curve_pair_align, 10
  curve_principal_directions, 11
  curve_srvf_align, 12
  curve_to_q, 13
  elastic.logistic, 14
  elastic.lpcr.regression, 15
  elastic.mlogistic, 16
  elastic.mlpcr.regression, 18
  elastic.pcr.regression, 19
  elastic.prediction, 20
  elastic.regression, 21
  f_to_srvf, 26
  gradient, 27
  horizFPCA, 28
  invertGamma, 29
  jointFPCA, 30
  kmeans_align, 32
  multiple_align_functions, 34
  optimum.reparam, 35
  pair_align_functions, 37
  pair_align_image, 42
  PhaseBoxplot, 44
  predict.lpcr, 45
  predict.mlpcr, 45
  predict.pcr, 46
  q_to_curve, 47
  reparam_curve, 48
  reparam_image, 49
  resamplecurve, 50
  sample_shapes, 51
  smooth.data, 53
  SqrtMean, 54
  SqrtMedian, 55
  srsf_to_f, 56
  time_warping, 56
  vertFPCA, 59
  warp_f_gamma, 60
  warp_q_gamma, 60
*Topic bayesian
  function_group_warp_bayes, 23
  function_mean_bayes, 24
  pair_align_functions_bayes, 38
*Topic bootstrap
  bootTB, 6
*Topic boxplot
  AmplitudeBoxplot, 5
  PhaseBoxplot, 44
*Topic clustering
  kmeans_align, 32
*Topic datasets
  beta, 6
  growth_vel, 28
  im, 29
  simu_data, 52
  simu_warp, 52
  simu_warp_median, 53
  toy_data, 58
  toy_warp, 58
*Topic detection
  outlier.detection, 36
*Topic distances
  elastic.distance, 13
*Topic function
  rgam, 50
*Topic image
  pair_align_image, 42
  reparam_image, 49
*Topic outlier
  outlier.detection, 36
*Topic pca
  align_fPCA, 3
  gauss_model, 26
INDEX

Topic regression
  elastic.logistic, 14
  elastic.lpcr.regression, 15
  elastic.mlogistic, 16
  elastic.mlpcr.regression, 18
  elastic.pcr.regression, 19
  elastic.prediction, 20
  elastic.regression, 21
  predict.lpcr, 45
  predict.mlpcr, 45
  predict.pcr, 46

Topic srsf
  f_to_srvf, 26
  kmeans_align, 32
  multiple_align_functions, 34
  optimum.reparam, 35
  pair_align_functions, 37
  srsf_to_f, 56
  time_warping, 56

Topic srvf
  AmplitudeBoxplot, 5
  calc_shape_dist, 7
  curve_geodesic, 8
  curve_karcher_cov, 8
  curve_karcher_mean, 9
  curve_pair_align, 10
  curve_principal_directions, 11
  curve_srvf_align, 12
  curve_to_q, 13
  elastic.logistic, 14
  elastic.lpcr.regression, 15
  elastic.mlogistic, 16
  elastic.mlpcr.regression, 18
  elastic.pcr.regression, 19
  elastic.prediction, 20
  elastic.regression, 21
  gradient, 27
  horizFPCA, 28
  invertGamma, 29
  jointFPCA, 30
  outlier.detection, 36
  PhaseBoxplot, 44
  predict.lpcr, 45
  predict.mlpcr, 45
  predict.pcr, 46
  q_to_curve, 47
  reparam_curve, 48
  resamplecurve, 50
  sample_shapes, 51
  smooth.data, 53
  SqrtMean, 54
  SqrtMedian, 55
  vertFPCA, 59
  warp_f_gamma, 60
  warp_q_gamma, 60

Topic tolerance
  bootTB, 6

Topic warping
  rgam, 50
  align_FPCA, 3
  AmplitudeBoxplot, 5
  beta, 6
  bootTB, 6
  calc_shape_dist, 7
  curve_geodesic, 8
  curve_karcher_cov, 8
  curve_karcher_mean, 9
  curve_pair_align, 10
  curve_principal_directions, 11
  curve_srvf_align, 12
  curve_to_q, 13
  elastic.distance, 13
  elastic.logistic, 14
  elastic.lpcr.regression, 15
  elastic.mlogistic, 16
  elastic.mlpcr.regression, 18
  elastic.pcr.regression, 19
  elastic.prediction, 20
  elastic.regression, 21
  f_to_srvf, 26
  fdasrvf, 22
  fdasrvf-package (fdasrvf), 22
  foreach, 4, 15, 17, 21, 33, 34, 57
  function_group_warp_bayes, 23
  function_mean_bayes, 24
  gauss_model, 26
  gradient, 27
  growth_vel, 28
  horizFPCA, 28
im, 29
invertGamma, 29

joint_gauss_model, 31
jointFPCA, 30

kmeans_align, 32
multiple_align_functions, 34

optimum_reparam, 35
outlier_detection, 36

pair_align_functions, 37
pair_align_functions_bayes, 38
pair_align_functions_expomap, 40
pair_align_image, 42
pcaTB, 43
PhaseBoxplot, 44
predict.lpcr, 45
predict.mlpcr, 45
predict.pcr, 46
q_to_curve, 47

reparam_curve, 48
reparam_image, 49
resamplecurve, 50
rgam, 50

sample_shapes, 51
simu_data, 52
simu_warp, 52
simu_warp_median, 53
smooth.data, 53
SqrtMean, 54
SqrtMedian, 55
srsf_to_f, 56

time_warping, 5, 27, 28, 30, 31, 36, 44, 56, 59
toy_data, 58
toy_warp, 58

vertFPCA, 59

warp_f_gamma, 60
warp_q_gamma, 60