Stochastic Block Model Prior with Ordering Constraints for Gaussian Graphical Models

Alessandro Colombi (Supervisor)* Teo Bucci† Filippo Cipriani‡ Filippo Pagella§
Flavia Petruso¶ Andrea Puricelli‖ Giulio Venturini**

Contents

1 Simulations

1.1 Load the necessary packages

1.2 Generate data

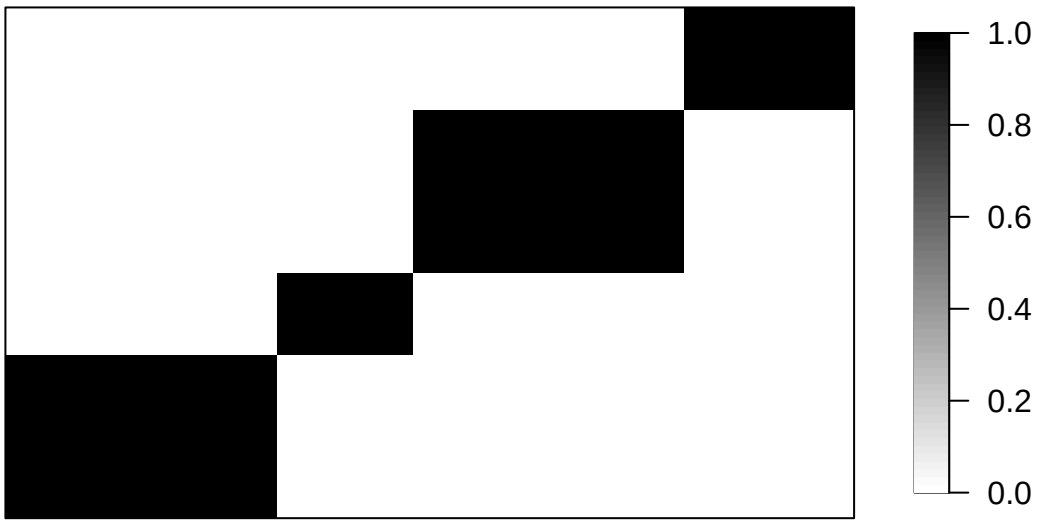
```
# Define true clustering
rho_true = c(8,4,8,5)

# Set seed for data generation
seed_data_gen = 22111996
```

*a.colombi10@campus.unimib.it
†teo.bucci@mail.polimi.it
‡filippo.cipriani@mail.polimi.it
§filippo.pagella@mail.polimi.it
¶flavia.petruso@mail.polimi.it
‖andrea3.puricelli@mail.polimi.it
**giulio.venturini@mail.polimi.it

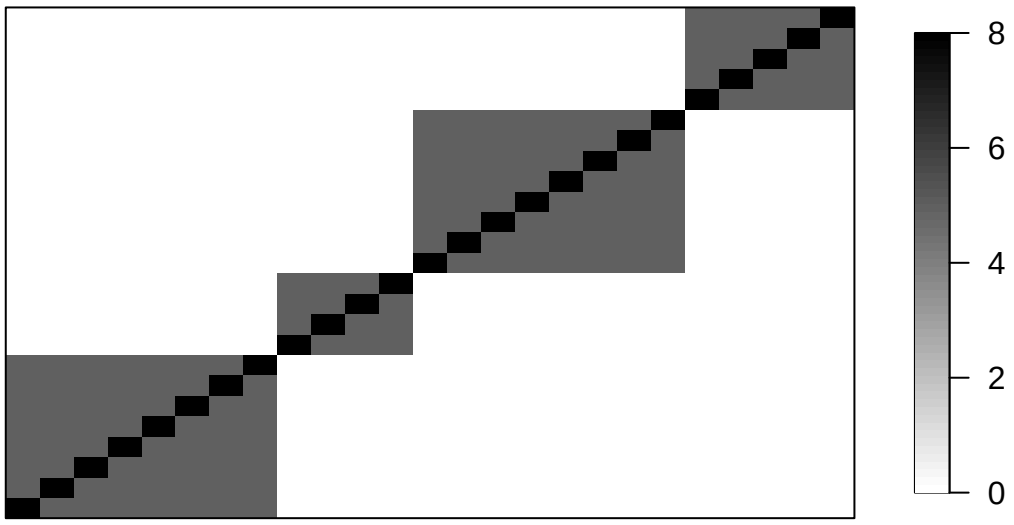
```r
# Define number of observations
n = 500

# Define variance of the Beta
beta_sig2 = 1/16
```

```r
z_true = rho_to_z(rho_true)
r_true = z_to_r(z_true)
p = length(z_true)

# Generate data
sim = Generate_BlockDiagonal(n = n, z_true = z_true)
# sim = Generate_Block(
#     n=n,
#     z_true=z_true,
#     p_block_diag = 1,
#     p_block_extra_diag = 0,
#     p_inside_block = 0.95,
#     p_outside_block = 0.1,
#     elem_out = 5,
#     min_eigenval_correction = 3,
#     seed = 1
# )
```

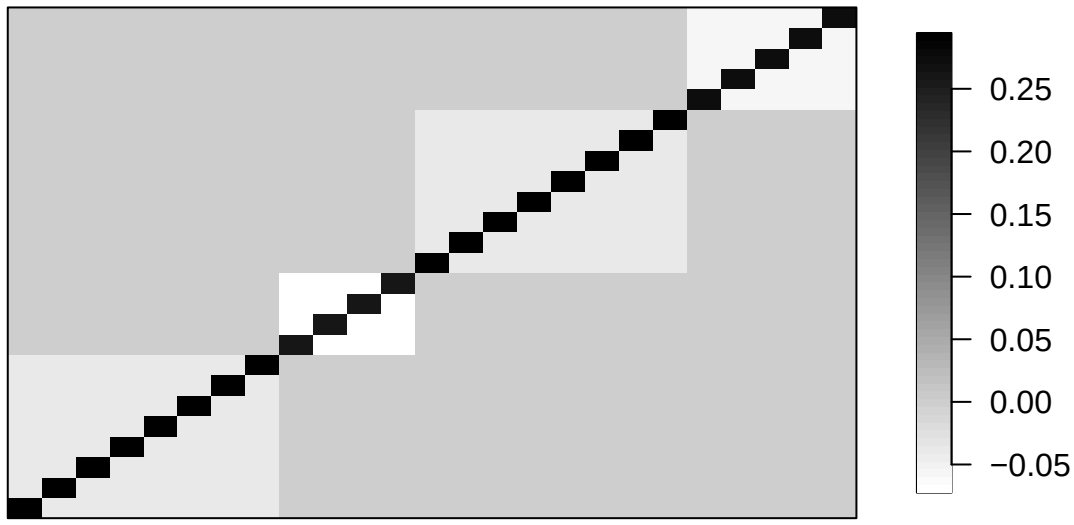
```r
ACutils::ACheatmap(
    sim$Graph,
    use_x11_device = F,
    horizontal = F,
    main = "Graph",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```




```
ACutils::ACheatmap(
    sim$Prec,
    use_x11_device = F,
    horizontal = F,
    main = "Precision matrix",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```




```
ACutils::ACheatmap(
    sim$Cov,
    use_x11_device = F,
    horizontal = F,
    main = "Covariance",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```



`sim$Prec` is the true precision matrix, `sim$Graph` is the true graph (adjacency matrix).

```
graph_density = sum(sim$Graph) / (p*(p-1))
graph_density
```

```
## [1] 0.2816667
```

1.3 Set options for the simulation

```
options = set_options(sigma_prior_0=0.5,
                      sigma_prior_parameters=list("a"=1,"b"=1,"c"=1,"d"=1),
                      theta_prior_0=1,
                      theta_prior_parameters=list("c"=1,"d"=1),
                      rho0=p, # start with one group
                      weights_a0=rep(1,p-1),
                      weights_d0=rep(1,p-1),
                      alpha_target=0.234,
                      beta_mu=graph_density,
                      beta_sig2=beta_sig2,
                      d=3,
                      alpha_add=0.5,
                      adaptation_step=1/(p*1000),
                      update_sigma_prior=TRUE,
                      update_theta_prior=TRUE,
                      update_weights=TRUE,
                      update_partition=TRUE,
                      update_graph=TRUE,
                      perform_shuffle=TRUE)
```

1.4 Running the simulation

Create output directory if needed

```
dir.create(file.path("output", "data"), showWarnings = FALSE, recursive = TRUE)
dir.create(file.path("output", "log"), showWarnings = FALSE, recursive = TRUE)
```

```
unique_ID = uuid::UUIDgenerate(use.time = TRUE, n = 1, output = c("string"))
unique_ID = dittodb::hash(unique_ID, n = 8)
cat("This simulation has been assigned ID:", unique_ID)
```

```
## This simulation has been assigned ID: 9d4663c6
```

```
filename_data = paste("output/data/simulation_", unique_ID, ".rds", sep = "")
filename_log = paste("output/log/simulation_", unique_ID, ".log", sep = "")
```

Run the simulation

```
#log_open(file_name = filename_log, show_notes=FALSE, logdir = FALSE)
res <- Gibbs_sampler(
    data = sim$data,
    niter = 1000,
    nburn = 2,
    thin = 1,
    options = options,
    seed = 123456,
    print = TRUE
)
```

```
##   |                                                                      |
```

```
#log_close()
```

Before saving, also append true data

```
res$true_rho = rho_true
res$true_precision = sim$Prec
res$true_graph = sim$Graph
# remove self loops
res$true_graph[col(res$true_graph)==row(res$true_graph)] = 0
```

```
# save an object to a file
saveRDS(res, file = filename_data)
```

2 Posterior analysis

Restore the object

```
#unique_ID = 'c7726e0e'
filename_data = paste("output/data/simulation_", unique_ID, ".rds", sep = "")
res = readRDS(file = filename_data)
```

2.1 Partition

Recomputing the partition in other forms and the number of groups

```
rho_true = res$true_rho
r_true = rho_to_r(rho_true)
z_true = rho_to_z(rho_true)
```

```
p = length(z_true)
num_clusters_true = length(rho_true)
rho = res$rho
r = do.call(rbind, lapply(res$rho, rho_to_r))
```

```
## Warning in (function (..., deparse.level = 1) : number of columns of result is
## not a multiple of vector length (arg 4)
```

```
z = do.call(rbind, lapply(res$rho, rho_to_z))
num_clusters = do.call(rbind, lapply(res$rho, length))
num_clusters = as.vector(num_clusters)
```

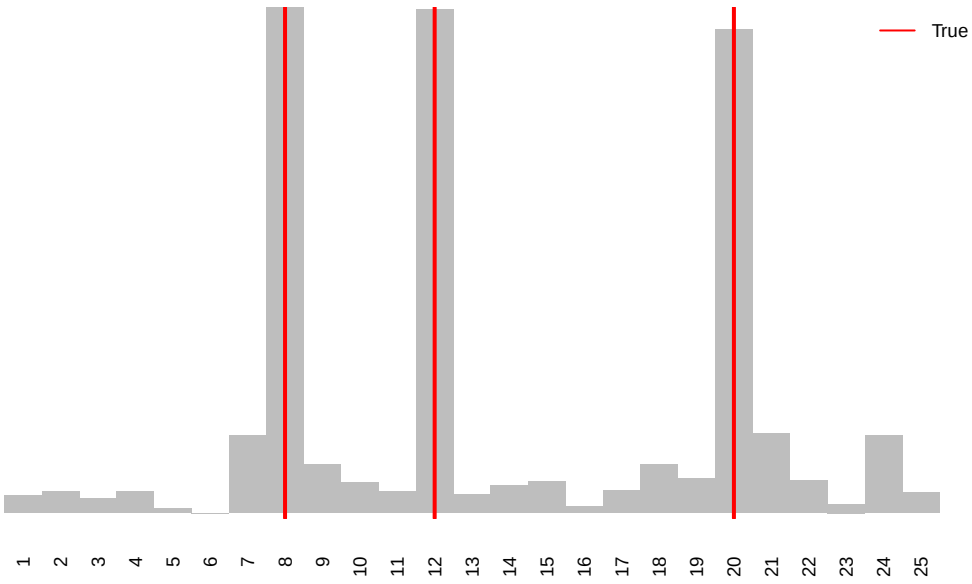
2.1.1 Acceptance frequency

```
mean(res$accepted)
```

```
## [1] 0.161
```

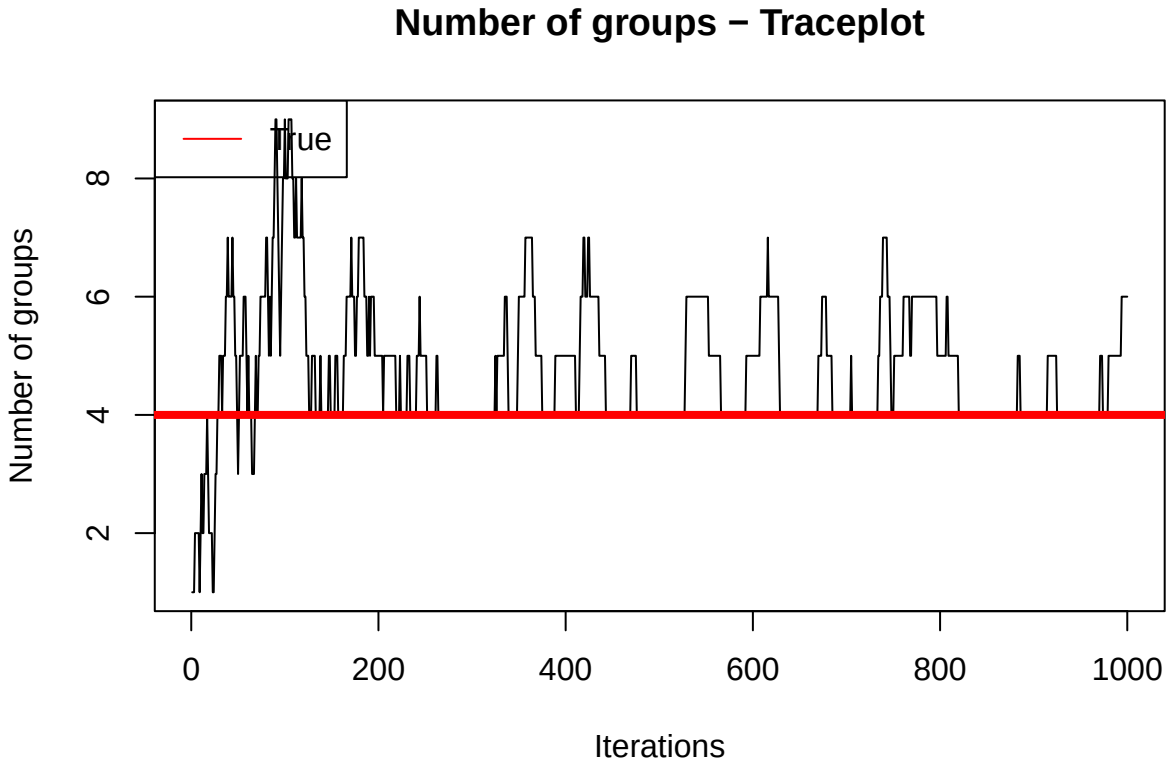
2.1.2 Barplot of changepoints

```
bar_heights = colSums(r)
cp_true = which(r_true==1)
color <- ifelse(seq_along(bar_heights) %in% c(cp_true), "red", "gray")

barplot(
    bar_heights,
    names = seq_along(bar_heights),
    border = "NA",
    space = 0,
    yaxt = "n",
    main="Changepoint frequency distribution",
    #col = color,
    cex.names=.6,
    las=2
)

abline(v=cp_true-0.5, col="red", lwd=2)
legend("topright", legend=c("True"), col=c("red"),
    bty = "n",
    lty = 1,
    cex = 0.6)
```

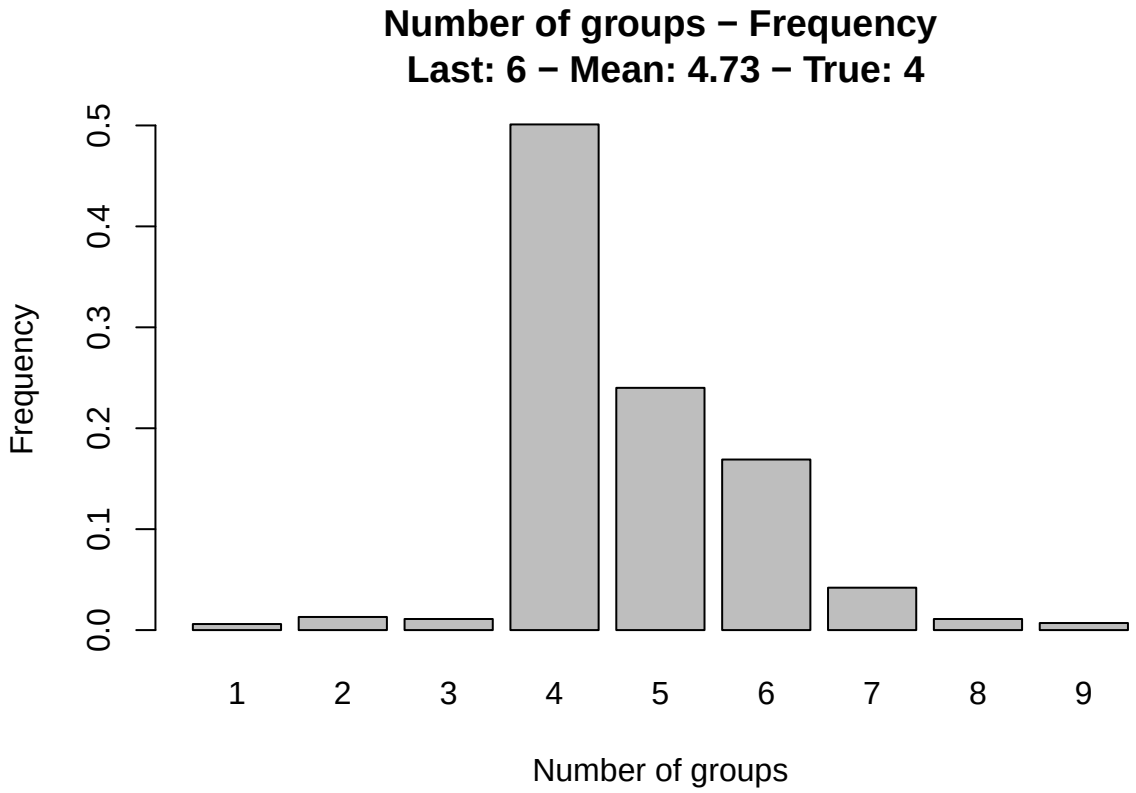


2.1.3 Evolution of the number of clusters

```
plot(
    x = seq_along(num_clusters),
    y = num_clusters,
    type = "n",
    xlab = "Iterations",
    ylab = "Number of groups",
    main = "Number of groups - Traceplot"
)
lines(x = seq_along(num_clusters), y = num_clusters)
abline(h = length(z_to_rho(z_true)),
       col = "red",
       lwd = 4)
legend("topleft", legend=c("True"), col=c("red"),
    lty = 1,
    cex = 1)
```

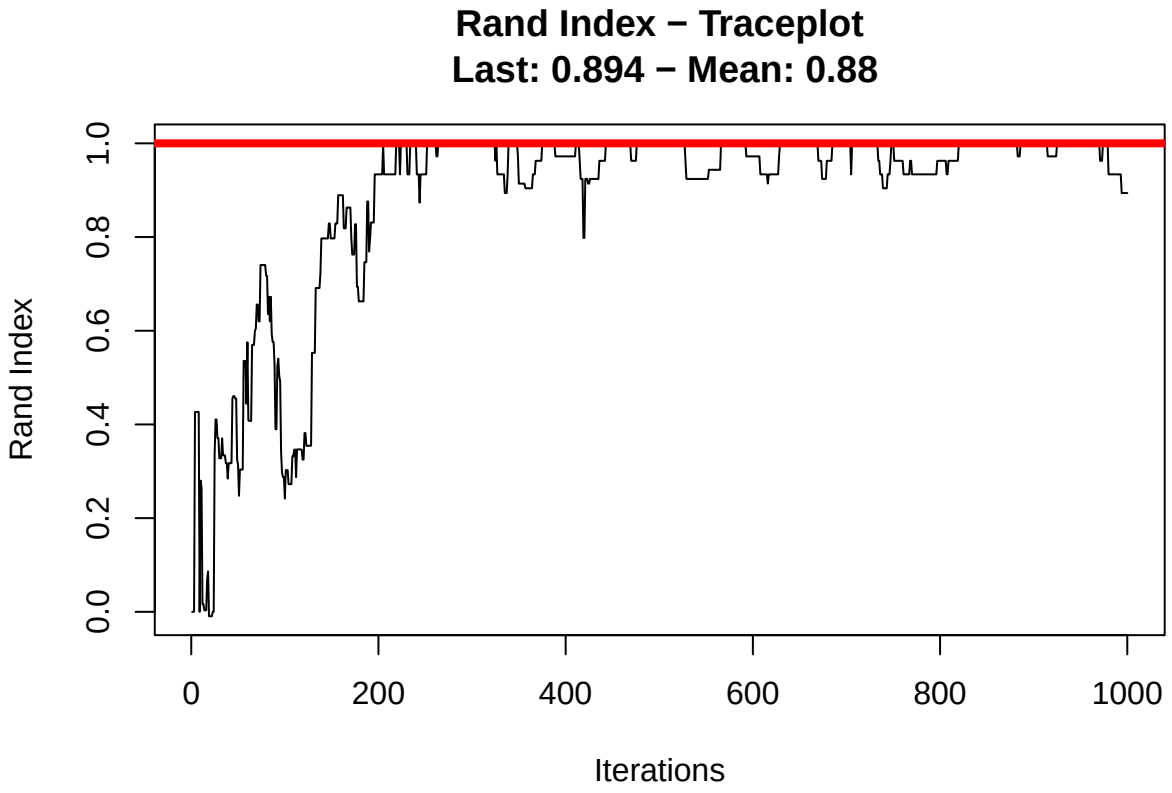



```
barplot(
    prop.table(table(num_clusters)),
    xlab = "Number of groups",
    ylab = "Frequency",
    main = paste(
        "Number of groups - Frequency\n",
        "Last:",
        tail(num_clusters, n = 1),
        "- Mean:",
        round(mean(num_clusters), 2),
        "- True:",
        num_clusters_true
    )
)
```



2.1.4 Evolution of the Rand Index

```
# computing rand index for each iteration
rand_index = apply(z, 1, mcclust::arandi, z_true)

# plotting the traceplot of the index
plot(
    x = seq_along(rand_index),
    y = rand_index,
    type = "n",
    xlab = "Iterations",
    ylab = "Rand Index",
    main = paste(
        "Rand Index - Traceplot\n",
        "Last:",
        round(tail(rand_index, n=1), 3),
        "- Mean:",
        round(mean(rand_index), 2)
    )
)
lines(x = seq_along(rand_index), y = rand_index)
abline(h = 1, col = "red", lwd = 4)
```



2.1.5 Retrieving best partition using VI on visited ones (order is guaranteed here)

Here we are satisfied with finding the optimal partition only in the set of those visited, not in all the possible ones. I expect it could work even worse. But at least it guarantees to find an admissible one. I would say that it is the implementation of formula (13) of the Corradin-Danese paper (https://doi.org/10.1016/j.ijar.2021.12.019).

```
library("Rcpp")
library("RcppArmadillo")
sourceCpp("src/wade.cpp")

# compute VI
sim_matrix <- salso::psm(z)
dists <- VI_LB(z, psm_mat = sim_matrix)

# select best partition (among the visited ones)
best_partition_index = which.min(dists)
rho_est = rho[[best_partition_index]]
z_est = z[best_partition_index,]

# VI loss
dists[best_partition_index]
```

```
## [1] 0.2730145
```

```
# select best partition
unname(z_est)
```

```
##  [1] 1 1 1 1 1 1 1 1 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4
```

```r
# compute Rand Index
mcclust::arandi(z_est, z_true)
```

```
## [1] 1
```

2.2 Graph

Extract last plinks

```r
last_plinks = tail(res$G, n=1)[[1]]
```

Criterion 1 to select the threshold (should not work very well) and assign final graph

```r
threshold = 0.5
G_est <- matrix(0,p,p)
G_est[which(last_plinks>threshold)] = 1
```

Criterion 2 to select the threshold

```r
bfdr_select = BFDR_selection(last_plinks, tol = seq(0.1, 1, by = 0.001))
```

Inspect the threshold and assign final graph

```r
bfdr_select$best_treshold
```

```
## [1] 0.878
```

```r
G_est = bfdr_select$best_truncated_graph
```

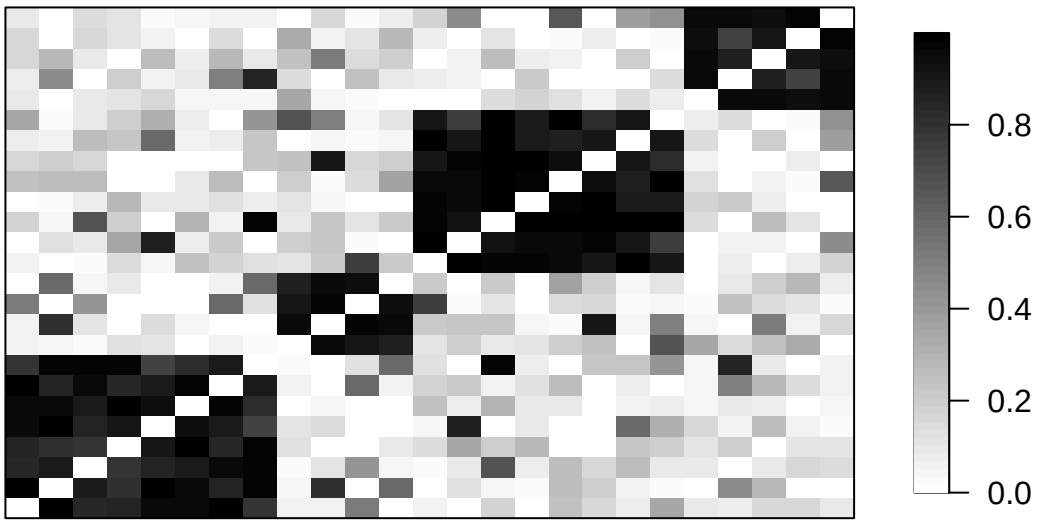
2.2.1 Standardized Hamming distance

```r
SHD = sum(abs(sim$true_graph - G_est)) / (p^2 - p)
SHD
```

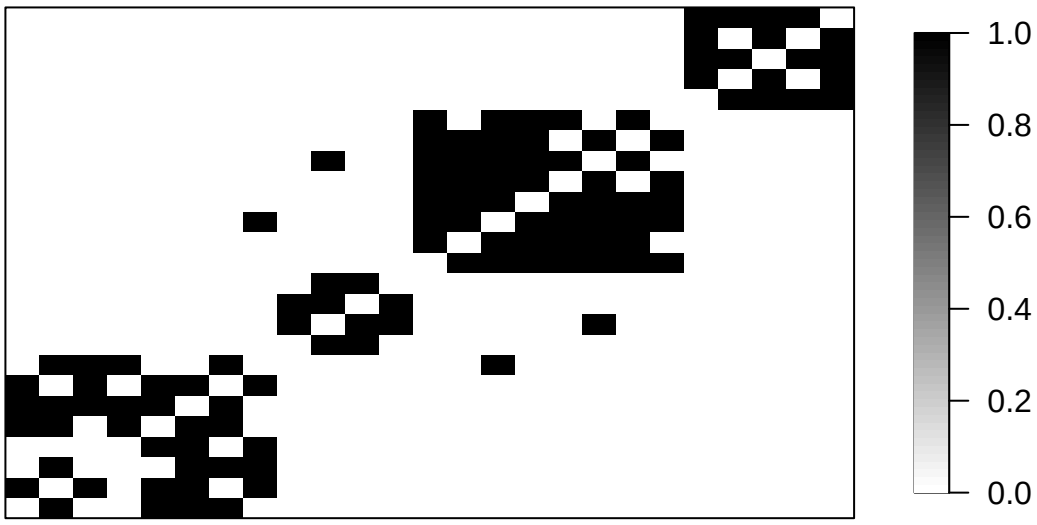
```
## [1] 0
```

2.2.2 Plot estimated matrices

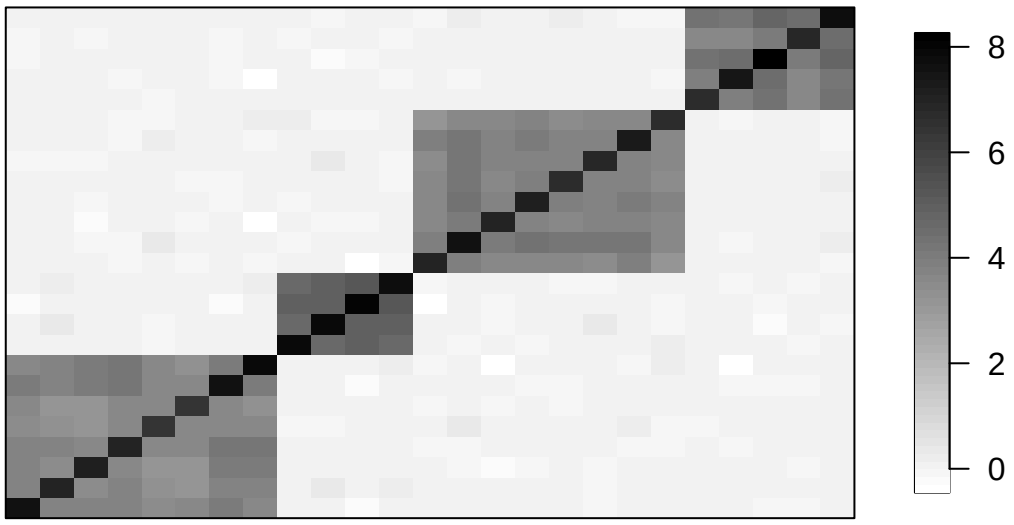
```r
ACutils::ACheatmap(
    last_plinks,
    use_x11_device = F,
    horizontal = F,
    main = "Estimated plinks matrix",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```




```
ACutils::ACheatmap(
    G_est,
    use_x11_device = F,
    horizontal = F,
    main = "Estimated Graph",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```

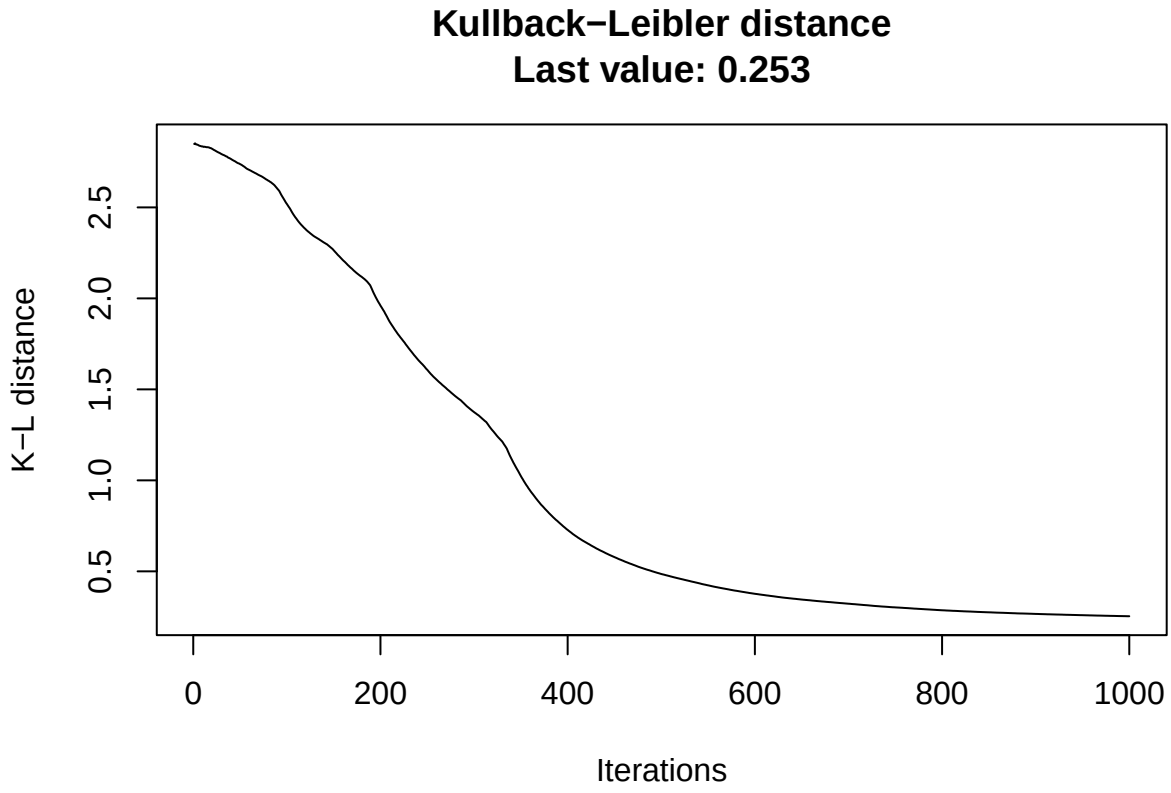



```
ACutils::ACheatmap(
    tail(res$K,n=1)[[1]],
    use_x11_device = F,
    horizontal = F,
    main = "Estimated Precision matrix",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```



2.2.3 Evolution of the Kullback-Leibler

```
kl_dist = do.call(rbind, lapply(res$K, function(k) {
    ACutils::KL_dist(res$true_precision, k)
}))

last = round(tail(kl_dist, n=1), 3)
plot(
    x = seq_along(kl_dist),
    y = kl_dist,
    type = "n",
    xlab = "Iterations",
    ylab = "K-L distance",
    main = paste("Kullback-Leibler distance\nLast value:", last)
)
lines(x = seq_along(kl_dist), y = kl_dist)
```



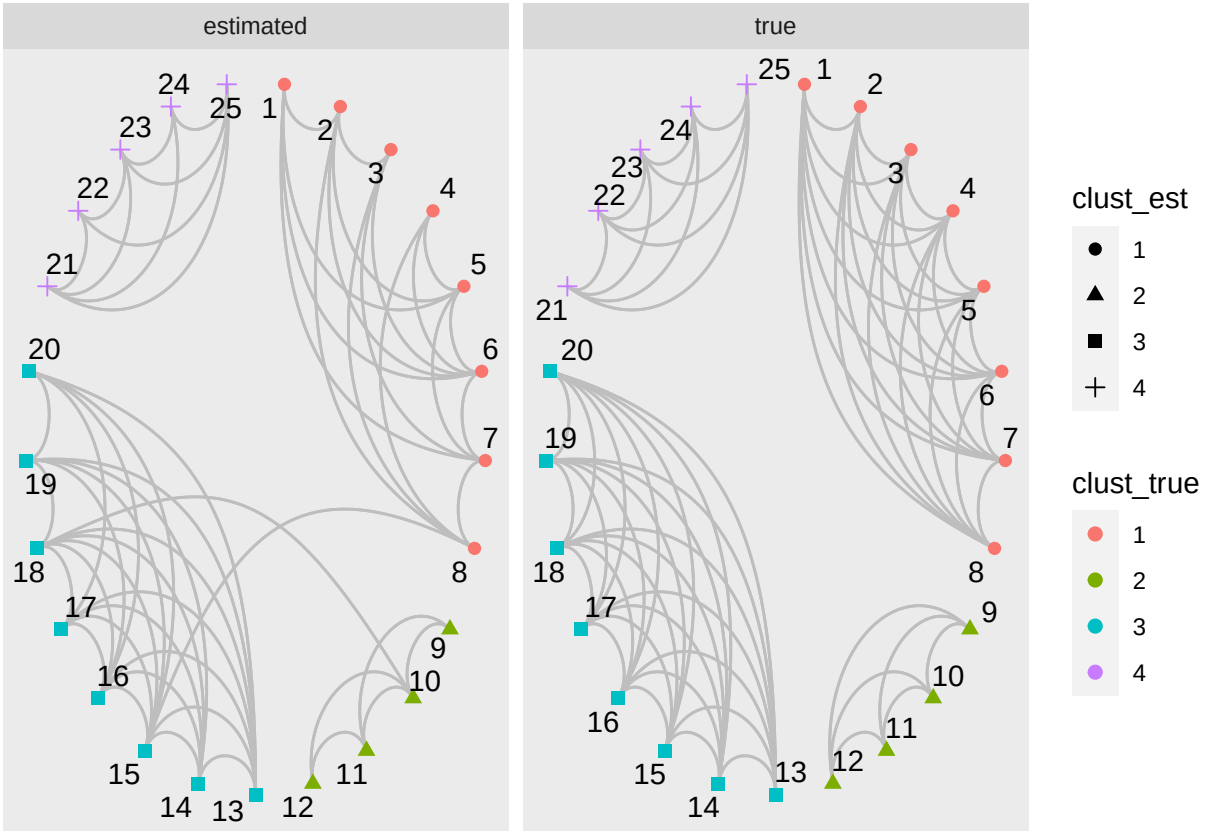
2.2.4 Graph visualization

Plot against the original

```
g1 <- graph.adjacency(res$true_graph)
edges1 <- get.edgelist(g1)
edges1 <- cbind(edges1, rep("true", nrow(edges1)))
g2 <- graph.adjacency(G_est)
edges2 <- get.edgelist(g2)
edges2 <- cbind(edges2, rep("estimated", nrow(edges2)))
edges <- as.data.frame(rbind(edges1, edges2))
names(edges) = c("from", "to", "graph")
nodes = data.frame(
    vertices = 1:p,
    clust_true = as.factor(z_true),
    clust_est = as.factor(z_est)
)
# nodes
g = graph_from_data_frame(edges, directed = FALSE, nodes)
lay = create_layout(g, layout = "linear", circular = TRUE)

output_plot <- ggraph(lay) +
    geom_edge_arc(edge_colour = "grey") +
    geom_node_point(aes(color = clust_true, shape = clust_est), size = 2) +
    geom_node_text(aes(label = name), repel = TRUE) +
    facet_edges( ~ graph)
print(output_plot)
```

```
## Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
```

```
## i Please use `linewidth` in the `default_aes` field and elsewhere instead.
```



ì