Package ‘conformalInference.fd’

March 2, 2022

Type Package

Title Tools for Conformal Inference for Regression in Multivariate Functional Setting

Version 1.1.0

Description It computes full conformal, split conformal and multi split conformal prediction regions when the response has functional nature. Moreover, the package also contain a plot function to visualize the output of the split conformal.

To guarantee consistency, the package structure mimics the univariate 'conformalInference' package of professor Ryan Tibshirani.

The main references for the code are:
Diquigiovanni, Fontana, and Vantini (2021) <arXiv:2102.06746>,
Diquigiovanni, Fontana, and Vantini (2021) <arXiv:2106.01792>,

URL https://github.com/ryantibs/conformal,
https://github.com/paolo-vergo/conformalInference.fd

License GPL-2

Depends R (>= 4.1.0)

Imports fda (>= 5.5.1), future (>= 1.23.0), future.apply (>= 1.8.1),
ggplot2 (>= 3.3.5), gridExtra (>= 2.3), stats, utils, methods

Suggests roahd, pbapply

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Jacopo Diquigiovanni [aut, ths],
Matteo Fontana [aut, ths],
Aldo Solari [aut, ths],
Simone Vantini [aut, ths],
Paolo Vergottini [aut, cre],
Ryan Tibshirani [ctb]

Maintainer Paolo Vergottini <paolo.vergottini@gmail.com>
computing_s_regression

Description

It computes modulation functions which allows local scaling of the prediction bands.

Usage

computing_s_regression(vec_residual, type, alpha, tau, grid_size)

Arguments

vec_residual A vector of the residuals obtained via functional modeling.
type A string indicating the type of modulation function chosen. The alternatives are "identity", "st-dev", "alpha-max".
alpha The value of the confidence interval.
tau A number between 0 and 1 used for the randomized version of the algorithm.
grid_size A vector containing the number of grid points in each dimension.

Details

More details can be found in the help of conformal.fun.split function.

Value

It returns a the values of a modulation function in each dimension of the response.
**concurrent**

*Concurrent Model for Functional Regression*

**Description**

It is a concurrent model, which may be fed to `conformal.fun.split`.

**Usage**

```r
concurrent()
```

**Details**

For more details about the structure of the inputs go to `split.R`.

**Value**

A training and a prediction function.

---

**conformal.fun.jackplus**

*Functional Jackknife + Prediction Regions*

**Description**

Compute prediction regions using functional Jackknife + inference.

**Usage**

```r
conformal.fun.jackplus(x, t_x, y, t_y, x0, train.fun, predict.fun, alpha = 0.1)
```

**Arguments**

- `x`: The input variable, a list of n elements. Each element is composed by a list of p vectors(with variable length, since the evaluation grid may change). If x is NULL, the function will sample it from a gaussian.
- `t_x`: The grid points for the evaluation of function x. It is a list of vectors. If the x data type is "fData" or "mfData" is must be NULL.
- `y`: The response variable. It is either, as with x, a list of list of vectors or an fda object (of type fd, fData, mfData).
- `t_y`: The grid points for the evaluation of function y_val. It is a list of vectors. If the y_val data type is "fData" or "mfData" is must be NULL.
- `x0`: The new points to evaluate, a list of n0 elements. Each element is composed by a list of p vectors(with variable length).
train.fun  A function to perform model training, i.e., to produce an estimator of $E(Y|X)$, the conditional expectation of the response variable $Y$ given features $X$. Its input arguments should be x: list of features, and y: list of responses.

predict.fun  A function to perform prediction for the (mean of the) responses at new feature values. Its input arguments should be out: output produced by train.fun, and newx: feature values at which we want to make predictions.

alpha  Miscoverage level for the prediction intervals, i.e., intervals with coverage $1-\alpha$ are formed. Default for alpha is 0.1.

Details

The work is an extension of the univariate approach to jackknife + inference to a multivariate functional context, exploiting the concept of depth measures.

This function is based on the package future.apply to perform parallelisation. If this package is not installed, then the function will abort.

Value

A list containing lo, up, tn. lo and up are lists of length $n_0$, containing lists of length $p$, with vectors of lower and upper bounds. tn is the list of the grid evaluations.

Examples

```r
### SIMULATED DATA

sample_size=50

oxout <- seq(from=0, to=1, length.out=301)

nb <- 6
w=c(1,1,1)

set.seed(-1)
coeff <- matrix(rnorm(nb*length(w)),nb,length(w))
bsbasis_obj <- fda::create.bspline.basis(rangeval=c(0,1), nbasis=nb, norder=4)
fd_obj <- fda::fd(coeff, bsbasis_obj)

beta <- t(matrix(w,length(w),1)%*%rep(1,length(xout)))*fda::eval.fd(xout, fd_obj)
beta_dim1=beta[,1:2]
beta_dim2=beta[,c(1,3)]

rm(coeff,bsbasis_obj,fd_obj,w,beta)

cov_dim1=matrix(1,sample_size+1,length(xout),2)
cov_dim2=matrix(rep(1:(sample_size+1)/(sample_size+1),length(xout)),
    sample_size+1,length(xout)) #x_{i2}=i/(n+1) per ogni i=1,...,n+1

cov_dim1[,1]=matrix(1,sample_size+1,length(xout)) #x_0: intercetta
cov_dim1[,2]=matrix(rep(1:(sample_size+1)/(sample_size+1),length(xout)),
    sample_size+1,length(xout)) #x_{i2}=i/(n+1) per ogni i=1,...,n+1

cov_dim2[,1]=matrix(1,sample_size+1,length(xout)) #x_0: intercetta
```
cov_dim2[,2]=matrix(rep((1:(sample_size+1))/(sample_size+1)*2, length(xout)),sample_size+1,length(xout))

deterministic_data_dim1=vapply(1:(sample_size+1),function(j) rowSums(beta_dim1*cov_dim1[,j,]),numeric(length(xout)))
deterministic_data_dim2=vapply(1:(sample_size+1),function(j) rowSums(beta_dim2*cov_dim2[,j,]),numeric(length(xout)))

w_error=rep(1,(sample_size+1)*2)
set.seed(1234)
coeff_error <- matrix(rnorm(nb*(sample_size+1)*2),nb,(sample_size+1)*2)
bsbasis_obj_error <- fda::create.bspline.basis(rangeval=c(0,1),
nbasis=nb, norder=4)
fd_obj_error <- fda::fd(coeff_error, bsbasis_obj_error)
error_data <- t(matrix(w_error,length(w_error),1))%*%rep(1,length(xout))%*%fda::eval.fd(xout, fd_obj_error)

rm(coeff_error,bsbasis_obj_error,fd_obj_error,w_error)

data_dim1=deterministic_data_dim1+error_data[,1:(sample_size+1)]
data_dim2=deterministic_data_dim2+error_data[(sample_size+2):(2*sample_size+2)]

data_dim1=data_dim1+error_data[,1:(sample_size+1)]
data_dim2=data_dim2+error_data[(sample_size+2):(2*sample_size+2)]

set.seed(1234)
obs_tbp=sample(1:(sample_size+1),1)
data_y=lapply(1:(sample_size+1),function(x) NULL)
for(i in 1:(sample_size+1)){
  data_y[[i]]=list(data_dim1[,i],data_dim2[,i])
}
new_data_y=list(data_y[[obs_tbp]])
data_y=data_y[-obs_tbp]
rm(data_dim1,data_dim2)

data_x=lapply(1:(sample_size+1),function(x) NULL)
for(i in 1:(sample_size+1)){
  data_x[[i]]=list(t(cov_dim1[,i,]),t(cov_dim2[,i,]))
}
new_x=list(data_x[[obs_tbp]])
data_x=data_x[-obs_tbp]
rm(obs_tbp,i,cov_dim1,cov_dim2)

n0=2
set.seed(1234)
### run once

true.jack = conformal.fun.jackplus (x=data_x[-id], t_x=NULL, y=data_y[-id], t_y=t, x0=data_x[id], fun$train.fun, fun$predict.fun, alpha=0.1)

coverage<-mean(sapply(1:n0, function(i){
  for(j in 1:p){
    coverage=rep(1,grid_len)
    for(k in 1:grid_len){
      coverage[k]<-prod(c(coverage[k],as.numeric(
        true.jack$lo[[i]][[j]][k]<y0[[i]][[j]][[k]] & y0[[i]][[j]][[k]] < true.jack$up[[i]][[j]][k])))
    }
  }
  return(sum(coverage)/grid_len)
})

coverage

---

**conformal.fun.msplit**  
*Functional Multi Split Conformal Prediction Regions*

**Description**

Compute prediction regions using functional multi split conformal inference.

**Usage**

conformal.fun.msplit(
  x, t_x, y, t_y, x0, train.fun, predict.fun, alpha = 0.1,
split = NULL,
seed = FALSE,
randomized = FALSE,
seed.rand = FALSE,
verbose = FALSE,
rho = NULL,
s.type = "alpha-max",
B = 100,
lambda = 0,
tau = 0.05
)

Arguments

x  
The input variable, a list of n elements. Each element is composed by a list of p vectors(with variable length, since the evaluation grid may change). If x is NULL, the function will sample it from a gaussian.

t_x  
The grid points for the evaluation of function x. It is a list of vectors. If the x data type is "fData" or "mfData", it must be NULL.

y  
The response variable. It is either, as with x, a list of list of vectors or an fda object (of type fd, fData, mfData).

t_y  
The grid points for the evaluation of function y_val. It is a list of vectors. If the y_val data type is "fData" or "mfData", it must be NULL.

x0  
The new points to evaluate, a list of n0 elements. Each element is composed by a list of p vectors(with variable length).

train.fun  
A function to perform model training, i.e., to produce an estimator of E(Y|X), the conditional expectation of the response variable Y given features X. Its input arguments should be x: list of features, and y: list of responses.

predict.fun  
A function to perform prediction for the (mean of the) responses at new feature values. Its input arguments should be out: output produced by train.fun, and newx: feature values at which we want to make predictions.

alpha  
Miscoverage level for the prediction intervals, i.e., intervals with coverage 1-\alpha are formed. Default for alpha is 0.1.

split  
Indices that define the data-split to be used (i.e., the indices define the first half of the data-split, on which the model is trained). Default is NULL, in which case the split is chosen randomly.

seed  
Integer to be passed to set.seed before defining the random data-split to be used. Default is FALSE, which effectively sets no seed. If both split and seed are passed, the former takes priority and the latter is ignored.

randomized  
Should the randomized approach be used? Default is FALSE.

seed.rand  
The seed for the randomized version of the conformal.split.fun. Default is FALSE.

verbose  
Should intermediate progress be printed out? Default is FALSE.

rho  
Vector containing the split proportion between training and calibration set. It has B components. Default is 0.5.
**s.type**  
The type of modulation function. Currently we have 3 options: "identity","std","alpha-max".

**B**  
Number of repetitions. Default is 100.

**lambda**  
Smoothing parameter. Default is 0.

**tau**  
It is a smoothing parameter: $\tau=1-1/B$ Bonferroni intersection method $\tau=0$ unadjusted intersection Default is 0.05, a value selected through sensitivity analysis.

**Details**

The work is an extension of the univariate approach to Multi Split conformal inference to a multivariate functional context, exploiting the concept of depth measures.

This function is based on the package future.apply to perform parallelisation. If this package is not installed, then the function will abort.

**Value**

A list containing lo, up, tn. lo and up are lists of length n0, containing lists of length p, with vectors of lower and upper bounds. tn is the list of the grid evaluations.

**References**

"Multi Split Conformal Prediction" by Solari, Djordjilovic (2021) is the baseline for the univariate case.

**Examples**

```r
### SIMULATED DATA
#
#---------------------- Genero i dati
library(fda)
sample_size=100

xout <- seq(from=0,to=1,length.out=301)

nb <- 6
w=c(1,1,1)

set.seed(-1)
coeff <- matrix(rnorm(nb*length(w)),nb,length(w))
bsbasis_obj <- create.bspline.basis(rangeval=c(0,1),
  nbasis=nb, norder=4)
fd_obj <- fd(coeff, bsbasis_obj)
beta <- t(matrix(w,length(w),1)%*%rep(1,length(xout)))*eval.fd(xout, fd_obj)
beta_dim1=beta[,1:2]
beta_dim2=beta[,c(1,3)]
rm(coeff,bsbasis_obj,fd_obj,w,beta)
```
cov_dim1 = array(NA, c(sample_size + 1, length(xout), 2))
cov_dim2 = array(NA, c(sample_size + 1, length(xout), 2))

cov_dim1[, , 1] = matrix(1, sample_size + 1, length(xout))
cov_dim1[, , 2] = matrix(rep(1:(sample_size + 1)/(sample_size + 1), length(xout)),
                          sample_size + 1, length(xout))
cov_dim2[, , 1] = matrix(1, sample_size + 1, length(xout))
cov_dim2[, , 2] = matrix(rep((1:(sample_size + 1)/(sample_size + 1))^2, length(xout)),
                          sample_size + 1, length(xout))

deterministic_data_dim1 = vapply(1:(sample_size + 1), function(j)
                                 rowSums(beta_dim1 * cov_dim1[, , j]), numeric(length(xout)))
deterministic_data_dim2 = vapply(1:(sample_size + 1), function(j)
                                 rowSums(beta_dim2 * cov_dim2[, , j]), numeric(length(xout)))

w_error = rep(1, (sample_size + 1) * 2)
set.seed(1234)
coeff_error <- matrix(rnorm(nb * (sample_size + 1) * 2), nb, (sample_size + 1) * 2)
bsbasis_obj_error <- create.bspline.basis(rangeval = c(0, 1),
                                         nbasis = nb, norder = 4)
fd_obj_error <- fd(coeff_error, bsbasis_obj_error)
error_data <- t(matrix(w_error, length(w_error), 1))
                          %*% rep(1, length(xout))) * eval.fd(xout, fd_obj_error)

rm(coeff_error, bsbasis_obj_error, fd_obj_error, w_error)

data_dim1 = deterministic_data_dim1 + error_data[, 1:(sample_size + 1)]
data_dim2 = deterministic_data_dim2 + error_data[, (sample_size + 2):(2 * sample_size + 2)]

rm(beta_dim1, beta_dim2, deterministic_data_dim1, deterministic_data_dim2, error_data, nb)

set.seed(1234)
obs_tbp = sample(1:(sample_size + 1), 1)
data_y = lapply(1:(sample_size + 1), function(x) NULL)
for(i in 1:(sample_size + 1)){
  data_y[[i]] = list(data_dim1[, i], data_dim2[, i])
}

new_data_y = list(data_y[[obs_tbp]])
data_y = data_y[-obs_tbp]

rm(data_dim1, data_dim2)
data_x = lapply(1:(sample_size + 1), function(x) NULL)
for(i in 1:(sample_size + 1)){
  data_x[[i]] = list(t(cov_dim1[i, , ]), t(cov_dim2[i, , ]))
}
new_x=list(data_x[[obs_tbp]])
data_x=data_x[-obs_tbp]
rm(obs_tbp,i,cov_dim1,cov_dim2)

n0=10
set.seed(1234)
len=length(data_x)
id=sample(1:len,n0)
n=len-n0
t=list(xout)
fun=mean_lists()
y0=data_y[id]
p<-length(y0[[1]])
grid_len<-length(y0[[1]][[1]])

### test multisplit depth
rrr<-conformal.fun.msplit(x=data_x[-id],t_x=NULL, y=data_y[-id],t_y=t, x0=data_x[id],
fun$train.fun, fun$predict.fun,alpha=0.5,
split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
verbose=FALSE, rho=NULL,B=100,lambda=0
)

mean(sapply(1:n0, function(i){
  for(j in 1:p){
    coverage=rep(1,grid_len)
    for(k in 1:grid_len){
      coverage[k]<-prod(c(coverage[k],as.numeric(rrr$lo[[i]][[j]][[k]]<y0[[i]][[j]][[k]] && y0[[i]][[j]][[k]] < rrr$up[[i]][[j]][[k]])
    }
  }
  return(sum(coverage)/grid_len)
})
})

---

conformal.fun.split  

**Functional Split Conformal Prediction Intervals**

**Description**

Compute prediction intervals using split conformal inference.
Usage

conformal.fun.split(
    x,
    t_x,
    y,
    t_y,
    x0,
    train.fun,
    predict.fun,
    alpha = 0.1,
    split = NULL,
    seed = FALSE,
    randomized = FALSE,
    seed.rand = FALSE,
    verbose = FALSE,
    rho = 0.5,
    s.type = "st-dev"
)

Arguments

x The input variable, a list of n elements. Each element is composed by a list of p vectors(with variable length, since the evaluation grid may change). If x is NULL, the function will sample it from a gaussian.

t_x The grid points for the evaluation of function x. It is a list of vectors. If the x data type is "fData" or "mfData" is must be NULL.

y The response variable. It is either, as with x, a list of list of vectors or an fda object (of type fd, fData, mfData).

t_y The grid points for the evaluation of function y_val. It is a list of vectors. If the y_val data type is "fData" or "mfData" is must be NULL.

x0 The new points to evaluate, a list of n0 elements. Each element is composed by a list of p vectors(with variable length).

train.fun A function to perform model training, i.e., to produce an estimator of E(Y|X), the conditional expectation of the response variable Y given features X. Its input arguments should be x: list of features, and y: list of responses.

predict.fun A function to perform prediction for the (mean of the) responses at new feature values. Its input arguments should be out: output produced by train.fun, and newx: feature values at which we want to make predictions.

alpha Miscoverage level for the prediction intervals, i.e., intervals with coverage 1-alpha are formed. Default for alpha is 0.1.

split Indices that define the data-split to be used (i.e., the indices define the first half of the data-split, on which the model is trained). Default is NULL, in which case the split is chosen randomly.

seed Integer to be passed to set.seed before defining the random data-split to be used. Default is FALSE, which effectively sets no seed. If both split and seed are passed, the former takes priority and the latter is ignored.
randomized  Should the randomized approach be used? Default is FALSE.
seed.rand   The seed for the randomized version. Default is FALSE.
verbose    Should intermediate progress be printed out? Default is FALSE.
rho        Split proportion between training and calibration set. Default is 0.5.
s.type      The type of modulation function. Currently we have 3 options: "identity","st-dev","alpha-max". Default is "std-dev".

Value
A list with the following components: t,pred,k,s.type,s,alpha,randomized,tau, extremes_are_included,average_width,product_integral, res, lo, up, t and s are lists of vectors, pred has the same interval structure of y_val, but the outside list is of length n0, k_s, average_width and product_integral are all positive floats, alpha and tau are positive floats less than 1, randomized and extremes_are_included are logical values, while s.type is a string. Finally lo and up are lists of length n0 of lists of length p, each containing a vector of lower and upper bounds respectively.

Examples
## fData #################################
N = 1e2
P = 1e3
grid = seq( 0, 1, length.out = P )
C = roahd::exp_cov_function( grid, alpha = 0.3, beta = 0.4 )
values = roahd::generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )
fD = roahd::fData( grid, values )
x0=list(as.list(grid))
fun=mean_lists()
final.fData = conformal.fun.split(x=NULL, t_x=NULL, y=fD, t_y=NULL, x0=x0, fun$train.fun, fun$predict.fun,
alpha=0.1, split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
verbose=TRUE, rho=0.5,s.type="alpha-max")
plot_fun(final.fData)

### mfData ###################################
N = 1e2
P = 1e3
t0 = 0
t1 = 1
grid = seq( t0, t1, length.out = P )
C = roahd::exp_cov_function( grid, alpha = 0.3, beta = 0.4 )
Data_1 = roahd::generate_gauss_fdata( N, centerline = sin( 2 * pi * grid ), Cov = C )
Data_2 = roahd::generate_gauss_fdata( N, centerline = log(1+ 2 * pi * grid ), Cov = C )
mfD=roahd::mfData( grid, list( Data_1, Data_2 ) )
x0=list(as.list(grid))
fun=mean_lists()
final.mfData = conformal.fun.split(NULL,NULL, mfD,NULL, x0, fun$train.fun, fun$predict.fun,
   alpha=0.1,
   split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
   verbose=TRUE, rho=0.5,s.type="alpha-max")

h=plot_fun(final.mfData)

### fd ############################################################

daybasis <- fda::create.fourier.basis(c(0, 365), nbasis=65)
tempfd <- fda::smooth.basis(fda::day.5, fda::CanadianWeather$dailyAv 
   [,"Temperature.C"],daybasis)$fd
Lbasis <- fda::create.constant.basis(c(0, 365))
Lcoef <- matrix(c(0,(2*pi/365)^2,0),1,3)
bfdojb <- fda::fd(Lcoef,Lbasis)
bwtlist <- fda::fd2list(bfdojb)
harmaccelLfd <- fda::Lfd(3, bwtlist)
Ltempmat <- fda::eval.fd(fda::day.5, tempfd, harmaccelLfd)
t=1:365
x0=list(as.list(grid))
fun=mean_lists()
final.fd = conformal.fun.split(NULL,NULL, tempfd,fda::day.5, x0, fun$train.fun, fun$predict.fun,
   alpha=0.1,
   split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
   verbose=TRUE, rho=0.5,s.type="alpha-max")

plot_fun(final.fd)

---

mean_lists  Mean of Functional Data

Description

This model, which averages functional data, is a fed to a Functional Conformal Prediction function.

Usage

mean_lists()

Details

For more details about the structure of the inputs go to the help of conformal.fun.split

Value

It outputs a training function and a prediction function.
Plot Functional Split Conformal Confidence Bands

Description

The function plots the confidence bands provided by the `conformal.fun.split` function, `conformal.fun.msplit` and `conformal.fun.jackplus`.

Usage

```r
plot_fun(out)
```

Arguments

- `out`: The output of the split/msplit/jackknife+ function.

Details

It exploits the package `ggplot` and `grid.arrange` to better visualize the results. It outputs n0=length(x0) plots.

It plots, for each value in x0, the predicted functional value and bands in all the dimensions of the multivariate functional response.

Value

None

Examples

```r
## fData
N = 1e2
P = 1e3
grid = seq( 0, 1, length.out = P )
C = roahd::exp_cov_function( grid, alpha = 0.3, beta = 0.4 )
values = roahd::generate_gauss_fdata( N,
  centerline = sin( 2 * pi * grid ),
  Cov = C )
fD = roahd::fData( grid, values )
x0=list(as.list(grid))
fun=mean_lists()
final.fData = conformal.fun.split(x=NULL,t_x=NULL, y=fD,
  t_y=NULL, x0=x0, fun$train.fun, fun$predict.fun,
  alpha=0.1,
  split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
  verbose=TRUE, rho=0.5,s.type="alpha-max")
plot_fun(final.fData)
```
### mfData #####################################################################

N = 1e2  
P = 1e3  
\( t_0 = 0 \)  
\( t_1 = 1 \)

\[
\text{grid} = \text{seq}( t_0, t_1, \text{length.out} = P )
\]

\[
\text{C} = \text{roahd::exp_cov_function}(\text{grid}, \alpha = 0.3, \beta = 0.4)
\]

\[
\text{Data}_1 = \text{roahd::generate_gauss_fdata}( \text{N}, \text{centerline} = \sin(2 \pi \text{grid}), \text{Cov} = \text{C})
\]

\[
\text{Data}_2 = \text{roahd::generate_gauss_fdata}( \text{N}, \text{centerline} = \log(1 + 2 \pi \text{grid}), \text{Cov} = \text{C})
\]

\[
\text{mfD} = \text{roahd::mfData}(\text{grid}, \text{list( Data}_1, \text{Data}_2 ))
\]

\[
x0 = \text{list(as.list(grid))}
\]

\[
\text{fun} = \text{mean_lists()}
\]

\[
\text{final.mfData} = \text{conformal.fun.split(NULL, NULL, mfD, NULL, x0, fun$train.fun, fun$predict.fun,}
\]

\[
\quad \text{alpha} = 0.1,
\]

\[
\quad \text{split} = \text{NULL, seed=FALSE, randomized=FALSE, seed.rand=FALSE,}
\]

\[
\quad \text{verbose=TRUE, rho=0.5, s.type="alpha-max"})
\]

\[
h = \text{plot_fun(final.mfData)}
\]

### fd ####################################################################

daybasis <- \text{fda::create.fourier.basis(c(0, 365), nbasis=65)}

tempfd <- \text{fda::smooth.basis(fda::day.5, fda::CanadianWeather$dailyAv[,"Temperature.C"], daybasis)$fd}

Lbasis <- \text{fda::create.constant.basis(c(0, 365))}

Lcoef <- \text{matrix(c(0,(2*pi/365)^2,0),1,3)}

bfdboj <- \text{fda::fd(Lcoef, Lbasis)}

bwtlist <- \text{fda::fd2list(bfdobj)}

harmaccelLfd <- \text{fda::Lfd(3, bwtlist)}

Ltempmat <- \text{fda::eval.fd(fda::day.5, tempfd, harmaccelLfd)}

t = 1:365

\[
x0 = \text{list(as.list(grid))}
\]

\[
\text{fun} = \text{mean_lists()}
\]

\[
\text{final.fd} = \text{conformal.fun.split(NULL, NULL, tempfd, fda::day.5, x0, fun$train.fun, fun$predict.fun,}
\]

\[
\quad \text{alpha} = 0.1,
\]

\[
\quad \text{split} = \text{NULL, seed=FALSE, randomized=FALSE, seed.rand=FALSE,}
\]

\[
\quad \text{verbose=TRUE, rho=0.5, s.type="alpha-max"})
\]

\[
h = \text{plot_fun(final.fd)}
\]
Index

computing_s_regression, 2
concurrent, 3
conformal.fun.jackplus, 3, 14
conformal.fun.msplit, 6, 14
conformal.fun.split, 2, 3, 10, 13, 14

ggplot, 14
grid.arrange, 14

mean_lists, 13

plot_fun, 14