Package 'GulFM'

December 17, 2025

Type Package

Title General Unilateral Load Estimator for Two-Layer Latent Factor Models

Version 0.2.0

Description Implements general unilateral loading estimator for two-layer latent factor models with smooth, element-wise factor transformations. We provide data simulation, loading estimation,finite-sample error bounds, and diagnostic tools for zero-mean and sub-Gaussian assumptions. A unified interface is given for evaluating estimation accuracy and cosine similarity. The philosophy of the package is described in Guo G. (2026) <doi:10.1016/j.apm.2025.116280>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

Imports MASS, matrixStats

Suggests testthat (>= 3.0.0), ggplot2

NeedsCompilation no

Language en-US

Author Guangbao Guo [aut, cre]

Maintainer Guangbao Guo <ggb11111111@163.com>

Repository CRAN

Date/Publication 2025-12-17 13:20:02 UTC

Contents

estimate_gul_loadings 2
generate_gfm_data 3
gul_simulation 3
g_fun 4
g_theorem 5
ionosphere 5

loading_metrics 6
verify_mean 7
verify_subgaussian 8
wholesale 8
Wine 9

Index **11**

`estimate_gul_loadings` *General unilateral load Estimator*

Description

General unilateral load Estimator

Usage

```
estimate_gul_loadings(X, m)
```

Arguments

`X`	n *p data matrix (already centred and scaled if desired).
`m`	number of latent factors (both layers).

Details

Step 1: PCA on X to get hat_A1 Step 2: Regress X on hat_A1 to get hat_gF1 Step 3: PCA on hat_gF1 to get hat_A2 Step 4: hat_Ag = hat_A1

Value

A list with hat_A1 : p * m 1st-layer loadings hat_A2 : m * m 2nd-layer loadings hat_Ag : p * m overall loadings Sigma1 : p * p sample cov(X) (for diagnostics) Sigma2 : m * m sample cov(hat_gF1) hat_gF1 : n * m estimated transformed latent factors eig1 : eigen-values of Sigma1 eig2 : eigen-values of Sigma2

Examples

```
dat  <- generate_gfm_data(500, 50, 5, tanh, seed = 1)
est  <- estimate_gul_loadings(dat$X, m = 5)
err  <- sqrt(mean((est$hat_Ag - dat$Ag)^2))  # overall RMSE
```

generate_gfm_data — *Generate general factor model with smooth latent transformation*

Description

Generate general factor model with smooth latent transformation

Usage

```
generate_gfm_data(n, p, m, g_fun, seed = 1, sigma_V = 0.1)
```

Arguments

n	Integer: sample size.
p	Integer: number of observed variables.
m	Integer: number of latent factors (both layers).
g_fun	Function: smooth, element-wise transformation applied to latent factors. Must be vectorised, e.g. 'sin', 'tanh', 'scale'.
seed	1.
sigma_V	Numeric: standard deviation of the idiosyncratic noise (default 0.1 => Var = 0.01).

Value

List with components X : n * p matrix of standardised observations. A1 : p * m first-layer loading matrix. A2 : m * m second-layer loading matrix. Ag : p * m overall loading matrix (Ag = A1 F1 : n * m latent factors (before transformation). gF1: n * m latent factors (after transformation). V1 : n * p noise matrix (for diagnostics).

Examples

```
dat <- generate_gfm_data(200, 50, 5, g_fun = tanh)
```

gul_simulation — *Single-replication GUL simulation*

Description

Generates one synthetic data set, estimates loadings with the GUL, and evaluates estimation accuracy.

Usage

```
gul_simulation(n, p, m, g_fun)
```

Arguments

- `n` Integer: sample size.
- `p` Integer: number of observed variables.
- `m` Integer: number of latent factors (both layers).
- `g_fun` Function: element-wise, smooth transformation applied to the latent factors (e.g. ‘tanh‘, ‘sin‘).

Value

Named numeric vector with components error_F : Frobenius norm ||hat(Ag) - Ag||_F

Examples

```
gul_simulation(200, 50, 5, g_fun = tanh)
```

`g_fun` *Smooth link functions compliant with Theorems 9&10*

Description

Returns a vectorised map $g(\cdot)$ and its exact Lipschitz constant L_g for three increasingly nonlinear choices.

Usage

```
g_fun(type = c("linear", "weak_nonlinear", "strong_nonlinear"))
```

Arguments

- `type` Character string selecting the map: `"linear"`, `"weak_nonlinear"`, or `"strong_nonlinear"`.

Value

Named list with components

- `g_fun` vectorised function $g(\cdot)$
- `L_g` scalar Lipschitz constant of g

Examples

```
## pick a link with L_g = 1
tmp  <- g_fun("linear")
dat  <- generate_gfm_data(n = 500, p = 200, m = 5, g_fun = tmp$g_fun)
est  <- estimate_gul_loadings(dat$X, m = 5)
err  <- norm(est$hat_Ag - dat$Ag, "F")
sprintf("F-error (L_g = %d) = %.3f", tmp$L_g, err)
```

g_theorem — *Simulation wrapper for Theorems 9 & 10*

Description

One Monte-Carlo replicate; returns empirical error, exceedance indicator, theoretical bounds, and assumption-check flags.

Usage

```
g_theorem(n, p, m, g_type, epsilon, zero_tol = 0.02)
```

Arguments

n	sample size
p	number of observed variables
m	number of latent factors
g_type	character: "linear", "weak_nonlinear", "strong_nonlinear"
epsilon	error threshold
zero_tol	zero-mean tolerance (default 0.02)

Value

one-row data-frame

Examples

```
df <- g_theorem(500, 200, 5, "linear", 0.6)
```

ionosphere — *ionosphere Data*

Description

This dataset contains radar returns from the ionosphere, collected by a system in Goose Bay, Labrador. The dataset is used for classifying radar returns as 'good' or 'bad' based on the presence of structure in the ionosphere.

Usage

```
data(ionosphere)
```

Format

A data frame with multiple rows and 35 columns representing different features related to radar returns.

- `Attribute1`: Continuous feature.
- `Attribute2`: Continuous feature.
- `Attribute3`: Continuous feature.
- `Attribute4`: Continuous feature.
- `Attribute5`: Continuous feature.
- `Attribute6`: Continuous feature.
- `Attribute7`: Continuous feature.
- `Attribute8`: Continuous feature.
- `Attribute9`: Continuous feature.
- `Attribute10`: Continuous feature.
- `...`: Additional continuous features (up to Attribute34).
- `Class`: Binary classification target ('good' or 'bad').

Examples

```
# Load the dataset
data(ionosphere)

# Print the first few rows of the dataset
print(head(ionosphere))
```

`loading_metrics` *Multi-metric evaluation of factor loading matrix estimation error*

Description

Multi-metric evaluation of factor loading matrix estimation error

Usage

```
loading_metrics(A_true, A_hat)
```

Arguments

`A_true`	True loading matrix (p x m)
`A_hat`	Estimated loading matrix (p x m)

Value

data.frame with MSE, RMSE, MAE, MaxDev, and Cosine similarity

Examples

```
## simulated example
p <- 100; m <- 5
Ag_true  <- matrix(rnorm(p*m), p, m)
Ag_hat   <- Ag_true + matrix(rnorm(p*m, 0, 0.1), p, m)
metrics  <- loading_metrics(Ag_true, Ag_hat)
print(metrics)
```

`verify_mean` *Verify zero-mean preservation (Theorem 10 assumption 2a)*

Description

Draws n i.i.d. N(0, I_m) latent factors, applies g component-wise, and checks whether |E[g(x)]| < tol on every coordinate.

Usage

```
verify_mean(g_fun, m = 5, n = 10000, tol = 0.001)
```

Arguments

`g_fun`	vectorised map g: R -> R
`m`	latent dimension
`n`	Monte-Carlo sample size
`tol`	numerical tolerance (default 1e-3)

Value

logical TRUE if |mean| < tol on all coords

Examples

```
tmp <- g_fun("weak_nonlinear")
verify_mean(tmp$g_fun, m = 5)
```

verify_subgaussian *Verify sub-Gaussian preservation*

Description

Draws n i.i.d. N(0, I_m) latent factors, applies g component-wise, and checks whether E[exp(g(x))] remains below an empirical cut-off. This is a quick proxy for finite sub-Gaussian norm.

Usage

```
verify_subgaussian(g_fun, m = 5, n = 1000, cut = exp(2))
```

Arguments

`g_fun`	vectorised map g: R -> R
`m`	latent dimension
`n`	Monte-Carlo sample size
`cut`	empirical threshold (default exp(2) & 7.389)

Value

logical TRUE if E[exp(g)] < cut on all coords

Examples

```
tmp <- g_fun("strong_nonlinear")
verify_subgaussian(tmp$g_fun, m = 5)
```

wholesale *Wholesale Customers Data*

Description

This dataset contains the annual spending amounts of wholesale customers on various product categories, along with their channel and region information.

Usage

```
data(wholesale)
```

Format

A data frame with 440 rows and 8 columns.

- `FRESH`: Annual spending (m.u.) on fresh products.
- `MILK`: Annual spending (m.u.) on milk products.
- `GROCERY`: Annual spending (m.u.) on grocery products.
- `FROZEN`: Annual spending (m.u.) on frozen products.
- `DETERGENTS_PAPER`: Annual spending (m.u.) on detergents and paper products.
- `DELICATESSEN`: Annual spending (m.u.) on delicatessen products.
- `CHANNEL`: Customers' channel - Horeca (Hotel/Restaurant/Café) or Retail channel (Nominal).
- `REGION`: Customers' region - Lisbon, Oporto or Other (Nominal).

Examples

```
data(wholesale)
```

`Wine` *Wine Data*

Description

The Wine dataset contains the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. This dataset is commonly used for classification tasks to determine the origin of wines based on their chemical properties.

Usage

```
data(Wine)
```

Format

A data frame with 178 rows and 14 columns representing different features of wines.

- `Class`: Categorical target variable indicating the type of wine (1, 2, or 3).
- `Alcohol`: Continuous feature representing the alcohol content.
- `Malic_acid`: Continuous feature representing the malic acid content.
- `Ash`: Continuous feature representing the ash content.
- `Alcalinity_of_ash`: Continuous feature representing the alcalinity of ash.
- `Magnesium`: Integer feature representing the magnesium content.
- `Total_phenols`: Continuous feature representing the total phenols content.
- `Flavanoids`: Continuous feature representing the flavanoids content.
- `Nonflavanoid_phenols`: Continuous feature representing the nonflavanoid phenols content.

- `Proanthocyanins`: Continuous feature representing the proanthocyanins content.
- `Color_intensity`: Continuous feature representing the color intensity.
- `Hue`: Continuous feature representing the hue.
- `OD280_OD315_of_diluted_wines`: Continuous feature representing the OD280/OD315 of diluted wines.
- `Proline`: Continuous feature representing the proline content.

Examples

```
# Load the dataset
data(Wine)

# Print the first few rows of the dataset
print(head(Wine))
```

Index

∗ **datasets**
    ionosphere, 5
    wholesale, 8
    Wine, 9

estimate_gul_loadings, 2

g_fun, 4
g_theorem, 5
generate_gfm_data, 3
gul_simulation, 3

ionosphere, 5

loading_metrics, 6

verify_mean, 7
verify_subgaussian, 8

wholesale, 8
Wine, 9