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

*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

1 Simulations

1.1 Load the necessary packages

1.2 Load data

```
suono_centered <- load(here::here("dataset","Suono_centered.Rdat"))
suono_normalized <- load(here::here("dataset","Suono_Quantile_Normalized.Rdat"))

n = nrow(suono_cent)
p = ncol(suono_quant_norm)
```

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=0.25,
                      beta_sig2=1/16,
                      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 Run the simulation

```
res <- Gibbs_sampler(
    data = suono_cent,
    niter = 8000,
    nburn = 2000,
    thin = 1,
    options = options,
    seed = 123456,
    print = FALSE
)

res_norm <- Gibbs_sampler(
    data = suono_quant_norm,
    niter = 8000,
    nburn = 2000,
    thin = 1,
    options = options,
    seed = 123456,
```

```
    print = FALSE
)
```

```
# create output directory if not present
dir.create(file.path("output", "real_dataset"), showWarnings = FALSE, recursive = TRUE)

# save an object to a file
filename_data = file.path("output","real_dataset","realtest_cent.rds")
saveRDS(res, file = filename_data)

filename_data = file.path("output","real_dataset","realtest_norm.rds")
saveRDS(res_norm, file = filename_data)
```

2 Posterior analysis (Case: Normalized)

Restore the object

```
filename_data = file.path("output","real_dataset","realtest_norm.rds")
res = readRDS(file = filename_data)
```

2.1 Partition

Recomputing the partition in other forms and the number of groups

```
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 2)
```

```
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)
```

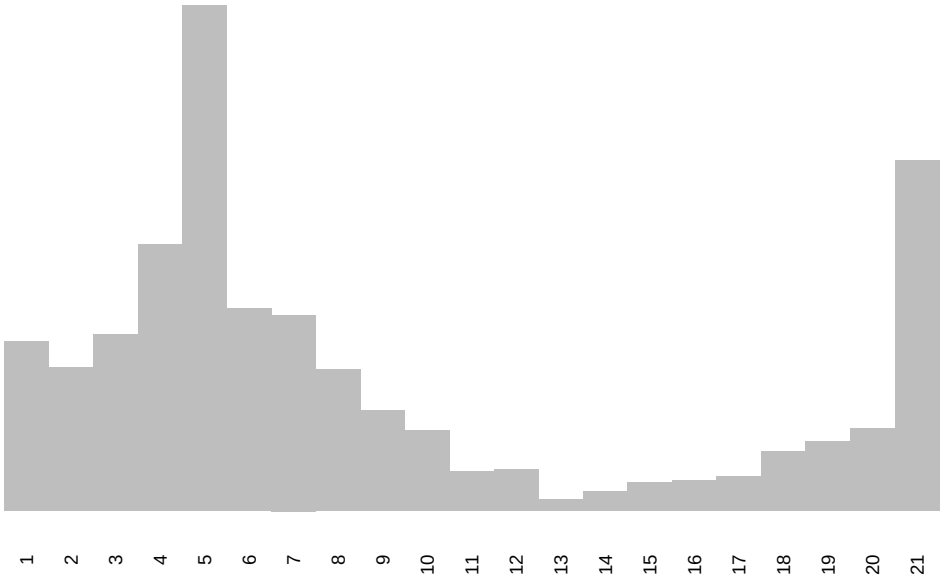
2.1.1 Acceptance frequency

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

```
## [1] 0.344
```

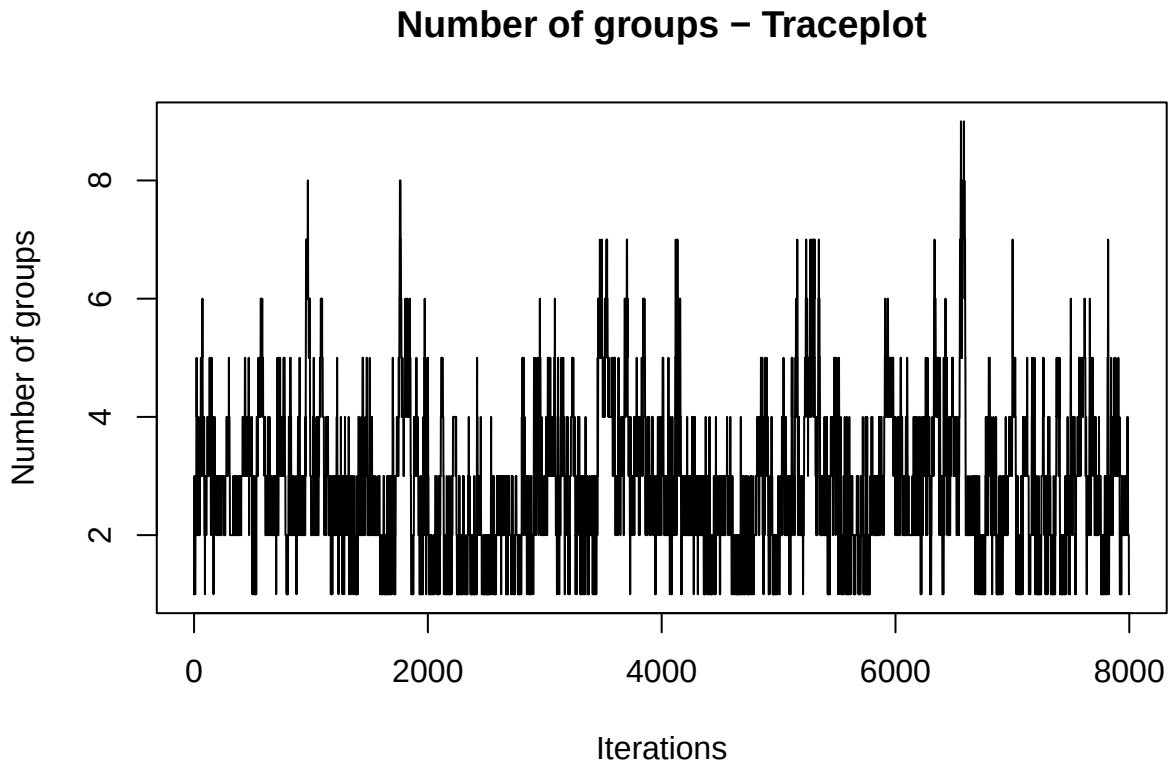
2.1.2 Barplot of changepoints

```
bar_heights = colSums(r)
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
)
```

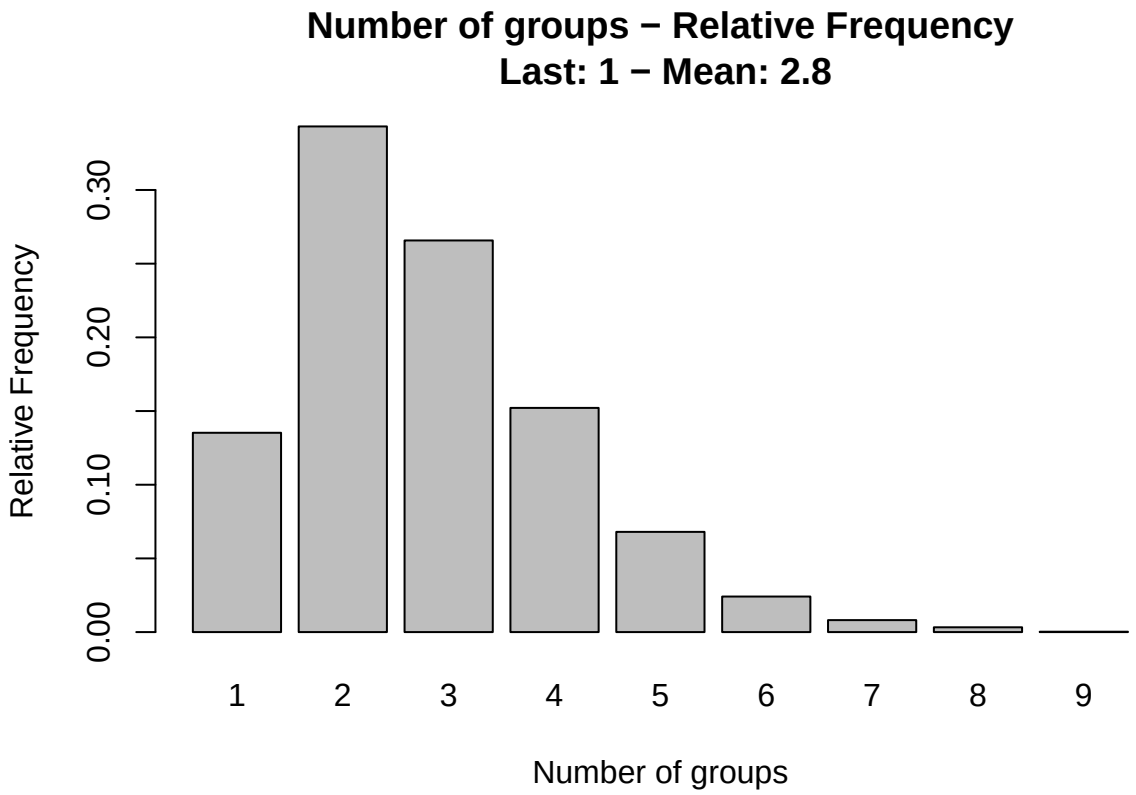


2.1.3 Evolution of the number of clusters

```
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)
```




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



2.1.4 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] 0.6494628
```

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

```
##  [1] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

2.2 Graph

Extract last plinks

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

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

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

Criterion 2 to select the threshold

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

Inspect the threshold and assign final graph

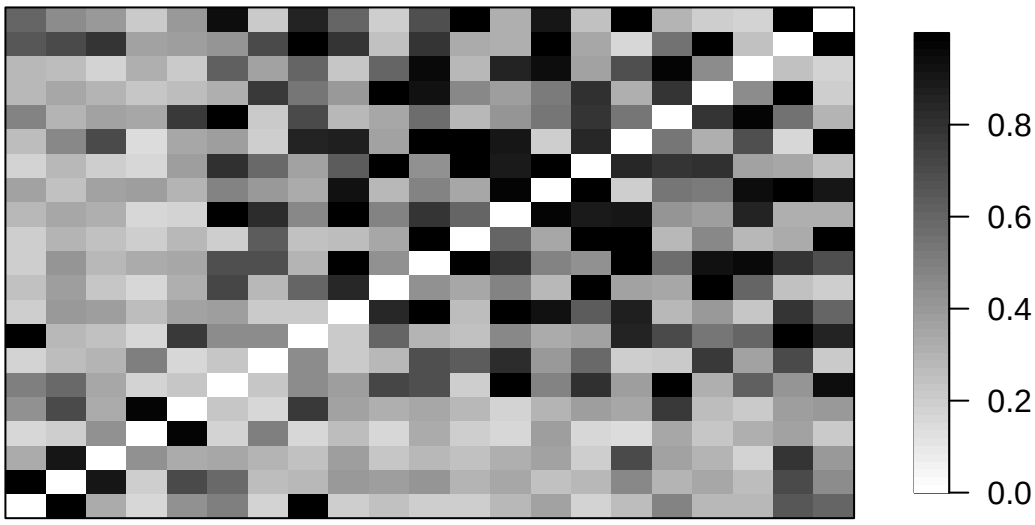
```
bfdr_select$best_treshold
```

```
## [1] 0.818
```

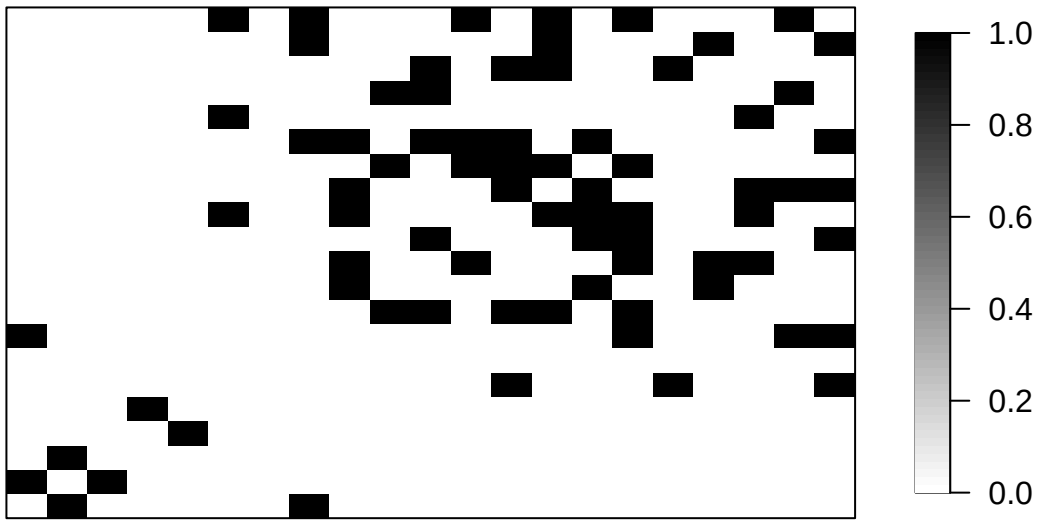
```
G_est = bfdr_select$best_truncated_graph
```

2.2.1 Plot estimated matrices

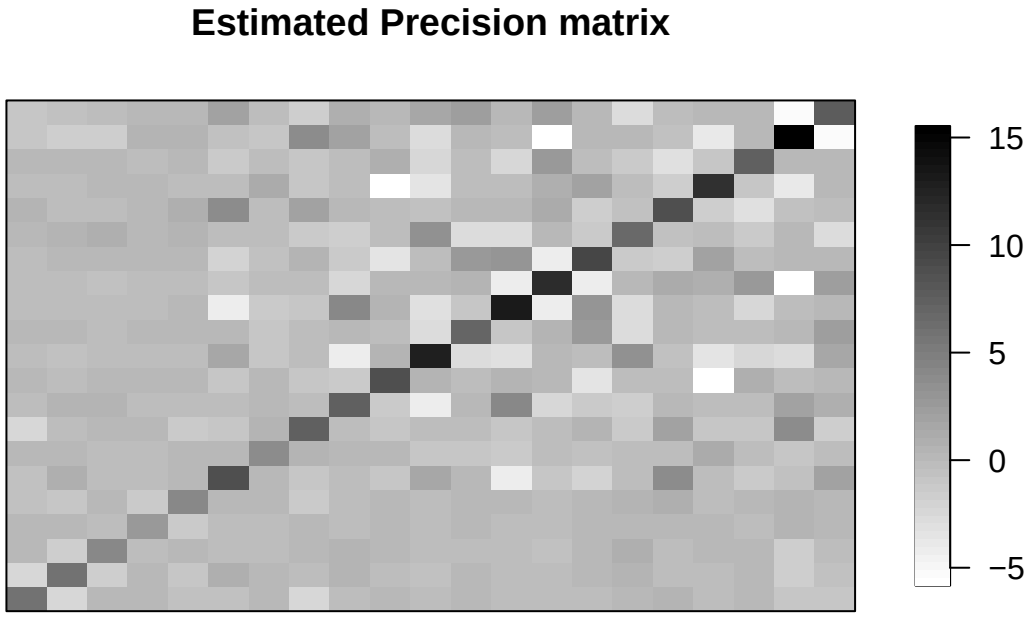
```
ACutils::ACheatmap(
    last_plinks,
    use_x11_device = F,
    horizontal = F,
    main = "Estimated plinks matrix",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```




```
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"
)
```



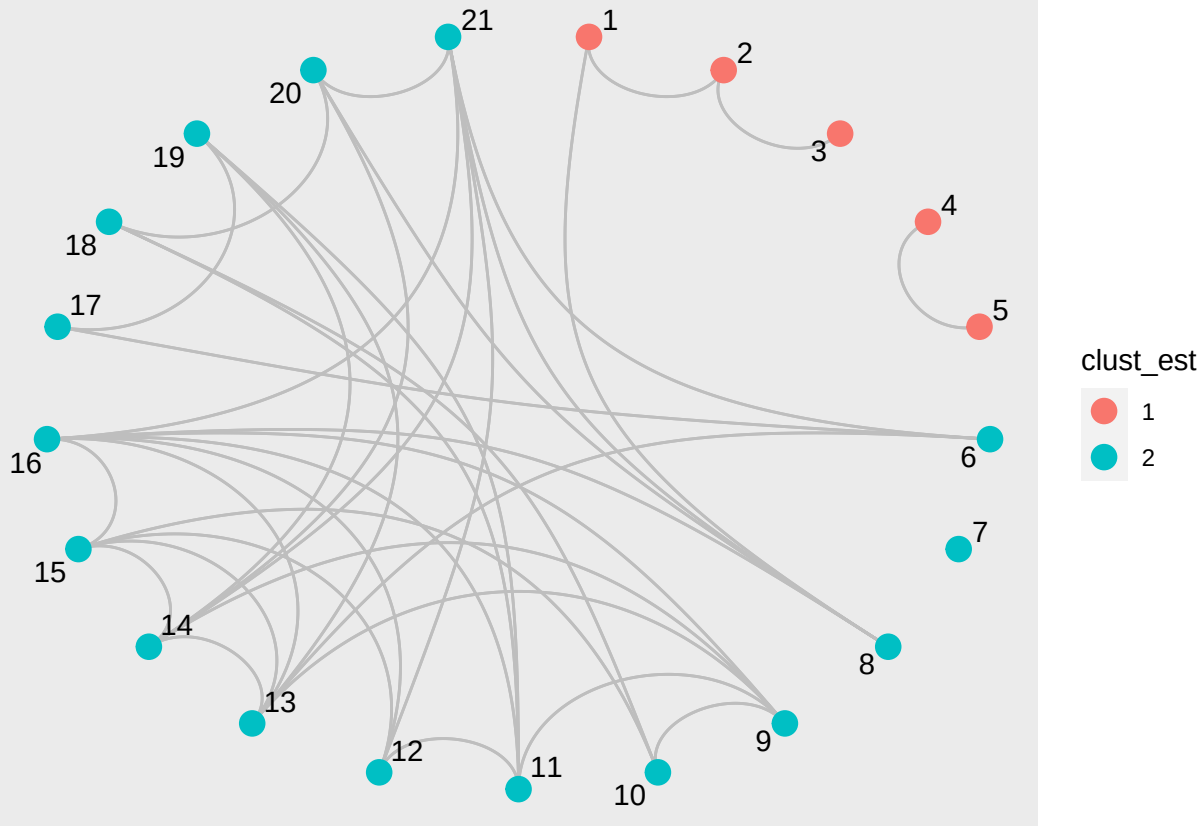
2.2.2 Graph visualization

Plot against the original

```
g2 <- graph.adjacency(G_est)
edges2 <- get.edgelist(g2)
edges2 <- cbind(edges2, rep("estimated", nrow(edges2)))
edges <- as.data.frame(edges2)
names(edges) = c("from", "to", "graph")
nodes = data.frame(
    vertices = 1:p,
    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_est), size = 4) +
    geom_node_text(aes(label = name), repel = TRUE)
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.
```



3 Posterior analysis (Case: Centered)

Restore the object

```
filename_data = file.path("output","real_dataset","realtest_cent.rds")
res = readRDS(file = filename_data)
```

3.1 Partition

Recomputing the partition in other forms and the number of groups

```
rho = res$rho
r = do.call(rbind, lapply(res$rho, rho_to_r))
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)
```

3.1.1 Acceptance frequency

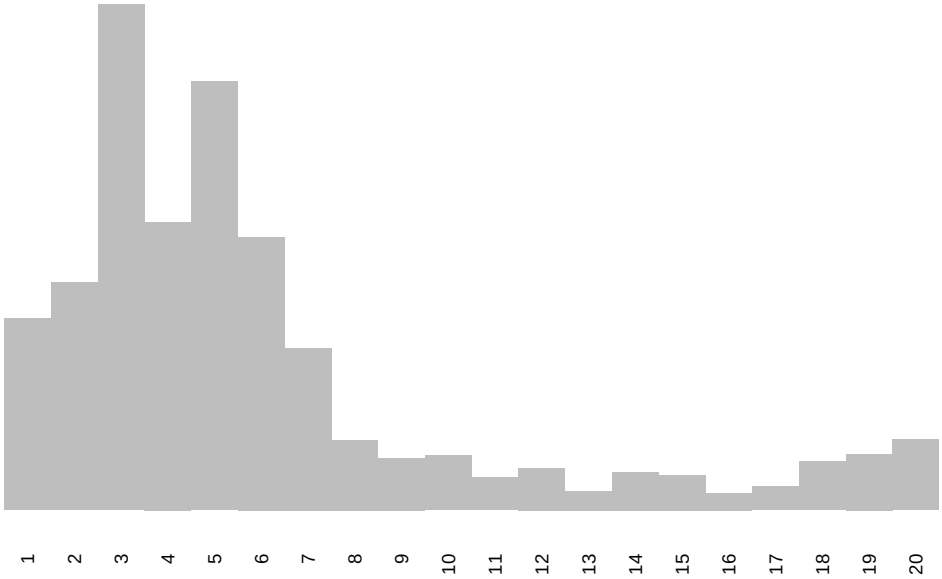
```
mean(res$accepted)
```

```
## [1] 0.322625
```

3.1.2 Barplot of changepoints

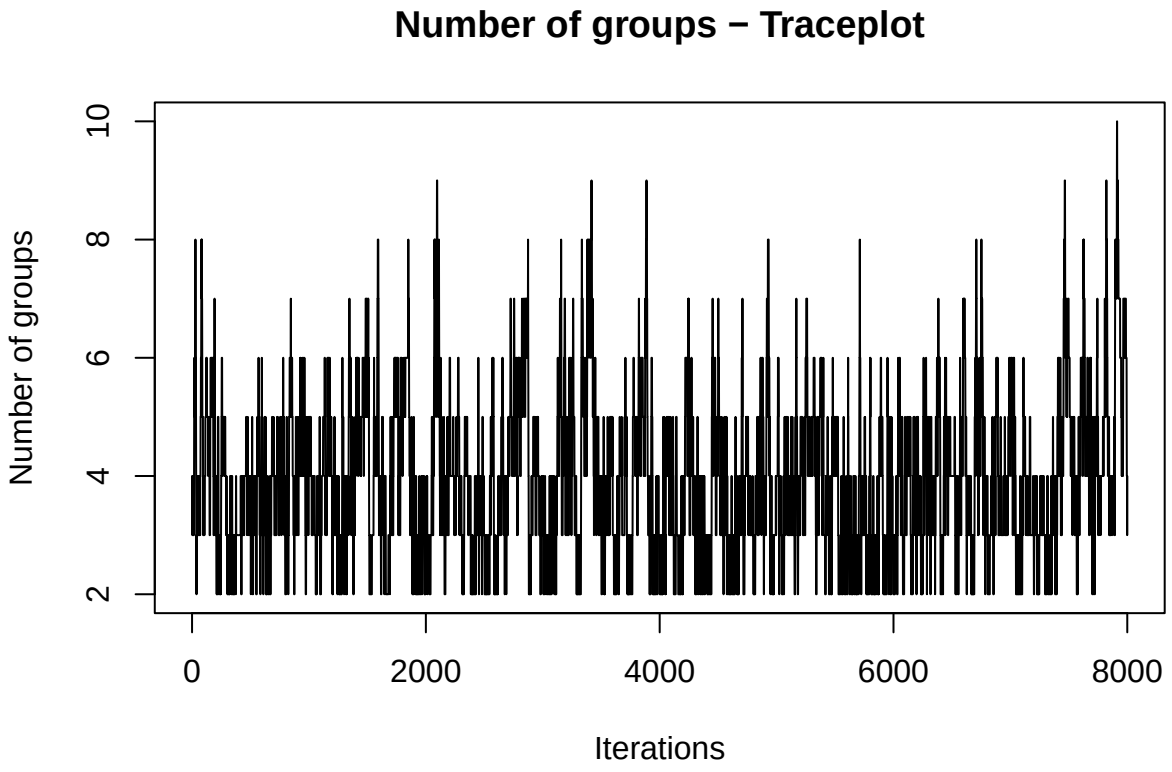
```
bar_heights = colSums(r)
barplot(
```

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

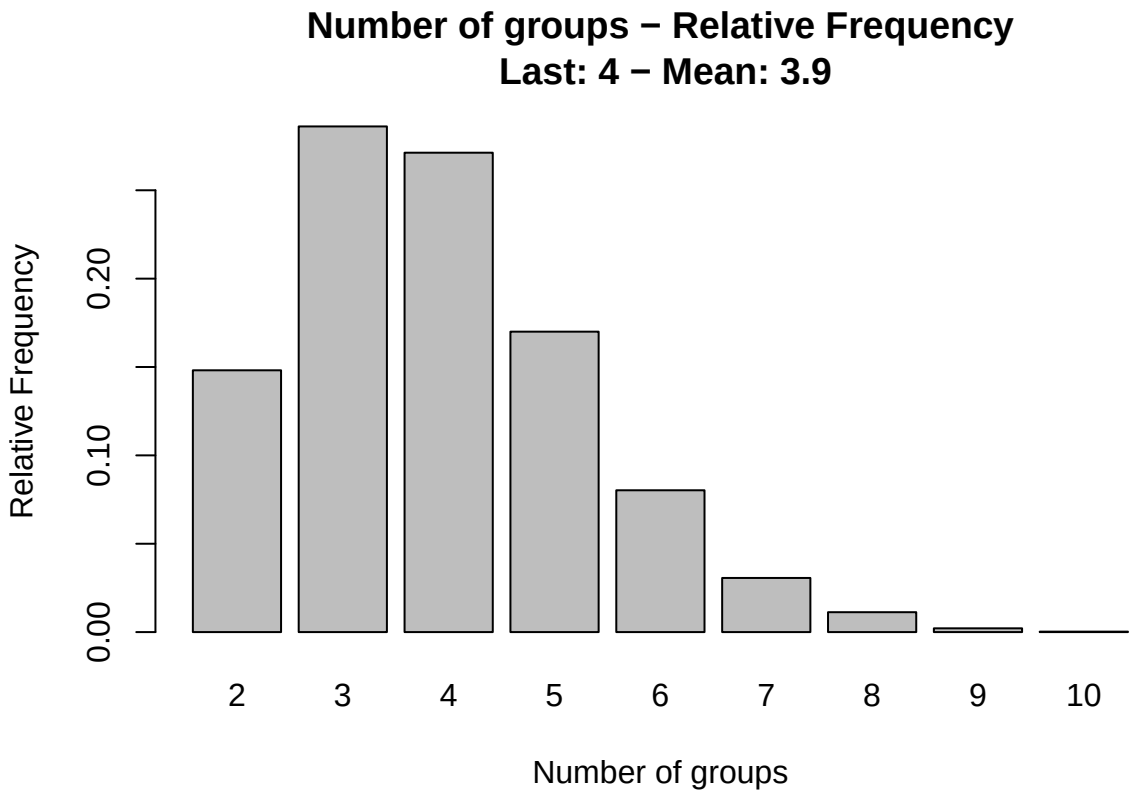


3.1.3 Evolution of the number of clusters

```
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)
```




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



3.1.4 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] 0.5885619
```

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

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

3.2 Graph

Extract last plinks

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

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

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

Criterion 2 to select the threshold

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

Inspect the threshold and assign final graph

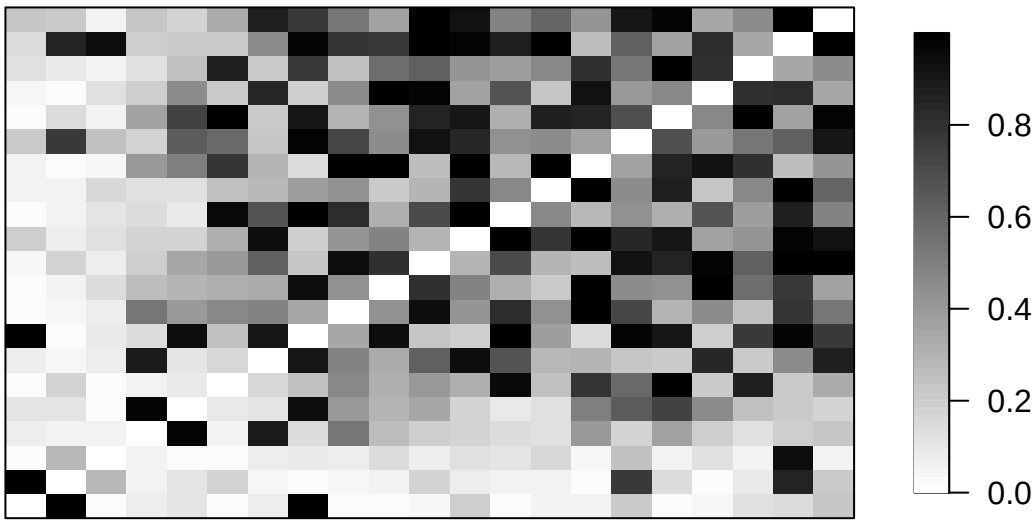
```
bfdr_select$best_treshold
```

```
## [1] 0.853
```

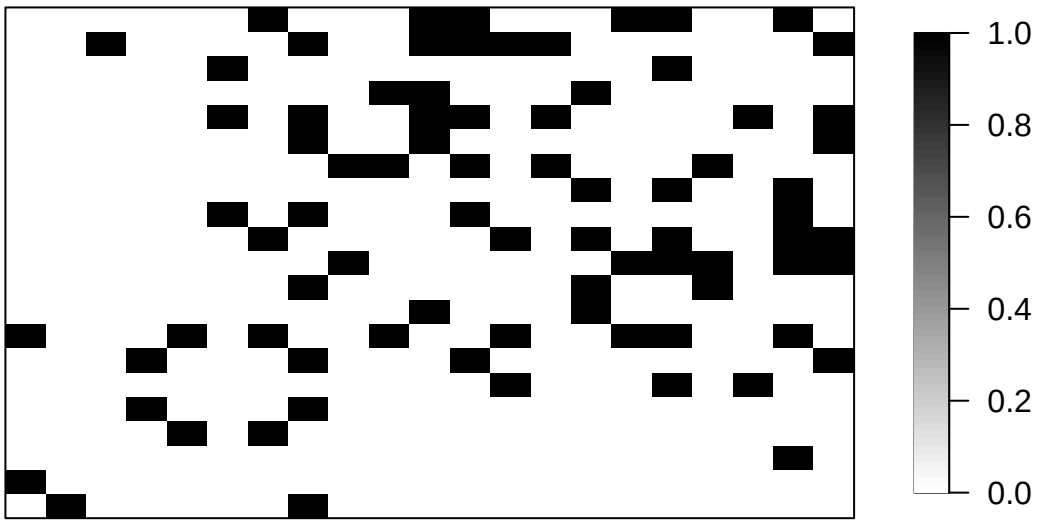
```
G_est = bfdr_select$best_truncated_graph
```

3.2.1 Plot estimated matrices

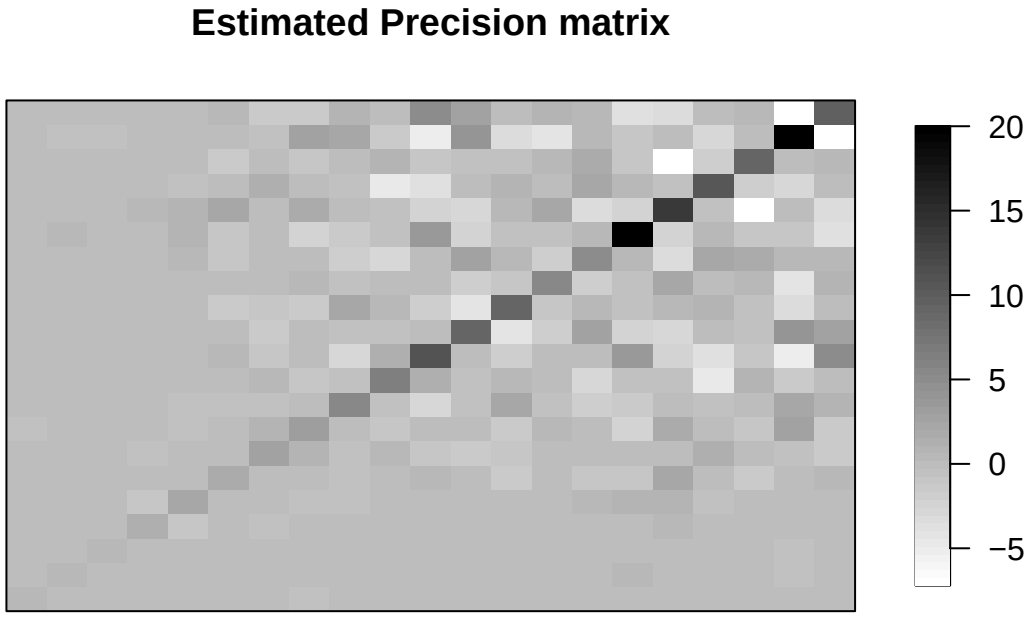
```
ACutils::ACheatmap(
    last_plinks,
    use_x11_device = F,
    horizontal = F,
    main = "Estimated plinks matrix",
    center_value = NULL,
    col.upper = "black",
    col.center = "grey50",
    col.lower = "white"
)
```




```
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"
)
```

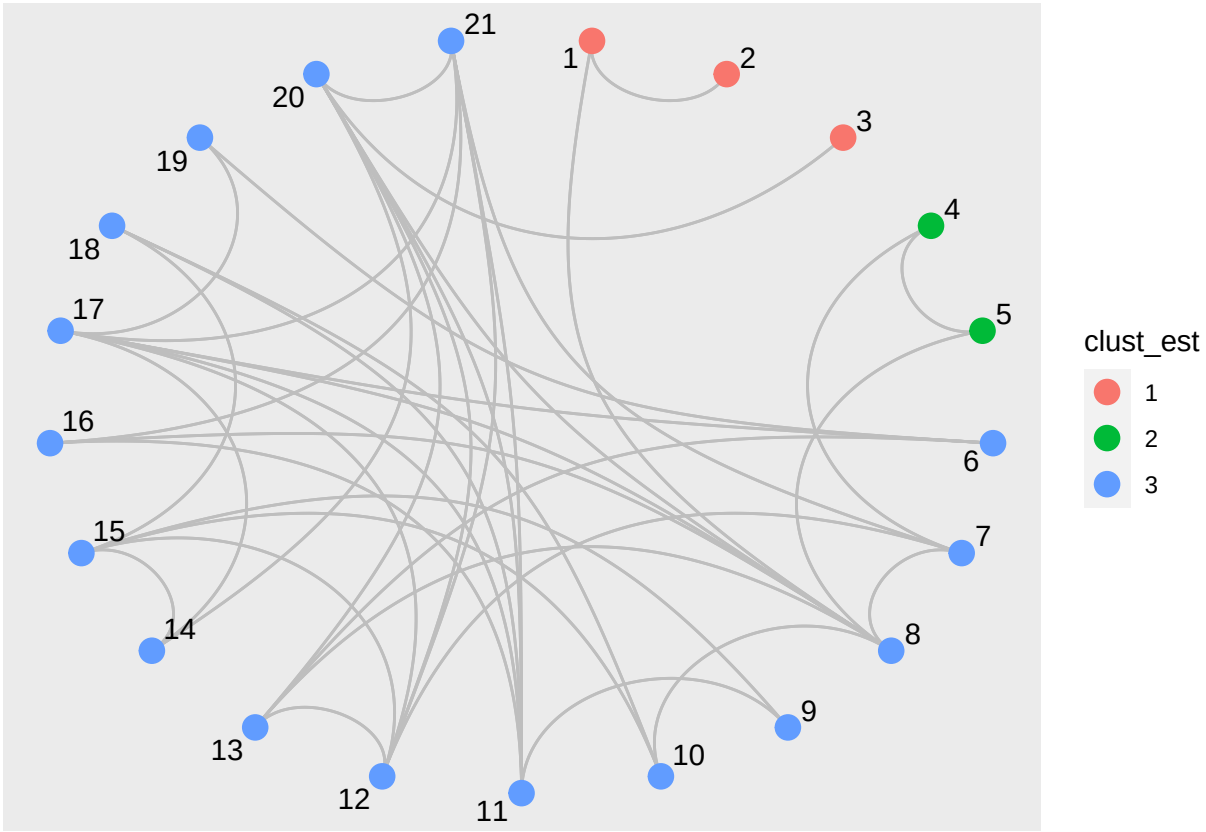


3.2.2 Graph visualization

Plot against the original

```
g2 <- graph.adjacency(G_est)
edges2 <- get.edgelist(g2)
edges2 <- cbind(edges2, rep("estimated", nrow(edges2)))
edges <- as.data.frame(edges2)
names(edges) = c("from", "to", "graph")
nodes = data.frame(
    vertices = 1:p,
    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_est), size = 4) +
    geom_node_text(aes(label = name), repel = TRUE)
print(output_plot)
```


21
1
2
20
3
19
4
18
17
5
16
6
15
7
14
8
9
13
10
12
11
clust_est
1
2
3