Package ‘Compack’

May 11, 2020

Title Regression with Compositional Covariates

Version 0.1.0

Date 2020-04-21

Description Regression methodologies with compositional covariates, including (1) sparse log-contrast regression with compositional covariates proposed by Lin et al. (2014) <doi:10.1093/biomet/asu031>, and (2) sparse log-contrast regression with functional compositional predictors proposed by Sun et al. (2020) <arXiv:1808.02403>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Depends R (>= 3.4.0)

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, MASS, plyr, splines, fda, orthogonalsplinebasis, methods

Suggests knitr, rmarkdown

NeedsCompilation yes

VignetteBuilder knitr

Author Zhe Sun [aut, cre] (<https://orcid.org/0000-0001-8699-8235>), Kun Chen [aut] (<https://orcid.org/0000-0003-3579-5467>)

Maintainer Zhe Sun <zhe.sun@uconn.edu>

Repository CRAN

Date/Publication 2020-05-11 16:00:11 UTC

R topics documented:

`cglasso` *Fit a linearly constrained linear regression model with group lasso regularization.*

Description

Fit a linearly constrained regression model with group lasso regularization.

Usage

```
cglasso(y, Z, Zc = NULL, k, W = rep(1, times = p), intercept = TRUE,
        A =  kronecker(matrix(1, ncol = p), diag(k)), b = rep(0, times = k),
        u = 1, mu_ratio = 1.01,
        lam = NULL, nlam = 100,lambda.factor = ifelse(n < p1, 0.05, 0.001),
        dfmax = p, pfmax = min(dfmax * 1.5, p), tol = 1e-8,
        outer_maxiter = 1e+6, outer_eps = 1e-8,
        inner_maxiter = 1e+4, inner_eps = 1e-8)
```

Arguments

- `y` — respones vector with length n.
- `Z` — design matrix of dimension $n \times p1$.
- `Zc` — design matrix for unpenalized variables. Default value is NULL.
- `k` — the group size in Z. The number of groups is $p = p1/k$.
- `W` — a vector in length p (the total number of groups), or a matrix with dimension `p1*p1`. Default value is rep(1, times = p).
  - a vector of penalization weights for the groups of coefficients. A zero weight implies no shrinkage.
  - a diagonal matrix with positive diagonal elements.
- `intercept` — Boolean, specifying whether to include an intercept. Default is TRUE.
- `A, b` — linear equalities of the form $A\beta_{p1} = b$, where b is a vector with length k, and A is a $k \times p1$ matrix. Default values: b is a vector of 0's and `A = kronecker(matrix(1,ncol = p),diag(k))`.
- `u` — the inital value of the penalty parameter of the augmented Lagrange method adopted in the outer loop. Default value is 1.
- `mu_ratio` — the increasing ratio of the penalty parameter `u`. Default value is 1.01. Inital values for scaled Lagrange multipliers are set as 0's. If `mu_ratio` < 1, the program automatically set the initial penalty parameter `u` as 0 and `outer_maxiter` as 1, indicating that there is no linear constraint.
- `lam` — a user supplied lambda sequence. If `lam` is provided as a scaler and `nlam`> 1, `lam` sequence is created starting from `lam`. To run a single value of `lam`, set `nlam`$= 1$. The program will sort user-defined `lambda` sequence in decreasing order.
- `nlam` — the length of the `lam` sequence. Default is 100. No effect if `lam` is provided.
- `lambda.factor` — the factor for getting the minimal lambda in `lam` sequence, where `min(lam)` = `lambda.factor` * `max(lam)`. `max(lam)` is the smallest value of `lam` for which all penalized group are 0's. If $n >= p1$, the default is `0.001`. If $n < p1$, the default is `0.05`.
- `dfmax` — limit the maximum number of groups in the model. Useful for handling very large p, if a partial path is desired. Default is p.
- `pfmax` — limit the maximum number of groups ever to be nonzero. For example once a group enters the model along the path, no matter how many times it re-enters the model through the path, it will be counted only once. Default is `min(dfmax*1.5,p)`.
- `tol` — tolerance for coefficient to be considered as non-zero. Once the convergence criterion is satisfied, for each element β_j in coefficient vector β, $\beta_j = 0$ if $\beta_j < tol$.
- `outer_maxiter, outer_eps` — `outer_maxiter` is the maximun number of loops allowed for the augmented Lagrange method; and `outer_eps` is the corresponding convergence tolerance.
- `inner_maxiter, inner_eps` — `inner_maxiter` is the maximum number of loops allowed for blockwise-GMD; and `inner_eps` is the corresponding convergence tolerance.

Value

A list of

`beta`	a matrix of coefficients.
`lam`	the sequence of lambda values.
`df`	a vector, the number of nonzero groups in estimated coefficients for `Z` at each value of lambda.
`npass`	total number of iteration.
`error`	a vector of error flag.

`classo` *Fit a linearly constrained linear regression model with lasso regularization.*

Description

Fit a linearly constrained linear model with lasso regularization.

Usage

```
classo(y, Z, Zc = NULL, intercept = TRUE, pf = rep(1, times = p),
       lam = NULL, nlam = 100,lambda.factor = ifelse(n < p, 0.05, 0.001),
       dfmax = p, pfmax = min(dfmax * 1.5, p),
       u = 1, mu_ratio = 1.01, tol = 1e-10,
       outer_maxiter = 3e+08, outer_eps = 1e-8,
       inner_maxiter = 1e+6, inner_eps = 1e-8,
       A = rep(1, times = p), b = 0, beta.ini)
```

Arguments

`y`	a response vector with length n.
`Z`	a design matrix, with dimension $n \times p$.
`Zc`	design matrix for unpenalized variables. Default value is NULL.
`intercept`	Boolean, specifying whether to include an intercept. Default is TRUE.
`pf`	penalty factor, a vector of length p. Zero implies no shrinkage. Default value for each entry is 1.
`lam`	a user supplied lambda sequence. If `lam` is provided as a scaler and `nlam`> 1, `lam` sequence is created starting from `lam`. To run a single value of `lam`, set `nlam`$= 1$. The program will sort user-defined `lambda` sequence in decreasing order.
`nlam`	the length of the `lam` sequence. Default is 100. No effect if `lam` is provided.
`lambda.factor`	the factor for getting the minimal lambda in the `lam` sequence, where `min(lam)` = `lambda.factor` * `max(lam)`. `max(lam)` is the smallest value of `lam` for which all penalized coefficients become zero. If $n >= p$, the default is `0.001`. If $n < p$, the default is `0.05`.

dfmax	limit the maximum number of groups in the model. Useful for handling very large p, if a partial path is desired. Default is p.
pfmax	limit the maximum number of groups ever to be nonzero. For example once a group enters the model along the path, no matter how many times it re-enters the model through the path, it will be counted only once. Default is `min(dfmax*1.5,p)`.
u	the inital value of the penalty parameter of the augmented Lagrange method adopted in the outer loop. Default value is 1.
mu_ratio	the increasing ratio, with value at least 1, for `u`. Default value is 1.01. Inital values for scaled Lagrange multipliers are set as 0. If `mu_ratio` < 1, the program automatically set `u` as 0 and `outer_maxiter` as 1, indicating that there is no linear constraint.
tol	tolerance for the estimated coefficients to be considered as non-zero, i.e., if $abs(\beta_j)$ < `tol`, set β_j as 0. Default value is 1e-10.
outer_maxiter, outer_eps	`outer_maxiter` is the maximum number of loops allowed for the augmented Lanrange method; and `outer_eps` is the corresponding convergence tolerance.
inner_maxiter, inner_eps	`inner_maxiter` is the maximum number of loops allowed for the coordinate descent; and `inner_eps` is the corresponding convergence tolerance.
A, b	linear equalities of the form $A\beta_p = b$, where b is a scaler, and A is a row-vector of length `p`. Default values: b is 0 and `A = matrix(1,ncol = p)`.
beta.ini	inital value of the coefficients. Can be unspecified.

Value

A list of

beta	a matrix of coefficients.
lam	the sequence of lambda values.
df	a vector, the number of nonzero coefficients for `Z` at each value of lambda.
npass	total number of iteration.
error	a vector of error flag.

`coef.compCL` *extracts model estimated coefficients from a* `"compCL"` *object.*

Description

gets the coefficients at the requested values for `lam` from a fitted `"compCL"` object.

Usage

```
## S3 method for class 'compCL'
coef(object, s = NULL, ...)
```

Arguments

object	fitted "compCL" object.
s	value(s) of the penalty parameter lam at which coefficients are requested. Default (NULL) is the entire sequence used to create the model.
...	Not used.

Details

s is a vector of lambda values at which the coefficients are requested. If s is not in the lam sequence used for fitting the model, the coef function will use linear interpolation, so the function should be used with caution.

Value

The coefficients at the requested tuning parameter values in s.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

compCL and predict, plot and print methods for "compCL" object.

Examples

```
Comp_data = comp_Model(n = 50, p = 30)
Comp_fit = compCL(y = Comp_data$y, Z = Comp_data$X.comp, Zc = Comp_data$Zc,
                  intercept = Comp_data$intercept)
coef(Comp_fit)
coef(Comp_fit, s = Comp_fit$lam[50])
coef(Comp_fit, s = c(1, 0.5, 0.1))
```

---

coef.cv.compCL *Extract estimated coefficients from a "cv.compCL" object.*

Description

This function gets coefficients from a compCL object, using the stored "compCL.fit" object.

Usage

```
## S3 method for class 'cv.compCL'
coef(object, trim = FALSE, s = c("lam.min", "lam.1se"), ...)
```

Arguments

`object`	fitted "`cv.compCL`" object.
`trim`	whether to use the trimmed result. Default is FASLE.
`s`	value(s) of the penalty parameter `lam` at which coefficients are requested. • `s="lam.min"` (default) stored in the `cv.compCL` object, which gives value of `lam` that achieves the minimum cross-vadilation error. • `s="lambda.min"` which gives the largest value of `lam` such that 1 standard error above the minimum of the cross-validation errors is achieved. • If `s` is numeric, it is taken as the value(s) of `lam` to be used. • If `s = NULL`, the whole sequence of `lam` stored in the `cv.compCGL` object is used.
`...`	not used.

Details

`s` is a vector of lambda values at which the coefficients are requested. If `s` is not in the `lam` sequence used for fitting the model, the `coef` function will use linear interpolation, so the function should be used with caution.

Value

The coefficients at the requested tuning parameter values in `s`.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

`cv.compCL` and `compCL`, and `predict` and `plot` methods for "`cv.compCL`" object.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
```

```
cvm1 <- cv.compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                  Zc = Comp_data$Zc, intercept = Comp_data$intercept)
coef(cvm1)
coef(cvm1, s = NULL)
coef(cvm1, s = c(1, 0.5, 0.1))
```

---

coef.cv.FuncompCGL *Extract estiamted coefficients from a* "cv.FuncompCGL" *object.*

Description

This function gets the coefficients from a cross-validated FuncompCGL model, using the stored "FuncompCGL.fit" object, and the optimal grid values of the penalty parameter lam and the degrees of freedom k.

Usage

```
## S3 method for class 'cv.FuncompCGL'
coef(object, trim = FALSE, s = c("lam.min", "lam.1se"), k = NULL, ...)
```

Arguments

object fitted cv.FuncompCGL object.

trim logical; whether to use the trimmed result. Default is FALSE.

s value(s) of the penalty parameter lam at which coefficients are requested.

- s="lam.min"(default), grid value of lam and k stored in the "cv.FuncompCGL" object such that the minimum cross-validation error is achieved.
- s="lam.1se", grid value of lam and k stored on the "cv.FuncompCGL" object such that the 1 standard error above the miminum cross-validation error is achieved.
- If s is numeric, it is taken as the value(s) of lam to be used. In this case, k must be provided.
- If s = NULL, the whole sequence of lam stored in the cv.FuncompCGL object is used.

k value(s) of the degrees of freedom of the basis function at which coefficents are requested. k can be NULL (default) or integer(s).

- k = NULL, s must be either "lam.min" or "lam.1se".
- if k is an integer(s), it is taken as the value of k to be used and it must be one(s) of these in the "cv.FuncompCGL" object.

... not used.

Details

s is a vector of lambda values at which the coefficients are requested. If s is not in the lam sequence used for fitting the model, the coef function will use linear interpolation, so the function should be used with caution.

Value

The coefficients at the requested values of s and k. If k is a vector, a list of coefficient matrices is returned.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`cv.FuncompCGL` and `FuncompCGL`, and `predict` and `plot` methods for `"cv.FuncompCGL"` object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = 50, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0, rho_T = 0.6, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))

cv_m1 <-  cv.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                        Zc = Data$data$Zc, intercept = Data$data$intercept,
                        k = c(4,5), nfolds = 5, nlam = 50,
                        keep = TRUE)
coef(cv_m1)
coef(cv_m1, s = "lam.1se")
coef(cv_m1, s = c(0.5, 0.1, 0.05), k = c(4,5))
coef(cv_m1, s = NULL, k = c(4,5))
```

`coef.FuncompCGL` *Extract estimated coefficients from a* `"FuncompCGL"` *object.*

Description

get the coefficients at the requested values for `lam` from a fitted `FuncompCGL` object.

Usage

```
## S3 method for class 'FuncompCGL'
coef(object, s = NULL, ...)
```

Arguments

`object`	fitted `FuncompCGL` object.
`s`	value(s) of the penalty parameter `lam` at which coefficients are requested. Default (`NULL`) is the entire sequence used to create the model.
`...`	Not used.

Details

`s` is a vector of lambda values at which the coefficients are requested. If `s` is not in the `lam` sequence used for fitting the model, the `coef` function will use linear interpolation, so the function should be used with caution.

Value

The coefficients at the requested tuning parameter values in `s`.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`FuncompCGL`, and `predict`, `plot` and `print` methods for `"FuncompCGL"` object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = 50, p = p, m = 2, intercept = TRUE,
                    SNR = 2, sigma = 2, rho_X = 0, rho_T = 0.5, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))

m1 <- FuncompCGL(y = Data$data$y, X = Data$data$Comp , Zc = Data$data$Zc,
                 intercept = Data$data$intercept, k = df_beta)
coef(m1)
coef(m1, s = c(0.5, 0.1, 0.01))
```

---

coef.GIC.compCL — *Extracts estimated coefficients from a* "GIC.compCL" *object.*

Description

This function gets coefficients from a `compCL` object, using the stored `"compCL.fit"` object.

Usage

```
## S3 method for class 'GIC.compCL'
coef(object, s = "lam.min", ...)
```

Arguments

`object`	fitted "`GIC.compCL`" object.
`s`	value(s) of the penalty parameter `lam` at which coefficients are requested. • `s="lam.min"` (default) stored in the `GIC.compCL` object, which gives value of `lam` that achieves the minimum value of GIC. • If `s` is numeric, it is taken as the value(s) of `lam` to be used. • If `s = NULL`, the whole sequence of `lam` stored in the `GIC.compCGL` object is used.
`...`	not used.

Details

`s` is a vector of lambda values at which the coefficients are requested. If `s` is not in the `lam` sequence used for fitting the model, the `coef` function will use linear interpolation, so the function should be used with caution.

Value

The coefficients at the requested tuning parameter values in s.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

GIC.compCL and compCL, and predict, and plot methods for "GIC.compCL" object.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
GICm1 <- GIC.compCL(y = Comp_data$y, Z = Comp_data$X.comp, Zc = Comp_data$Zc,
                    intercept = Comp_data$intercept)
coef(GICm1)
coef(GICm1, s = c(1, 0.5, 0.1))
```

---

coef.GIC.FuncompCGL *Extract model estimated coefficients from a "GIC.FuncompCGL" object.*

Description

This function gets coefficients from a "GIC.FuncompCGL" object, using the stored "FuncompCGL.fit" object, and the optimal values of lam and k.

Usage

```
## S3 method for class 'GIC.FuncompCGL'
coef(object, s = "lam.min", k = NULL, ...)
```

Arguments

`object`	fitted `GIC.FuncompCGL` object.
`s`	value(s) of the regularization parameter `lam` at which coefficients are requested. • `s="lam.min"` (default), grid value of `lam` and `k` stored in `"GIC.FuncompCGL"` object such that the minimun value of GIC is achieved. • If `s` is numeric, it is taken as the value(s) of `lam` to be used. In this case, `k` must be provided. • If `s = NULL`, used the whole sequence of `lam` stored in the `GIC.FuncompCGL` object.
`k`	value(s) of degrees of freedom of the basis function at which coefficents are requested. `k` can be `NULL` (default) or integer(s). • `k = NULL`, `s` must be `"lam.min"`. • if `k` is integer(s), it is taken as the value of `k` to be used and it must be one(s) of these in `"GIC.FuncompCGL"` model.
`...`	not used.

Details

`s` is a vector of lambda values at which the coefficients are requested. If `s` is not in the `lam` sequence used for fitting the model, the `coef` function will use linear interpolation, so the function should be used with caution.

Value

The coefficients at the requested tuning parameter values in `s`.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`GIC.FuncompCGL` and `FuncompCGL`, and `predict` and `plot` methods for `"GIC.FuncompCGL"` object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
```

```
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
n = 50
k_list <- c(4,5)
Data <- Fcomp_Model(n = n, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0.6, rho_T = 0,
                    df_beta = df_beta, n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))

GIC_m1 <-  GIC.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                          Zc = Data$data$Zc, intercept = Data$data$intercept,
                          k = k_list)
coef(GIC_m1)
coef(GIC_m1, s = c(0.05, 0.01), k = c(4,5))
coef(GIC_m1, s = NULL, k = c(4,5))
```

---

compCL — *Fit regularization path for log-contrast model of compositional data with lasso penalty.*

Description

Fit regression with compositional predictors via penalized *log-contrast* model which was proposed by Lin et al. (2014) <doi:10.1093/biomet/asu031>. The model estimation is conducted by minimizing a linearly constrained lasso criterion. The regularization paths are computed at a grid of tuning parameter `lambda`.

Usage

```
compCL(y, Z, Zc = NULL, intercept = TRUE,
       lam = NULL, nlam = 100, lambda.factor = ifelse(n < p, 0.05, 0.001),
       pf = rep(1, times = p), dfmax = p, pfmax = min(dfmax * 1.5, p),
       u = 1, mu_ratio = 1.01, tol = 1e-10,
       inner_maxiter = 1e+4, inner_eps = 1e-6,
       outer_maxiter = 1e+08, outer_eps = 1e-8)
```

Arguments

`y`	a response vector with length n.
`Z`	a $n \times p$ design matrix of compositional data or categorical data. If `Z` is categorical data, i.e., row-sums of `Z` differ from 1, the program automatically transforms `Z` into compositional data by dividing each row by its sum. `Z` could NOT include entry of 0's.
`Zc`	a $n * p_c$ design matrix of control variables (not penalized). Default is `NULL`.
`intercept`	Boolean, specifying whether to include an intercept. Default is `FALSE`.

`lam` — a user supplied lambda sequence. If `lam` is provided as a scaler and `nlam`> 1, `lam` sequence is created starting from `lam`. To run a single value of `lam`, set `nlam`$= 1$. The program will sort user-defined `lambda` sequence in decreasing order.

`nlam` — the length of the `lam` sequence. Default is 100. No effect if `lam` is provided.

`lambda.factor` — the factor for getting the minimal lambda in the `lam` sequence, where `min(lam)` = `lambda.factor` * `max(lam)`. `max(lam)` is the smallest value of `lam` for which all penalized coefficients become zero. If $n >= p$, the default is `0.001`. If $n < p$, the default is `0.05`.

`pf` — penalty factor, a vector of length p. Zero implies no shrinkage. Default value for each entry is 1.

`dfmax` — limit the maximum number of groups in the model. Useful for handling very large p, if a partial path is desired. Default is p.

`pfmax` — limit the maximum number of groups ever to be nonzero. For example once a group enters the model along the path, no matter how many times it re-enters the model through the path, it will be counted only once. Default is `min(dfmax*1.5,p)`.

`u` — the inital value of the penalty parameter of the augmented Lagrange method adopted in the outer loop. Default value is 1.

`mu_ratio` — the increasing ratio, with value at least 1, for `u`. Default value is 1.01. Inital values for scaled Lagrange multipliers are set as 0's. If `mu_ratio` < 1, the program automatically set `u` as 0 and `outer_maxiter` as 1, indicating that there is no linear constraints included.

`tol` — tolerance for the estimated coefficients to be considered as non-zero, i.e., if $abs(\beta_j)$ < `tol`, set β_j as 0. Default value is 1e-10.

`inner_maxiter, inner_eps` — `inner_maxiter` is the maximun number of loops allowed in the coordinate descent; and `inner_eps` is the corresponding convergence tolerance.

`outer_maxiter, outer_eps` — `outer_maxiter` is the maximum number of loops allowed in the Augmented Lagrange method; and `outer_eps` is the corresponding convergence tolerance.

Details

The *log-contrast* regression model with compositional predictors is expressed as

$$y = Z\beta + e, s.t. \sum_{j=1}^{p} \beta_j = 0,$$

where Z is the n-by-p design matrix of log-transforemd compositional data, β is the p-vector of regression cofficients, and e is an n-vector of random errors. If zero(s) exists in the original compositional data, user should pre-process these zero(s).
To enable variable selection, we conduct model estimation via linearly constrained lasso

$$argmin_{\beta}(\frac{1}{2n}\|y - Z\beta\|_2^2 + \lambda\|\beta\|_1), s.t. \sum_{j=1}^{p} \beta_j = 0.$$

Value

An object with S3 calss "compCL" is a list containing:

- `beta` — a matrix of coefficients for $p + p_c + 1$ rows. If `intercept=FALSE`, then the last row of `beta` is set to 0's.
- `lam` — the sequence of `lam` values used.
- `df` — the number of non-zero β_p's in estimated coefficients for `Z` at each value of `lam`.
- `npass` — total iterations.
- `error` — error messages. If 0, no error occurs.
- `call` — the call that produces this object.
- `dim` — dimension of the coefficient matrix `beta`.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979

See Also

`coef`, `predict`, `print` and `plot` methods for "compCL" object and `cv.compCL` and `GIC.compCL`.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
m1 <- compCL(y = Comp_data$y, Z = Comp_data$X.comp,
             Zc = Comp_data$Zc, intercept = Comp_data$intercept)
print(m1)
plot(m1)
beta = coef(m1)
Test_data = comp_Model(n = 30, p = p, beta = Comp_data$beta, intercept = FALSE)
predmat = predict(m1, Znew = Test_data$X.comp, Zcnew = Test_data$Zc)
```

comp_Model *Simulation for log-contrast model.*

Description

Simulate data for log-contrast model with a single set of compositional data.

Usage

```
comp_Model(n, p, rho = 0.2, sigma = 0.5, gamma = 0.5, add.on = 1:5,
           beta = c(c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2), rep(0, times = p - 8)),
             beta0  = 1, intercept = TRUE)
```

Arguments

n	sample size
p	number of components in the compositional data
rho	parameter used to generate the $p \times p$ autocorrelation matrix for correlations among the components. Default is 0.2.
sigma	standard deviation for the noise terms, which are iid normal with mean 0. Default is 0.5.
gamma	a scaler. For the high level mean component(s), `log(p * gamma)` is added to the "non-normalized" data `w_i` before the data are converted to compositional.
add.on	an index vector with value(s) in `[1,p]`, specifying which component(s) of compositions is of high level mean. Default is `1:5`.
beta	coefficients for the compositional variables.
beta0	coefficient for the intercept. Default is 1.
intercept	whether to include an intercept. Default is `FALSE`.

Details

The setup of this simulation follows Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. Specifically, we first generate the correlation matrix among the components `X.Sigma` by `rho` with an autoregressive correlation structure. we then generate the "non-normalized" data w_i for each subject from multivariate normal distribution with covariance `X.Sigma` and mean determined by `add.on` and `gamma`. Each w_i is a vector of length `p`. Finally, the compositional covariates are obtained as

$$x_{ij} = exp(w_{ij}) / \sum_{k=1}^{p} exp(w_{ik}),$$

for each subject $i = 1, ..., n$ and component $j = 1, ..., p$.

Value

A list containing:

`y`	a n-vector of the simulated response
`X.comp`	a matrix of the simulated compositional predictors of dimension $n \times p$
`Z`	a matrix of the log-transformed compositional predictors
`Zc`	a matrix of the simulated covariates
`intercept`	whether an intercept is included
`beta`	the true coefficient vector

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

Examples

```
p = 30
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c( beta, rep(0, times = p - length(beta)) )
Data = comp_Model(n = 50, p = p, intercept = FALSE,
                  rho = 0.2, sigma = 0.5, gamma  = 0.5, add.on = 1:5,
                  beta = beta)
```

`cv.compCL` *Cross-validation for compCL.*

Description

k-fold cross-validation for compCL; produce a plot and return optimal values of `lam`.

Usage

```
cv.compCL(y, Z, Zc = NULL, intercept = FALSE, lam = NULL,
          nfolds = 10, foldid, trim = 0, keep = FALSE, ...)
```

Arguments

y	response vector with length n.
Z	z matrix as in compCL.
Zc	Zc matrix as in compCL. Default is NULL.
intercept	whether to include an intercept. Default is FALSE.
lam	a user supplied lambda sequence. If lam is provided as a scaler and nlam>1, lam sequence is created starting from lam. To run a single value of lam, set nlam$=1$. The program will sort user-defined lambda sequence in decreasing order.
nfolds	number of folds, default is 10. The smallest allowable value is nfolds=3.
foldid	an optional vector of values between 1 and the sample size n, providing the fold assignments. If supplied, nfold can be missing.
trim	percentage to be trimmed off the prediction errors from either side; default is 0.
keep	If keep=TRUE, fitted models in cross validation are reported. Default is keep=FALSE.
...	other arguments that can be passed to compCL.

Details

cross-validation and fit full data with selected model.

Value

an object of S3 class "cv.compCL" is returned, which is a list constaining:

compCL.fit	a fitted compCL object for the full data.
lam	the sequence of lam.
Ftrim	a list of cross-validation results without trimming: • cvm the mean cross-validated error - a vector of length length(lam). • cvsd standard error of cvm. • cvupper upper curve = cvm+cvsd. • cvlo lower curve = cvm-cvsd. • lam.min the optimal value of lam that gives minimum cross validation error. • lam.1se the largest value of lam such that the error is within 1 standard error of the minimum cvm.
Ttrim	a list of cross-validation result with trim*100%, The structure is the same as that for Ftrim.
foldid	the values of foldid.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979

See Also

`compCL` and `cv.compCL`, and `coef`, `predict` and `plot` methods for `"cv.compCL"` object.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
cvm1 <- cv.compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                  Zc = Comp_data$Zc, intercept = Comp_data$intercept)

plot(cvm1)
coef(cvm1)
## selection by "lam.min" criterion
which(abs(coef(cvm1, s = "lam.min")[1:p]) > 0)
## selection by "lam.1se" criterion
which(abs(coef(cvm1, s= "lam.1se")[1:p]) > 0)

Comp_data2 = comp_Model(n = 30, p = p, beta = Comp_data$beta, intercept = FALSE)
y_hat = predict(cvm1, Znew = Comp_data2$X.comp, Zcnew = Comp_data2$Zc)
plot(Comp_data2$y, y_hat,
     xlab = "Observed response", ylab = "Predicted response")
```

`cv.FuncompCGL` *Cross-validation for FuncompCGL.*

Description

k-fold cross-validation for FuncompCGL; produce a plot and return optimal values of `lam` and `k`.

Usage

```
cv.FuncompCGL(y, X, Zc = NULL, lam = NULL, nlam = 100, k = 4:10, ref = NULL,
              foldid, nfolds = 10, W = rep(1,times = p - length(ref)),
              trim = 0, outer_maxiter = 1e+06, keep = FALSE, ...)
```

Arguments

- `y` — response vector with length n.
- `X` — a data frame or matrix.
  - If `nrow(X)` > n, `X` should be a data frame or matrix of the functional compositional predictors with p columns for the values of the compositional components, one column indicating the subject ID and one column of observed time points. The order of the Subject ID should be the SAME as that of `y`.
  - If `nrow(X)[1]`=n, `X` is considered as the integrated design matrix, a `n*(k*p -length(ref))` matrix.
- `Zc` — a $n \times p_c$ design matrix of unpenalized variables. Default is NULL.
- `lam` — a user supplied lambda sequence. If `lam` is provided as a scaler and `nlam`> 1, `lam` sequence is created starting from `lam`. To run a single value of `lam`, set `nlam`$= 1$. The program will sort user-defined `lambda` sequence in decreasing order.
- `nlam` — the length of the `lam` sequence. Default is 100. No effect if `lam` is provided.
- `k` — a vector of integer values of the degrees of freedom; default is 4:10.
- `ref` — reference level (baseline), either an integer between $[1, p]$ or `NULL`. Default value is `NULL`.
  - If `ref` is set to be an integer between `[1,p]`, the group lasso penalized *log-contrast* model (with log-ratios) is fitted with the `ref`-th component chosed as baseline.
  - If `ref` is set to be `NULL`, the linearly constrained group lasso penalized *log-contrast* model is fitted.
- `foldid` — an optional vector of values between 1 and the sample size `n`, providing the fold assignments. If supplied, `nfold` can be missing.
- `nfolds` — number of folds, default is 10. The smallest allowable value is `nfolds=3`.
- `W` — a vector of length p (the total number of groups), or a matrix with dimension $p_1 \times p_1$, where `p1=(p -length(ref)) * k`, or character specifying the function used to calculate weight matrix for each group.
  - a vector of penalization weights for the groups of coefficients. A zero weight implies no shrinkage.
  - a diagonal matrix with positive diagonal elements.
  - if character string of function name or an object of type `function` to compute the weights.
- `trim` — percentage to be trimmed off the prediction errors from either side; default is 0.
- `outer_maxiter` — maximum number of loops allowed for the augmented Lagrange method.
- `keep` — If `keep=TRUE`, fitted models in cross validation are reported. Default is `keep=FALSE`.
- `...` — other arguments that can be passed to `FuncompCGL`.

Details

k-fold cross validation.

Value

An object of S3 class "cv.FuncompCGL" is return, which is a list containing:

FuncompCGL.fit
: a list of length length(k), with elements being the fitted FuncompCGL objects of different degrees of freedom.

lam
: the sequence of lam.

Ftrim
: a list for cross validation results with trim = 0.
  - cvm the mean cross-validated error - a matrix of dimension length(k)*length(lam).
  - cvsd estimated standard error of cvm.
  - cvup upper curve = cvm + cvsd.
  - cvlo lower curve = cvm -cvsd.
  - lam.min the optimal values of k and lam that give minimum cross validation error cvm.
  - lam.1se the optimal values of k and lam that give cross validation error withnin 1 standard error of the miminum cvm.

Ttrim
: a list of cross validation result with trim*100%. The structure is the same as that for Ftrim.

fit.preval, foldid
: fit.preval is the array of fitted models. Only kept when keep=TRUE.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

FuncompCGL and GIC.FuncompCGL, and predict, coef and plot methods for "cv.FuncompCGL" object.

Examples

```
## generate training and testing data
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)

n_train = 50
```

```
n_test = 30
nfolds = 5
foldid <- sample(rep(seq(nfolds), length = n_train))
k_list <- c(4,5)

Data <- Fcomp_Model(n = n_train, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0.2, rho_T = 0.5,
                    df_beta = df_beta, n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
arg_list <- as.list(Data$call)[-1]
arg_list$n <- n_test
Test <- do.call(Fcomp_Model, arg_list)

## cv_cgl: Constrained group lasso
cv_cgl <-  cv.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                         Zc = Data$data$Zc, intercept = Data$data$intercept,
                         k = k_list, foldid = foldid,
                         keep = TRUE)
plot(cv_cgl,k = k_list)
cv_cgl$Ftrim[c("lam.min", "lam.1se")]
beta <-  coef(cv_cgl, trim = FALSE, s = "lam.min")
k_opt <- cv_cgl$Ftrim$lam.min['df']
## plot path against L2-norm of group coefficients
plot(cv_cgl$FuncompCGL.fit[[as.character(k_opt)]])
## or plot path against L1-norm of group coefficients
plot(cv_cgl$FuncompCGL.fit[[as.character(k_opt)]], ylab = "L1")

m1 <- ifelse(is.null(ncol(Data$data$Zc)), 0, ncol(Data$data$Zc))
m1 <- m1 + Data$data$intercept
if(k_opt == df_beta) {
  plot(Data$beta, col = "red", pch = 19,
       ylim = range(c(range(Data$beta), range(beta))))
  abline(v= seq(from = 0, to = (p*df_beta), by = df_beta ))
  abline(h = 0)
  points(beta)
  if(m1 > 0) points(p*df_beta + 1:m1, tail(Data$beta, m1),
                    col = "blue", pch = 19)
} else {
  plot(beta, ylim = range(c(range(Data$beta), range(beta))) )
  abline(v= seq(from = 0, to = (p*k_opt), by = k_opt ))
  abline(h = 0, col = "red")
  if(m1 > 0) points(p*k_opt + 1:m1, tail(Data$beta, m1),
                    col = "blue", pch = 19)
}

beta_C <- matrix(beta[1:(p*k_opt)], byrow = TRUE, nrow = p)
## satisfies zero-sum constraints
cat("colSums:", colSums(beta_C))
Nonzero <- (1:p)[apply(beta_C, 1, function(x) max(abs(x)) >0)]
cat("selected groups:", Nonzero)

oldpar <- par(mfrow=c(2,1))
sseq <- Data$basis.info[, 1]
```

```
beta_curve_true <- Data$basis.info[, -1] %*%  t(beta_C_true)
Nonzero_true <- (1:p)[apply(beta_C_true, 1, function(x) max(abs(x)) >0)]
matplot(sseq, beta_curve_true, type = "l", ylim = range(beta_curve_true),
        ylab = "True coeffcients curves", xlab = "TIME")
abline(a = 0, b = 0, col = "grey", lwd = 2)
text(0, beta_curve_true[1, Nonzero_true], labels = Nonzero_true)

beta_curve <- splines::bs(sseq, df = k_opt, intercept = TRUE) %*% t(beta_C)
matplot(sseq, beta_curve, type = "l", ylim = range(beta_curve_true),
        ylab = "Estimated coefficient curves", xlab = "TIME")
abline(a = 0, b = 0, col = "grey", lwd = 2)
text(0, beta_curve[1, Nonzero], labels = Nonzero)
par(oldpar)

## plot L1-norm of the estimated coefficients for each component of the composition
plot(apply(abs(beta_C),1,sum), ylab = "L1-norm", xlab = "Component index")
## or plot L2-norm
plot(apply(abs(beta_C),1, function(x) sqrt(sum(x^2))),
     ylab = "L2-norm", xlab = "Component index")

## set a thresholding for variable selection via cross-validation model
## example 1: cut by average L2-norm for estimated coefficient curves
Curve_L2 <- colSums(beta_curve^2)
Curve_L2 <- Curve_L2 - colSums(beta_curve[c(1, nrow(beta_curve)), ]^2) / 2
Curve_L2 <- Curve_L2 * (Data$basis.info[2,1] - Data$basis.info[1,1])
Curve_L2 <- sqrt(Curve_L2)
plot(Curve_L2, xlab = "Component index", ylab = "L2-norm for coefficient curves")
cutoff <- sum(Curve_L2) / p
Nonzero_cut <- (1:p)[which(Curve_L2 >= cutoff)]
Nonzero_cut
## example 2: cut by average L2-norm for estimated coefficient vectors
cutoff <- sum(apply(beta_C, 1, function(x) norm(x, "2")))/p
Nonzero_cut2 <- (1:p)[apply(beta_C, 1, function(x, a) norm(x, "2") >= a, a = cutoff)]
## example 3: cut by average L1-norm for estimated coefficient vectors
cutoff <- sum(abs(beta_C))/p
Nonzero_cut3 <- (1:p)[apply(beta_C, 1, function(x, a) sum(abs(x)) >= a, a = cutoff)]

y_hat <- predict(cv_cgl, Data$data$Comp, Data$data$Zc, s = "lam.min")
MSE <- sum((drop(Data$data$y) - y_hat)^2) / n_train
y_hat <- predict(cv_cgl, Test$data$Comp, Test$data$Zc, s = "lam.min")
PRE <- sum((drop(Test$data$y) - y_hat)^2) / n_test
cgl_result <- list(cv.result = cv_cgl, beta = beta,
                   Nonzero = c("Original" = Nonzero, "Cut" = Nonzero_cut),
                   MSE = MSE, PRE = PRE)

## cv_naive: ignoring the zero-sum constraints
## set mu_raio = 0 to identifying without linear constraints,
## no outer_loop for Lagrange augmented multiplier
cv_naive <-  cv.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                           Zc = Data$data$Zc, intercept = Data$data$intercept,
                           k = k_list, foldid = foldid, keep = TRUE,
                           mu_ratio = 0)
plot(cv_naive, k = k_list)
```

```
beta <-  coef(cv_naive, trim = FALSE, s = "lam.min")
k_opt <- cv_naive$Ftrim$lam.min['df']
beta_C <- matrix(beta[1:(p*k_opt)], byrow = TRUE, nrow = p)
## does NOT satisfy zero-sum constraints
cat("colSums:", colSums(beta_C))
Nonzero <- (1:p)[apply(beta_C, 1, function(x) max(abs(x)) >0)]
beta_curve <- splines::bs(sseq, df = k_opt, intercept = TRUE) %*% t(beta_C)
Curve_L2 <- colSums(beta_curve^2) - colSums(beta_curve[c(1, nrow(beta_curve)), ]^2) / 2
Curve_L2 <- sqrt(Curve_L2 * (Data$basis.info[2,1] - Data$basis.info[1,1]))
cutoff <- sum(Curve_L2) / p
Nonzero_cut <- (1:p)[which(Curve_L2 >= cutoff)]
y_hat <- predict(cv_naive, Data$data$Comp, Data$data$Zc, s = "lam.min")
MSE <- sum((drop(Data$data$y) - y_hat)^2) / n_train
y_hat <- predict(cv_naive, Test$data$Comp, Test$data$Zc, s = "lam.min")
PRE <- sum((drop(Test$data$y) - y_hat)^2) / n_test
naive_result <- list(cv.result = cv_naive, beta = beta,
                     Nonzero = c("Original" = Nonzero, "Cut" = Nonzero_cut),
                     MSE = MSE, PRE = PRE)

## cv_base: random select a component as reference
## mu_ratio is set to 0 automatically once ref is set to a integer
ref = sample(1:p, 1)
cv_base <- cv.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                         Zc = Data$data$Zc, intercept = Data$data$intercept,
                         k = k_list, foldid = foldid, keep = TRUE,
                         ref = ref)
plot(cv_base, k = k_list)
beta <-  coef(cv_base, trim = FALSE, s = "lam.min")
k_opt <- cv_base$Ftrim$lam.min['df']
beta_C <- matrix(beta[1:(p*k_opt)], byrow = TRUE, nrow = p)
## satisfies zero-sum constraints
cat("colSums:", colSums(beta_C))
Nonzero <- (1:p)[apply(beta_C, 1, function(x) max(abs(x)) >0)]
beta_curve <- splines::bs(sseq, df = k_opt, intercept = TRUE) %*% t(beta_C)
Curve_L2 <- colSums(beta_curve^2) - colSums(beta_curve[c(1, nrow(beta_curve)), ]^2) / 2
Curve_L2 <- sqrt(Curve_L2 * (Data$basis.info[2,1] - Data$basis.info[1,1]))
cutoff <- sum(Curve_L2) / p
Nonzero_cut <- (1:p)[which(Curve_L2 >= cutoff)]
y_hat <- predict(cv_base, Data$data$Comp, Data$data$Zc, s = "lam.min")
MSE <- sum((drop(Data$data$y) - y_hat)^2) / n_train
y_hat <- predict(cv_base, Test$data$Comp, Test$data$Zc, s = "lam.min")
PRE <- sum((drop(Test$data$y) - y_hat)^2) / n_test
base_result <- list(cv.result = cv_base, beta = beta,
                    Nonzero = c("Original" = Nonzero, "Cut" = Nonzero_cut),
                    MSE = MSE, PRE = PRE)
```

`Fcomp_Model` *Simulation for functional composition data.*

Description

simulate functional compositional data.

Usage

```
Fcomp_Model(n, p, m = 0, intercept = TRUE,
            interval = c(0, 1), n_T = 100, obs_spar = 0.6, discrete = FALSE,
            SNR = 1, sigma = 2, Nzero_group = 4,
            rho_X, Corr_X = c("CorrCS", "CorrAR"),
            rho_T, Corr_T = c("CorrAR", "CorrCS"),
            range_beta = c(0.5, 1), beta_c = 1, beta_C ,
            theta.add = c(1, 2, 5, 6), gamma = 0.5,
         basis_beta = c("bs", "OBasis", "fourier"), df_beta = 5, degree_beta = 3,
          insert = c("FALSE", "X", "basis"), method = c("trapezoidal", "step"))
```

Arguments

`n`	sample size.
`p`	number of the components in the functional compositional data.
`m`	size of unpenalized variables. The first `ceiling(m/2)` ones are generated with independent `bin(1,0.5)` entries; while the last `(m -ceiling(m/2))` ones are generated with independent `norm(0,1)` entries. Default is 0.
`intercept`	whether to include an intercept. Default is `TRUE`.
`interval`	a vector of length 2 indicating the time domain. Default is `c(0,1)`.
`n_T`	an integer specifying length of the equally spaced time sequence on domian `interval`.
`obs_spar`	a percentage used to get sparse ovbservation. Each time point is with probability `obs_spar` to be observed. It allows different subject to be observed on different time points. `obs_spar * n_T > 5` is required.
`discrete`	logical (default is `FALSE`) specifying whether the functional compositional data X is generated at different time points. If `distrete = TRUE`, generate X on dense sequence created by `max(ns_dense = 200 * diff(interval),5 * n_T)` and then for each subject, randomly sample `n_T` points.
`SNR`	signal to noise ratio.
`sigma`	variance used to generate the covariance matrix `CovMIX = sigma^2 * kronecker(T.Sigma,X.Sigma)`. The "non-normalized" data w_i for each subject is genearted from multivariate normal distribution with covariance `CovMIX`. `T.Sigma` and `X.Sigma` are correlation matrices for time points and components, respectively.
`Nzero_group`	an even integer specifying that the first `Nzero_group` compositional predictors are with non-zero effects. Default is 4.
`rho_X, rho_T`	parameters used to generate correlation matrices.
`Corr_X, Corr_T`	character string specifying correlation structure bewteen components and between time points, respectively. • `"CorrCS"`(Default for `Corr_X`) compound symmetry.

- "CorrAR"(Default for Corr_T) autoregressive.

range_beta
: a sorted vector of length 2, specifying the range of coefficient matrix B of demension $p \times k$. Specifically, each column of B is filled with Nzero_group/2 values from the unifom distribution over range_beta and their negative counterparts. Default is c(0.5,1).

beta_c
: value of coefficients for beta0 and beta_c (coefficients for intercept and time-invariant predictors). Default is 1.

beta_C
: vectorized coefficient matrix. If missing, the program will generate beta_C according to range_beta and Nzero_group.

theta.add
: logical or integer(s).
  - If integer(s), a vector with value(s) in [1,p], indicating which component(s) of compostions is of high level mean curve.
  - If TRUE, the components c(1:ceiling(Nzero_group/2) and Nzero_group + (1:ceiling(Nzero_group/2))) are set to with high level mean.
  - if FALSE, all mean curves are set to 0's.

gamma
: for the high-level mean groups, log(p * gamma) is added on the "non-normalized" data w_i before the data are converted to be compositional.

basis_beta, df_beta, degree_beta
: basis_fun, k and degree in FuncompCGL respectively.

insert
: a character string sepcifying method to perform functional interpolation.
  - "FALSE"(Default) no interpolation.
  - "X" linear interpolation of functional compositional data along the time grid.
  - "basis" the functional compositional data is interplolated as a step function along the time grid.

  If insert = "X" or "basis", interplolation is conducted on sseq, where sseq is the sorted sequence of all the observed time points.

method
: a character string sepcifying method used to approximate integral.
  - "trapezoidal"(Default) Sum up areas under the trapezoids.
  - "step" Sum up area under the rectangles.

Details

The setup of this simulation follows Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*.

Specifically, we first generate correlation matrix X.sigma for components of a composition based on rho_X and Corr_X, and correlation matrix T.sigma for time points based on rho_T and Corr_T. Then, the "non-normalized" data $w_i = [w_i(t_1)^T, ..., w_i(t_{n_T})^T]$ for each subject are generated from multivariate normal distrubtion with covariance CovMIX = sigma^2 * kronecker(T.Sigma,X.Sigma), and the mean vector is determined by theta.add and gamma. Each $w_i(t_v)$ is a p-vector for each time point $v = 1, ..., T_n$. Finally, the compositional data are obtained as

$$x_{ij}(t_v) = exp(w_{ij}(t_v))/sum_{k=1}^{p} exp(w_{ik}(t_v)),$$

for each subject $i = 1, ..., n$, component of a composition $j = 1, ..., p$ and time point $v = 1, ..., n_T$.

Value

a list including

data a list of observed data,

- y a vector of response variable,
- Comp a data frame of observed functional compositional data, a column of Subject_ID, and a column of TIME,
- Zc a matrix of unpenalized variables with dimension $n \times m$,
- intercept whether an intercept is included.

beta a length p*df_beta + m + 1 vector of coefficients

basis.info matrix of the basis function to generate the coefficient curves

data.raw a list consisting of

- Z_t.full the functional compositional data.
- Z_ITG integrated functional compositional data.
- Y.tru true response vector without noise.
- X functional "non-normalized" data W.

parameter a list of parameters used in the simulation.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

Examples

```
Data <- Fcomp_Model(n = 50, p = 30, m = 0, intercept = TRUE, Nzero_group = 4,
                    n_T = 20, SNR = 3, rho_X = 0, rho_T = 0.6,
                    df_beta = 5, obs_spar = 1, theta.add = FALSE)
```

FuncompCGL *Fit regularization paths of sparse log-contrast regression with functional compositional predictors.*

Description

Fit the penalized *log-contrast* regression with functional compositional predictors proposed by Zhe et al. (2020) <arXiv:1808.02403>. The model estimation is conducted by minimizing a linearly constrained group lasso criterion. The regularization paths are computed for the group lasso penalty at grid values of the regularization parameter lam and the degree of freedom of the basis function K.

Usage

```
FuncompCGL(y, X, Zc = NULL, intercept = TRUE, ref = NULL,
           k, degree = 3, basis_fun = c("bs", "OBasis", "fourier"),
          insert = c("FALSE", "X", "basis"), method = c("trapezoidal", "step"),
           interval = c("Original", "Standard"), Trange,
           T.name = "TIME", ID.name = "Subject_ID",
           W = rep(1,times = p - length(ref)),
           dfmax = p - length(ref), pfmax = min(dfmax * 1.5, p - length(ref)),
           lam = NULL, nlam = 100, lambda.factor = ifelse(n < p1, 0.05, 0.001),
           tol = 1e-8, mu_ratio = 1.01,
           outer_maxiter = 1e+6, outer_eps = 1e-8,
           inner_maxiter = 1e+4, inner_eps = 1e-8)
```

Arguments

`y` response vector with length n.

`X` data frame or matrix.

- If `nrow(X)` $> n$, `X` should be a data frame or matrix of the functional compositional predictors with p columns for the values of the composition components, a column indicating subject ID and a column of observation times. Order of Subject ID should be the SAME as that of `y`. Zero entry is not allowed.
- If `nrow(X)[1]`$=n$, `X` is considered as after taken integration, a `n*(k*p -length(ref))` matrix.

`Zc` a $n \times p_c$ design matrix of unpenalized variables. Default is NULL.

`intercept` Boolean, specifying whether to include an intercept. Default is TRUE.

`ref` reference level (baseline), either an integer between $[1, p]$ or `NULL`. Default value is `NULL`.

- If `ref` is set to be an integer between `[1,p]`, the group lasso penalized *log-contrast* model (with log-ratios) is fitted with the `ref`-th component chosed as baseline.
- If `ref` is set to be `NULL`, the linearly constrained group lasso penalized *log-contrast* model is fitted.

`k` an integer, degrees of freedom of the basis function.

`degree` degrees of freedom of the basis function. Default value is 3.

`basis_fun` method to generate basis:

- `"bs"`(Default) B-splines. See fucntion `bs`.
- `"OBasis"` Orthoganal B-splines. See function `OBasis` and package **orthogonalsplinebasis**.
- `"fourier"` Fourier basis. See fucntion `create.fourier.basis` and package **fda**.

`insert` a character string sepcifying method to perform functional interpolation.

- `"FALSE"`(Default) no interpolation.

- `"X"` linear interpolation of functional compositional data along the time grid.
- `"basis"` the functional compositional data is interplolated as a step function along the time grid.

If `insert = "X"` or `"basis"`, interplolation is conducted on `sseq`, where `sseq` is the sorted sequence of all the observed time points.

`method` a character string sepcifying method used to approximate integral.

- `"trapezoidal"`(Default) Sum up areas under the trapezoids.
- `"step"` Sum up area under the rectangles.

`interval` a character string sepcifying the domain of the integral.

- `"Original"`(Default) On the original time scale, `interval = range(Time)`.
- `"Standard"` Time points are mapped onto $[0, 1]$, `interval = c(0,1)`.

`Trange` range of time points

`T.name, ID.name`

a character string specifying names of the time variable and the Subject ID variable in `X`. This is only needed when X is a data frame or matrix of the functional compositional predictors. Default are `"TIME"` and `"Subject_ID"`.

`W` a vector of length p (the total number of groups), or a matrix with dimension $p_1 \times p_1$, where `p1=(p -length(ref)) * k`, or character specifying the function used to calculate weight matrix for each group.

- a vector of penalization weights for the groups of coefficients. A zero weight implies no shrinkage.
- a diagonal matrix with positive diagonal elements.
- if character string of function name or an object of type `function` to compute the weights.

`dfmax` limit the maximum number of groups in the model. Useful for handling very large p, if a partial path is desired. Default is p.

`pfmax` limit the maximum number of groups ever to be nonzero. For example once a group enters the model along the path, no matter how many times it re-enters the model through the path, it will be counted only once. Default is `min(dfmax*1.5,p)`.

`lam` a user supplied lambda sequence. If `lam` is provided as a scaler and `nlam`> 1, `lam` sequence is created starting from `lam`. To run a single value of `lam`, set `nlam`$= 1$. The program will sort user-defined `lambda` sequence in decreasing order.

`nlam` the length of the `lam` sequence. Default is 100. No effect if `lam` is provided.

`lambda.factor` the factor for getting the minimal lambda in `lam` sequence, where `min(lam)` = `lambda.factor` * `max(lam)`. `max(lam)` is the smallest value of `lam` for which all penalized group are 0's. If $n >= p1$, the default is `0.001`. If $n < p1$, the default is `0.05`.

`tol` tolerance for coefficient to be considered as non-zero. Once the convergence criterion is satisfied, for each element β_j in coefficient vector β, $\beta_j = 0$ if $\beta_j < tol$.

`mu_ratio` the increasing ratio of the penalty parameter `u`. Default value is 1.01. Inital values for scaled Lagrange multipliers are set as 0's. If `mu_ratio` < 1, the program automatically set the initial penalty parameter `u` as 0 and `outer_maxiter` as 1, indicating that there is no linear constraint.

`outer_maxiter, outer_eps`
: `outer_maxiter` is the maximum number of loops allowed for the augmented Lanrange method; and `outer_eps` is the corresponding convergence tolerance.

`inner_maxiter, inner_eps`
: `inner_maxiter` is the maximum number of loops allowed for blockwise-GMD; and `inner_eps` is the corresponding convergence tolerance.

Details

The *functional log-contrast regression model* for compositional predictors is defined as

$$y = 1_n\beta_0 + Z_c\beta_c + \int_T Z(t)\beta(t)dt + e, s.t.(1_p)^T\beta(t) = 0 \forall t \in T,$$

where β_0 is the intercept, β_c is the regression coefficient vector with length p_c corresponding to the control variables, $\beta(t)$ is the functional regression coefficient vector with length p as a funtion of t and e is the random error vector with zero mean with length n. Moreover, `Z(t)` is the log-transformed functional compostional data. If zero(s) exists in the original functional compositional data, user should pre-process these zero(s). For example, if count data provided, user could replace 0's with 0.5.
After adopting a truncated basis expansion approach to re-express $\beta(t)$

$$\beta(t) = B\Phi(t),$$

where B is a p-by-k unkown but fixed coefficient matrix, and $\Phi(t)$ consists of basis with degree of freedom k. We could write *functional log-contrast regression model* as

$$y = 1_n\beta_0 + Z_c\beta_c + Z\beta + e, s.t. \sum_{j=1}^{p} \beta_j = 0_k,$$

where Z is a n-by-pk matrix corresponding to the integral, $\beta = vec(B^T)$ is a pk-vector with every each k-subvector corresponding to the coefficient vector for the j-th compositional component.
To enable variable selection, `FuncompCGL` model is estimated via linearly constrained group lasso,

$$argmin_{\beta_0,\beta_c,\beta}(\frac{1}{2n}\|y - 1_n\beta_0 - Z_c\beta_c - Z\beta\|_2^2 + \lambda\sum_{j=1}^{p}\|\beta_j\|_2), s.t. \sum_{j=1}^{p}\beta_j = 0_k.$$

Value

An object with S3 class `"FuncompCGL"`, which is a list containing:

`Z` the integral matrix for the functional compositional predictors with dimension $n \times (pk)$.

`lam` the sequence of `lam` values.

df	the number of non-zero groups in the estimated coefficients for the functional compositional predictors at each value of lam.
beta	a matrix of coefficients with length(lam) columns and $p_1 + p_c + 1$ rows, where p_1=p*k. The first p_1 rows are the estimated values for the coefficients for the functional compositional preditors, and the last row is for the intercept. If intercept = FALSE, the last row is 0's.
dim	dimension of the coefficient matrix.
sseq	sequence of the time points.
call	the call that produces this object.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*.

Yang, Y. and Zou, H. (2015) *A fast unified algorithm for computing group-lasso penalized learning problems*, https://link.springer.com/article/10.1007/s11222-014-9498-5 *Statistics and Computing* **25(6)** 1129-1141.

Aitchison, J. and Bacon-Shone, J. (1984) *Log-contrast models for experiments with mixtures*, *Biometrika* **71** 323-330.

See Also

cv.FuncompCGL and GIC.FuncompCGL, and predict, coef, plot and print methods for "FuncompCGL" object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = 50, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0, rho_T = 0.6, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
m1 <- FuncompCGL(y = Data$data$y, X = Data$data$Comp, Zc = Data$data$Zc,
                 intercept = Data$data$intercept, k = df_beta, tol = 1e-10)
print(m1)
plot(m1)
beta <- coef(m1)
arg_list <- as.list(Data$call)[-1]
```

```
arg_list$n <- 30
TEST <- do.call(Fcomp_Model, arg_list)
y_hat <- predict(m1, Znew = TEST$data$Comp, Zcnew = TEST$data$Zc)
plot(y_hat[, floor(length(m1$lam)/2)], TEST$data$y,
     ylab = "Observed Response", xlab = "Predicted Response")

beta <- coef(m1, s = m1$lam[20])
beta_C <- matrix(beta[1:(p*df_beta)], nrow = p, byrow = TRUE)
colSums(beta_C)
Non.zero <- (1:p)[apply(beta_C, 1, function(x) max(abs(x)) > 0)]
Non.zero
```

`GIC.compCL` *Compute information crieteria for the* `compCL` *model.*

Description

Tune the penalty parameter codelam in the `compCGL` model by GIC, BIC, or AIC. This function calculates the GIC, BIC, or AIC curve and returns the optimal value of `lam`.

Usage

```
GIC.compCL(y, Z, Zc = NULL, intercept = FALSE, lam = NULL, ...)
```

Arguments

`y`	a response vector with length n.
`Z`	a $n \times p$ design matrix of compositional data or categorical data. If `Z` is categorical data, i.e., row-sums of `Z` differ from 1, the program automatically transforms `Z` into compositional data by dividing each row by its sum. `Z` could NOT include entry of 0's.
`Zc`	a $n * p_c$ design matrix of control variables (not penalized). Default is `NULL`.
`intercept`	Boolean, specifying whether to include an intercept. Default is `FALSE`.
`lam`	a user supplied lambda sequence. If `lam` is provided as a scaler and `nlam`> 1, `lam` sequence is created starting from `lam`. To run a single value of `lam`, set `nlam`$= 1$. The program will sort user-defined `lambda` sequence in decreasing order.
`...`	other arguments that can be passed to compCL.

Details

The model estimation is conducted through minimizing the following criterion:

$$\frac{1}{2n}\|y - Z\beta\|_2^2 + \lambda\|\beta\|_1, s.t. \sum_{j=1}^{p} \beta_j = 0.$$

The GIC is defined as:

$$GIC(\lambda) = \log \hat{\sigma}^2(\lambda) + (s(\lambda) - 1) \log (max(p, n)) * \log (\log n)/n,$$

where $\hat{\sigma}^2(\lambda) = \|y - Z\hat{\beta}(\lambda)\|_2^2/n$, $\hat{\beta}(\lambda)$ is the regularized estimator, and $s(\lambda)$ is the number of nonzero coefficients in $\hat{\beta}(\lambda)$. Because of the zero-sum constraint, the effective number of free parameters is $s(\lambda) - 1$ for $s(\lambda) \geq 2$. The optimal λ is selected by minimizing `GIC`(λ).

Value

an object of S3 class `GIC.compCL` is returned, which is a list:

`compCL.fit` a fitted `compCL` object.

`lam` the sequence of `lam`.

`GIC` a vector of GIC value(s).

`lam.min` the `lam` value that minimizes `GIC`(λ).

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979

Fan, Y., and Tang, C. Y. (2013) *Tuning parameter selection in high dimensional penalized likelihood*, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12001 *Journal of the Royal Statistical Society. Series B* **75** 531-552

See Also

`compCL` and `cv.compCL`, and `coef`, `predict` and `plot` methods for "`GIC.compCL`" object.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
GICm1 <- GIC.compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                    Zc = Comp_data$Zc, intercept = Comp_data$intercept)
coef(GICm1)
plot(GICm1)
test_data = comp_Model(n = 100, p = p, beta = Comp_data$beta, intercept = FALSE)
y_hat = predict(GICm1, Znew = test_data$X.comp, Zcnew = test_data$Zc)
plot(test_data$y, y_hat, xlab = "Observed value", ylab = "Predicted value")
abline(a = 0, b = 1, col = "red")
```

GIC.FuncompCGL *Compute information crieteria for the* FuncompCGL *model.*

Description

Tune the grid values of the penalty parameter codelam and the degrees of freedom of the basis function k in the FuncompCGL model by GIC, BIC, or AIC. This function calculates the GIC, BIC, or AIC curve and returns the optimal values of lam and k.

Usage

```
GIC.FuncompCGL(y, X, Zc = NULL, lam = NULL, nlam = 100, k = 4:10, ref = NULL,
               intercept = TRUE, W = rep(1,times = p - length(ref)),
               type = c("GIC", "BIC", "AIC"),
               mu_ratio = 1.01, outer_maxiter = 1e+6, ...)
```

Arguments

y — response vector with length n.

X — data frame or matrix.

- If nrow(X) > n, X should be a data frame or matrix of the functional compositional predictors with p columns for the values of the composition components, a column indicating subject ID and a column of observation times. Order of Subject ID should be the SAME as that of y. Zero entry is not allowed.
- If nrow(X)[1]=n, X is considered as after taken integration, a n*(k*p -length(ref)) matrix.

Zc — a $n \times p_c$ design matrix of unpenalized variables. Default is NULL.

lam — a user supplied lambda sequence. If lam is provided as a scaler and nlam> 1, lam sequence is created starting from lam. To run a single value of lam, set nlam$= 1$. The program will sort user-defined lambda sequence in decreasing order.

nlam — the length of the lam sequence. Default is 100. No effect if lam is provided.

k — an integer vector specifying the degrees of freedom of the basis function.

ref — reference level (baseline), either an integer between $[1, p]$ or NULL. Default value is NULL.

- If ref is set to be an integer between [1,p], the group lasso penalized *log-contrast* model (with log-ratios) is fitted with the ref-th component chosed as baseline.
- If ref is set to be NULL, the linearly constrained group lasso penalized *log-contrast* model is fitted.

intercept — Boolean, specifying whether to include an intercept. Default is TRUE.

W	a vector of length p (the total number of groups), or a matrix with dimension $p_1 \times p_1$, where p1=(p -length(ref)) * k, or character specifying the function used to calculate weight matrix for each group. • a vector of penalization weights for the groups of coefficients. A zero weight implies no shrinkage. • a diagonal matrix with positive diagonal elements. • if character string of function name or an object of type function to compute the weights.
type	a character string specifying which crieterion to use. The choices include "GIC" (default), "BIC", and "AIC".
mu_ratio	the increasing ratio of the penalty parameter u. Default value is 1.01. Inital values for scaled Lagrange multipliers are set as 0's. If mu_ratio < 1, the program automatically set the initial penalty parameter u as 0 and outer_maxiter as 1, indicating that there is no linear constraint.
outer_maxiter	maximum number of loops allowed for the augmented Lanrange method.
...	other arguments that could be passed to FuncompCL.

Details

The FuncompCGL model estimation is conducted through minimizing the linearly constrained group lasso criterion

$$\frac{1}{2n}\|y - 1_n\beta_0 - Z_c\beta_c - Z\beta\|_2^2 + \lambda \sum_{j=1}^{p} \|\beta_j\|_2, s.t. \sum_{j=1}^{p} \beta_j = 0_k.$$

The tuning parameters can be selected by the generalized information crieterion (GIC),

$$GIC(\lambda, k) = \log(\hat{\sigma}^2(\lambda, k)) + (s(\lambda, k) - 1)k \log(max(p * k + p_c + 1, n)) \log(\log n)/n,$$

where $\hat{\sigma}^2(\lambda, k) = \|y - 1_n\hat{\beta}_0(\lambda, k) - Z_c\hat{\beta}_c(\lambda, k) - Z\hat{\beta}(\lambda, k)\|_2^2/n$ with $\hat{\beta}_0(\lambda, k)$, $\hat{\beta}_c(\lambda, k)$ and $\hat{\beta}(\lambda, k)$ being the regularized estimators of the regression coefficients, and $s(\lambda, k)$ is the number of nonzero coefficient groups in $\hat{\beta}(\lambda, k)$.

Value

An object of S3 class "GIC.FuncompCGL" is returned, which is a list containing:

FuncompCGL.fit	a list of length length(k), with fitted FuncompCGL objects of different degrees of freedom of the basis function.
lam	the sequence of the penalty parameter lam.
GIC	a k by length(lam) matirx of GIC values.
lam.min	the optimal values of the degrees of freedom k and the penalty parameter lam.
MSE	a k by length(lam) matirx of mean squared errors.

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*.

Fan, Y., and Tang, C. Y. (2013) *Tuning parameter selection in high dimensional penalized likelihood*, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12001 *Journal of the Royal Statistical Society. Series B* **75** 531-552.

See Also

`FuncompCGL` and `cv.FuncompCGL`, and `predict`, `coef` and `plot` methods for `"GIC.FuncompCGL"` object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
n_train = 50
n_test = 30
k_list <- c(4,5)
Data <- Fcomp_Model(n = n_train, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0.2, rho_T = 0.5,
                    df_beta = df_beta, n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
arg_list <- as.list(Data$call)[-1]
arg_list$n <- n_test
Test <- do.call(Fcomp_Model, arg_list)

## GIC_cgl: Constrained group lasso
GIC_cgl <- GIC.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                          Zc = Data$data$Zc, intercept = Data$data$intercept,
                          k = k_list)
coef(GIC_cgl)
plot(GIC_cgl)
y_hat <- predict(GIC_cgl, Znew = Test$data$Comp, Zcnew = Test$data$Zc)
plot(Test$data$y, y_hat, xlab = "Observed response", ylab = "Predicted response")

## GIC_naive: ignoring the zero-sum constraints
## set mu_raio = 0 to identifying without linear constraints,
## no outer_loop for Lagrange augmented multiplier
GIC_naive <- GIC.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                            Zc = Data$data$Zc, intercept = Data$data$intercept,
                            k = k_list, mu_ratio = 0)
coef(GIC_naive)
plot(GIC_naive)
```

```
y_hat <- predict(GIC_naive, Znew = Test$data$Comp, Zcnew = Test$data$Zc)
plot(Test$data$y, y_hat, xlab = "Observed response", ylab = "Predicted response")

## GIC_base: random select a component as reference
## mu_ratio is set to 0 automatically once ref is set to a integer
ref <- sample(1:p, 1)
GIC_base <- GIC.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                            Zc = Data$data$Zc, intercept = Data$data$intercept,
                            k = k_list, ref = ref)
coef(GIC_base)
plot(GIC_base)
y_hat <- predict(GIC_base, Znew = Test$data$Comp, Zcnew = Test$data$Zc)
plot(Test$data$y, y_hat, xlab = "Observed response", ylab = "Predicted response")
```

`plot.compCL` *Plot solution paths from a "*`compCL`*" object.*

Description

Produce a coefficient profile plot from a fitted "compCL" object.

Usage

```
## S3 method for class 'compCL'
plot(x, xlab = c("lam", "norm"), label = FALSE, ...)
```

Arguments

`x`	fitted "compCL" model.
`xlab`	what is on the X-axis. "lam" plots against the log-lambda sequence (default) and "norm" against the L1-norm of the coefficients.
`label`	if TRUE, label the curve with the variable sequence numbers. Default is `FALSE`.
`...`	other graphical parameters.

Details

A coefficient profile plot for the compositional predictors is produced.

Value

No return value. Side effect is a base R plot.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

`compCL` and `print`, `predict` and `coef` methods for `"compCL"` object.

Examples

```
Comp_data = comp_Model(n = 50, p = 30)
Comp_fit = compCL(y = Comp_data$y, Z = Comp_data$X.comp, Zc = Comp_data$Zc,
                  intercept = Comp_data$intercept)
plot(Comp_fit)
plot(Comp_fit, xlab = "norm", label = TRUE)
```

`plot.cv.compCL` *Plot the cross-validation curve produced by* `"cv.compCL"` *object.*

Description

Plot the cross-validation curve with its upper and lower standard deviation curves.

Usage

```
## S3 method for class 'cv.compCL'
plot(x, xlab = c("log", "-log", "lambda"), trim = FALSE, ...)
```

Arguments

`x`	fitted `"cv.compCL"` object.
`xlab`	what is on the X-axis, `"log"` plots against `log(lambda)` (default), `"-log"` against `-log(lambda)`, and `"lambda"` against `lambda`.
`trim`	logical; whether to use the trimmed result. Default is `FALSE`.
`...`	other graphical parameters.

Details

A cross-validation curve is produced.

Value

No return value. Side effect is a base R plot.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

`cv.compCL` and `compCL`, and `coef` and `plot` methods for `"cv.compCL"` object.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, intercept = FALSE)
cvm1 <- cv.compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                  Zc = Comp_data$Zc, intercept = Comp_data$intercept)
plot(cvm1)
plot(cvm1, xlab = "-log")
```

---

plot.cv.FuncompCGL   *Plot the cross-validation curve produced by* "cv.FuncompCGL".

Description

Plot the cross-validation curve with its upper and lower standard deviation curves.

Usage

```
## S3 method for class 'cv.FuncompCGL'
plot(x, xlab = c("log", "-log", "lambda"), trim = FALSE, k, ...)
```

Arguments

- `x` — fitted `"cv.FuncompCGL"` model.
- `xlab` — what is on the X-axis, `"log"` plots against `log(lambda)` (default), `"-log"` against `-log(lambda)`, and `"lambda"` against `lambda`.
- `trim` — logical; whether to use the trimmed result. Default is `FALSE`.
- `k` — a vector or character string
  - if character string, either `"lam.1se"` or `"lam.min"`.
  - if it is an integer vector, specify the set of degrees of freedom `k` to plot.
  - if it is missing (default), cross-validation curves for `k` that are associated with `lambda.min` (blue) and `lambda.1se` (red) are plotted.
- `...` — other graphical parameters.

Details

A cross-validation curve is produced.

Value

No return value. Side effect is a base R plot.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`cv.FuncompCGL` and `FuncompCGL`, and `predict` and `coef` methods for `"cv.FuncompCGL"` object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = 50, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0, rho_T = 0.6, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
k_list <- 4:5
cv_m1 <-  cv.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                        Zc = Data$data$Zc, intercept = Data$data$intercept,
                        k = k_list, nfolds = 5, keep = TRUE)
plot(cv_m1)
plot(cv_m1, xlab = "-log", k = k_list)
```

`plot.FuncompCGL` *Plot solution paths from a* `"FuncompCGL"` *object.*

Description

Produce a coefficient profile plot of the coefficient paths for a fitted `"FuncompCGL"` object.

Usage

```
## S3 method for class 'FuncompCGL'
plot(x, ylab = c("L2", "L1"), xlab = c("log", "-log", "lambda"), ...)
```

Arguments

`x`	fitted `"FuncompCGL"` object.
`ylab`	what is the on Y-axis, `"L2"` (default) plots against the L2-norm of each group of coefficients, `"L1"` against L1-norm.
`xlab`	what is on the X-axis, `"log"` plots against `log(lambda)` (default), `"-log"` against `-log(lambda)`, and `"lambda"` against `lambda`.
`...`	other graphical parameters.

Details

A solution path plot is produced.

Value

No return value. Side effect is a base R plot.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`FuncompCGL`, and `predict`, `coef` and `print` methods for `"FuncompCGL"` object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = 50, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0, rho_T = 0.6, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
m1 <- FuncompCGL(y = Data$data$y, X = Data$data$Comp, Zc = Data$data$Zc,
                 intercept = Data$data$intercept, k = df_beta, tol = 1e-10)
plot(m1)
plot(m1, ylab = "L1", xlab = "-log")
```

---

plot.GIC.compCL — *Plot the GIC curve produced by* "GIC.compCL" *object.*

Description

Plot the CIC curve as a function of the lam values.

Usage

```
## S3 method for class 'GIC.compCL'
plot(x, xlab = c("log", "-log", "lambda"), ...)
```

Arguments

x	fitted "GIC.compCL" object.
xlab	what is on the X-axis, "log" plots against log(lambda) (default), "-log" against -log(lambda), and "lambda" against lambda.
...	other graphical parameters.

Details

A GIC curve is produced.

Value

No return value. Side effect is a base R plot.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

`GIC.compCL` and `compCL`, and `predict` and `coef` methods for `"GIC.compCL"` object.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
GICm1 <- GIC.compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                    Zc = Comp_data$Zc, intercept = Comp_data$intercept)
plot(GICm1)
plot(GICm1, xlab = "-log")
```

`plot.GIC.FuncompCGL` *Plot the GIC curve produced by* `"GIC.FuncompCGL"` *object.*

Description

Plot the GIC curve as a function of the `lam` values used for different degree of freedom `k`.

Usage

```
## S3 method for class 'GIC.FuncompCGL'
plot(x, xlab = c("log", "-log", "lambda"), k, ...)
```

Arguments

`x`	fitted `"GIC.FuncompCGL"` object.
`xlab`	what is on the X-axis, `"log"` plots against `log(lambda)` (default), `"-log"` against `-log(lambda)`, and `"lambda"` against `lambda`.
`k`	value(s) of the degrees of freedom at which GIC cuvre(s) are plotted. • if missing (default), GIC curve for `k` that is associated with `"lam.min"` (RED) stored on `x` is plotted. • if it is an integer vector, specify what set of degrees of freedom to plot.
`...`	other graphical parameters.

Details

A GIC curve is produced.

Value

No return value. Side effect is a base R plot.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`GIC.FuncompCGL` and `FuncompCGL`, and `predict` and `coef` methods for `"GIC.FuncompCGL"` object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = 50, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0.6, rho_T = 0,
                    df_beta = df_beta, n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))

k_list <- c(4,5)
GIC_m1 <-  GIC.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                          Zc = Data$data$Zc, intercept = Data$data$intercept,
                          k = k_list)
plot(GIC_m1)
plot(GIC_m1, xlab = "-log", k = k_list)
```

---

predict.compCL   *Make predictions based on a* `"compCL"` *object.*

Description

Make predictions based on a fitted `"compCL"` object.

Usage

```
## S3 method for class 'compCL'
predict(object, Znew, Zcnew = NULL, s = NULL, ...)
```

Arguments

`object`	fitted "`compCL`" object.
`Znew`	`z` matrix as in `compCL` with new compositional data or categorical data.
`Zcnew`	`Zc` matrix as in `compCL` with new data for other covariates. Default is `NULL`
`s`	value(s) of the penalty parameter `lam` at which predictions are required. Default is the entire sequence used in the fitted object.
`...`	not used.

Details

`s` is the vector at which predictions are requested. If `s` is not in the lambda sequence used for fitting the model, the `predict` function uses linear interpolation.

Value

predicted values at the requested values of `s`.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

`compCL` and `coef`, `predict` and `plot` methods for "`compCL`" object.

Examples

```
Comp_data = comp_Model(n = 50, p = 30)
Comp_data2 = comp_Model(n = 30, p = 30, beta = Comp_data$beta)
m1 = compCL(y = Comp_data$y, Z = Comp_data$X.comp,
            Zc = Comp_data$Zc, intercept = Comp_data$intercept)
predict(m1, Znew = Comp_data2$X.comp, Zcnew = Comp_data2$Zc)
predict(m1, Znew = Comp_data2$X.comp, Zcnew = Comp_data2$Zc, s = c(1, 0.5, 0.1))
```

predict.cv.compCL — *Make predictions based on a "cv.compCL" object.*

Description

This function makes prediction based on a cross-validated `compCL` model, using the stored `compCL.fit` object.

Usage

```
## S3 method for class 'cv.compCL'
predict(object, Znew, Zcnew = NULL, s = c("lam.min", "lam.1se" ),
        trim = FALSE, ...)
```

Arguments

`object`	fitted "`cv.compCL`" model.
`Znew`	`z` matrix as in `compCL` with new compositional data or categorical data.
`Zcnew`	`Zc` matrix as in `compCL` with new data for other covariates. Default is `NULL`
`s`	specify the `lam` at which prediction(s) is requested. • `s = "lam.min"` (default), value of `lam` that obtains the minimun value of cross-validation error. • `s = "lam.1se"` value of `lam` that obtains 1 standard error above the miminum of the cross-validation errors. • if `s` is numeric, it is taken as the value(s) of lam to be used. • if `s = NULL`, uses the whole sequence of `lam` stored in the "`cv.compCL`" object.
`trim`	Whether to use the trimmed result. Default is FASLE.
`...`	not used.

Details

`s` is the vector at which predictions are requested. If `s` is not in the lambda sequence used for fitting the model, the `predict` function uses linear interpolation.

Value

predicted values at the requested values of `s`.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

`cv.compCL` and `compCL`, and `coef` and `plot` methods for "`cv.compCL`" object.

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c( beta, rep(0, times = p - length(beta)) )
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
test_data = comp_Model(n = 30, p = p, beta = beta, intercept = FALSE)
cvm1 <- cv.compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                  Zc = Comp_data$Zc, intercept = Comp_data$intercept)
y_hat = predict(cvm1, Znew = test_data$X.comp, Zcnew = test_data$Zc)
predmat = predict(cvm1, Znew = test_data$X.comp, Zcnew = test_data$Zc, s = NULL)
plot(test_data$y, y_hat, xlab = "Observed response", ylab = "Predicted response")
abline(a = 0, b = 1, col = "red")
```

`predict.cv.FuncompCGL` *Make predictions based on a "`cv.FuncompCGL`" object.*

Description

This function makes prediction based on a `cv.FuncompCGL` object, using the stored "`FuncompCGL.fit`" object and the optimal values of the regularization parameter `lam` and the degrees of freedom `k`.

Usage

```
## S3 method for class 'cv.FuncompCGL'
predict(object, Znew, Zcnew = NULL,
        s = c("lam.1se", "lam.min"), k = NULL, trim = FALSE, ...)
```

Arguments

`object`	fitted `cv.FuncompCGL` object.
`Znew`	data frame or matrix `X` as in `FuncompCGL` with new functional compositional data at which prediction is to be made.
`Zcnew`	matrix `Zc` as in `FuncompCGL` with new values of time-invariant covariates at which prediction is to be made. Default is `NULL`.
`s`	value(s) of the penalty parameter `lam` at which coefficients are requested.

- s="lam.min"(default), grid value of lam and k stored in the "cv.FuncompCGL" object such that the minimum cross-validation error is achieved.
- s="lam.1se", grid value of lam and k stored on the "cv.FuncompCGL" object such that the 1 standard error above the miminum cross-validation error is achieved.
- If s is numeric, it is taken as the value(s) of lam to be used. In this case, k must be provided.
- If s = NULL, the whole sequence of lam stored in the cv.FuncompCGL object is used.

k value(s) of the degrees of freedom of the basis function at which coefficents are requested. k can be NULL (default) or integer(s).

- k = NULL, s must be either "lam.min" or "lam.1se".
- if k is an integer(s), it is taken as the value of k to be used and it must be one(s) of these in the "cv.FuncompCGL" object.

trim logical; whether to use the trimmed result. Default is FALSE.

... Other arguments passed to predict.FuncompCGL

Details

s is the vector at which predictions are requested. If s is not in the lam sequence used for fitting the model, the predict function uses linear interpolation.
If the data frame X is provided in FuncompCGL mode, the integral for new data newx is taken the same as that in the fitted FuncompCGL model. This means that the parameters degree, basis_fun, insert, method, inteval, Trange, and K are exactly the same as these in the provided object. If insert="X" or "basis", sseq is the sorted sequence of all the observed time points in fitting FuncompCGL model and all the observed time points in newx. Then interpolation is conducted on sseq. If matrix X after integral is provided in the FuncompCGL object, these parameters are required.

Value

The prediction values at the requested value(s) for s and k. If k is a vector, a list of prediction matrix is returned, otherwise a prediction matrix is returned.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

cv.FuncompCGL and FuncompCGL, and coef and plot methods for "cv.FuncompCGL" object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
n_train = 50
n_test = 30
Data <- Fcomp_Model(n = n_train, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0, rho_T = 0.6, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
arg_list <- as.list(Data$call)[-1]
arg_list$n <- n_test
Test <- do.call(Fcomp_Model, arg_list)
k_list = c(4,5)
cv_m1 <-  cv.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                        Zc = Data$data$Zc, intercept = Data$data$intercept,
                        k = k_list, nfolds = 5)
y_hat = predict(cv_m1, Znew = Test$data$Comp, Zcnew = Test$data$Zc)
predict(cv_m1, Znew = Test$data$Comp, Zcnew = Test$data$Zc, s = "lam.1se")
predict(cv_m1, Znew = Test$data$Comp, Zcnew = Test$data$Zc, s = c(0.5, 0.1, 0.05), k = k_list)
plot(Test$data$y, y_hat, xlab = "Observed Response", ylab = "Predicted Response")
```

`predict.FuncompCGL` *Make prediction from a* "FuncompCGL" *object.*

Description

Make prediction based on a fitted FuncompCGL object.

Usage

```
## S3 method for class 'FuncompCGL'
predict(object, Znew, Zcnew = NULL, s = NULL,
        T.name = "TIME", ID.name = "Subject_ID",
        Trange, interval, insert, basis_fun, degree, method, sseq,
        ...)
```

Arguments

object	fitted FuncompCGL object.
Znew	data frame or matrix X as in FuncompCGL with new functional compositional data at which prediction is to be made.

`Zcnew`	matrix `Zc` as in `FuncompCGL` with new values of time-invariate covariates at which prediction is to be made. Default is `NULL`.
`s`	value(s) of the penalty parameter `lam` at which predictions are requested. Default is the entire sequence used to fit the model.
`T.name`	a character string specifying names of the time variable and the Subject ID variable in `X`. This is only needed when X is a data frame or matrix of the functional compositional predictors. Default are `"TIME"` and `"Subject_ID"`.
`ID.name`	a character string specifying names of the time variable and the Subject ID variable in `X`. This is only needed when X is a data frame or matrix of the functional compositional predictors. Default are `"TIME"` and `"Subject_ID"`.
`Trange, interval, insert, basis_fun, degree, method`	the same as those in `FuncompCGL`.
`sseq`	full set of potential time points of observations; used for interpolation when `insert = "X"` or `insert = "basis"`.
`...`	not used.

Details

`s` is the vector at which predictions are requested. If `s` is not in the `lam` sequence used for fitting the model, the `predict` function uses linear interpolation.
If the data frame `X` is provided in `FuncompCGL` mode, the integral for new data `newx` is taken the same as that in the fitted `FuncompCGL` model. This means that the parameters `degree`, `basis_fun`, `insert`, `method`, `inteval`, `Trange`, and `K` are exactly the same as these in the provided `object`. If `insert="X"` or `"basis"`, `sseq` is the sorted sequence of all the observed time points in fitting `FuncompCGL` model and all the observed time points in `newx`. Then interpolation is conducted on `sseq`. If matrix `X` after integral is provided in the `FuncompCGL` object, these parameters are required.

Value

predicted values at the requested value(s) for `s`.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`FuncompCGL`, and `coef`, `plot` and `print` methods for `"FuncompCGL"` object.

Examples

```
p = 30
n_train = 50
n_test = 30
df_beta = 5
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = n_train, p = p, m = 0, intercept = TRUE,
                    SNR = 2, sigma = 2,
                    rho_X = 0, rho_T = 0.5, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = c(3,4,5),
                    beta_C = as.vector(t(beta_C_true)))
m1 <- FuncompCGL(y = Data$data$y, X = Data$data$Comp , Zc = Data$data$Zc,
                 intercept = Data$data$intercept, k = df_beta)
arg_list <- as.list(Data$call)[-1]
arg_list$n <- n_test
TEST <- do.call(Fcomp_Model, arg_list)
predmat <- predict(m1, Znew = TEST$data$Comp, Zcnew = TEST$data$Zc)
predmat <- predict(m1, Znew = TEST$data$Comp, Zcnew = TEST$data$Zc, s = c(0.5, 0.1, 0.05))
```

---

predict.GIC.compCL *Make predictions based on a "GIC.compCL" object.*

Description

This function makes prediction based on a "GIC.compCL" model, using the stored "compCL.fit" object and the optimal value of lambda.

Usage

```
## S3 method for class 'GIC.compCL'
predict(object, Znew, Zcnew = NULL, s = "lam.min", ...)
```

Arguments

object	fitted "GIC.compCL" model.
Znew	z matrix as in compCL with new compositional data or categorical data.
Zcnew	Zc matrix as in compCL with new data for other covariates. Default is NULL
s	specify the lam at which prediction(s) is requested. • s = "lam.min" (default), lam that obtains the minimun value of GIC values. • if s is numeric, it is taken as the value(s) of lam to be used. • if s = NULL, uses the whole sequence of lam stored in the "GIC.compCL" object.
...	not used.

Details

s is the vector at which predictions are requested. If s is not in the lambda sequence used for fitting the model, the predict function uses linear interpolation.

Value

predicted values at the requested values of s.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

GIC.compCL and compCL, and coef and plot methods for "GIC.compCL".

Examples

```
p = 30
n = 50
beta = c(1, -0.8, 0.6, 0, 0, -1.5, -0.5, 1.2)
beta = c(beta, rep(0, times = p - length(beta)))
Comp_data = comp_Model(n = n, p = p, beta = beta, intercept = FALSE)
test_data = comp_Model(n = 100, p = p, beta = beta, intercept = FALSE)
GICm1 <- GIC.compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                    Zc = Comp_data$Zc, intercept = Comp_data$intercept)
y_hat = predict(GICm1, Znew = test_data$X.comp, Zcnew = test_data$Zc)
predmat = predict(GICm1, Znew = test_data$X.comp, Zcnew = test_data$Zc, s = c(1, 0.5, 1))
plot(test_data$y, y_hat, xlab = "Observed value", ylab = "Predicted value")
abline(a = 0, b = 1, col = "red")
```

---

predict.GIC.FuncompCGL

Make predictions based on a "GIC.FuncompCGL" object.

Description

This function makes prediction based on a "GIC.FuncompCGL" object, using the stored "FuncompCGL.fit" object and the optimal values of the regularization parameter lam and the degrees of freedom k.

Usage

```
## S3 method for class 'GIC.FuncompCGL'
predict(object, Znew, Zcnew = NULL,
        s = "lam.min", k = NULL, ...)
```

Arguments

object	fitted GIC.FuncompCGL object.
Znew	data frame or matrix X as in FuncompCGL with new functional compositional data at which prediction is to be made.
Zcnew	matrix Zc as in FuncompCGL with new values of time-invariate covariates at which prediction is to be made. Default is NULL.
s	value(s) of the regularization parameter lam at which coefficients are requested. • s="lam.min" (default), grid value of lam and k stored in "GIC.FuncompCGL" object such that the minimun value of GIC is achieved. • If s is numeric, it is taken as the value(s) of lam to be used. In this case, k must be provided. • If s = NULL, used the whole sequence of lam stored in the GIC.FuncompCGL object.
k	value(s) of degrees of freedom of the basis function at which coefficents are requested. k can be NULL (default) or integer(s). • k = NULL, s must be "lam.min". • if k is integer(s), it is taken as the value of k to be used and it must be one(s) of these in "GIC.FuncompCGL" model.
...	Other arguments passed to predict.FuncompCGL

Details

s is the vector at which predictions are requested. If s is not in the lam sequence used for fitting the model, the predict function uses linear interpolation.
If the data frame X is provided in FuncompCGL mode, the integral for new data newx is taken the same as that in the fitted FuncompCGL model. This means that the parameters degree, basis_fun, insert, method, inteval, Trange, and K are exactly the same as these in the provided object. If insert="X" or "basis", sseq is the sorted sequence of all the observed time points in fitting FuncompCGL model and all the observed time points in newx. Then interpolation is conducted on sseq. If matrix X after integral is provided in the FuncompCGL object, these parameters are required.

Value

The prediction values at the requested value(s) for s and k. If k is a vector, a list of prediction matrix is returned, otherwise a prediction matrix is returned.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

`GIC.FuncompCGL` and `FuncompCGL`, and `coef` and `plot` methods for "`GIC.FuncompCGL`" object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
n_train = 50
n_test = 30
k_list <- c(4,5)
Data <- Fcomp_Model(n = n_train, p = p, m = 0, intercept = TRUE,
                    SNR = 4, sigma = 3, rho_X = 0.6, rho_T = 0,
                    df_beta = df_beta, n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
arg_list <- as.list(Data$call)[-1]
arg_list$n <- n_test
Test <- do.call(Fcomp_Model, arg_list)
GIC_m1 <-  GIC.FuncompCGL(y = Data$data$y, X = Data$data$Comp,
                          Zc = Data$data$Zc, intercept = Data$data$intercept,
                          k = k_list)
y_hat <- predict(GIC_m1, Znew = Test$data$Comp, Zcnew = Test$data$Zc)
predict(GIC_m1, Znew = Test$data$Comp, Zcnew = Test$data$Zc, s = NULL, k = k_list)
plot(Test$data$y, y_hat, xlab = "Observed response", ylab = "Predicted response")
```

`print.compCL` *Print a* "`compCL`" *object.*

Description

print the number of nonzero coefficients for the compositional varaibles at each step along the compCL path.

Usage

```
## S3 method for class 'compCL'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

x	fitted "compCL" object.
digits	significant digits in printout.
...	not used.

Value

a two-column matrix; the first column DF gives the number of nonzero coefficients for the compositional predictors and the second column Lam gives the corresponding lam values.

Author(s)

Zhe Sun and Kun Chen

References

Lin, W., Shi, P., Peng, R. and Li, H. (2014) *Variable selection in regression with compositional covariates*, https://academic.oup.com/biomet/article/101/4/785/1775476. *Biometrika* **101** 785-979.

See Also

compCL and coef, predict and plot methods for "compCL" object.

Examples

```
Comp_data = comp_Model(n = 50, p = 30)
Comp_fit = compCL(y = Comp_data$y, Z = Comp_data$X.comp,
                  Zc = Comp_data$Zc, intercept = Comp_data$intercept)
print(Comp_fit)
```

---

print.FuncompCGL *Print a "FuncompCGL" object.*

Description

print the number of nonzero coefficient curves for the functional compositional predictors at each lam along the FuncompCGL path.

Usage

```
## S3 method for class 'FuncompCGL'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

x	fitted FuncompCGL object.
digits	significant digits in printout.
...	not used.

Value

a two-column matrix; the first column DF gives the number of nonzero coefficients and the second column Lam gives the correspondint lam values.

Author(s)

Zhe Sun and Kun Chen

References

Sun, Z., Xu, W., Cong, X., Li G. and Chen K. (2020) *Log-contrast regression with functional compositional predictors: linking preterm infant's gut microbiome trajectories to neurobehavioral outcome*, https://arxiv.org/abs/1808.02403 *Annals of Applied Statistics*

See Also

FuncompCGL, and coef, predict and plot methods for "FuncompCGL" object.

Examples

```
df_beta = 5
p = 30
beta_C_true = matrix(0, nrow = p, ncol = df_beta)
beta_C_true[1, ] <- c(-0.5, -0.5, -0.5 , -1, -1)
beta_C_true[2, ] <- c(0.8, 0.8,  0.7,  0.6,  0.6)
beta_C_true[3, ] <- c(-0.8, -0.8 , 0.4 , 1 , 1)
beta_C_true[4, ] <- c(0.5, 0.5, -0.6  ,-0.6, -0.6)
Data <- Fcomp_Model(n = 50, p = p, m = 2, intercept = TRUE,
                    SNR = 2, sigma = 2, rho_X = 0, rho_T = 0.5, df_beta = df_beta,
                    n_T = 20, obs_spar = 1, theta.add = FALSE,
                    beta_C = as.vector(t(beta_C_true)))
m1 <- FuncompCGL(y = Data$data$y, X = Data$data$Comp ,
                 Zc = Data$data$Zc, intercept = Data$data$intercept,
                 k = df_beta)
print(m1)
```

Index