Package: sparseGFM (via r-universe)

May 13, 2026

Type Package

Title Sparse Generalized Factor Models with Multiple Penalty Functions

Version 0.1.0

Description Implements sparse generalized factor models (sparseGFM) for dimension reduction and variable selection in high-dimensional data with automatic adaptation to weak factor scenarios. The package supports multiple data types (continuous, count, binary) through generalized linear model frameworks and handles missing values automatically. It provides 12 different penalty functions including Least Absolute Shrinkage and Selection Operator (Lasso), adaptive Lasso, Smoothly Clipped Absolute Deviation (SCAD), Minimax Concave Penalty (MCP), group Lasso, and their adaptive versions for inducing row-wise sparsity in factor loadings. Key features include cross-validation for regularization parameter selection using Sparsity Information Criterion (SIC), automatic determination of the number of factors via multiple information criteria, and specialized algorithms for row-sparse loading structures. The methodology employs alternating minimization with Singular Value Decomposition (SVD)-based identifiability constraints and is particularly effective for high-dimensional applications in genomics, economics, and social sciences where interpretable sparse dimension reduction is crucial. For penalty functions, see Tibshirani (1996) <doi:10.1111/j.2517-6161.1996.tb02080.x>, Fan and Li (2001) <doi:10.1198/016214501753382273>, and Zhang (2010) <doi:10.1214/09-AOS729>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports stats, GFM, MASS, irlba

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

RoxygenNote 7.3.2

URL https://github.com/zjwang1013/sparseGFM

BugReports https://github.com/zjwang1013/sparseGFM/issues

Repository https://zjwang1013.r-universe.dev

Date/Publication 2025-09-15 01:56:10 UTC

RemoteUrl https://github.com/zjwang1013/sparsegfm

RemoteRef HEAD

RemoteSha 5d4527e9c3be2f4920416a98854c6e0e5f8c1834

Contents

`add_identifiability` *Apply Identifiability Constraints*

Description

Apply identifiability constraints to ensure unique solutions for factor and loading matrices

Usage

```
add_identifiability(H, B, mu)
```

Arguments

`H`	Factor matrix (n x q)
`B`	Loading matrix (p x q)
`mu`	Intercept vector (p x 1)

Details

This function applies SVD-based identifiability constraints to factor models

Value

List with constrained H, B, and mu matrices

`cv.sparseGFM` *Cross-Validation for Sparse Generalized Factor Model*

Description

Performs cross-validation to select the optimal lambda parameter for sparse generalized factor models using SIC (Sparsity Information Criterion). The method can handle weak factor scenarios and is particularly effective for row-sparse loading structures.

Usage

```
cv.sparseGFM(
  x,
  type = c("continuous", "count", "binary"),
  q = 2,
 penalty = c("lasso", "SCAD", "MCP", "group", "agroup", "gSCAD", "agSCAD", "gMCP",
    "agMCP", "alasso", "glasso", "aglasso"),
  lambda_range = seq(0.1, 1, by = 0.1),
  gam = 1,
  tau = NULL,
  mat_sd = 1,
  delta = 1e-04,
  maxiter = 30,
  C = 5,
  bic_type = "dd",
  verbose = TRUE
)
```

Arguments

`x`	A numeric matrix of observations (n x p)
`type`	Character string specifying the data type ("continuous", "count", or "binary")
`q`	Integer specifying the number of latent factors (default = 2)
`penalty`	Character string specifying the penalty type. See sparseGFM for all 12 available options. Recommended: "aglasso" for row-sparse loading matrices
`lambda_range`	Numeric vector of lambda values to evaluate (default = seq(0.1, 1, by = 0.1))
`gam`	Numeric value for the adaptive weight parameter (default = 1)
`tau`	Numeric value for the shape parameter in SCAD/MCP penalties
`mat_sd`	Standard deviation for continuous data (default = 1)
`delta`	Convergence tolerance (default = 1e-4)
`maxiter`	Maximum number of iterations (default = 30)
`C`	Constraint bound for stability (default = 5)
`bic_type`	Character string specifying BIC type. Options are:

- "dd": Default SIC using estimated degrees of freedom
- "as": Alternative simplified BIC

verbose Logical indicating whether to print progress (default = TRUE)

Details

The function fits sparse GFM models for each lambda value in lambda_range and calculates two types of SIC for model selection. The optimal lambda is chosen as the one minimizing the selected SIC criterion. The method automatically handles missing values and adapts to weak factor structures.

Value

A list containing:

- optimal_lambda: Selected optimal lambda value
- optimal_model: Model fitted with optimal lambda
- all_results: List of all fitted models for each lambda
- objloglik: Vector of log-likelihood values for each lambda
- bic_dd: Vector of default SIC values
- bic_as: Vector of alternative SIC values
- df_dd: Vector of degrees of freedom (default method)
- df_as: Vector of degrees of freedom (alternative method)
- lambda_range: Vector of evaluated lambda values

Examples

```
# Generate data with sparse loading structure
library(sparseGFM)
set.seed(123)
n <- 200; p <- 200; q <- 2
a_param <- 0.9; s <- ceiling(p^a_param)

FF <- matrix(runif(n * q, min = -3, max = 3), nrow = n, ncol = q)
BB <- rbind(matrix(runif(s * q, min = 1, max = 2), nrow = s, ncol = q),
            matrix(0, nrow = (p - s), ncol = q))
alpha_true <- runif(p, min = -1, max = 1)

ident_res <- add_identifiability(FF, BB, alpha_true)
FF0 <- ident_res$H; BB0 <- ident_res$B; alpha0 <- ident_res$mu

mat_para <- FF0 %*% t(BB0) + as.matrix(rep(1, n)) %*% t(as.matrix(alpha0))
x <- matrix(nrow = n, ncol = p)
for (i in 1:n) {
  for (j in 1:p) {
    x[i, j] <- rnorm(1, mean = mat_para[i, j])
  }
}
```

```
# Cross-validation for optimal lambda selection
cv_result <- cv.sparseGFM(x, type = "continuous", q = 2,
                          penalty = "aglasso", C = 5,
                          lambda_range = seq(0.1, 1, by = 0.1),
                          verbose = FALSE)

print(paste("Optimal lambda:", cv_result$optimal_lambda))
optimal_model <- cv_result$optimal_model
```

`eval.space` *Evaluate Subspace Angles Between Two Matrices*

Description

Calculate the Vector Correlation Coefficient (VCC) and Tucker's Congruence Coefficient (TCC) between two matrices to evaluate the similarity of subspaces.

Usage

```
eval.space(A, B, orthnm = TRUE)
```

Arguments

`A`	First matrix (n x k)
`B`	Second matrix (n x k)
`orthnm`	Logical indicating whether to orthonormalize the matrices (default = TRUE)

Details

This function computes subspace similarity measures commonly used in factor analysis and matrix factorization to compare estimated and true factor spaces.

Value

A numeric vector of length 2 containing:

- VCC: Vector Correlation Coefficient
- TCC: Tucker's Congruence Coefficient

Examples

```
# Generate example matrices
set.seed(123)
A <- matrix(rnorm(50), 10, 5)
B <- A + matrix(rnorm(50, sd = 0.1), 10, 5)  # Add small noise

# Calculate subspace angles
angles <- eval.space(A, B, orthnm = TRUE)
print(paste("VCC:", round(angles[1], 4)))
print(paste("TCC:", round(angles[2], 4)))
```

---

evaluate_performance *Evaluate Variable Selection Performance*

Description

Calculate performance metrics for variable selection including sensitivity, specificity, and other classification measures.

Usage

```
evaluate_performance(true_vector, pred_vector)
```

Arguments

true_vector	True binary vector indicating selected variables (0/1)
pred_vector	Predicted binary vector indicating selected variables (0/1)

Details

This function evaluates the performance of variable selection methods by comparing predicted selections against the true sparse structure.

Value

A list containing performance metrics:

- sensitivity: True positive rate
- specificity: True negative rate
- precision: Positive predictive value
- f1_score: F1 score
- accuracy: Overall accuracy

Examples

```
# Generate example selection vectors
set.seed(123)
p <- 100
true_selected <- c(rep(1, 20), rep(0, 80))  # First 20 variables selected
pred_selected <- c(rep(1, 18), rep(0, 2), rep(1, 5), rep(0, 75))  # Some errors

# Evaluate performance
perf <- evaluate_performance(true_selected, pred_selected)
print(perf)
```

`facnum.sparseGFM` *Determine the Number of Factors for Sparse Generalized Factor Model*

Description

Determines the optimal number of factors for sparse generalized factor models using multiple information criteria (SIC - Sparsity Information Criterion variants). The method can effectively handle weak factor scenarios and high-dimensional data with sparse loading structures.

Usage

```
facnum.sparseGFM(
  x,
  type = c("continuous", "count", "binary"),
  q_range = 1:5,
 penalty = c("lasso", "SCAD", "MCP", "group", "agroup", "gSCAD", "agSCAD", "gMCP",
    "agMCP", "alasso", "glasso", "aglasso"),
  lambda_range = seq(0.1, 1, by = 0.1),
  gam = 1,
  tau = NULL,
  mat_sd = 1,
  delta = 1e-04,
  maxiter = 30,
  C = 5,
  sic_type = "sic1",
  verbose = TRUE
)
```

Arguments

`x`	A numeric matrix of observations (n x p)
`type`	Character string specifying the data type ("continuous", "count", or "binary")
`q_range`	Integer vector of factor numbers to evaluate (default = 1:5)

penalty	Character string specifying the penalty type. See sparseGFM for all 12 available options. Recommended: "aglasso" for row-sparse loading matrices
lambda_range	Numeric vector of lambda values for cross-validation (default = seq(0.1, 1, by = 0.1))
gam	Numeric value for the adaptive weight parameter (default = 1)
tau	Numeric value for the shape parameter in SCAD/MCP penalties
mat_sd	Standard deviation for continuous data (default = 1)
delta	Convergence tolerance (default = 1e-4)
maxiter	Maximum number of iterations (default = 30)
C	Constraint bound for stability (default = 5)
sic_type	Character string specifying SIC type for selection. Options are: • "sic1": SIC with estimated df and (n+p) denominator • "sic2": SIC with simplified df and (n+p) denominator • "sic3": SIC with estimated df and max(n,p) denominator • "sic4": SIC with simplified df and max(n,p) denominator
verbose	Logical indicating whether to print progress (default = TRUE)

Details

For each q value, the function performs cross-validation to select optimal lambda, then calculates various SIC measures. The optimal q minimizes the selected SIC. This provides automatic selection of the latent dimension in factor models, with particular effectiveness in weak factor scenarios where traditional methods may struggle.

Value

A list containing:

- optimal_q: Selected optimal number of factors
- optimal_model: Model fitted with optimal q
- all_results: List of all fitted models for each q
- objpen: Vector of penalized objective values
- objlogli: Vector of log-likelihood values
- sic1, sic2, sic3, sic4: Vectors of different SIC values
- sic21, sic22, sic23, sic24: Alternative SIC formulations
- lambda_opt: Vector of optimal lambda values for each q
- df_dd: Vector of degrees of freedom (default method)
- df_as: Vector of degrees of freedom (alternative method)
- q_range: Vector of evaluated q values
- used_sic_type: The SIC type used for selection
- optimal_sic_value: The optimal SIC value

Examples

```
# Generate data with sparse loading structure
library(sparseGFM)
set.seed(123)
n <- 200; p <- 200; q <- 2
a_param <- 0.9; s <- ceiling(p^a_param)

FF <- matrix(runif(n * q, min = -3, max = 3), nrow = n, ncol = q)
BB <- rbind(matrix(runif(s * q, min = 1, max = 2), nrow = s, ncol = q),
            matrix(0, nrow = (p - s), ncol = q))
alpha_true <- runif(p, min = -1, max = 1)

ident_res <- add_identifiability(FF, BB, alpha_true)
FF0 <- ident_res$H; BB0 <- ident_res$B; alpha0 <- ident_res$mu

mat_para <- FF0 %*% t(BB0) + as.matrix(rep(1, n)) %*% t(as.matrix(alpha0))
x <- matrix(nrow = n, ncol = p)
for (i in 1:n) {
  for (j in 1:p) {
    x[i, j] <- rnorm(1, mean = mat_para[i, j])
  }
}

# Determine optimal number of factors using multiple criteria
facnum_result <- facnum.sparseGFM(x, type = "continuous",
                                  q_range = 1:5, penalty = "aglasso",
                                  lambda_range = c(0.1), sic_type = "sic1",
                                  C = 6, verbose = FALSE)

# Extract optimal factor numbers from different criteria
optimal_q_sic1 <- facnum_result$optimal_q
optimal_q_sic2 <- which.min(facnum_result$sic2)
optimal_q_sic3 <- which.min(facnum_result$sic3)
optimal_q_sic4 <- which.min(facnum_result$sic4)

print(paste("Optimal q (SIC1):", optimal_q_sic1))
print(paste("Optimal q (SIC2):", optimal_q_sic2))
```

`sparseGFM` *Sparse Generalized Factor Model with Multiple Penalty Functions*

Description

Implements sparse generalized factor models with 12 different penalty functions for dimension reduction and variable selection in high-dimensional data. The method is designed to handle row-sparse loading structures and can adapt to weak factor scenarios where factors have relatively small eigenvalues. Missing values are automatically handled through imputation.

Usage

```
sparseGFM(
  x,
  type = c("continuous", "count", "binary"),
  q = 2,
 penalty = c("lasso", "SCAD", "MCP", "group", "agroup", "gSCAD", "agSCAD", "gMCP",
    "agMCP", "alasso", "glasso", "aglasso"),
  lambda = 1,
  gam = 1,
  tau = NULL,
  mat_sd = 1,
  delta = 1e-04,
  maxiter = 30,
  C = 5,
  verbose = TRUE
)
```

Arguments

x	A numeric matrix of observations (n x p), where n is the number of observations and p is the number of variables
type	Character string specifying the data type. Options are: • "continuous": Gaussian family for continuous data • "count": Poisson family for count data • "binary": Binomial family for binary data
q	Integer specifying the number of latent factors (default = 2)
penalty	Character string specifying the penalty type for sparsity. Available options: • "lasso": L1 penalty (adaptive version: "alasso") • "SCAD": Smoothly clipped absolute deviation penalty (adaptive: "agSCAD") • "MCP": Minimax concave penalty (adaptive: "agMCP") • "group"/"glasso": Group lasso penalty (adaptive: "agroup"/"aglasso") • "gSCAD": Group SCAD penalty (adaptive: "agSCAD") • "gMCP": Group MCP penalty (adaptive: "agMCP")
lambda	Numeric value for the penalty tuning parameter (default = 1)
gam	Numeric value for the adaptive weight parameter in adaptive penalties (default = 1)
tau	Numeric value for the shape parameter in SCAD/MCP penalties. Default is 3.7 for SCAD and 3 for MCP if not specified
mat_sd	Standard deviation for continuous data (default = 1)
delta	Convergence tolerance for the iterative algorithm (default = 1e-4)
maxiter	Maximum number of iterations (default = 30)
C	Numeric value for the constraint bound to ensure stability (default = 5)
verbose	Logical indicating whether to print iteration progress (default = TRUE)

Details

The algorithm employs alternating minimization with the following steps: 1. Initialization using the GFM package for initial estimates 2. Iteratively updating factors (F) and loadings (B) 3. Applying penalty functions to achieve variable selection 4. Ensuring identifiability through SVD-based constraints 5. Monitoring convergence through objective function changes

The adaptive group lasso (aglasso) penalty is particularly effective for row-sparse loading matrices as it can select entire rows (variables) rather than individual elements.

Value

A list containing:

- FF_hat: Estimated factor matrix (n x q)
- BB_hat: Estimated loading matrix (p x q)
- alpha_hat: Estimated intercept vector (p x 1)
- obj_loglik: Log-likelihood value
- obj_pen: Penalized objective function value
- index: Indices of variables with zero loadings (selected out)
- df_est: Estimated degrees of freedom
- iter: Number of iterations performed

Examples

```
# Generate data with sparse loading matrix
library(sparseGFM)
set.seed(123)
n <- 200; p <- 200; q <- 2
a_param <- 0.9
s <- ceiling(p^a_param)

# Generate factors and sparse loadings
FF <- matrix(runif(n * q, min = -3, max = 3), nrow = n, ncol = q)
BB <- rbind(matrix(runif(s * q, min = 1, max = 2), nrow = s, ncol = q),
            matrix(0, nrow = (p - s), ncol = q))
alpha_true <- runif(p, min = -1, max = 1)

# Apply identifiability constraints
ident_res <- add_identifiability(FF, BB, alpha_true)
FF0 <- ident_res$H; BB0 <- ident_res$B; alpha0 <- ident_res$mu

# Generate data matrix
mat_para <- FF0 %*% t(BB0) + as.matrix(rep(1, n)) %*% t(as.matrix(alpha0))
x <- matrix(nrow = n, ncol = p)
for (i in 1:n) {
  for (j in 1:p) {
    x[i, j] <- rnorm(1, mean = mat_para[i, j])
  }
}
```

```
# Fit sparse GFM with adaptive group lasso
result <- sparseGFM(x, type = "continuous", q = 2,
                    penalty = "aglasso", lambda = 0.1, C = 5)

# View results
print(paste("Selected variables:", length(setdiff(1:p, result$index))))
```

Index