# Package 'grasps'

May 3, 2026

**Type** Package

**Title** Groupwise Regularized Adaptive Sparse Precision Solution

**Version** 0.1.1

**Maintainer** Shiying Xiao <shiying.xiao@outlook.com>

**Description** Provides a unified framework for sparse-group regularization and precision matrix estimation in Gaussian graphical models. It implements multiple sparse-group penalties, including sparse-group lasso, sparse-group adaptive lasso, sparse-group SCAD, and sparse-group MCP, and solves them efficiently using ADMM-based optimization. The package is designed for high-dimensional network inference where both sparsity and group structure are present.

**License** GPL (>= 3)

**URL** https://github.com/Carol-seven/grasps, https://shiying-xiao.com/grasps/

**BugReports** https://github.com/Carol-seven/grasps/issues

**Encoding** UTF-8

**Imports** igraph, ggforce, ggplot2, grDevices, Rcpp, Rdpack, scales

**LinkingTo** Rcpp, RcppArmadillo

**RdMacros** Rdpack

**Suggests** knitr, MASS, quarto, rmarkdown

**VignetteBuilder** knitr, quarto

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** yes

**Author** Shiying Xiao [aut, cre] (ORCID: <https://orcid.org/0000-0002-8846-3258>), Jun Yan [aut] (ORCID: <https://orcid.org/0000-0003-4401-7296>), Panpan Zhang [aut] (ORCID: <https://orcid.org/0000-0002-8211-5930>)

**Repository** CRAN

**Date/Publication** 2026-05-02 22:50:09 UTC

# Contents



---

`compute_derivative` *Penalty Derivative Computation*

---

## Description

Compute one or more derivative values for a given `omega`, allowing vectorized specifications of `penalty`, `lambda`, and `gamma`.

## Usage

```
compute_derivative(omega, penalty, lambda, gamma = NA)
```

## Arguments

| | |
|---|---|
| `omega` | A numeric value or vector at which the penalty is evaluated. |
| `penalty` | A character string or vector specifying one or more penalty types. Available options include:<br>1. "lasso": Least absolute shrinkage and selection operator (Tibshirani 1996; Friedman et al. 2008).<br>2. "atan": Arctangent type penalty (Wang and Zhu 2016).<br>3. "exp": Exponential type penalty (Wang et al. 2018).<br>4. "lq": Lq penalty (Frank and Friedman 1993; Fu 1998; Fan and Li 2001).<br>5. "lsp": Log-sum penalty (Candès et al. 2008).<br>6. "mcp": Minimax concave penalty (Zhang 2010).<br>7. "scad": Smoothly clipped absolute deviation (Fan and Li 2001; Fan et al. 2009).<br>If `penalty` has length 1, it is recycled to the common length determined by `penalty`, `lambda`, and `gamma`. |
| `lambda` | A non-negative numeric value or vector specifying the regularization parameter. If `lambda` has length 1, it is recycled to the common length determined by `penalty`, `lambda`, and `gamma`. |

`gamma` A numeric value or vector specifying the additional parameter for the penalty function. If `lambda` has length 1, it is recycled to the common length determined by `penalty`, `lambda`, and `gamma`. The penalty-specific defaults are:

1. "atan": 0.005
2. "exp": 0.01
3. "lq": 0.5
4. "lsp": 0.1
5. "mcp": 3
6. "scad": 3.7

For `"lasso"`, `gamma` is ignored.

## Value

A data frame with S3 class `"derivative"` containing:

**omega** The input `omega` values.

**penalty** The penalty type for each row.

**lambda** The regularization parameter used.

**gamma** The additional penalty parameter used.

**value** The computed derivative value.

The number of rows equals `max(length(penalty), length(lambda), length(gamma))`. Any of `penalty`, `lambda`, or `gamma` with length 1 is recycled to this common length.

## References

Candès EJ, Wakin MB, Boyd SP (2008). "Enhancing Sparsity by Reweighted $\ell_1$ Minimization." *Journal of Fourier Analysis and Applications*, **14**(5), 877–905. doi:10.1007/s000410089045x.

Fan J, Feng Y, Wu Y (2009). "Network Exploration via the Adaptive LASSO and SCAD Penalties." *The Annals of Applied Statistics*, **3**(2), 521–541. doi:10.1214/08aoas215.

Fan J, Li R (2001). "Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties." *Journal of the American Statistical Association*, **96**(456), 1348–1360. doi:10.1198/016214501753382273.

Frank LE, Friedman JH (1993). "A Statistical View of Some Chemometrics Regression Tools." *Technometrics*, **35**(2), 109–135. doi:10.1080/00401706.1993.10485033.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441. doi:10.1093/biostatistics/kxm045.

Fu WJ (1998). "Penalized Regressions: The Bridge versus the Lasso." *Journal of Computational and Graphical Statistics*, **7**(3), 397–416. doi:10.1080/10618600.1998.10474784.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society: Series B (Methodological)*, **58**(1), 267–288. doi:10.1111/j.25176161.1996.tb02080.x.

Wang Y, Fan Q, Zhu L (2018). "Variable Selection and Estimation using a Continuous Approximation to the $L_0$ Penalty." *Annals of the Institute of Statistical Mathematics*, **70**(1), 191–214. doi:10.1007/s1046301605883.

Wang Y, Zhu L (2016). "Variable Selection and Parameter Estimation with the Atan Regularization Method." *Journal of Probability and Statistics*, **2016**, 6495417. doi:10.1155/2016/6495417.

Zhang C (2010). "Nearly Unbiased Variable Selection under Minimax Concave Penalty." *The Annals of Statistics*, **38**(2), 894–942. doi:10.1214/09AOS729.

## Examples

```
library(grasps)
library(ggplot2)

deriv_df <- compute_derivative(
  omega = seq(0, 4, by = 0.01),
  penalty = c("atan", "exp", "lasso", "lq", "lsp", "mcp", "scad"),
  lambda = 1)

plot(deriv_df) +
  scale_y_continuous(limits = c(0, 1.5)) +
  guides(color = guide_legend(nrow = 2, byrow = TRUE))
```

---

`compute_penalty` *Penalty Function Computation*

---

## Description

Compute one or more penalty values for a given `omega`, allowing vectorized specifications of `penalty`, `lambda`, and `gamma`.

## Usage

```
compute_penalty(omega, penalty, lambda, gamma = NA)
```

## Arguments

| | |
|---|---|
| `omega` | A numeric value or vector at which the penalty is evaluated. |
| `penalty` | A character string or vector specifying one or more penalty types. Available options include:<br>1. "lasso": Least absolute shrinkage and selection operator (Tibshirani 1996; Friedman et al. 2008).<br>2. "atan": Arctangent type penalty (Wang and Zhu 2016).<br>3. "exp": Exponential type penalty (Wang et al. 2018).<br>4. "lq": Lq penalty (Frank and Friedman 1993; Fu 1998; Fan and Li 2001). |

5. "lsp": Log-sum penalty (Candès et al. 2008).
6. "mcp": Minimax concave penalty (Zhang 2010).
7. "scad": Smoothly clipped absolute deviation (Fan and Li 2001; Fan et al. 2009).

If `penalty` has length 1, it is recycled to the common length determined by `penalty`, `lambda`, and `gamma`.

`lambda` A non-negative numeric value or vector specifying the regularization parameter. If `lambda` has length 1, it is recycled to the common length determined by `penalty`, `lambda`, and `gamma`.

`gamma` A numeric value or vector specifying the additional parameter for the penalty function. If `lambda` has length 1, it is recycled to the common length determined by `penalty`, `lambda`, and `gamma`. The penalty-specific defaults are:

1. "atan": 0.005
2. "exp": 0.01
3. "lq": 0.5
4. "lsp": 0.1
5. "mcp": 3
6. "scad": 3.7

For `"lasso"`, `gamma` is ignored.

## Value

A data frame with S3 class `"penalty"` containing:

**omega** The input `omega` values.

**penalty** The penalty type for each row.

**lambda** The regularization parameter used.

**gamma** The additional penalty parameter used.

**value** The computed penalty value.

The number of rows equals `max(length(penalty), length(lambda), length(gamma))`. Any of `penalty`, `lambda`, or `gamma` with length 1 is recycled to this common length.

## References

Candès EJ, Wakin MB, Boyd SP (2008). "Enhancing Sparsity by Reweighted $\ell_1$ Minimization." *Journal of Fourier Analysis and Applications*, **14**(5), 877–905. doi:10.1007/s000410089045x.

Fan J, Feng Y, Wu Y (2009). "Network Exploration via the Adaptive LASSO and SCAD Penalties." *The Annals of Applied Statistics*, **3**(2), 521–541. doi:10.1214/08aoas215.

Fan J, Li R (2001). "Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties." *Journal of the American Statistical Association*, **96**(456), 1348–1360. doi:10.1198/016214501753382273.

Frank LE, Friedman JH (1993). "A Statistical View of Some Chemometrics Regression Tools."

*Technometrics*, **35**(2), 109–135. doi:10.1080/00401706.1993.10485033.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441. doi:10.1093/biostatistics/kxm045.

Fu WJ (1998). "Penalized Regressions: The Bridge versus the Lasso." *Journal of Computational and Graphical Statistics*, **7**(3), 397–416. doi:10.1080/10618600.1998.10474784.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society: Series B (Methodological)*, **58**(1), 267–288. doi:10.1111/j.25176161.1996.tb02080.x.

Wang Y, Fan Q, Zhu L (2018). "Variable Selection and Estimation using a Continuous Approximation to the $L_0$ Penalty." *Annals of the Institute of Statistical Mathematics*, **70**(1), 191–214. doi:10.1007/s1046301605883.

Wang Y, Zhu L (2016). "Variable Selection and Parameter Estimation with the Atan Regularization Method." *Journal of Probability and Statistics*, **2016**, 6495417. doi:10.1155/2016/6495417.

Zhang C (2010). "Nearly Unbiased Variable Selection under Minimax Concave Penalty." *The Annals of Statistics*, **38**(2), 894–942. doi:10.1214/09AOS729.

## Examples

```
library(grasps)
library(ggplot2)

pen_df <- compute_penalty(
  omega = seq(-4, 4, by = 0.01),
  penalty = c("atan", "exp", "lasso", "lq", "lsp", "mcp", "scad"),
  lambda = 1)

plot(pen_df, xlim = c(-1, 1), ylim = c(0, 1), zoom.size = 1) +
  guides(color = guide_legend(nrow = 2, byrow = TRUE))
```

---

`gen_prec_sbm` *Block-Structured Precision Matrix based on SBM*

---

## Description

Generate a precision matrix that exhibits block structure induced by a stochastic block model (SBM).

## Usage

```
gen_prec_sbm(
  p,
  block.sizes = NULL,
  K = 3,
```

```
  prob.mat = NULL,
  within.prob = 0.25,
  between.prob = 0.05,
  weight.mat = NULL,
  weight.dists = list("gamma", "unif"),
  weight.paras = list(c(shape = 100, rate = 10), c(min = 0, max = 5)),
  cond.target = 100
)
```

## Arguments

| | |
|---|---|
| p | An integer specifying the number of variables (dimensions). |
| block.sizes | An integer vector (default = NULL) specifying the size of each group. If NULL, the $p$ variables are divided as evenly as possible across $K$ groups. |
| K | An integer (default = 3) specifying the number of groups. Ignored if block.sizes is provided; then K <- length(block.sizes). |
| prob.mat | A $K \times K$ symmetric matrix (default = NULL) specifying the Bernoulli rates. Element $(i, j)$ gives the probability of creating an edge between vertices from groups $i$ and $j$. If NULL, a matrix with within.prob on the diagonal and between.prob on the off-diagonal is used. |
| within.prob | A numeric value in [0,1] (default = 0.25) specifying the probability of creating an edge between vertices within the same group. This argument is used only when prob.mat = NULL. |
| between.prob | A numeric value in [0,1] (default = 0.05) specifying the probability of creating an edge between vertices from different groups. This argument is used only when prob.mat = NULL. |
| weight.mat | A $p \times p$ symmetric matrix (default = NULL) specifying the edge weights. If NULL, weights are generated block-wise according to weight.dists and weight.paras. |
| weight.dists | A list (default = list("gamma", "unif")) specifying the sampling distribution for each block of weights. Its length determines how the distributions are assigned:<br>• length = 1: Same specification for all blocks.<br>• length = 2: First for within-group blocks, second for between-group blocks.<br>• length = $K + K(K-1)/2$: Full specification for each block. The first $K$ elements correspond to within-group blocks with indices $1, \ldots, K$, and the remaining $K(K-1)/2$ elements correspond to between-group blocks ordered as $(1,2), (1,3), (1,4), \ldots, (1,K), (2,3), \ldots, (K-1,K)$.<br>Each element of weight.dists can be:<br>1. A user-supplied sampling function. The function must accept an argument n specifying the number of samples.<br>2. A character string specifying the distribution family. Accepted distributions (base R samplers in parentheses) include:<br>• "beta": Beta distribution (rbeta)<br>• "cauchy": Cauchy distribution (rcauchy).<br>• "chisq": Chi-squared distribution (rchisq). |

- "exp": Exponential distribution (rexp).
- "f": F distribution (rf).
- "gamma": Gamma distribution (rgamma).
- "lnorm": Log normal distribution (rlnorm).
- "norm": Normal distribution (rnorm).
- "t": Student's t distribution (rt).
- "unif": Uniform distribution (runif).
- "weibull": Weibull distribution (rweibull).

`weight.paras` A list (default = `list(c(shape = 100, rate = 10), c(min = 0, max = 5))`) specifying the parameters associated with `weight.dists`. It must follow the same length rules as `weight.dists`. Each element should be a named vector or list suitable for the corresponding sampler.

`cond.target` A numeric value > 1 (default = 100) specifying the target condition number for the precision matrix. When necessary, a diagonal shift is applied to ensure positive definiteness and numerical stability.

## Details

**Edge sampling.** Within- and between-group edges are sampled independently according to Bernoulli distributions specified by `prob.mat`, or by `within.prob` and `between.prob` if `prob.mat` is not supplied.

**Weight sampling.** Conditional on the adjacency structure, edge weights are sampled block-wise from samplers specified in `weight.dists` and `weight.paras`. The length of `weight.dists` (and `weight.paras`) determines how weight distributions are assigned:

- length = 1: Same specification for all blocks.
- length = 2: first for within-group blocks, second for between-group blocks.
- length = $K + K(K-1)/2$: Full specification for each block.

**Block indexing.** The order for blocks is:

- Within-group blocks: Indices $1, \ldots, K$.
- Between-group blocks: $K(K-1)/2$ blocks in order $(1,2)$, $(1,3)$, $(1,4)$, $\ldots$, $(1,K)$, $(2,3)$, $\ldots$, $(K-1,K)$.

**Positive definiteness.** The weighted adjacency matrix is symmetrized and used as the precision matrix $\Omega_0$. Since arbitrary block-structured weights may not be positive definite, a diagonal adjustment is applied to control the eigenvalue spectrum. Specifically, let $\lambda_{\max}$ and $\lambda_{\min}$ denote the largest and smallest eigenvalues of a matrix. A non-negative numeric value $\tau$ is added to the diagonal so that

$$\begin{cases} \dfrac{\lambda_{\max}(\Omega_0 + \tau I)}{\lambda_{\min}(\Omega_0 + \tau I)} \leq \texttt{cond.target} \\ \lambda_{\min}(\Omega_0 + \tau I) > 0 \\ \tau \geq 0 \end{cases}$$

which ensures both positive definiteness and guarantees that the condition number does not exceed `cond.target`, providing numerical stability even in high-dimensional settings.

## Value

An object with S3 class "gen_prec_sbm" containing the following components:

**Omega** The precision matrix with SBM block structure.

**Sigma** The covariance matrix, i.e., the inverse of Omega.

**sparsity** Proportion of zero entries in Omega.

**membership** An integer vector specifying the group membership.

## Examples

```
library(grasps)

## reproducibility for everything
set.seed(1234)

## block-structured precision matrix based on SBM
#### case 1: base R distribution
sim1 <- gen_prec_sbm(p = 100, K = 5,
                     within.prob = 0.25, between.prob = 0.1,
                     weight.dists = list("gamma", "unif"),
                     weight.paras = list(c(shape = 100, scale = 1e2),
                                         c(min = 0, max = 10)),
                     cond.target = 100)
#### visualization
plot(sim1)

#### case 2: user-defined sampler
my_gamma <- function(n) {
  rgamma(n, shape = 1e4, scale = 1e2)
}
sim2 <- gen_prec_sbm(p = 100, K = 5,
                     within.prob = 0.2, between.prob = 0.05,
                     weight.dists = list(my_gamma, "unif"),
                     weight.paras = list(NULL,
                                         c(min = 0, max = 1)),
                     cond.target = 100)
#### visualization
plot(sim2)
```

---

grasps *Groupwise Regularized Adaptive Sparse Precision Solution*

---

## Description

Provide a collection of statistical methods that incorporate both element-wise and group-wise penalties to estimate a precision matrix.

## Usage

```
grasps(
  X,
  n = nrow(X),
  membership,
  penalty,
  diag.ind = TRUE,
  diag.grp = TRUE,
  diag.include = FALSE,
  lambda = NULL,
  alpha = NULL,
  gamma = NULL,
  nlambda = 10,
  lambda.min.ratio = 0.01,
  growiter.lambda = 30,
  tol.lambda = 0.001,
  maxiter.lambda = 50,
  rho = 2,
  tau.incr = 2,
  tau.decr = 2,
  nu = 10,
  tol.abs = 1e-04,
  tol.rel = 1e-04,
  maxiter = 10000,
  crit = "BIC",
  kfold = 5,
  ebic.tuning = 0.5
)
```

## Arguments

| | |
|---|---|
| `X` | 1. An $n \times p$ data matrix with sample size $n$ and dimension $p$.<br>2. A $p \times p$ sample covariance matrix with dimension $p$. |
| `n` | An integer (default = `nrow(X)`) specifying the sample size. This is only required when the input matrix `X` is a $p \times p$ sample covariance matrix with dimension $p$. |
| `membership` | An integer vector specifying the group membership. The length of `membership` must be consistent with the dimension $p$. |
| `penalty` | A character string specifying the penalty for estimating precision matrix. Available options include:<br>1. "lasso": Least absolute shrinkage and selection operator (Tibshirani 1996; Friedman et al. 2008).<br>2. "adapt": Adaptive lasso (Zou 2006; Fan et al. 2009).<br>3. "atan": Arctangent type penalty (Wang and Zhu 2016).<br>4. "exp": Exponential type penalty (Wang et al. 2018).<br>5. "lq": Lq penalty (Frank and Friedman 1993; Fu 1998; Fan and Li 2001).<br>6. "lsp": Log-sum penalty (Candès et al. 2008). |

7. "mcp": Minimax concave penalty (Zhang 2010).
8. "scad": Smoothly clipped absolute deviation (Fan and Li 2001; Fan et al. 2009).

`diag.ind` A logical value (default = TRUE) specifying whether to penalize the diagonal elements.

`diag.grp` A logical value (default = TRUE) specifying whether to penalize the within-group blocks.

`diag.include` A logical value (default = FALSE) specifying whether to include the diagonal entries in the penalty for within-group blocks when `diag.grp = TRUE`.

`lambda` A non-negative numeric vector specifying the grid for the regularization parameter. The default is `NULL`, which generates its own `lambda` sequence based on `nlambda` and `lambda.min.ratio`.

`alpha` A numeric vector in [0,1] specifying the grid for the mixing parameter balancing the element-wise individual L1 penalty and the block-wise group L2 penalty. An alpha of 1 corresponds to the individual penalty only; an alpha of 0 corresponds to the group penalty only. The default value is a sequence from 0.1 to 0.9 with increments of 0.1.

`gamma` A numeric value specifying the additional parameter fo the chosen `penalty`. The default value depends on the penalty:

1. "adapt": 0.5
2. "atan": 0.005
3. "exp": 0.01
4. "lq": 0.5
5. "lsp": 0.1
6. "mcp": 3
7. "scad": 3.7

`nlambda` An integer (default = 10) specifying the number of `lambda` values to generate when `lambda = NULL`.

`lambda.min.ratio`
A numeric value > 0 (default = 0.01) specifying the fraction of the maximum `lambda` value $\lambda_{\max}$ to generate the minimum `lambda` $\lambda_{\min}$. If `lambda = NULL`, a `lambda` grid of length `nlambda` is automatically generated on a log scale, ranging from $\lambda_{\max}$ down to $\lambda_{\min}$.

`growiter.lambda`
An integer (default = 30) specifying the maximum number of exponential growth steps during the initial search for an admissible upper bound $\lambda_{\max}$.

`tol.lambda` A numeric value > 0 (default = 1e-03) specifying the relative tolerance for the bisection stopping rule on the interval width.

`maxiter.lambda` An integer (default = 50) specifying the maximum number of bisection iterations in the line search for $\lambda_{\max}$.

`rho` A numeric value > 0 (default = 2) specifying the ADMM augmented-Lagrangian penalty parameter (often called the ADMM step size). Larger values typically put more weight on enforcing the consensus constraints at each iteration; smaller values yield more conservative updates.

`tau.incr` A numeric value > 1 (default = 2) specifying the multiplicative factor used to increase `rho` when the primal residual dominates the dual residual in ADMM.

`tau.decr` A numeric value > 1 (default = 2) specifying the multiplicative factor used to decrease `rho` when the dual residual dominates the primal residual in ADMM.

`nu` A numeric value > 1 (default = 10) controlling how aggressively `rho` is rescaled in the adaptive-`rho` scheme (residual balancing).

`tol.abs` A numeric value > 0 (default = 1e-04) specifying the absolute tolerance for ADMM stopping (applied to primal/dual residual norms).

`tol.rel` A numeric value > 0 (default = 1e-04) specifying the relative tolerance for ADMM stopping (applied to primal/dual residual norms).

`maxiter` An integer (default = 1e+04) specifying the maximum number of ADMM iterations.

`crit` A character string (default = "BIC") specifying the parameter selection criterion to use. Available options include:

1. "AIC": Akaike information criterion (Akaike 1973).
2. "BIC": Bayesian information criterion (Schwarz 1978).
3. "EBIC": extended Bayesian information criterion (Chen and Chen 2008; Foygel and Drton 2010).
4. "HBIC": high dimensional Bayesian information criterion (Wang et al. 2013; Fan et al. 2017).
5. "CV": k-fold cross validation with negative log-likelihood loss.

`kfold` An integer (default = 5) specifying the number of folds used for `crit = "CV"`.

`ebic.tuning` A numeric value in [0,1] (default = 0.5) specifying the tuning parameter to calculate for `crit = "EBIC"`.

## Value

An object with S3 class `"grasps"` containing the following components:

**hatOmega** The estimated precision matrix.

**lambda** The optimal regularization parameter.

**alpha** The optimal mixing parameter.

**initial** The initial estimate of `hatOmega` when a non-convex penalty is chosen via `penalty`.

**gamma** The optimal addtional parameter when a non-convex penalty is chosen via `penalty`.

**iterations** The number of ADMM iterations.

**lambda.grid** The actual lambda grid used in the program.

**alpha.grid** The actual alpha grid used in the program.

**lambda.safe** The bisection-refined upper bound $\lambda_{\max}$, corresponding to `alpha.grid`, when `lambda = NULL`.

**loss** The optimal k-fold loss when `crit = "CV"`.

**CV.loss** Matrix of CV losses, with rows for parameter combinations and columns for CV folds, when `crit = "CV"`.

**score** The optimal information criterion score when `crit` is set to `"AIC"`, `"BIC"`, `"EBIC"`, or `"HBIC"`.

**IC.score** The information criterion score for each parameter combination when `crit` is set to `"AIC"`, `"BIC"`, `"EBIC"`, or `"HBIC"`.

**membership** The group membership.

## References


Akaike H (1973). "Information Theory and an Extension of the Maximum Likelihood Principle." In Petrov BN, Csáki F (eds.), *Second International Symposium on Information Theory*, 267–281. Akadémiai Kiadó, Budapest, Hungary.

Candès EJ, Wakin MB, Boyd SP (2008). "Enhancing Sparsity by Reweighted $\ell_1$ Minimization." *Journal of Fourier Analysis and Applications*, **14**(5), 877–905. doi:10.1007/s000410089045x.

Chen J, Chen Z (2008). "Extended Bayesian Information Criteria for Model Selection with Large Model Spaces." *Biometrika*, **95**(3), 759–771. doi:10.1093/biomet/asn034.

Fan J, Feng Y, Wu Y (2009). "Network Exploration via the Adaptive LASSO and SCAD Penalties." *The Annals of Applied Statistics*, **3**(2), 521–541. doi:10.1214/08aoas215.

Fan J, Li R (2001). "Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties." *Journal of the American Statistical Association*, **96**(456), 1348–1360. doi:10.1198/016214501753382273.

Fan J, Liu H, Ning Y, Zou H (2017). "High Dimensional Semiparametric Latent Graphical Model for Mixed Data." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **79**(2), 405–421. doi:10.1111/rssb.12168.

Foygel R, Drton M (2010). "Extended Bayesian Information Criteria for Gaussian Graphical Models." In Lafferty J, Williams C, Shawe-Taylor J, Zemel R, Culotta A (eds.), *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, 604–612. https://dl.acm.org/doi/10.5555/2997189.2997257.

Frank LE, Friedman JH (1993). "A Statistical View of Some Chemometrics Regression Tools." *Technometrics*, **35**(2), 109–135. doi:10.1080/00401706.1993.10485033.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441. doi:10.1093/biostatistics/kxm045.

Fu WJ (1998). "Penalized Regressions: The Bridge versus the Lasso." *Journal of Computational and Graphical Statistics*, **7**(3), 397–416. doi:10.1080/10618600.1998.10474784.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society: Series B (Methodological)*, **58**(1), 267–288. doi:10.1111/j.25176161.1996.tb02080.x.

Wang L, Kim Y, Li R (2013). "Calibrating Nonconvex Penalized Regression in Ultra-High Dimension." *The Annals of Statistics*, **41**(5), 2505–2536. doi:10.1214/13AOS1159.

Wang Y, Fan Q, Zhu L (2018). "Variable Selection and Estimation using a Continuous Approximation to the $L_0$ Penalty." *Annals of the Institute of Statistical Mathematics*, **70**(1), 191–214. doi:10.1007/s1046301605883.

Wang Y, Zhu L (2016). "Variable Selection and Parameter Estimation with the Atan Regularization Method." *Journal of Probability and Statistics*, **2016**, 6495417. doi:10.1155/2016/6495417.

Zhang C (2010). "Nearly Unbiased Variable Selection under Minimax Concave Penalty." *The Annals of Statistics*, **38**(2), 894–942. doi:10.1214/09AOS729.

Zou H (2006). "The Adaptive Lasso and Its Oracle Properties." *Journal of the American Statistical Association*, **101**(476), 1418–1429. doi:10.1198/016214506000000735.

## Examples

```
library(grasps)

## reproducibility for everything
set.seed(1234)

## block-structured precision matrix based on SBM
sim <- gen_prec_sbm(p = 30, K = 3,
                    within.prob = 0.25, between.prob = 0.05,
                    weight.dists = list("gamma", "unif"),
                    weight.paras = list(c(shape = 20, rate = 10),
                                        c(min = 0, max = 5)),
                    cond.target = 100)
## ground truth visualization
plot(sim)

## n-by-p data matrix
library(MASS)
X <- mvrnorm(n = 20, mu = rep(0, 30), Sigma = sim$Sigma)

## precision matrix: adaptive lasso; BIC
prec <- grasps(X = X, membership = sim$membership, penalty = "adapt", crit = "BIC")

## precision matrix visualization
plot(prec)

## performance
performance(hatOmega = prec$hatOmega, Omega = sim$Omega)

## adjacency matrix: diagonal = 0; raw partial correlations;
##                   no thresholding; weighted network
adj <- prec_to_adj(prec$hatOmega,
                   diag.zero = TRUE, absolute = FALSE,
```

```
                    threshold = NULL, weighted = TRUE)

## adjacency matrix visualization
plot(adj)
```

---

`performance` *Performance Measures for Precision Matrix Estimation*

---

## Description

Compute a collection of loss-based and structure-based measures to evaluate the performance of an estimated precision matrix.

## Usage

```
performance(hatOmega, Omega)
```

## Arguments

| | |
|---|---|
| `hatOmega` | A numeric $p \times p$ matrix giving the estimated precision matrix. |
| `Omega` | A numeric $p \times p$ matrix giving the reference (typically true) precision matrix. |

## Details

Let $\Omega_{p\times p}$ and $\hat{\Omega}_{p\times p}$ be the reference (true) and estimated precision matrices, respectively, with $\Sigma = \Omega^{-1}$ being the corresponding covariance matrix. Edges are defined by nonzero off-diagonal entries in the upper triangle of the precision matrices.

Sparsity is treated as a structural summary, while the remaining measures are grouped into loss-based measures, raw confusion-matrix counts, and classification-based (structure-recovery) measures.

"sparsity": Sparsity is computed as the proportion of zero entries among the off-diagonal elements in the upper triangle of $\hat{\Omega}$.

**Loss-based measures:**

- "Frobenius": Frobenius (Hilbert-Schmidt) norm loss $= \|\Omega - \hat{\Omega}\|_F$.
- "KL": Kullback-Leibler divergence $= \mathrm{tr}(\Sigma\hat{\Omega}) - \log\det(\Sigma\hat{\Omega}) - p$.
- "quadratic": Quadratic norm loss $= \|\Sigma\hat{\Omega} - I_p\|_F^2$.
- "spectral": Spectral (operator) norm loss $= \|\Omega - \hat{\Omega}\|_{2,2} = e_1$, where $e_1^2$ is the largest eigenvalue of $(\Omega - \hat{\Omega})^2$.

**Confusion-matrix counts:**

- "TP": True positive = number of edges in both $\Omega$ and $\hat{\Omega}$.
- "TN": True negative = number of edges in neither $\Omega$ nor $\hat{\Omega}$.
- "FP": False positive = number of edges in $\hat{\Omega}$ but not in $\Omega$.

- "FN": False negative = number of edges in $\Omega$ but not in $\hat{\Omega}$.

**Classification-based (structure-recovery) measures:**

- "TPR": True positive rate (TPR), recall, sensitivity $= \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$.
- "FPR": False positive rate (FPR) $= \mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$.
- "F1": $F_1$ score $= 2\,\mathrm{TP}/(2\,\mathrm{TP}+\mathrm{FN}+\mathrm{FP})$
- "MCC": Matthews correlation coefficient (MCC) $= (\mathrm{TP}\times\mathrm{TN}-\mathrm{FP}\times\mathrm{FN})/\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}$

The following table summarizes the confusion matrix and related rates:

| | **Predicted Positive** | **Predicted Negative** |
|---|---|---|
| **Real Positive** (P) | True positive (TP) | False negative (FN) |
| **Real Negative** (N) | False positive (FP) | True negative (TN) |
| | Positive predictive value (PPV), precision = TP / (TP + FP) = 1 - FDR | False omission rate (FOR) = FN |
| | False discovery rate (FDR) = FP / (TP + FP) = 1 - PPV | Negative predictive value (NPV |

## Value

A data frame of S3 class `"performance"`, with one row per performance metric and two columns:

**measure** The name of each performance metric. The reported metrics include: sparsity, Frobenius norm loss, Kullback-Leibler divergence, quadratic norm loss, spectral norm loss, true positive, true negative, false positive, false negative, true positive rate, false positive rate, F1 score, and Matthews correlation coefficient.

**value** The corresponding numeric value.

## Examples

```
library(grasps)

## reproducibility for everything
set.seed(1234)

## block-structured precision matrix based on SBM
sim <- gen_prec_sbm(p = 30, K = 3,
                    within.prob = 0.25, between.prob = 0.05,
                    weight.dists = list("gamma", "unif"),
                    weight.paras = list(c(shape = 20, rate = 10),
                                        c(min = 0, max = 5)),
                    cond.target = 100)
## ground truth visualization
plot(sim)

## n-by-p data matrix
library(MASS)
X <- mvrnorm(n = 20, mu = rep(0, 30), Sigma = sim$Sigma)

## precision matrix: adaptive lasso; BIC
prec <- grasps(X = X, membership = sim$membership, penalty = "adapt", crit = "BIC")
```

```
## precision matrix visualization
plot(prec)

## performance
performance(hatOmega = prec$hatOmega, Omega = sim$Omega)

## adjacency matrix: diagonal = 0; raw partial correlations;
##                   no thresholding; weighted network
adj <- prec_to_adj(prec$hatOmega,
                   diag.zero = TRUE, absolute = FALSE,
                   threshold = NULL, weighted = TRUE)

## adjacency matrix visualization
plot(adj)
```

---

`plot.adjmat` *Plot Function for S3 Class "adjmat"*

---

## Description

Visualize an adjacency matrix as a heatmap. This function is shared by objects returned from `prec_to_adj`.

## Usage

```
## S3 method for class 'adjmat'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object inheriting from S3 class `"adjmat"`, typically returned by `prec_to_adj`. |
| `...` | Additional arguments passed to `ggplot`. |

## Value

A heatmap of class `ggplot` showing the matrix entries. The plot title also reports matrix dimension and sparsity.

## Examples

```
library(grasps)

## reproducibility for everything
set.seed(1234)

## block-structured precision matrix based on SBM
sim <- gen_prec_sbm(p = 30, K = 3,
                    within.prob = 0.25, between.prob = 0.05,
```

```
                    weight.dists = list("gamma", "unif"),
                    weight.paras = list(c(shape = 20, rate = 10),
                                        c(min = 0, max = 5)),
                    cond.target = 100)
## ground truth visualization
plot(sim)

## n-by-p data matrix
library(MASS)
X <- mvrnorm(n = 20, mu = rep(0, 30), Sigma = sim$Sigma)

## precision matrix: adaptive lasso; BIC
prec <- grasps(X = X, membership = sim$membership, penalty = "adapt", crit = "BIC")

## precision matrix visualization
plot(prec)

## performance
performance(hatOmega = prec$hatOmega, Omega = sim$Omega)

## adjacency matrix: diagonal = 0; raw partial correlations;
##                    no thresholding; weighted network
adj <- prec_to_adj(prec$hatOmega,
                   diag.zero = TRUE, absolute = FALSE,
                   threshold = NULL, weighted = TRUE)

## adjacency matrix visualization
plot(adj)
```

---

`plot.blkmat` *Plot Function for Block-Structured Precision Matrices (Visualize a Matrix with Group Boundaries)*

---

## Description

Visualize a precision matrix as a heatmap with dashed boundary lines separating group blocks. This function is shared by objects returned from `grasps`, `gen_prec_sbm`, and `sparsify_block_banded`, all of which inherit from the S3 class `"blkmat"`.

## Usage

```
## S3 method for class 'blkmat'
plot(x, colors = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object inheriting from S3 class `"blkmat"`, typically returned by `grasps`, `gen_prec_sbm` or `sparsify_block_banded`. |
| `colors` | A vector of colors specifying an n-color gradient scale for the fill aesthetics. |
| `...` | Additional arguments passed to `ggplot`. |

## Value

A heatmap of class ggplot showing the matrix entries. Dashed lines indicate group boundaries. The plot title also reports matrix dimension and sparsity.

## Examples

```
library(grasps)

## reproducibility for everything
set.seed(1234)

## block-structured precision matrix based on SBM
sim <- gen_prec_sbm(p = 100, K = 10,
                    within.prob = 0.2, between.prob = 0.05,
                    weight.dists = list("gamma", "unif"),
                    weight.paras = list(c(shape = 100, scale = 10),
                                        c(min = 0, max = 5)),
                    cond.target = 100)

## visualization
plot(sim)
```

---

# plot.penderiv *Plot Function for S3 Class "penderiv"*

---

## Description

Generate a visualization of penalty functions produced by compute_penalty, or penalty derivatives produced by compute_derivative. The plot automatically summarizes multiple configurations of penalty type, $\lambda$, and $\gamma$. Optional zooming is supported through facet_zoom.

## Usage

```
## S3 method for class 'penderiv'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| x | An object inheriting from S3 class "penderiv", typically returned by compute_penalty, or compute_derivative. |
| ... | Optional arguments passed to facet_zoom to zoom in on a subset of the data, while keeping the view of the full dataset as a separate panel. |

## Value

An object of class ggplot.

## Examples

```
library(grasps)
library(ggplot2)

pen_df <- compute_penalty(
  omega = seq(-4, 4, by = 0.01),
  penalty = c("atan", "exp", "lasso", "lq", "lsp", "mcp", "scad"),
  lambda = 1)

plot(pen_df, xlim = c(-1, 1), ylim = c(0, 1), zoom.size = 1) +
  guides(color = guide_legend(nrow = 2, byrow = TRUE))

deriv_df <- compute_derivative(
  omega = seq(0, 4, by = 0.01),
  penalty = c("atan", "exp", "lasso", "lq", "lsp", "mcp", "scad"),
  lambda = 1)

plot(deriv_df) +
  scale_y_continuous(limits = c(0, 1.5)) +
  guides(color = guide_legend(nrow = 2, byrow = TRUE))
```

---

# prec_to_adj *Adjacency Matrix from Precision Matrix*

---

## Description

Convert a precision matrix to a partial-correlation-based adjacency matrix.

## Usage

```
prec_to_adj(
  prec.mat,
  diag.zero = TRUE,
  absolute = FALSE,
  threshold = NULL,
  weighted = TRUE
)
```

## Arguments

| | |
|---|---|
| prec.mat | A numeric precision matrix. |
| diag.zero | A logical value (default = TRUE) specifying whether to set the diagonal entries of the adjacency matrix to 0. If diag.zero = FALSE, the diagonal entries are set to 1 for a weighted network. For an unweighted network (weighted = FALSE), the diagonal is always forced to 0 to avoid self-loops. |
| absolute | A logical value (default = FALSE) specifying whether to take the absolute values of the partial correlations. |

`threshold` A nonnegative numeric value (default = `NULL`) specifying the threshold for edge filtering. Entries with absolute values smaller than the threshold are set to 0.

`weighted` A logical value (default = TRUE) specifying whether to return a weighted adjacency matrix. If `weighted` = `FALSE`, the matrix is a binary adjacency matrix with entries equal to 0 or 1.

## Details

For a precision matrix $\Omega$, the partial correlation between nodes $i$ and $j$ is computed as

$$\rho_{ij} = -\frac{\Omega_{ij}}{\sqrt{\Omega_{ii}\Omega_{jj}}}.$$

## Value

A numeric adjacency matrix with S3 class `"adjmat"`.

## Examples

```
library(grasps)

## reproducibility for everything
set.seed(1234)

## block-structured precision matrix based on SBM
sim <- gen_prec_sbm(p = 30, K = 3,
                    within.prob = 0.25, between.prob = 0.05,
                    weight.dists = list("gamma", "unif"),
                    weight.paras = list(c(shape = 20, rate = 10),
                                        c(min = 0, max = 5)),
                    cond.target = 100)
## ground truth visualization
plot(sim)

## n-by-p data matrix
library(MASS)
X <- mvrnorm(n = 20, mu = rep(0, 30), Sigma = sim$Sigma)

## precision matrix: adaptive lasso; BIC
prec <- grasps(X = X, membership = sim$membership, penalty = "adapt", crit = "BIC")

## precision matrix visualization
plot(prec)

## performance
performance(hatOmega = prec$hatOmega, Omega = sim$Omega)

## adjacency matrix: diagonal = 0; raw partial correlations;
##                   no thresholding; weighted network
adj <- prec_to_adj(prec$hatOmega,
                   diag.zero = TRUE, absolute = FALSE,
                   threshold = NULL, weighted = TRUE)
```

```
## adjacency matrix visualization
plot(adj)
```

---

`sparsify_block_banded` *Groupwise Block-Banded Sparsifier*

---

## Description

Make a precision-like matrix block-banded according to group membership, keeping only entries within specified group neighborhoods.

## Usage

```
sparsify_block_banded(mat, membership, neighbor.range = 1)
```

## Arguments

| | |
|---|---|
| `mat` | A $p \times p$ precision-like matrix specifying the base matrix to be masked. |
| `membership` | An integer vector specifying the group membership. The length of `membership` must be consistent with the dimension $p$. |
| `neighbor.range` | An integer (default = 1) specifying the neighbor range, where groups whose labels differ by at most `neighbor.range` are considered neighbors and kept in the mask. |

## Value

An object with S3 class `"sparsify_block_banded"` containing the following components:

**Omega** The masked precision matrix.

**Sigma** The covariance matrix, i.e., the inverse of `Omega`.

**sparsity** Proportion of zero entries in `Omega`.

**membership** An integer vector specifying the group membership.

## Examples

```
library(grasps)

## reproducibility for everything
set.seed(1234)

## precision matrix estimation
X <- matrix(rnorm(200), 10, 20)
membership <- c(rep(1,5), rep(2,5), rep(3,4), rep(4,6))
est <- grasps(X, membership = membership, penalty = "lasso", crit = "BIC")

## default: keep blocks within ±1 of each group
res1 <- sparsify_block_banded(est$hatOmega, membership, neighbor.range = 1)
```

```
plot(res1)

## wider band: keep blocks within ±2 of each group
res2 <- sparsify_block_banded(est$hatOmega, membership, neighbor.range = 2)
plot(res2)

## special case: block-diagonal matrix
res3 <- sparsify_block_banded(est$hatOmega, membership, neighbor.range = 0)
plot(res3)
```

# Index