Package 'DLFM'

January 26, 2026

Type Package

Version 0.2.2

Title Distributed Laplace Factor Model

Description Distributed estimation method is based on a Laplace factor model to solve the estimates of load and specific variance. The philosophy of the package is described in Guangbao Guo. (2022). <doi:10.1007/s00180-022-01270-z>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports MASS, LaplacesDemon, matrixcalc, stats

Depends R (>= 3.5.0)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

BuildManual yes

NeedsCompilation no

Language en-US

 # Date/Publication 2025-01-25

Repository CRAN

Author Guangbao Guo [aut, cre],
Siqi Liu [aut]

Maintainer Guangbao Guo <ggb11111111@163.com>

Date/Publication 2026-01-26 09:50:29 UTC

Contents

`Dfactor.tests` *Distributed Factor Model Testing with Wald, GRS, PY tests and FDR control*

Description

Performs comprehensive factor model testing in distributed environment across multiple nodes, including joint tests (Wald, GRS, PY), individual asset t-tests, and False Discovery Rate control.

Usage

```
Dfactor.tests(ret, fac, n1, K, q.fdr = 0.05)
```

Arguments

`ret`	A T × N matrix representing the excess returns of N assets at T time points.
`fac`	A T × K matrix representing the returns of K factors at T time points.
`n1`	The number of assets allocated to each node
`K`	The number of nodes
`q.fdr`	The significance level for FDR (False Discovery Rate) testing, defaulting to 5%.

Value

A list containing the following components:

`alpha_list`	List of alpha vectors from each node
`tstat_list`	List of t-statistics from each node
`pval_list`	List of p-values from each node
`Wald_list`	List of Wald test statistics from each node
`p_Wald_list`	List of p-values for Wald tests from each node

GRS_list	List of GRS test statistics from each node
p_GRS_list	List of p-values for GRS tests from each node
PY_list	List of Pesaran and Yamagata test statistics from each node
p_PY_list	List of p-values for PY tests from each node
reject_fdr_list	List of logical vectors indicating significant assets after FDR correction from each node
power_proxy_list	List of number of significant assets after FDR correction from each node
combined_alpha	Combined alpha vector from all nodes
combined_pval	Combined p-value vector from all nodes
combined_reject_fdr	Combined FDR rejection vector from all nodes
total_power_proxy	Total number of significant assets across all nodes after FDR correction

Examples

```
set.seed(42)
T <- 120
N <- 100  # Larger dataset for distributed testing
K_factors <- 3
fac <- matrix(rnorm(T * K_factors), T, K_factors)
beta <- matrix(rnorm(N * K_factors), N, K_factors)
alpha <- rep(0, N)
alpha[1:10] <- 0.4 / 100  # 10 non-zero alphas
eps <- matrix(rnorm(T * N, sd = 0.02), T, N)
ret <- alpha + fac %*% t(beta) + eps

# Distributed testing with 4 nodes, each handling 25 assets
results <- Dfactor.tests(ret, fac, n1 = 25, K = 4, q.fdr = 0.05)

# View combined results
cat("Total significant assets after FDR:", results$total_power_proxy, "\n")
cat("Combined results across all nodes:\n")
print(summary(results$combined_alpha))
```

DGulPC *Distributed general unilateral loading principal component*

Description

Distributed general unilateral loading principal component

Usage

```
DGulPC(data, m, n1, K)
```

Arguments

data	is a total data set
m	is the number of principal component
n1	is the length of each data subset
K	is the number of nodes

Value

AU1,AU2,DU3,Shat

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
DGulPC(data,m=3,n1=128,K=2)
```

DIPC *Distributed Incremental Principal Component Analysis (DIPC)*

Description

Apply IPC in a distributed manner across K nodes.

Usage

```
DIPC(data, m, eta, K)
```

Arguments

data	Matrix of input data (n × p).
m	Number of principal components.
eta	Proportion of initial batch to total data within each node.
K	Number of nodes (distributed splits).

Value

List with per-node results and aggregated averages.

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
results <- DIPC(data, m, eta=0.8, K=5)
```

DPC *Distributed principal component*

Description

Distributed principal component

Usage

```
DPC(data, m, n1, K)
```

Arguments

data	is a total data set
m	is the number of principal component
n1	is the length of each data subset
K	is the number of nodes

Value

Ahat,Dhat,Sigmahathat

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
DPC(data,m=3,n1=128,K=2)
```

DPPC *Distributed projection principal component*

Description

Distributed projection principal component

Usage

```
DPPC(data, m, n1, K)
```

Arguments

data	is a total data set
m	is the number of principal component
n1	is the length of each data subset
K	is the number of nodes

Value

Apro,pro,Sigmahathatpro

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
```

```
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
DPPC(data,m=3,n1=128,K=2)
```

DSAPC *The distributed stochastic approximation principal component for handling online data sets with highly correlated data across multiple nodes.*

Description

The distributed stochastic approximation principal component for handling online data sets with highly correlated data across multiple nodes.

Usage

```
DSAPC(data, m, eta, n1, K)
```

Arguments

data	is a highly correlated online data set
m	is the number of principal component
eta	is the proportion of online data to total data
n1	is the length of each data subset
K	is the number of nodes

Value

Asa, Dsa (lists containing results from each node)

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
```

```
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
DSAPC(data=data, m=3, eta=0.8, n1=128, K=2)
```

`factor.tests` *Factor Model Testing with Wald, GRS, PY tests and FDR control*

Description

Performs comprehensive factor model testing including joint tests (Wald, GRS, PY), individual asset t-tests, and False Discovery Rate control.

Usage

```
factor.tests(ret, fac, q.fdr = 0.05)
```

Arguments

`ret`	A T × N matrix representing the excess returns of N assets at T time points.
`fac`	A T × K matrix representing the returns of K factors at T time points.
`q.fdr`	The significance level for FDR (False Discovery Rate) testing, defaulting to 5%.

Value

A list containing the following components:

`alpha`	N-vector of estimated alphas for each asset
`tstat`	N-vector of t-statistics for testing individual alphas
`pval`	N-vector of p-values for individual alpha tests
`Wald`	Wald test statistic for joint alpha significance
`p_Wald`	p-value for Wald test
`GRS`	GRS test statistic (finite-sample F-test)
`p_GRS`	p-value for GRS test
`PY`	Pesaran and Yamagata test statistic
`p_PY`	p-value for PY test
`reject_fdr`	Logical vector indicating which assets have significant alphas after FDR correction
`fdr_p`	Adjusted p-values using Benjamini-Hochberg procedure
`power_proxy`	Number of significant assets after FDR correction

Examples

```
set.seed(42)
T <- 120
N <- 25
K <- 3
fac <- matrix(rnorm(T * K), T, K)
beta <- matrix(rnorm(N * K), N, K)
alpha <- rep(0, N)
alpha[1:3] <- 0.4 / 100  # 3 non-zero alphas
eps <- matrix(rnorm(T * N, sd = 0.02), T, N)
ret <- alpha + fac %*% t(beta) + eps
results <- factor.tests(ret, fac, q.fdr = 0.05)

# View results
cat("Wald test p-value:", results$p_Wald, "\n")
cat("GRS test p-value:", results$p_GRS, "\n")
cat("PY test p-value:", results$p_PY, "\n")
cat("Significant assets after FDR:", results$power_proxy, "\n")
```

FanPC *Apply the FanPC method to the Laplace factor model*

Description

This function performs Factor Analysis via Principal Component (FanPC) on a given data set. It calculates the estimated factor loading matrix (AF), specific variance matrix (DF), and the mean squared errors.

Usage

```
FanPC(data, m)
```

Arguments

data	A matrix of input data.
m	is the number of principal component

Value

AF,DF,SigmahatF

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
```

```
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
results <- FanPC(data, m)
print(results)
```

`Ftest` *Apply the Farmtest method to the Laplace factor model*

Description

This function simulates data from a Lapalce factor model and applies the FarmTest for multiple hypothesis testing. It calculates the false discovery rate (FDR) and power of the test.

Usage

```
Ftest(
  data,
  p1,
  alpha = 0.05,
  K = -1,
  alternative = c("two.sided", "less", "greater")
)
```

Arguments

`data`	A matrix or data frame of simulated or observed data from a Laplace factor model.
`p1`	The number or proportion of non-zero hypotheses.
`alpha`	The significance level for controlling the false discovery rate (default: 0.05).
`K`	The number of factors to estimate (default: -1, meaning auto-detect).
`alternative`	The alternative hypothesis: "two.sided", "less", or "greater" (default: "two.sided").

Value

A list containing the following elements:

`FDR`	The false discovery rate, which is the proportion of false positives among all discoveries (rejected hypotheses).
`Power`	The statistical power of the test, which is the probability of correctly rejecting a false null hypothesis.

PValues	A vector of p-values associated with each hypothesis test.
RejectedHypotheses	The total number of hypotheses that were rejected by the FarmTest.
reject	Indices of rejected hypotheses.
means	Estimated means.

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
p1=40
results <- Ftest(data, p1)
print(results$FDR)
print(results$Power)
```

GulPC *General unilateral loading principal component*

Description

General unilateral loading principal component

Usage

```
GulPC(data, m)
```

Arguments

data	is a total data set
m	is the number of first layer principal component

Value

AU1,AU2,DU3,SigmaUhat

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
GulPC(data=data,m=5)
```

IPC *Incremental principal component method*

Description

The incremental principal component can handle online data sets with highly correlated.

Usage

```
IPC(data, m, eta)
```

Arguments

data	is a highly correlated online data set
m	is the number of principal component
eta	is the proportion of online data to total data

Value

Ai,Di

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
```

```
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
IPC(data=data,m=3,eta=0.8)
```

LFM *Generate Laplace factor models*

Description

The function is to generate Laplace factor model data. The function supports various distribution types for generating the data, including: - 'truncated_laplace': Truncated Laplace distribution - 'log_laplace': Univariate Symmetric Log-Laplace distribution - 'Asymmetric Log_Laplace': Log-Laplace distribution - 'Skew-Laplace': Skew-Laplace distribution

Usage

```
LFM(n, p, m, distribution_type)
```

Arguments

n	An integer specifying the sample size.
p	An integer specifying the sample dimensionality or the number of variables.
m	An integer specifying the number of factors in the model.
distribution_type	A character string indicating the type of distribution to use for generating the data.

Value

A list containing the following elements:

data	A numeric matrix of the generated data.
A	A numeric matrix representing the factor loadings.
D	A numeric matrix representing the uniquenesses, which is a diagonal matrix.

Examples

```
n <- 1000
p <- 10
m <- 5
sigma1 <- 1
sigma2 <- matrix(c(1,0.7,0.7,1), 2, 2)
distribution_type <- "Asymmetric Log_Laplace"
results <- LFM(n, p, m, distribution_type)
print(results)
```

`online_sir_lfm` *Online Sufficient Dimension Reduction for Laplace Factor Model (LFM)*

Description

Implements an online SIR algorithm tailored for LFM data, using a proxy response constructed from the current subspace estimate and robust updates to handle heavy-tailed noise. The algorithm supports two optimization methods: gradient-based updates and perturbation-based updates.

Usage

```
online_sir_lfm(
  X,
  K_true = NULL,
  K_max = NULL,
  c_robust = 1.345,
  eta = "auto",
  method = "gradient",
  verbose = FALSE
)
```

Arguments

`X`	A matrix or data stream of size n x p (rows = observations, cols = features). Can be processed row-by-row in streaming setting.
`K_true`	Optional true dimension (for monitoring). If NULL, will estimate online via BIC-like criterion.
`K_max`	Maximum candidate dimension for online selection (default = min(10, ncol(X))).
`c_robust`	Robustness scale for tanh transformation (default = 1.345, approx. 0.95 efficiency for Gaussian).
`eta`	Learning rate schedule: either a function of t, or "auto" for 1/t.
`method`	Optimization method: "gradient" for gradient-based updates with learning rate, or "perturbation" for direct eigenvector computation of the moment matrix (default = "gradient").
`verbose`	Logical; if TRUE, prints progress and estimated K at each step.

Value

A list with:

`B_hat`	Final estimated basis matrix (p x K_est)
`K_est`	Estimated structural dimension
`B_path`	List of B estimates over time (optional, for debugging)
`loss`	Reconstruction loss trace (optional)
`method_used`	The optimization method actually used

Examples

```
set.seed(123)
n <- 500; p <- 20; m <- 3
B_true <- qr.Q(qr(matrix(rnorm(p * m), p, m)))
f <- matrix(rnorm(n * m), n, m)
eps <- matrix(rexp(n * p, rate = 1) - 1, n, p) # Asymmetric Laplace-like noise
X <- f %*% t(B_true) + eps

# Using gradient method (default)
out_grad <- online_sir_lfm(X, K_true = m, verbose = TRUE)

# Using perturbation method
out_pert <- online_sir_lfm(X, K_true = m, method = "perturbation", verbose = TRUE)
```

`osdr_lfm` *Online Sufficient Dimension Reduction for Laplace Factor Models (OSDR-LFM)*

Description

Implements an online SIR-based sufficient dimension reduction method tailored for Laplace Factor Models (LFM) with symmetric, asymmetric, or skewed error structures. Supports distributed deployment via local updates and global aggregation.

Usage

```
osdr_lfm(
  X,
  Y = NULL,
  laplace_type = c("symmetric", "asymmetric", "skewed"),
  K_max = NULL,
  H = NULL,
  method_svd = c("gradient", "perturbation"),
  is_distributed = FALSE,
  node_id = 1,
  sync_interval = 50,
  verbose = FALSE
)
```

Arguments

`X`	numeric matrix (n x p), observations in rows.
`Y`	optional numeric vector (n) of proxy responses (e.g., factor scores). If NULL, uses norm of projection as proxy (unsupervised LFM mode).
`laplace_type`	character; one of "symmetric", "asymmetric", or "skewed".
`K_max`	integer; maximum candidate dimension (default = min(10, p)).

H	integer; number of slices for SIR (default = max(5, floor(sqrt(n)))).
method_svd	character; "perturbation" or "gradient" (default = "gradient").
is_distributed	logical; if TRUE, simulate distributed node behavior.
node_id	integer; node identifier (only used if is_distributed = TRUE).
sync_interval	integer; how often to "aggregate" in distributed mode (ignored if not distributed).
verbose	logical; print progress.

Value

list with B_hat (p x K_est), K_est, lambda_trace, and (if distributed) local_B.

Examples

```
set.seed(42)
n <- 600; p <- 30; m <- 4
A <- qr.Q(qr(matrix(rnorm(p * m), p, m)))
F <- matrix(rnorm(n * m), n, m)
eps <- matrix(rexp(n * p) - rexp(n * p), n, p)
X <- F %*% t(A) + eps

out <- osdr_lfm(X, laplace_type = "asymmetric", K_max = 6, verbose = TRUE)
cat("Estimated K:", out$K_est, "\n")
```

PC *Principal component*

Description

Principal component

Usage

```
PC(data, m)
```

Arguments

data	is a total data set
m	is the number of principal component

Value

Ahat, Dhat, Sigmahat

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
PC(data,m=5)
```

PPC *Projection principal component*

Description

Projection principal component

Usage

```
PPC(data, m)
```

Arguments

data	is a total data set
m	is the number of principal component

Value

Apro, Dpro, Sigmahatpro

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
```

```
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
PPC(data=data,m=5)
```

SAPC *The stochastic approximation principal component can handle online data sets with highly correlated.*

Description

The stochastic approximation principal component can handle online data sets with highly correlated.

Usage

```
SAPC(data, m, eta)
```

Arguments

data	is a highly correlated online data set
m	is the number of principal component
eta	is the proportion of online data to total data

Value

Asa,Dsa

Examples

```
library(LaplacesDemon)
library(MASS)
n=1000
p=10
m=5
mu=t(matrix(rep(runif(p,0,1000),n),p,n))
mu0=as.matrix(runif(m,0))
sigma0=diag(runif(m,1))
F=matrix(mvrnorm(n,mu0,sigma0),nrow=n)
A=matrix(runif(p*m,-1,1),nrow=p)
lanor <- rlaplace(n*p,0,1)
epsilon=matrix(lanor,nrow=n)
D=diag(t(epsilon)%*%epsilon)
data=mu+F%*%t(A)+epsilon
SAPC(data=data,m=3,eta=0.8)
```

Index