

Simulation Examples for Graphical Models

Package Setup

```
if (!require("devtools", quietly = TRUE))  
  install.packages("devtools")
```

Warning: package 'usethis' was built under R version 4.4.1

```
devtools::install_github("anonstats123/Nullstrap")
```

Skipping install of 'Nullstrap' from a github remote, the SHA1 (4519a73e) has not changed since last install.

Use `force = TRUE` to force installation

```
library(PRRoc)  
library(Nullstrap)  
library(MASS)
```

Warning: package 'MASS' was built under R version 4.4.1

```
library(SILGGM)
```

Loading required package: Rcpp

Warning: package 'Rcpp' was built under R version 4.4.1

Simulation Data Generation

```
generate_precision_matrix <- function(p, b, graph_type) {  
  # Initialize Omega matrix  
  Omega_0 <- matrix(0, nrow = p, ncol = p)  
  
  if (graph_type == "band") {  
    # Band graph  
    diag(Omega_0) <- 1  
    for (i in 1:(p-1)) {  
      for (j in (i+1):min(p, i+10)) {  
        Omega_0[i, j] <- sign(b) * abs(b)^(abs(i - j) / 10)  
        Omega_0[j, i] <- Omega_0[i, j] # Symmetric matrix  
      }  
    }  
  }  
  
  } else if (graph_type == "block") {  
    # Block graph: 10 blocks, each block size 20  
    for (k in seq(1, p, by = 20)) {
```

```

    block_size <- min(20, p - k + 1)
    Omega_0[k:(k+block_size-1), k:(k+block_size-1)] <- 1
    Omega_0[k:(k+block_size-1), k:(k+block_size-1)] <- b
  }

} else if (graph_type == "erdos") {
  # Erdos-Renyi graph
  diag(Omega_0) <- 1
  for (i in 1:(p-1)) {
    for (j in (i+1):p) {
      if (rbinom(1, 1, 0.1)) {
        samples_1 <- runif(n, min = -0.6, max = -0.2) # [-0.6, -0.2]
        samples_2 <- runif(n, min = 0.2, max = 0.6)   # [0.2, 0.6]
        phi_ij = combined_samples <- sample(c(samples_1, samples_2), 1)
        # phi_ij <- sample(c(-0.6, -0.2, 0.2, 0.6), 1)
        Omega_0[i, j] <- Omega_0[j, i] <- phi_ij
      }
    }
  }
}

} else if (graph_type == "cluster") {
  # Cluster graph: 5 blocks, each block size 40
  for (k in seq(1, p, by = 40)) {
    block_size <- min(40, p - k + 1)
    Omega_0[k:(k+block_size-1), k:(k+block_size-1)] <- 1
    for (i in k:(k+block_size-1)) {
      if (i < (k+block_size-1)) { # in case j exceeds range
        for (j in (i+1):(k+block_size-1)) {
          if (rbinom(1, 1, 0.5)) {
            samples_1 <- runif(n, min = -0.6, max = -0.2) # [-0.6, -0.2]
            samples_2 <- runif(n, min = 0.2, max = 0.6)   # [0.2, 0.6]
            phi_ij = combined_samples <- sample(c(samples_1, samples_2), 1)
            Omega_0[i, j] <- Omega_0[j, i] <- phi_ij
          }
        }
      }
    }
  }
}

# Make the precision matrix positive definite
lambda_min <- eigen(Omega_0)$values[p]
Omega <- Omega_0 + (abs(lambda_min) + 0.5) * diag(p)

return(Omega)
}

generate_data <- function(n, p, Omega) {
  # Generate the precision matrix Omega
  # Omega <- generate_precision_matrix(p, b, graph_type)

```

```

# Generate the covariance matrix Sigma
Sigma <- solve(Omega)

# Generate n samples from N_p(0, Sigma)
data <- mvrnorm(n = n, mu = rep(0, p), Sigma = Sigma)

return(data)
}

# Function to obtain the true support of the precision matrix
obtain_true_support <- function(Omega, threshold = 1e-10) {
  # Obtain the true support of the precision matrix
  true_support = matrix(0, nrow = ncol(Omega), ncol = ncol(Omega))
  true_support[abs(Omega) > threshold] = 1
  return(true_support)
}

n <- 2000
p <- 200
edge_strength <- -0.8
Omega <- generate_precision_matrix(p, edge_strength, "band")
true_support = obtain_true_support(Omega)
Signal_index = which(true_support[upper.tri(true_support)] == 1)
true_labels <- true_support[upper.tri(true_support)]
X <- generate_data(n, p, Omega)
data_scale = scale(X)

```

Statistical Metrics Function

```

aupr = function(statistic) pr.curve(scores.class0 = statistic,
                                     weights.class0 = true_labels,
                                     curve = FALSE)$auc.integral

```

Nullstrap [↗](#)

```

result_Nullstrap <- nullstrap_filter(
  X = data_scale, fdr_value = 0.2, best_lambda = NULL, B_reps = NULL, model_type = "graph"
)

```

Loading required package: Matrix

Warning: package 'Matrix' was built under R version 4.4.1

Loaded glmnet 4.1-8

Warning: package 'survival' was built under R version 4.4.1

Warning: package 'eha' was built under R version 4.4.1

[illegible]

[illegible]

Conducting the graphical lasso (glasso) with lossless screening....in progress: 99%
Conducting the graphical lasso (glasso)....done.

```
Nullstrap_FDR <- length(which(result_Nullstrap$statistic[setdiff(1:length(result_Nullstra
Signal_index)] >=
result_Nullstrap$threshold)) / max(length(result_Nullstrap$s
Nullstrap_Power <- (length(which(result_Nullstrap$statistic[Signal_index] >=
result_Nullstrap$threshold))) / (length(Signal_index)
Nullstrap_AUPR <- aupr(result_Nullstrap$statistic)
cat("Nullstrap FDR:", Nullstrap_FDR, "\n")
```

Nullstrap FDR: 0.09904153

```
cat("Nullstrap Power:", Nullstrap_Power, "\n")
```

Nullstrap Power: 0.8699229

```
cat("Nullstrap AUPR:", Nullstrap_AUPR, "\n")
```

Nullstrap AUPR: 0.9519845

GFC-L

```
result_GFCL <- SILGGM(data_scale, method = "GFC_L", alpha = 0.2,
true_graph = Omega)
```

Use method "GFC_L"

Center each column.

Standardize each column.

Pre-calculate inner product matrixes.

Use default number of delta = 40

Perform global inference.

Use pre-specified level(s): 0.2

True graph is available.

Calculate Lasso of each variable with tuning parameters under each delta.

Record test statistics under each delta.

delta 1

Lasso for variable 1

Lasso for variable 101

delta 2

Lasso for variable 1

Lasso for variable 101

delta 3

Lasso for variable 1

Lasso for variable 101

delta 4

Lasso for variable 1

Lasso for variable 101
delta 5
Lasso for variable 1
Lasso for variable 101
delta 6
Lasso for variable 1
Lasso for variable 101
delta 7
Lasso for variable 1
Lasso for variable 101
delta 8
Lasso for variable 1
Lasso for variable 101
delta 9
Lasso for variable 1
Lasso for variable 101
delta 10
Lasso for variable 1
Lasso for variable 101
delta 11
Lasso for variable 1
Lasso for variable 101
delta 12
Lasso for variable 1
Lasso for variable 101
delta 13
Lasso for variable 1
Lasso for variable 101
delta 14
Lasso for variable 1
Lasso for variable 101
delta 15
Lasso for variable 1
Lasso for variable 101
delta 16
Lasso for variable 1
Lasso for variable 101
delta 17
Lasso for variable 1
Lasso for variable 101
delta 18
Lasso for variable 1
Lasso for variable 101
delta 19
Lasso for variable 1
Lasso for variable 101
delta 20
Lasso for variable 1
Lasso for variable 101
delta 21
Lasso for variable 1

Lasso for variable 101
delta 22
Lasso for variable 1
Lasso for variable 101
delta 23
Lasso for variable 1
Lasso for variable 101
delta 24
Lasso for variable 1
Lasso for variable 101
delta 25
Lasso for variable 1
Lasso for variable 101
delta 26
Lasso for variable 1
Lasso for variable 101
delta 27
Lasso for variable 1
Lasso for variable 101
delta 28
Lasso for variable 1
Lasso for variable 101
delta 29
Lasso for variable 1
Lasso for variable 101
delta 30
Lasso for variable 1
Lasso for variable 101
delta 31
Lasso for variable 1
Lasso for variable 101
delta 32
Lasso for variable 1
Lasso for variable 101
delta 33
Lasso for variable 1
Lasso for variable 101
delta 34
Lasso for variable 1
Lasso for variable 101
delta 35
Lasso for variable 1
Lasso for variable 101
delta 36
Lasso for variable 1
Lasso for variable 101
delta 37
Lasso for variable 1
Lasso for variable 101
delta 38
Lasso for variable 1

Lasso for variable 101
delta 39
Lasso for variable 1
Lasso for variable 101
delta 40
Lasso for variable 1
Lasso for variable 101
Choose delta for FDR control.

```
GFCL_Power = result_GFCL$power
GFCL_FDR = result_GFCL$FDR
GFCL_AUPR = aupr(abs(result_GFCL$T_stat[upper.tri(result_GFCL$T_stat)]))
cat("GFC-L FDR:", GFCL_FDR, "\n")
```

GFC-L FDR: 0.1829653

```
cat("GFC-L Power:", GFCL_Power, "\n")
```

GFC-L Power: 0.3994859

```
cat("GFC-L AUPR:", GFCL_AUPR, "\n")
```

GFC-L AUPR: 0.6647654

GFC-SL

```
result_GFCS <- SILGGM(data_scale, method = "GFC_SL", alpha = 0.2,
                      true_graph = Omega)
```

Use method '"GFC_SL"'

Use default $\lambda = \sqrt{2 \cdot \log(p/\sqrt{n})/n}$

In this case, $\lambda = 0.0387023$

Center each column.

Standardize each column.

Pre-calculate inner product matrixes.

Calculate scaled Lasso for each variable.

scaled Lasso for variable 1

scaled Lasso for variable 101

Perform global inference.

Use pre-specified level(s): 0.2

True graph is available.

```
GFCS_Power = result_GFCS$power
GFCS_FDR = result_GFCS$FDR
GFCS_AUPR = aupr(abs(result_GFCS$T_stat[upper.tri(result_GFCS$T_stat)]))
cat("GFC-SL FDR:", GFCS_FDR, "\n")
```

GFC-SL FDR: 0.1836735

```
cat("GFC-SL Power:", GFC_S_Power, "\n")
```

GFC-SL Power: 0.4113111

```
cat("GFC-SL AUPR:", GFC_S_AUPR, "\n")
```

GFC-SL AUPR: 0.6757225

KO2

```
source("./Knockoff.R")
trueNoneConnections <- which(true_support == 0)
ko.est <- GraphEstimation(data_scale, FDRtarget = 0.2, plus = FALSE)
```

Warning in pcor(dat): The inverse of variance-covariance matrix is calculated using Moore-Penrose generalized matrix invers due to its determinant of zero.

```
ko.adj <- ko.est$adjacency.matrix
if(dim(ko.est$edge.set)[1]==0){
  ko.err<-0
} else {
  ko.FP <- ko.adj[trueNoneConnections]
  ko.err <- length(which(ko.FP == 1))
}
ko2_FDR <- ko.err / max(1, length(which(ko.adj == 1)) )
ko2_Power <- (length(which(ko.adj == 1)) -
              ko.err) / max(1, length(which(true_support == 1)))
ko2_AUPR = aupr(abs(ko.est$continuous.raw[upper.tri(ko.est$continuous.raw)]))
cat("K02 FDR:", ko2_FDR, "\n")
```

K02 FDR: 0.211480362538

```
cat("K02 Power:", ko2_Power, "\n")
```

K02 Power: 0.255256723716

```
cat("K02 AUPR:", ko2_AUPR, "\n")
```

K02 AUPR: 0.557439098129

Data Splitting

```
source("./DS_graph.R")
source("./fdp_power_graph.R")
```

```
source("./analys_graph.R")
selected_ds <- DS_graph(data_scale, q = 0.2, num_split = 1)
result_ds = fdp_power_graph(selected_ds$DS_selected_edge, true_support)
ds_Power <- result_ds$power
ds_FDR <- result_ds$fdp
ds_AUPR = aupr(abs(selected_ds$DS_statistic[upper.tri(selected_ds$DS_statistic)]))
cat("Data Splitting FDR:", ds_FDR, "\n")
```

Data Splitting FDR: 0.192849949648

```
cat("Data Splitting Power:", ds_Power, "\n")
```

Data Splitting Power: 0.783863080685

```
cat("Data Splitting AUPR:", ds_AUPR, "\n")
```

Data Splitting AUPR: 0.844597853735