Graphical Lasso

Viviana Gavilanes y Mikael Ketema

2024-02-08

Introduction

The graphical lasso algorithm is a powerful tool for estimating sparse graphical models, specifically focusing on the precision matrix (Θ) of a multivariate Gaussian distribution with mean μ and covariance matrix Σ. By incorporating L_1 regularization, it promotes sparsity in Θ, which is instrumental in revealing the conditional independence structure among variables.

In recent years a number of authors have proposed the estimation of sparse undirected graphical models through the use of L_1 (lasso) regularization. The basic model for continuous data assumes that the observations have a multivariate Gaussian distribution If the ij th component of Σ^{-1} is zero, then variables i and j are conditionally independent, given the other variables. Thus, it makes sense to impose an L_1 penalty for the estimation of Σ^{-1} to increase its sparsity.

Sally has direct connections to both Mike and Joe, signifying friendship. BUT there is no direct edge between Mike and Joe, which implies that their attendance at the same party is not directly related to each other but is instead related through their mutual friend Sally.

This reflects the idea of conditional independence where the relationship between Mike and Joe attending the same event is independent, given their mutual friendship with Sally.

Implementation of Graphical Lasso

Theoretical Background and Optimization Problem

The core optimization problem in graphical lasso involves maximizing the penalized log-likelihood function of a Gaussian graphical model, where sparsity in the precision matrix $\Theta = \Sigma^{-1}$ is induced through L_1 regularization. This approach reflects the assumption that zeros in Θ represent conditional independencies between variables, providing a sparse graphical model of the data.

Optimization Problem

The optimization problem can be expressed as:

$$\max_{\Theta} \left\{ \log\det(\Theta) - \operatorname{tr}(S\Theta) - \rho\|\Theta\|_1 \right\}, \tag{1}$$

where:

- Θ is the precision matrix to be estimated,
- S is the empirical covariance matrix derived from the data,

- ρ is the regularization parameter controlling the sparsity level,
- $\log\det(\Theta)$ encourages the likelihood of the model,
- $\text{tr}(S\Theta)$ ensures the model fits the observed data,
- $\|\Theta\|_1$ is the L_1 norm of Θ, promoting sparsity.

Duality and Coordinate Descent

The solution to this optimization problem powers duality in optimization, translating the primal problem into a dual problem that facilitates a coordinate descent approach. Specifically, the dual formulation allows for iteratively solving a series of lasso regression problems, effectively decoupling the joint optimization into manageable subproblems.

- Remark: The optimization problem of the graphical lasso is convex, which is crucial for ensuring that any local optimum is also a global optimum. The objective function combines the log-determinant of Θ, which encourages the model to fit the data, with the L_1 penalty term, which encourages sparsity in the estimated matrix.

Solution via Dual Problem

The dual formulation of this optimization problem allows for a coordinate descent approach, breaking down the complex problem into simpler lasso regression tasks:

$$\min_{\beta} \left\{ \frac{1}{2} \|W_{11}^{1/2}\beta - b\|^2 + \rho\|\beta\|_1 \right\}, \tag{2}$$

where $b = W_{11}^{-1/2} s_{12}$ and W_{11} is a submatrix of W, excluding the variable of interest.

Detailed Explanation of $\hat{\beta}$ and Optimization in Graphical Lasso

A key step in this optimization involves solving for $\hat{\beta}$, which represents the coefficients of a lasso regression problem for each variable against all others, under a specific regularization parameter, ρ.

Updating $\hat{\beta}$ and Θ

For each variable, the algorithm updates $\hat{\beta}$ using the coordinate descent method:

$$\hat{\beta}_j \leftarrow \frac{S(u_j - \sum_{k \neq j} V_{kj}\hat{\beta}_k, \rho)}{V_{jj}}, \tag{3}$$

with $V = W_{11}$, $u = s_{12}$, and S being the soft-thresholding operator,

$$S(x, t) = \text{sign}(x)(|x| - t)_+$$

. The updates for $\hat{\beta}$ are then used to revise W and indirectly Θ, through the relation $W_{12} = W_{11}\hat{\beta}$ and $\Theta_{12} = -\Theta_{22}\hat{\beta}$, ensuring that the optimization respects the graphical model's structure and the sparsity induced by ρ.

Convergence and Output

The iterative process continues until the change in W (or equivalently, Θ) falls below a predefined threshold, signaling convergence to a solution that maximizes the penalized log-likelihood while adhering to the sparsity constraints imposed by the L_1 penalty.

Algorithm: Coordinate Descent LASSO

- **Input**: W11 (matrix), s12 (vector), rho (scalar), max_iter (integer), tol (tolerance level)
- **Output**: beta (vector of coefficients)

1. Initialize the length of s12 to `p`.
2. Create a vector `beta` of length `p`, initialized with zeros.
3. For `iter` from 1 to `max_iter`:
    a. Copy `beta` to `beta_old`.
    b. For each element `j` in 1 to `p`:
        i. Define the soft-thresholding operator `S` as a function that applies the soft-thresholding rule given `x` and `lambda`.
        ii. Calculate residual `r_j` for `j`th element by subtracting the dot product of `W11` excluding `j`th column with `beta` excluding `j`th element from `s12[j]`.
        iii. Update `beta[j]` using the soft-thresholding operator `S` with `r_j` and `rho`, divided by `W11[j, j]`.
    c. If the Euclidean norm of the difference between `beta` and `beta_old` is less than `tol`, break the loop.
4. Return `beta`.

Algorithm: Graphical LASSO

- **Input**: S (covariance matrix), rho (regularization parameter), threshold (convergence threshold), max_iter (maximum iterations)
- **Output**: Dictionary with `W` (updated precision matrix) and `Theta` (inverse of `W`)

1. Initialize `p` as the number of columns in `S`.
2. Initialize `W` by adding `rho` times identity matrix of size `p` to `S`.
3. For `iter` from 1 to `max_iter`:
    a. Copy `W` to `W_old`.
    b. For each element `j` in 1 to `p`:
        i. Partition `W` into `W11` (excluding `j`th row and column) and `s12` (excluding `j`th column).
        ii. Compute `beta_hat` by solving the LASSO problem using `coordinate_descent_lasso` with inputs `W11`, `s12`, and `rho`.
        iii. Update `W` by setting `W[-j, j]` and `W[j, -j]` to the dot product of `W11` and `beta_hat`.
    c. If the mean absolute difference between `W` and `W_old` is less than `threshold`, break the loop.
4. Invert `W` to obtain `Theta`, and set the column and row names of `Theta` to match those of `S`.
5. Return a list containing `W` and `Theta`.

```
coordinate_descent_lasso <- function(W11, s12, rho, max_iter = 10000, tol = 1e-4) {
  p <- length(s12)
  beta <- rep(0, p)
  for (iter in 1:max_iter) {
    beta_old <- beta
    for (j in 1:p) {
      # Soft-thresholding operator
      S <- function(x, lambda) sign(x) * max(abs(x) - lambda, 0)

      # Update rule for beta_j
```

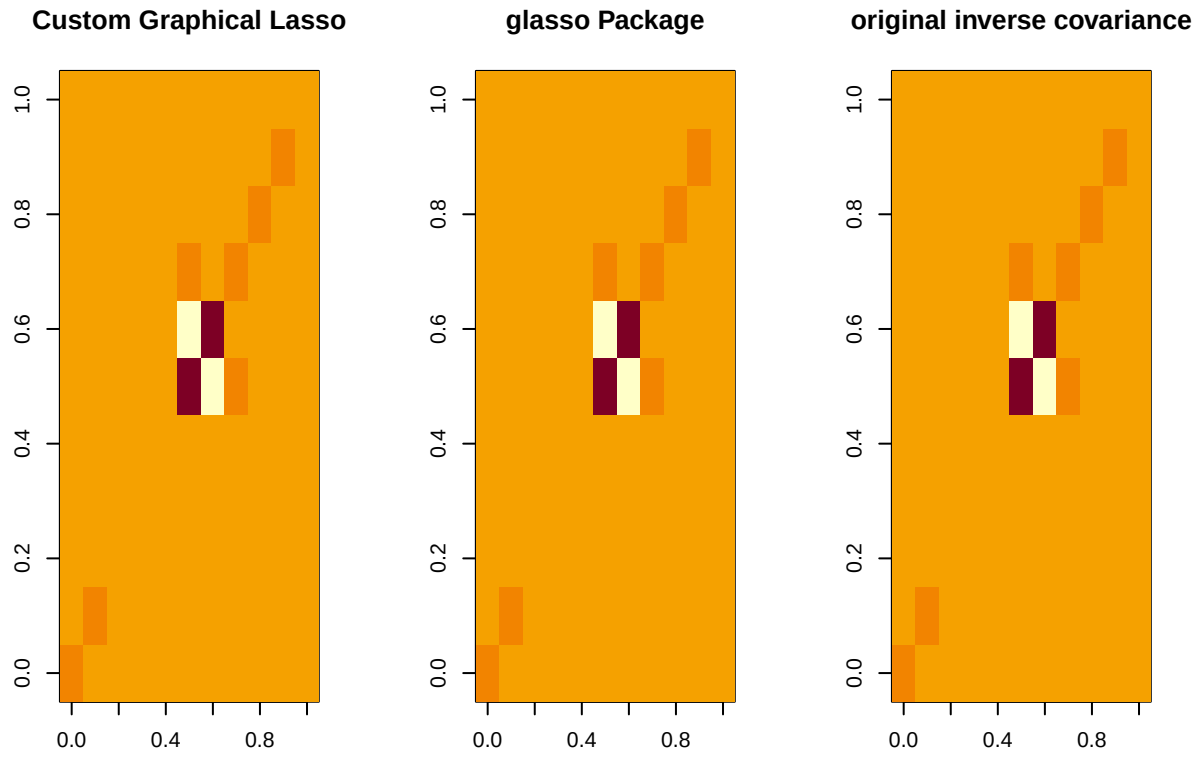
```
      r_j <- s12[j] - W11[-j, j] %*% beta[-j]
      beta[j] <- S(r_j, rho) / W11[j, j]
    }
    # Check for convergence
    if (sqrt(sum((beta - beta_old)^2)) < tol) {
      break
    }
  }
  return(beta)
}


graphical_lasso <- function(S, rho, threshold = 0.001, max_iter = 10000) {
  p <- ncol(S)
  W <- S + rho * diag(p)
  for (iter in 1:max_iter) {
    W_old <- W
    for (j in 1:p) {
      # Partition W and S
      W11 <- W[-j, -j]
      s12 <- S[-j, j]

      # Solve the lasso problem for beta_hat
      beta_hat <- coordinate_descent_lasso(W11, s12, rho)

      # Update W
      W[-j, j] <- W11 %*% beta_hat
      W[j, -j] <- W[-j, j]
    }

    # Check for convergence
    if (mean(abs(W - W_old)) < threshold) {
      break
    }
  }

  # Invert W to get Theta
  Theta <- solve(W)
  colnames(Theta) <- colnames(S)
  rownames(Theta) <- rownames(S)
  return(list(W = W, Theta = Theta))
}
```

Contrast between our implementation and the glasso from glasso package using the Sachs Data.

```
X <- read.table("sachs.data.txt", header = TRUE, sep = "", dec = ".")
X <- scale(X)
head(X, 5)
```

```
##               Raf        Mek       Plcg       PIP2       PIP3         Erk
## [1,] -0.786589530 -0.6131178 -0.7275215 -0.6764100  0.8286647 -0.17324690
## [2,] -0.559558724 -0.4928938 -0.4903494 -0.6924503 -0.6528526 -0.03986437
## [3,]  0.002043796  0.5126155 -0.3335977 -0.7630278 -0.5104608 -0.08102495
## [4,]  0.327056317  1.9225145  0.2457020 -0.7277391 -0.8528442 -0.18192399
## [5,] -0.612134279 -0.3726699 -0.9749166 -0.7680538 -0.1654460 -0.01205317
##              Akt         PKA        PKC        P38         Jnk
## [1,] -0.19723359 -0.35769988  0.1708973  0.5599682  0.03577374
## [2,] -0.07485901 -0.50262745 -1.0049193 -0.9022981  0.53469193
## [3,] -0.07485901 -0.38341284 -0.3121968 -0.1093790 -0.43993896
## [4,] -0.23828829 -0.09122016 -0.1137832 -0.2792903 -0.35639917
## [5,]  0.03251481 -0.61249190 -0.8936351 -0.4286062  0.99416077
```

```r
# Install and load the glasso package install.packages("glasso")
library(glasso)

rho <- 0.00001

S <- cov(X)

# Run your custom graphical lasso implementation
custom_result <- graphical_lasso(S, rho)

# Run the glasso package implementation
glasso_result <- glasso::glasso(S, rho)

# Qualitative comparison: Visualize the sparsity patterns
#windows()
par(mfrow=c(1,3))
image(custom_result$Theta, main="Custom Graphical Lasso")
image(glasso_result$wi, main="glasso Package")
image(solve(S), main="original inverse covariance")
```



Ploting of the graphs

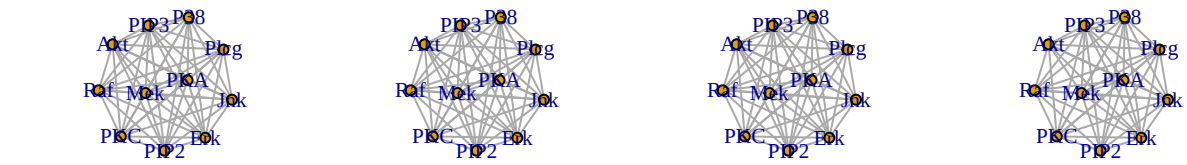
```
create_graph <- function(theta) {
  graph <- graph_from_adjacency_matrix(abs(theta) > 1e-4, mode = "undirected", diag = FALSE)
  return(graph)
}


#rhos <- c(0.00000001, 0.01,1, 20, 50)  #
rhos <- c(0.0001, 0.001, 0.01, 0.1, 0.2,0.3)
#Example values
graphs_custom <- list()
graphs_glasso <- list()

plot_custom <- list()
plot_glasso <- list()

for (rho in rhos) {
  custom_result <- graphical_lasso(S, rho, threshold = 0.1, max_iter = 30)
  glasso_result <- glasso::glasso(S, rho, thr = 0.1, maxit = 30)

  # Extraction of the precision matrix

  precision_matrix_custom <- custom_result$Theta
```

```
  precision_matrix_glasso <- glasso_result$wi

  colnames(precision_matrix_glasso) <- colnames(S)
  rownames(precision_matrix_glasso) <- rownames(S)

  # Convert the precision matrix into a data frame
  df_precision_custom <- melt(precision_matrix_custom)
  df_precision_glasso <- melt(precision_matrix_glasso)

  # Creation of the ploting precision matrix
  g_custom <- ggplot(df_precision_custom, aes(Var1, Var2, fill = value)) +
    geom_tile() +
    labs(title = paste("Precision Matrix (our glasso, rho =", rho, ")"),
         x = "Variables",
         y = "Variables") +
    scale_fill_gradientn(colours = heat.colors(12)) +
    theme_minimal() +
    theme(axis.title.x = element_text(size = 0.05),
          axis.title.y = element_text(size = 0.05),
          plot.title = element_text(size = 10))

    g_glasso <- ggplot(df_precision_glasso, aes(Var1, Var2, fill = value)) +
    geom_tile() +
    labs(title = paste("Precision Matrix (glasso, rho =", rho, ")"),
         x = "Variables",
         y = "Variables") +
    scale_fill_gradientn(colours = heat.colors(12)) +
    theme_minimal() +
    theme(axis.title.x = element_text(size = 0.05),
          axis.title.y = element_text(size = 0.05),
          plot.title = element_text(size = 10)
          )

    #plot(g_custom, cex.axis = 0.8, cex.lab = 0.8, cex.main = 0.8)
    #plot(g_glasso, cex.axis = 0.8, cex.lab = 0.8, cex.main = 0.8)

    # Adding graphiques to the lists
    plot_custom[[as.character(rho)]] <- g_custom
    plot_glasso[[as.character(rho)]] <- g_glasso

  graphs_custom[[as.character(rho)]] <- create_graph(precision_matrix_custom)
  graphs_glasso[[as.character(rho)]] <- create_graph(precision_matrix_glasso)
}
```

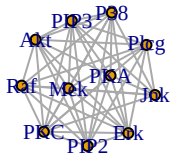
Ploting graphs

```
# Generate a consistent layout for all plots
layout <- layout_with_fr(graphs_custom[[as.character(rhos[1])]])

# Open a new graphics window
#windows()
```

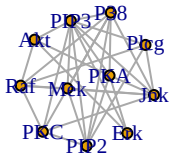
```
# Set up the plotting area for a 4x6 grid
par(mfrow=c(1, 4))

# Plot each graph
for (rho in rhos) {
  plot(graphs_custom[[as.character(rho)]], layout=layout, main=paste("Our glasso (rho =", rho, ")"))
  plot(graphs_glasso[[as.character(rho)]], layout=layout, main=paste("glasso (rho =", rho, ")"))
}
```



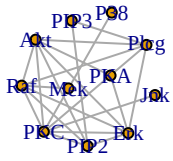
Our glasso (rho = 0.01



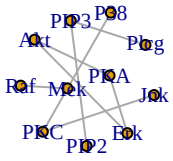
glasso (rho = 0.01)



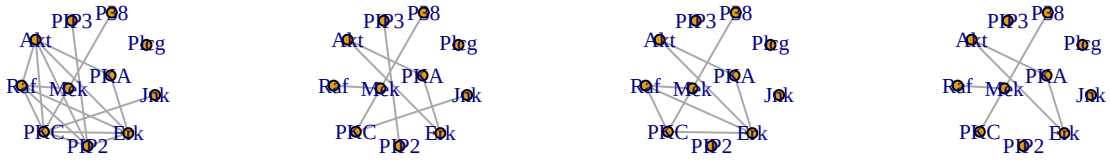
Our glasso (rho = 0.1



glasso (rho = 0.1)

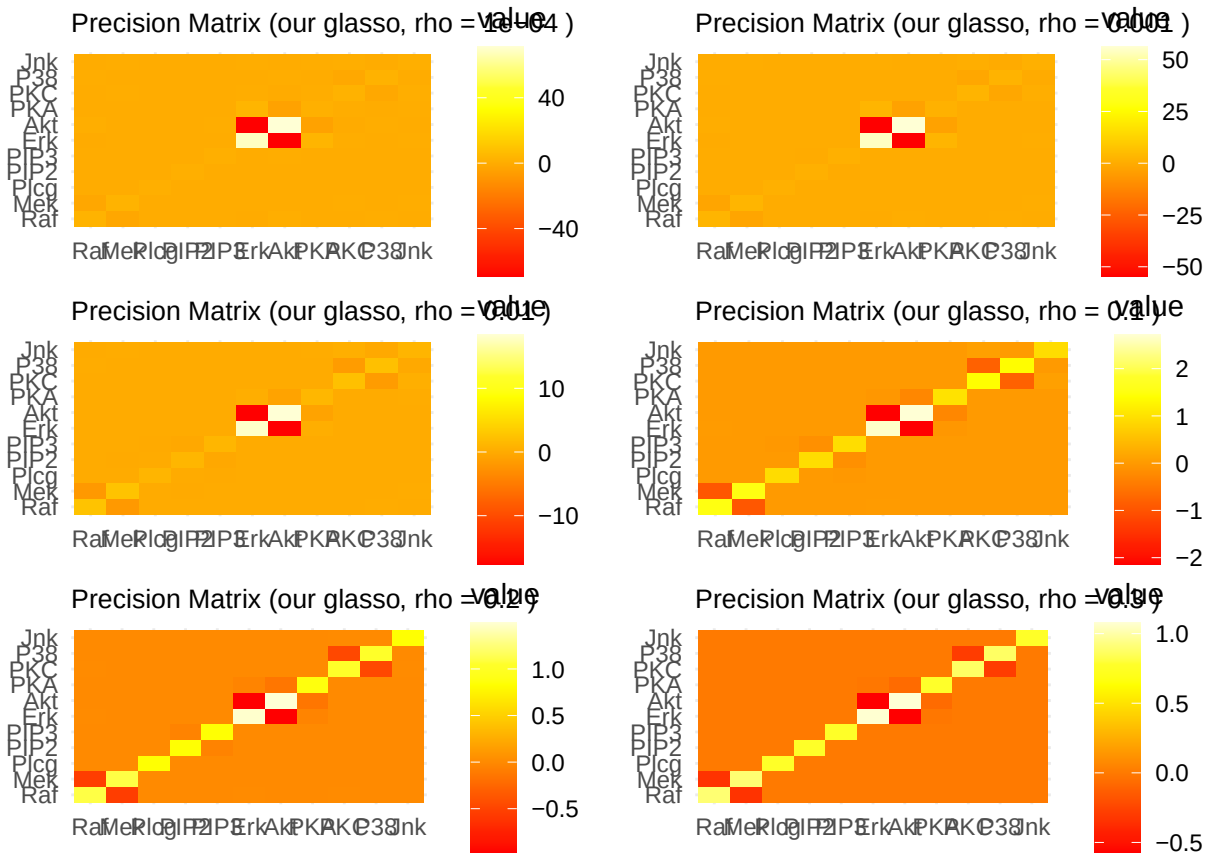


Our glasso (rho = 0.2 **glasso (rho = 0.2)** **Our glasso (rho = 0.3** **glasso (rho = 0.3)**

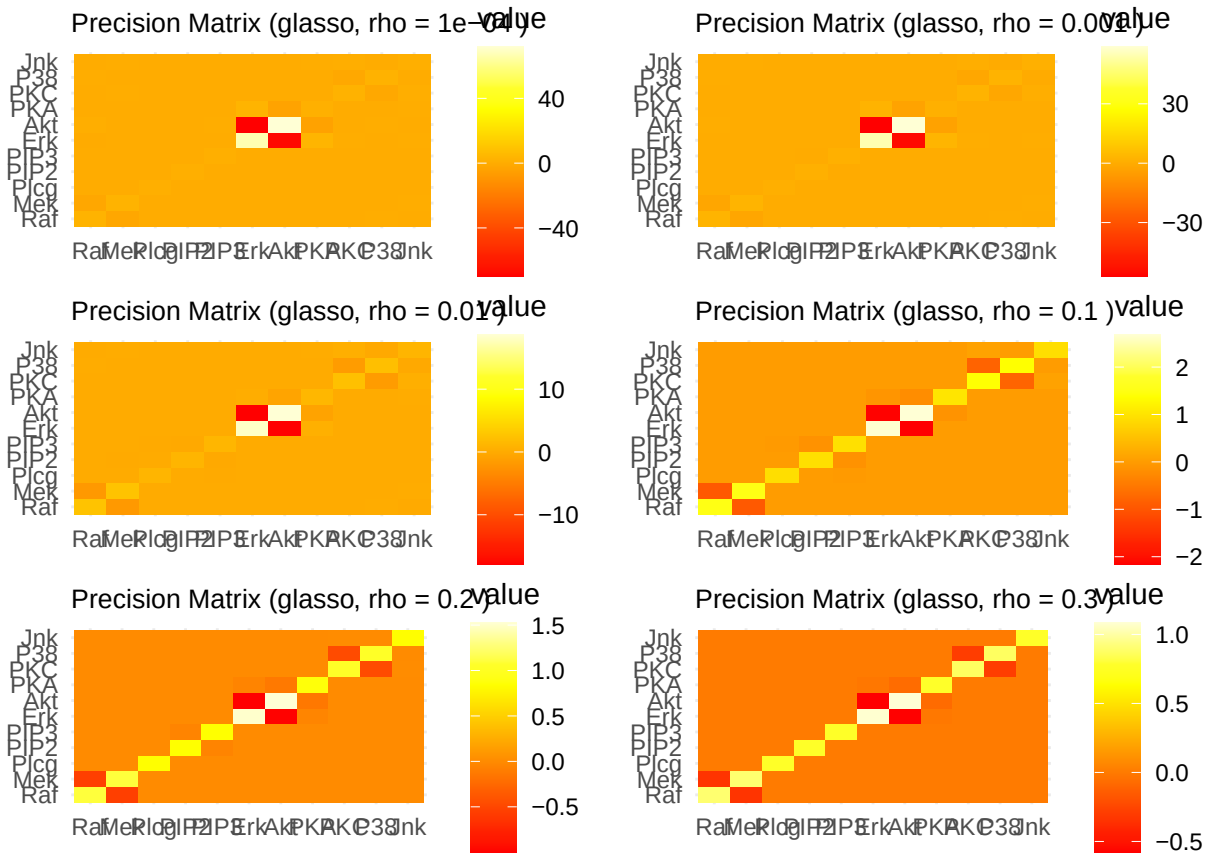


Ploting precision matrix

```
# Combination of graphiques in the same figure with grid.arrange()
grid.arrange(grobs = plot_custom, ncol = 2)
```




```
grid.arrange(grobs = plot_glasso, ncol = 2)
```



Selecting the Best ρ in Graphical Lasso

The regularization parameter, ρ, plays a critical role in controlling the sparsity of the estimated Θ. Selecting an optimal value for ρ is crucial for balancing model complexity with goodness of fit to the data. This selection process can be guided by criteria such as the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), and the model's log-likelihood.

Log-Likelihood

The log-likelihood of the data under the estimated precision matrix is given by:

$$\text{LogLik} = -\frac{n}{2}\left(\text{tr}(S\Theta) - \log|\Theta| - p\right), \tag{4}$$

where S is the empirical covariance matrix of the observed data, n is the number of observations, p is the number of variables, $\text{tr}(S\Theta)$ is the trace of the product of S and Θ, and $|\Theta|$ is the determinant of the precision matrix. This metric measures how well the model, characterized by Θ, explains the observed data.

Akaike Information Criterion (AIC)

The AIC is calculated as:

$$\text{AIC} = 2k - 2\text{LogLik}, \tag{5}$$

where k is the number of non-zero parameters in Θ. The AIC penalizes model complexity to prevent overfitting, with lower values indicating a better model.

Bayesian Information Criterion (BIC)

The BIC introduces a stronger penalty for model complexity and is given by:

$$\text{BIC} = \log(n)k - 2\text{LogLik}. \tag{6}$$

Similar to the AIC, lower BIC values are preferred, but the BIC tends to favor simpler models due to its heavier penalty on the number of parameters.

Model Selection

The selection of the best ρ involves computing the AIC, BIC, and log-likelihood for a range of ρ values. The ρ that minimizes the AIC or BIC is considered optimal, indicating a good balance between the fit of the model to the data and its complexity. This process ensures that the selected model is neither too simplistic to capture the essential structure of the data nor too complex to generalize well to new observations.

In summary, the choice of ρ in graphical lasso is a critical step that affects the model's performance. By carefully evaluating the AIC, BIC, and log-likelihood, one can select a ρ that leads to a model which is both interpretable and predictive, providing valuable insights into the underlying dependencies among the variables.

```r
select_rho <- function(S, rho_range, penalize.diagonal = TRUE, threshold = 0.001, max_iter = 10000) {
  # Initialize a data frame to store the results
  results <- data.frame(rho = numeric(length(rho_range)),
                        AIC = numeric(length(rho_range)),
                        BIC = numeric(length(rho_range)),
                        LogLik = numeric(length(rho_range)))

  n <- nrow(S) # Number of observations, assuming S is square (p = ncol(S))

  for (i in seq_along(rho_range)) {
    rho <- rho_range[i]
    glasso_result <- graphical_lasso(S, rho, threshold, max_iter)
    Theta <- glasso_result$Theta

    # Log-Likelihood calculation
    loglik <- -n/2 * (sum(diag(S %*% Theta)) - log(det(Theta)))

    # Number of nonzero parameters in Theta
    if (penalize.diagonal) {
      k <- sum(Theta != 0) # If penalizing diagonal, count all non-zero elements
    } else {
      k <- sum(Theta[upper.tri(Theta)] != 0) + sum(Theta[lower.tri(Theta)] != 0) # Count only off-diago
    }

    # AIC and BIC calculations
    aic <- 2 * k - 2 * loglik
    bic <- log(n) * k - 2 * loglik
```

```
    # Store results in the data frame
    results[i, ] <- c(rho, aic, bic, loglik)
  }

  colnames(results) <- c("Rho", "AIC", "BIC", "LogLik")

  return(results)
}



rho_range <- seq(0.0001, 0.5, by = 0.05)


select_rho <- function(S, rho_range, threshold=0.001, max_iter=100) {
  results <- data.frame(rho = numeric(length(rho_range)),
                        AIC = numeric(length(rho_range)),
                        BIC = numeric(length(rho_range)),
                        LogLik = numeric(length(rho_range)))

  n <- nrow(S) # Number of observations

  for (i in seq_along(rho_range)) {
    rho <- rho_range[i]
    glasso_result <- graphical_lasso(S, rho, threshold, max_iter)
    Theta <- glasso_result$Theta

    # Improved Log-Likelihood calculation
    loglik <- -n/2 * (sum(diag(S %*% Theta)) - log(det(Theta)) - ncol(S))

    # Count of non-zero off-diagonal elements in Theta for penalized parameters
    k <- sum(Theta[lower.tri(Theta)] != 0)

    # AIC and BIC calculations (make sure to use the correct formula and counts)
    aic <- 2 * k - 2 * loglik
    bic <- log(n) * k - 2 * loglik

    # Store results
    results[i, ] <- c(rho, aic, bic, loglik)
  }

  colnames(results) <- c("Rho", "AIC", "BIC", "LogLik")
  return(results)
}
# Find the best rho value
results_selecting_rho <- select_rho(S, rho_range)
```

```
kable(results_selecting_rho, caption = "Values of AIC, BIC and LogLikelihood") %>%
  kable_styling(position = "center", latex_options = "hold_position", font_size = 12)
```

Table 1: Values of AIC, BIC and LogLikelihood

Rho	AIC	BIC	LogLik
0.0001	38.88968	60.77392	35.555160
0.0501	60.94249	82.82673	24.528754
0.1001	67.00759	88.09604	19.496204
0.1501	58.47715	76.38244	15.761423
0.2001	58.74591	75.45751	12.627046
0.2501	50.16267	64.08901	9.918665
0.3001	24.94692	32.90482	7.526542
0.3501	27.25582	34.81583	5.372090
0.4001	33.20246	41.16037	3.398770
0.4501	26.81030	32.77873	1.594848

Simulate Gaussian Data - Timing COMPARISONS

We simulated Gaussian data from both sparse and dense scenarios, for a range of problem sizes p. The sparse scenario is the AR(1) model taken from Yuan and Lin (2007): $\left(\Sigma^{-1}\right)_{ii} = 1, \left(\Sigma^{-1}\right)_{i,i-1} = \left(\Sigma^{-1}\right)_{i-1,i} = 0.5$, and zero otherwise. In the dense scenario, $\left(\Sigma^{-1}\right)_{ii} = 2, \left(\Sigma^{-1}\right)_{ii'} = 1$ otherwise. All timings were carried out on a Asus Zenbook.

The next table shows the comparition between the graphical lasso programmed by us and the glasso. It is clear that the glasso is more efficient in times of computational time, it was implemented in fortran. The computation time is $O\left(p^3\right)$ for dense problems and considerably less than that for sparse problems. Even in the dense scenario, the glasso solves a 1000-node problem ($\sim$ 500000 parameters) in about a minute, and our implementation in less than two minutes. However, the computation time depends strongly on the value of ρ.

Timing results for dense scenario, $p = 400$, for different values of the regularization parameter ρ. The middle column is the number of nonzero coefficients.

```
library(glasso)
library(MASS)

generate_sparse_AR1 <- function(p) {
  # Precision matrix for sparse AR(1) scenario
  sigma_inv <- diag(1, p) # Initialize with 1s on the diagonal
  for(i in 2:p){
    sigma_inv[i, i-1] <- sigma_inv[i-1, i] <- 0.5 # Set off-diagonal elements
  }
  sigma <- solve(sigma_inv) # Invert to get the covariance matrix
  return(sigma)
}

generate_dense_scenario <- function(p) {
  # Precision matrix for dense scenario
  sigma_inv <- matrix(1, p, p) # All elements set to 1
  diag(sigma_inv) <- 2 # Diagonal elements set to 2
  sigma <- solve(sigma_inv) # Invert to get the covariance matrix
  return(sigma)
}
```

```r
p <- 400
n <- 400  # Number of observations; adjust as needed
rho_values <- c(0.01, 0.03, 0.06, 0.60)  # Varying values of rho

# Initialize an empty data frame to store the results
results <- data.frame(rho = numeric(), fraction_nonzero = numeric(), cpu_time_glasso = numeric(), cpu_t

# Generate data for the dense scenario
dense_data <- mvrnorm(n, mu = rep(0, p), Sigma = generate_dense_scenario(p))

for (rho in rho_values) {
  # Run glasso
  start_time <- Sys.time()
  glasso_res <- glasso(cov(dense_data), rho = rho)
  end_time <- Sys.time()
  cpu_time_glasso <- end_time - start_time
  fraction_nonzero_glasso <- sum(glasso_res$wi != 0) / (p^2)

  # Run custom graphical_lasso
  start_time <- Sys.time()
  custom_res <- graphical_lasso(cov(dense_data), rho = rho)
  end_time <- Sys.time()
  cpu_time_custom <- end_time - start_time
  fraction_nonzero_custom <- sum(custom_res$Theta != 0) / (p^2)

  # Collect results
  results <- rbind(results, data.frame(rho = rho, fraction_nonzero = fraction_nonzero_glasso, cpu_time_g
}
```

```r
kable(results, caption = "Timing results for dense scenario") %>%
  kable_styling(position = "center", latex_options = "hold_position", font_size = 12)
```

Table 2: Timing results for dense scenario

rho	fraction_nonzero	cpu_time_glasso	cpu_time_custom
0.01	0.7497125	5.7965710 secs	1.6806218 mins
0.03	0.4719875	2.0640378 secs	0.7802730 mins
0.06	0.2128125	1.0953891 secs	0.4834625 mins
0.60	0.0025000	0.1546159 secs	0.1091952 mins

Sources

1. Friedman, J., Hastie, T., & Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. Statistics and Computing, 18(3), 303-313. https://doi.org/10.1007/s11222-008-9206-1
2. Banerjee, O., El Ghaoui, L., & d'Aspremont, A. (2008). Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. Journal of Machine Learning Research, 9, 485-516.
3. Meinshausen, N., & Bühlmann, P. (2006). High-dimensional graphs and variable selection with the lasso. The Annals of Statistics, 34(3), 1436-1462. https://doi.org/10.1214/009053606000000281

4. Ravikumar, P., Wainwright, M. J., Raskutti, G., & Yu, B. (2011). High-dimensional covariance estimation by minimizing l1-penalized log-determinant divergence. Electronic Journal of Statistics, 5, 935-980. https://doi.org/10.1214/11-EJS631

5. Yuan, M., & Lin, Y. (2007). Model selection and estimation in the Gaussian graphical model. Biometrika, 94(1), 19-35. https://doi.org/10.1093/biomet/asm018