Package 'DLMRMV'

July 24, 2025

Type Package

Version 1.0.0

Title Distributed Linear Regression Models with Response Missing Variables

Depends R (>= 4.1.0)

Description As a distributed imputation strategy, the Distributed full information Multiple Imputation method is developed to impute missing response variables in distributed linear regression. The philosophy of the package is described in 'Guo' (2025) <doi:10.1038/s41598-025-93333-6>.

License Apache License (== 2.0)

RoxygenNote 7.3.2

Encoding UTF-8

Imports stats,MASS,glmnet

Date/Publication 2025-07-24 04:30:16 UTC

Config/testthat/edition 3

NeedsCompilation no

Author Guangbao Guo [aut, cre] (ORCID: <`https://orcid.org/0000-0002-4115-6218`>), Limin Song [aut]

Maintainer Guangbao Guo <`ggb11111111@163.com`>

Repository CRAN

Contents

`AVGM` *Averaged Generalized Method of Moments Imputation (AVGM)*

Description

This function performs multiple imputations on missing values in the response variable Y, using AVGMMI logic with support for grouped data. It is fully self-contained.

Usage

```
AVGM(data, M, midx = 1)
```

Arguments

`data`	A data frame where the first column is the response variable (Y), and others are predictors (X).
`M`	Number of multiple imputations.
`midx`	Integer indicating which column is the response variable (default = 1).

Value

A list containing:

`betahat`	Final averaged regression coefficient estimates.
`Yhat`	Imputed response variable with all missing values filled in.
`comm`	Completion flag (1 = success).

Examples

```
set.seed(123)
data <- data.frame(
  y = c(rnorm(50), rep(NA, 10)),
  x1 = rnorm(60),
  x2 = rnorm(60)
)
result <- AVGM(data, M = 10)
head(result$Yhat)
```

`CSLMI` *CSLMI: Consensus-based Stochastic Linear Multiple Imputation (Simplified Version)*

Description

Performs multiple imputation and parameter estimation using a consensus-based approach. The response variable is in the first column, all other columns are predictors, missing values are automatically detected, the whole dataset is treated as one block.

Usage

```
CSLMI(data, M)
```

Arguments

`data`	Dataframe with response variable in 1st column and predictors in others
`M`	Number of imputations

Value

A list containing:

`Yhat`	Imputed response values.
`betahat`	Average regression coefficients across imputations.
`comm`	Communication cost (number of messages passed).

A list containing the following components:

`Yhat`	Imputed response vector with missing values filled in.
`betahat`	Final regression coefficients.

Examples

```
set.seed(123)
data <- data.frame(
  y = c(rnorm(50), rep(NA, 10)),
  x1 = rnorm(60),
  x2 = rnorm(60)
)
result <- CSLMI(data = data, M = 10)
head(result$Yhat)
print(result$betahat)
print(result$comm)
```

DAVGMMI *Impute Missing Values in Response Variable Y Using Distributed AVG-MMI Method (With Grouping)*

Description

This function implements the Distributed Averaged Generalized Method of Moments Imputation (DAVGMMI) to fill in missing values in the response variable Y based on observed covariates X. Assumes a single group structure and does not require group size input (‘n‘).

Usage

```
DAVGMMI(data, R, M)
```

Arguments

`data`	A data frame or matrix where the first column is the response variable Y (may contain NA), and remaining columns are covariates X.
`R`	Number of simulations for stable Beta estimation.
`M`	Number of multiple imputations.

Value

A list containing:

`Yhat`	The vector of Y with missing values imputed.
`betahat`	Final averaged regression coefficient estimates used for imputation.

Examples

```
set.seed(123)
data <- data.frame(
  y = c(rnorm(50), rep(NA, 10)),
  x1 = rnorm(60),
  x2 = rnorm(60)
)
```

```
result <- DAVGMMI(data, R = 50, M = 10)
head(result$Yhat)
```

DCSLMI	*Distributed and Consensus-Based Stochastic Linear Multiple Imputation (DCSLMI)*

Description

Performs multiple imputation for missing response variables in linear regression models. This method iteratively updates parameter estimates using ordinary least squares (OLS) and generates M complete datasets by imputing missing values with different parameter draws.

Usage

```
DCSLMI(data, R = 1000, M = 20)
```

Arguments

`data`	A data frame or matrix. The first column contains the response variable ‘y‘ (which may include NA values), and the remaining columns are predictors ‘X‘.
`R`	Number of internal iterations for parameter estimation per imputation.
`M`	Number of multiple imputations to generate.

Value

A list containing:

Yhat A matrix of size n x M, where each column is a completed response vector.

betahat A matrix of size (p+1) x M, where each column contains the estimated regression coefficients.

missing_count The number of missing values in the original response variable.

Examples

```
# Simulate data with missing responses
set.seed(123)
data <- data.frame(
  y = c(rnorm(50), rep(NA, 10)),
  x1 = rnorm(60),
  x2 = rnorm(60)
)

# Perform multiple imputation
result <- DCSLMI(data, R = 500, M = 10)

# View imputed response values
```

```
head(result$Yhat)

# View coefficient estimates
apply(result$betahat, 1, mean)  # average estimates
apply(result$betahat, 1, sd)    # uncertainty across imputations
```

DERLS — *Distributed Exponentially Weighted Recursive Least Squares (DERLS)*

Description

Impute missing values in the response variable Y using distributed ERLS method. Multiple independent runs are performed to stabilize coefficient estimates. Missing values are imputed recursively and refined over multiple iterations.

Usage

```
DERLS(data, rho, lambda, R, nb)
```

Arguments

`data`	A data frame where: **First column:** Response Y (with possible NAs) **Remaining columns:** Predictors X
`rho`	Regularization parameter.
`lambda`	Forgetting factor.
`R`	Number of independent runs to stabilize estimates.
`nb`	Number of iterations per run.

Details

This function implements the Distributed Exponentially Weighted Recursive Least Squares (DERLS) method for imputing missing values in the response variable Y. The key steps include:

1. Initial imputation of missing values.
2. Recursive updates of the regression coefficients using the ERLS algorithm.
3. Multiple independent runs to stabilize the coefficient estimates.
4. Final prediction of missing values using the averaged coefficients.

The ERLS algorithm is particularly useful for online learning and adaptive filtering.

Value

A list containing:

Yhat Imputed response vector.

betahat Estimated coefficient vector.

Examples

```
set.seed(123)
n <- 60
data <- data.frame(
  Y = c(rnorm(n - 10), rep(NA, 10)),  # 50 observed + 10 missing
  X1 = rnorm(n),
  X2 = rnorm(n)
)
result <- DERLS(data, rho = 0.01, lambda = 0.95, R = 3, nb = 50)
head(result$Yhat)  # inspect imputed Y
result$betahat      # inspect estimated coefficients
```

`DERLS_InfoFilter` *Distributed Exponentially Weighted Recursive Least Squares (DERLS) using Information Filter*

Description

Impute missing values in the response variable Y using a distributed Exponentially Weighted Recursive Least Squares (DERLS) method that employs an Information Filter. Multiple independent runs are performed to stabilize coefficient estimates, and missing values are imputed recursively and refined over multiple iterations.

Usage

```
DERLS_InfoFilter(data, rho, lambda, R, nb)
```

Arguments

`data`	A data frame whose first column is the response variable Y (which may contain NAs), and the remaining columns are predictor variables X.
`rho`	Regularization parameter.
`lambda`	Forgetting factor.
`R`	Number of independent runs to stabilize estimates.
`nb`	Number of iterations per run.

Value

A list with two components:

`Yhat`	A numeric vector of length n equal to the number of rows in data. Missing values in the original Y have been imputed.
`betahat`	Numeric vector of final averaged regression coefficient estimates (length p, where p is the number of predictors).

Examples

```
set.seed(123)
n <- 60
data <- data.frame(
  Y = c(rnorm(n - 10), rep(NA, 10)),
  X1 = rnorm(n),
  X2 = rnorm(n)
)
result <- DERLS_InfoFilter(data, rho = 0.01, lambda = 0.95, R = 3, nb = 50)
head(result$Yhat)  # inspect imputed Y
result$betahat     # inspect estimated coefficients
```

DERLS_Woodbury — *Distributed Exponentially Weighted Recursive Least Squares (DERLS) using Woodbury Identity*

Description

Impute missing values in the response variable Y using the distributed ERLS method with the Woodbury Identity. Multiple independent runs are performed to stabilize coefficient estimates. Missing values are imputed recursively and refined over multiple iterations.

Usage

```
DERLS_Woodbury(data, rho, lambda, R, nb)
```

Arguments

`data`	A data frame where: **First column:** Response Y (with possible NAs) **Remaining columns:** Predictors X
`rho`	Regularization parameter.
`lambda`	Forgetting factor.
`R`	Number of independent runs to stabilize estimates.
`nb`	Number of iterations per run.

Details

This function implements the Distributed Exponentially Weighted Recursive Least Squares (DERLS) method using the Woodbury Identity for efficient updates of the covariance matrix. The key steps include:

1. Initial imputation of missing values.
2. Recursive updates of the regression coefficients using the ERLS algorithm with the Woodbury Identity.
3. Multiple independent runs to stabilize the coefficient estimates.

4. Final prediction of missing values using the averaged coefficients.

The Woodbury Identity is used to efficiently update the covariance matrix Pstar during each iteration, making the algorithm computationally efficient and suitable for large datasets.

Value

A list containing:

Yhat Imputed response vector.

betahat Estimated coefficient vector.

Examples

```
set.seed(123)
n <- 60
data <- data.frame(
  Y = c(rnorm(n - 10), rep(NA, 10)),  # 50 observed+10 missing
  X1 = rnorm(n),
  X2 = rnorm(n)
)
result <- DERLS_Woodbury(data, rho = 0.01, lambda = 0.95, R = 3, nb = 50)
head(result$Yhat)  # inspect imputed Y
result$betahat      # inspect estimated coefficients
```

`DfiMI` *Distributed Full-information Multiple Imputation (DfiMI)*

Description

Perform multiple imputation of the response variable Y via R independent runs and M stochastic imputations per run. Missing values in Y are imputed by means of (intercept-adjusted) OLS regression on the complete predictors.

Usage

```
DfiMI(data, R, M)
```

Arguments

`data`	A data frame whose first column contains the response variable Y (possibly with NAs) and whose remaining columns contain numeric predictors.
`R`	Positive integer – number of simulation runs used to stabilise the coefficient estimates.
`M`	Positive integer – number of multiple imputations drawn within each run.

Details

This function implements a distributed full-information multiple imputation (DfiMI) approach. It iteratively imputes missing values in the response variable Y using OLS regression on the complete predictors. The process is repeated R times to stabilise the coefficient estimates, and within each run, M imputations are performed to account for the uncertainty in the imputation process.

Value

A named list with components:

Yhat Numeric vector – the original Y with missing values replaced by their imputed counterparts.

betahat Numeric vector – final regression coefficients (including intercept).

Examples

```
set.seed(123)
n  <- 60
data <- data.frame(
  Y  = c(rnorm(n - 10), rep(NA, 10)),  # 50 observed + 10 missing
  X1 = rnorm(n),
  X2 = rnorm(n)
)

res <- DfiMI(data, R = 3, M = 5)
head(res$Yhat)   # inspect imputed Y
res$betahat      # inspect coefficients
```

DfiMI_lasso — *Distributed Full-information Multiple Imputation (DfiMI) using LASSO*

Description

Performs multiple imputation of the response variable Y via R independent runs and M stochastic imputations per run. Missing Y values are imputed using LASSO regression on predictors.

Usage

```
DfiMI_lasso(data, R, M)
```

Arguments

data	A `data.frame` where: **First column:** Response Y (may contain NA) **Remaining columns:** Numeric predictors
R	Positive integer – number of simulation runs for stable coefficient estimation.
M	Positive integer – number of multiple imputations per run.

Details

This function extends the Distributed Full-information Multiple Imputation (DfiMI) approach by using LASSO regression for imputing missing values in the response variable Y. LASSO regression is particularly useful for high-dimensional predictor spaces and can handle multicollinearity among predictors. The function performs the following steps:

1. Initialize missing values in Y.
2. Fit LASSO regression models on complete cases.
3. Average coefficients across multiple imputations and runs.
4. Predict missing values using the final averaged coefficients.

The function requires the glmnet package for LASSO regression.

Value

A named list containing:

Yhat Numeric vector – original Y values with missing values replaced by imputations.

betahat Numeric vector – final regression coefficients.

Examples

```
set.seed(123)
data <- data.frame(
  Y = c(rnorm(50), rep(NA, 10)),  # 50 observed + 10 missing
  X1 = rnorm(60),
  X2 = rnorm(60)
)
res <- DfiMI_lasso(data, R = 3, M = 5)
head(res$Yhat)
```

`DMCEM` *Distributed Monte Carlo Expectation-Maximization (DMCEM) Algorithm*

Description

Implements a distributed version of the Monte Carlo EM algorithm for handling missing response variables in linear regression models. By running multiple simulations and averaging the results, it provides more stable parameter estimates compared to standard EM.

Usage

```
DMCEM(data, R = 50, tol = 0.01, nb = 50)
```

Arguments

`data`	A data frame where the first column is the response variable (with missing values) and subsequent columns are predictors.
`R`	Integer specifying the number of Monte Carlo simulations. Larger values improve stability but increase computation time (default = 50).
`tol`	Numeric value indicating the convergence tolerance. The algorithm stops when the change in coefficients between iterations is below this threshold (default = 0.01).
`nb`	Integer specifying the maximum number of iterations per simulation. Prevents infinite loops if convergence is not achieved (default = 50).

Details

The DMCEM algorithm works by:

1. Splitting data into observed and missing response subsets.
2. Running multiple MCEM simulations with random imputations.
3. Averaging results across simulations to reduce variance.
4. Using robust matrix inversion to handle near-singular designs.

This approach is particularly useful for datasets with a large proportion of missing responses or high variability in the data.

Value

A list containing:

`Yhat` A vector of imputed response values with missing data filled in.

`betahat` A vector of final regression coefficients, averaged across simulations.

Examples

```
# Generate data with 20% missing responses
set.seed(123)
data <- data.frame(
  Y = c(rnorm(80), rep(NA, 20)),
  X1 = rnorm(100),
  X2 = runif(100)
)

# Run DMCEM with 50 simulations
result <- DMCEM(data, R = 50, tol = 0.001, nb = 100)

# View imputed values and coefficients
head(result$Yhat)
result$betahat

# Check convergence and variance
result$converged_ratio
result$sigma2
```

EMRE *EM Algorithm for Linear Regression with Missing Data*

Description

EM Algorithm for Linear Regression with Missing Data

Usage

```
EMRE(data, d = 1, tol = 1e-06, nb = 100, niter = 1)
```

Arguments

`data`	Dataframe with first column as response (Y) and others as predictors (X)
`d`	Initial convergence threshold (default=1)
`tol`	Termination tolerance (default=1e-6)
`nb`	Maximum iterations (default=100)
`niter`	Starting iteration counter (default=1)

Value

List containing:

`Yhat`	Imputed response vector
`betahat`	Estimated coefficients

Examples

```
# Generate data with 20% missing Y values
set.seed(123)
data <- data.frame(Y=c(rnorm(80),rep(NA,20)), X1=rnorm(100), X2=rnorm(100))

# Run EM algorithm
result <- EMRE(data, d=1, tol=1e-5, nb=50)
print(result$betahat) # View coefficients
```

`ERLS` *Exponentially Weighted Recursive Least Squares with Missing Value Imputation*

Description

Exponentially Weighted Recursive Least Squares with Missing Value Imputation

Usage

```
ERLS(data, rho = 0.01, lambda = 0.95, nb = 100, niter = 1)
```

Arguments

`data`	Linear regression dataset (1st column as Y, others as X)
`rho`	Regularization parameter
`lambda`	Forgetting factor
`nb`	Maximum iterations
`niter`	Initial iteration count (typically 1)

Value

List containing:

`Yhat`	Imputed response vector
`betahat`	Estimated coefficients

Examples

```
set.seed(123)
data <- data.frame(
  y = c(rnorm(50), rep(NA, 10)),
  x1 = rnorm(60),
  x2 = rnorm(60)
)
result <- ERLS(data, rho = 0.01, lambda = 0.95, nb = 100, niter = 1)
head(result$Yhat)
```

`fiMI` *fiMI: Predict Missing Response Variables using Multiple Imputation*

Description

This function predicts missing response variables in a linear regression dataset using multiple imputation. It leverages the `FimIMI` function to perform multiple runs of improved multiple imputation and averages the regression coefficients to predict the missing response values.

Usage

```
fiMI(data, R, n, M)
```

Arguments

`data`	`data.frame` containing the linear regression model dataset with missing response variables.
`R`	Number of runs for multiple imputation.
`n`	Number of rows in the dataset.
`M`	Number of multiple imputations per run.

Details

This function assumes that the first column of `data` is the response variable and the remaining columns are the independent variables. The function uses the `FimIMI` function to perform multiple runs of improved multiple imputation and averages the regression coefficients to predict the missing response values.

Value

A list containing:

`Yhat`	Predicted response values with missing values imputed.

Examples

```
# Example data
set.seed(123)
n <- 1000  # Number of rows
p <- 5  # Number of independent variables
data <- data.frame(Y = rnorm(n), X1 = rnorm(n), X2 = rnorm(n))
data[sample(n, 100), 1] <- NA  # Introduce missing response values

# Call fiMI function
result <- fiMI(data, R = 10, n = n, M = 20)

# View results
print(result$Yhat)  # Predicted response values
```

FimIMI — *FimIMI: Multiple Runs of Improved Multiple Imputation (IMI)*

Description

This function performs multiple runs of the Improved Multiple Imputation (IMI) estimation and collects the results. It is designed to facilitate batch processing and repeated runs of IMI.

Usage

```
FimIMI(d, R, n, M, batch = 0)
```

Arguments

`d`	The data structure.
`R`	Number of runs to perform.
`n`	Vector of sample sizes for each group.
`M`	Number of multiple imputations per run.
`batch`	Batch number (default is 0). This can be used to distinguish different batches of runs.

Details

This function assumes that the data structure `d` is properly defined and contains the necessary information. The function repeatedly calls the `IMI` function and collects the regression coefficients and indicator variables.

Value

A list containing:

`R`	Vector of run numbers.
`Beta`	Matrix of regression coefficients for each run.
`comm`	Vector of indicator variables for each run.

Examples

```
# Example data
set.seed(123)
n <- c(300, 300, 400)  # Sample sizes for each group
p <- 5  # Number of independent variables
d <- list(p = p, Y = rnorm(sum(n)), X0 = matrix(rnorm(sum(n) * p), ncol = p))

# Call FimIMI function
result <- FimIMI(d = d, R = 10, n = n, M = 20, batch = 1)

# View results
```

```
print(result$Beta)  # Regression coefficients for each run
```

GMD *Generate Missing Data function*

Description

This function generates missing data in a specified column of a data frame according to a given missing ratio.

Usage

```
GMD(data, ratio)
```

Arguments

`data`	A data frame containing the linear regression model dataset
`ratio`	The missing ratio (e.g., 0.5 means 1/2 of data will be made missing)

Value

`data0`	A modified version of ‘data‘ with missing values inserted.

Examples

```
set.seed(123) # for reproducibility
data <- data.frame(x = 1:10, y = rnorm(10))
modified_data <- GMD(data, ratio = 0.5)
summary(modified_data)
```

IMI *Improved Multiple Imputation (IMI) Estimation*

Description

This function performs Improved Multiple Imputation (IMI) estimation for grouped data with missing values. It iteratively imputes missing values using the `LS` function and estimates regression coefficients using the `PPLS` function. The final regression coefficients are averaged across multiple imputations.

Usage

```
IMI(d, M, midx, n)
```

Arguments

`d`	`data.frame` containing the dependent variable (`Y`) and independent variables (`X`).
`M`	Number of multiple imputations to perform.
`midx`	Column indices of the missing variables in `d`.
`n`	Vector of sample sizes for each group.

Details

The function assumes the data is grouped and contains missing values in specified columns (`midx`). It uses the `LS` function to impute missing values and the `PPLS` function to estimate regression coefficients. The process is repeated `M` times, and the final regression coefficients are averaged.

Value

A list containing the following elements:

`betahat`	Average regression coefficients across all imputations.
`comm`	Indicator variable (0 for single group, 1 for multiple groups).

Examples

```
# Example data

set.seed(123)
n <- c(300, 300, 400)  # Sample sizes for each group
p <- 5  # Number of independent variables
Y <- rnorm(sum(n))  # Dependent variable
X0 <- matrix(rnorm(sum(n) * p), ncol = p)  # Independent variables matrix
d <- list(p = p, Y = Y, X0 = X0)  # Data list
d$all <- cbind(Y, X0)
# Indices of missing variables (assuming some variables are missing)
midx <- c(2, 3)  # For example, the second and third variables are missing
# Call IMI function
result <- IMI(d, M = 5, midx = midx, n = n)
# View results
print(result$betahat)  # Average regression coefficients
```

`LS` *Least Squares Estimation for Grouped Data with Ridge Regularization*

Description

This function implements the least squares estimation for grouped data, supporting ridge regression regularization. It can handle missing data and returns regression coefficients and the sum of squared residuals for each group.

Usage

```
LS(d, yidx, Xidx, n, lam = 0.005)
```

Arguments

`d`	A data frame containing dependent and independent variables.
`yidx`	The column index of the dependent variable.
`Xidx`	The column indices of the independent variables.
`n`	A vector of starting indices for the groups.
`lam`	Regularization parameter for ridge regression, default is 0.005.

Value

A list containing the following elements:

`beta`	A matrix of regression coefficients for each group.
`SSE`	The sum of squared residuals for each group.
`df`	The sample size for each group.
`gram`	The Gram matrix for each group.
`cgram`	The Cholesky decomposition result for each group.
`comm`	An unused variable (reserved for future expansion).

Examples

```
# Example data
set.seed(123)
n <- 1000
p <- 5
d <- list(all = cbind(rnorm(n), matrix(rnorm(n*p), ncol=p)))

# Call the LS function
result <- LS(d, yidx = 1, Xidx = 2:(p + 1), n = c(1, 300, 600, 1000))

# View the results
print(result$beta)  # Regression coefficients
print(result$SSE)   # Sum of squared residuals
```

MCEM *MCEM Algorithm for Missing Response Variables*

Description

Implements the Monte Carlo EM algorithm for handling missing response data in linear regression models.

Usage

```
MCEM(data, d = 5, tol = 0.01, nb = 50)
```

Arguments

`data`	A data frame with the response variable in the first column and predictors in the remaining columns.
`d`	Initial convergence threshold. Defaults to 5.
`tol`	Termination tolerance. Defaults to 0.01.
`nb`	Maximum number of iterations. Defaults to 50.

Details

This function implements the Monte Carlo Expectation-Maximization (MCEM) algorithm to handle missing response variables in linear regression models. The algorithm iteratively imputes missing responses and updates regression coefficients until convergence.

Value

A list containing the following components:

`Yhat`	Imputed response vector with missing values filled in.
`betahat`	Final regression coefficients.
`iterations`	Number of iterations performed.

Examples

```
# Create dataset with 20% missing responses
set.seed(123)
data <- data.frame(
  Y = c(rnorm(80), rep(NA, 20)),
  X1 = rnorm(100),
  X2 = runif(100)
)
result <- MCEM(data, d = 5, tol = 0.001, nb = 100)
print(result$Yhat)  # Imputed response vector
print(result$betahat)  # Final regression coefficients
print(result$iterations)  # Number of iterations performed
```

PMMI *Predictive Mean Matching with Multiple Imputation*

Description

Implements PMM algorithm for handling missing data in linear regression models. Uses chained equations approach to generate multiple imputed datasets and pools results using Rubin's rules.

Usage

```
PMMI(data, k = 5, m = 5)
```

Arguments

`data`	Dataframe with response variable in 1st column and predictors in others
`k`	Number of nearest neighbors for matching (default=5)
`m`	Number of imputations (default=5)

Value

List containing:

`Y`	Original response vector with NAs
`Yhat`	Final imputed response vector (averaged across imputations)
`betahat`	Pooled regression coefficients
`imputations`	List of m completed datasets
`m`	Number of imputations performed
`k`	Number of neighbors used

Examples

```
# Create dataset with 30% missing values
data <- data.frame(Y=c(rnorm(70),rep(NA,30)), X1=rnorm(100))
results <- PMMI(data, k=5, m=5)
```

PPLS *Penalized Partial Least Squares (PPLS) Estimation*

Description

This function performs Penalized Partial Least Squares (PPLS) estimation for grouped data. It supports ridge regression regularization and handles missing data by excluding incomplete cases. The function returns regression coefficients, residual sum of squares, and other diagnostic information.

Usage

```
PPLS(d, yidx, Xidx, n, lam = 0.005)
```

Arguments

`d`	Containing the dependent and independent variables.
`yidx`	Column index of the dependent variable in `d`.
`Xidx`	Column indices of the independent variables in `d`.
`n`	Vector of sample sizes for each group.
`lam`	Regularization parameter for ridge regression (default is 0.005).

Details

This function assumes that the data is grouped and that the sample sizes for each group are provided. It excludes cases with missing values in the dependent or independent variables. The function uses Cholesky decomposition to solve the regularized least squares problem.

Value

A list containing the following elements:

`beta`	Regression coefficients.
`SSE`	Residual sum of squares.
`df`	Number of complete cases used in the estimation.
`gram`	Gram matrix ($X^T X + \lambda I$).
`cgram`	Cholesky decomposition of the Gram matrix.
`comm`	Indicator variable (0 for single group, 1 for multiple groups).

Examples

```
# Example data
set.seed(123)
n_total <- 1000
p <- 5
n_groups <- c(300, 300, 400)
d <- list(all = cbind(rnorm(n_total), matrix(rnorm(n_total*p), ncol=p)),p = p)
```

```
# Call PPLS function
result <- PPLS(d, yidx=1, Xidx=2:(p+1), n=n_groups)

# View results
print(result$beta)  # Regression coefficients
print(result$SSE)   # Residual sum of squares
```

Index