es3

2024-01-13

Load libraries

```
shhh = function(lib_name){ # It's a library, so shhh!
  suppressWarnings(suppressMessages(require(lib_name, character.only = TRUE)))
}
shhh("tidyverse")
shhh("ACutils")
shhh("mvtnorm")
shhh("salso")
shhh("FGM")
shhh("gmp")
shhh("mcclust")
shhh("mcclust.ext")
shhh("logr")
shhh("tidygraph")
shhh("ggraph")
shhh("igraph")
shhh("Rcpp")
shhh("RcppArmadillo")
shhh("RcppEigen")

## Load custom functions
source("functions/utility_functions.R");
source("functions/bulky_functions.R");
source("functions/data_generation.R")
sourceCpp("functions/wade.cpp")
Rcpp::sourceCpp('functions/UpdateParamsGSL.cpp')

library('RcppGSL')
library(fda)
library(tidyverse)
library(coda)
library(lattice)
```

Goal

Stima della partizione e grafo con bulky functions 2024.

Simulated data

Simuliamo i dati in sim_data.R con p=20basi e 3 cluster di dimensioni [6,6,8]. Grafo e matrice di precisione sono generati attraverso la funzione Generate_BlockDiagonal.

```
# Import simulated data
BaseMat = as.matrix(read_csv('simulated_data/BaseMat.csv'))         # matrix Phi, dim = 200 x 20
y_hat_true = as.matrix(read_csv('simulated_data/y_hat_true.csv'))   # y true, dim = 300 x 200
beta_true = as.matrix(read_csv('simulated_data/beta_true.csv'))     # beta, dim = 300 x 20
mu_true = as.matrix(read_csv('simulated_data/mu_true.csv'))         # mu true, dim = 20 x 1

# object containing the true graph G, the true precision matrix K
simKG <- readRDS("simulated_data/simKG.rds")

n <- dim(y_hat_true)[1]  # n=300
r <- dim(y_hat_true)[2]  # r=200
p <- dim(BaseMat)[2]     # p=20
```

Initialization

```
# Compute quantities for function UpdateParamGSL
tbase_base = t(BaseMat)%*%BaseMat              # p x p (phi_t * phi)
tbase_data = t(BaseMat)%*%t(y_hat_true)        # p x n (phi_t * Y_t)
Sdata = sum(diag(y_hat_true%*%t(y_hat_true))) # initialize phi*beta = 0

set_UpdateParamsGSL_list = set_UpdateParamsGSL (
    tbase_base  = tbase_base,
    tbase_data  = tbase_data,
    Sdata       = Sdata,
    a_tau_eps   = 2000,
    b_tau_eps   = 2,
    sigma_mu    = 100,
    r           = r,
    Update_Beta = TRUE,
    Update_Mu   = TRUE,
    Update_Tau  = TRUE
)

# Set the number of iterations and burn-in
niter <- 50000
burn_in <- 1000

# Set the initialization values of the chains
initialization_values = set_initialization(
    Beta          = matrix(rnorm(n = p*n), nrow = p, ncol = n),
    mu            = rnorm(n=p),
    tau_eps       = 100,
    K             = matrix(0,p,p),
    G             = matrix(0,p,p),
    z             = rep(1,p),
    rho           = p,
```

```
    a_sigma       = 1,
    b_sigma       = 1,
    c_sigma       = 1,
    d_sigma       = 1,
    c_theta       = 100,
    d_theta       = 10,
    sigma         = 0.5,
    theta         = 1,
    weights_a0    = rep(1,p-1),
    weights_d0    = rep(1,p-1),
    total_weights = 0,
    total_K       = matrix(0,p,p),
    total_graphs  = matrix(0,p,p),
    graph_start   = NULL,
    graph_density = sum(simKG$Graph) / (p*(p-1)),
    beta_sig2     = 0.2,
    d             = 3
)
```

Gibbs sampler

```
chains = Gibbs_sampler_update(
  set_UpdateParamsGSL_list,
  niter,
  initialization_values,
  alpha_target           = 0.234,
  alpha_add              = 0.5,
  adaptation_step        = 1/(p*1000),
  seed                   = 123456,
  update_sigma_prior     = TRUE,
  update_theta_prior     = TRUE,
  update_weights         = TRUE,
  update_partition       = TRUE,
  update_graph           = TRUE,
  perform_shuffle        = TRUE
)
```

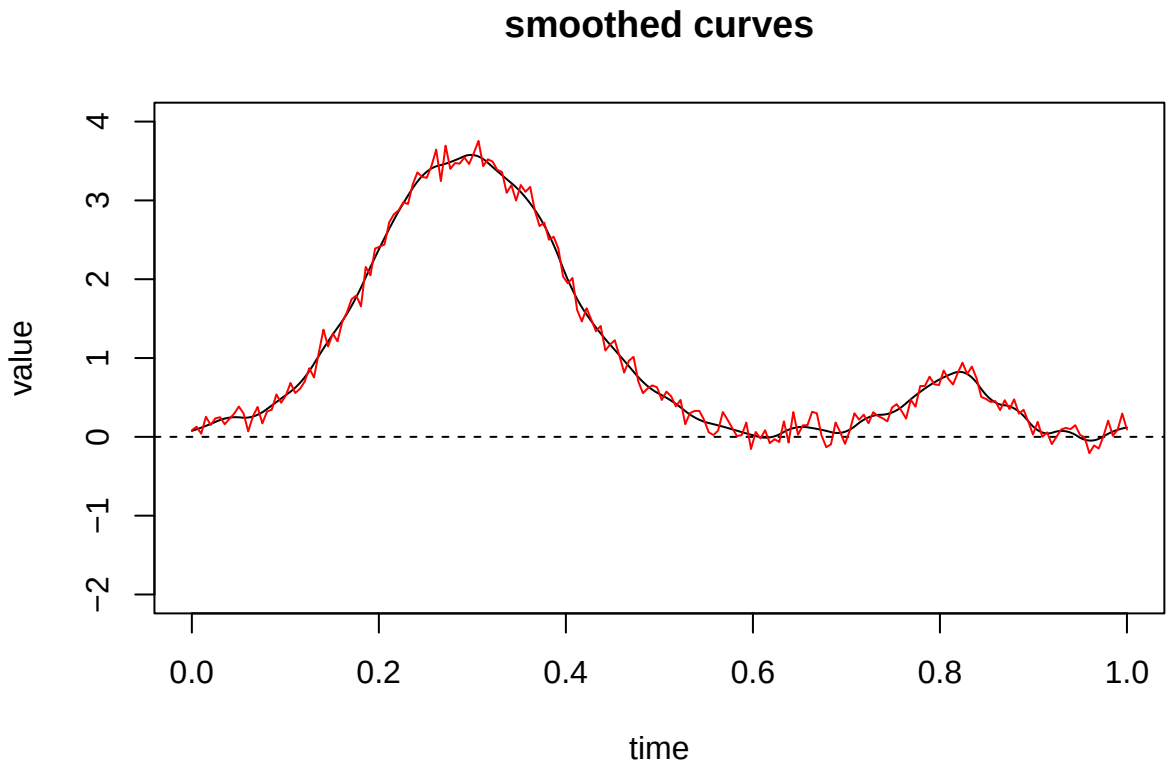
Plot smoothed curves

```
# Compute the mean of Beta in order to have data_post
sum_Beta <- matrix(0, p, n)
for(i in (burn_in+1):niter){
  sum_Beta <- sum_Beta + chains$Beta[[i]]
}
mean_Beta <- sum_Beta/(niter-burn_in)
data_post <- BaseMat %*% mean_Beta

# Compute the x value, create the basis and the functional object
```

```r
x <- seq(0, 1, length.out=r)
basis <- create.bspline.basis(rangeval=range(x), nbasis=p, norder=3)
data.fd <- Data2fd(y = data_post, argvals = x, basisobj = basis)

# Plot smoothed curves
plot.fd(data.fd[1,], main="smoothed curves", ylim=c(-2,4))
```

```
## [1] "done"
```

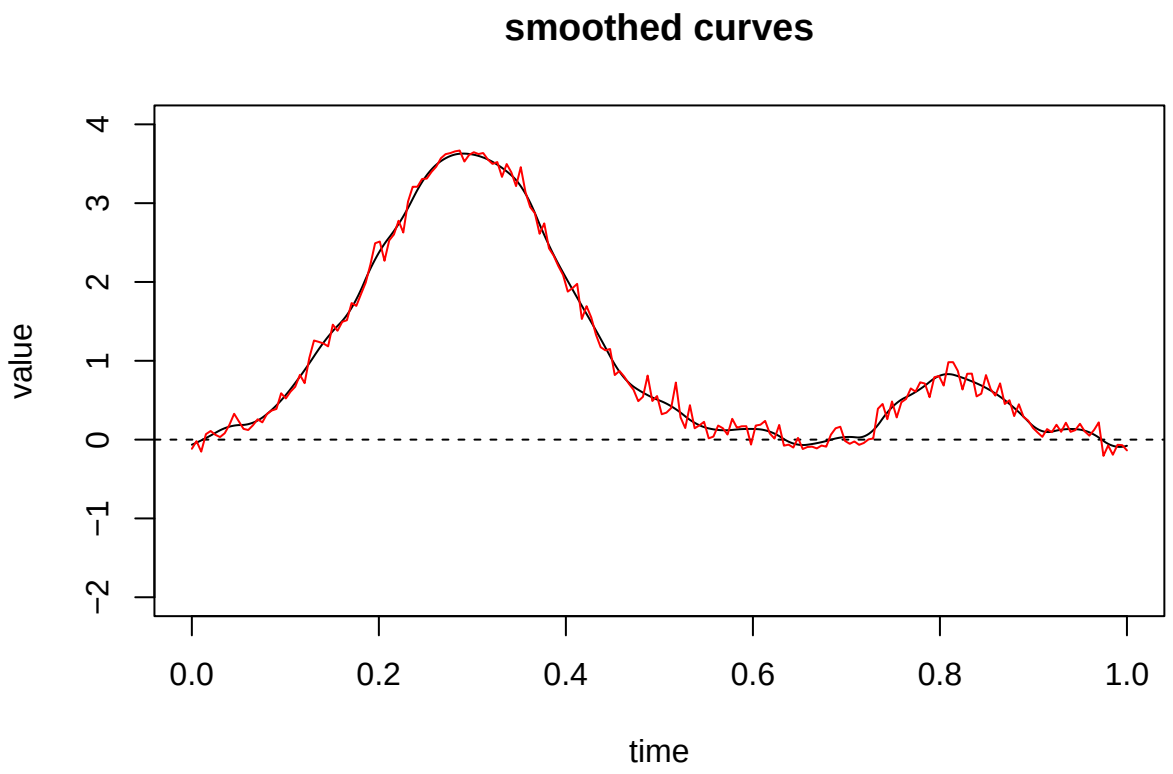
```r
# plot(x, data_post[,1], type='l', ylim=c(-2,4))
lines(x,y_hat_true[1,], main="smoothed curves", col='red')
```




```r
plot.fd(data.fd[300,], main="smoothed curves", ylim=c(-2,4))
```

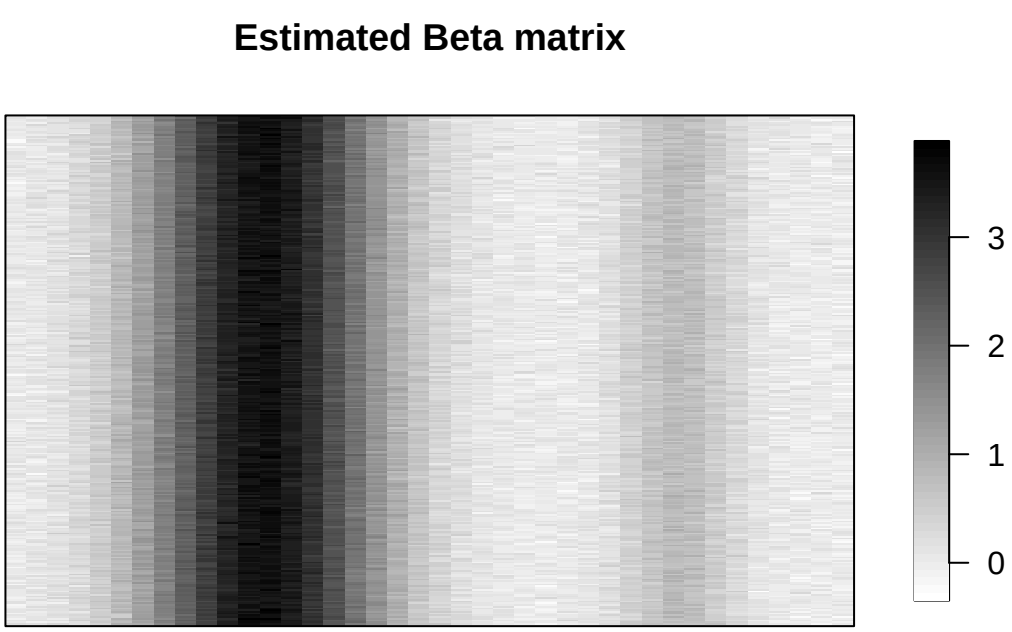
```
## [1] "done"
```

```r
# plot(x, data_post[,1], type='l', ylim=c(-2,4))
lines(x,y_hat_true[300,], main="smoothed curves", col='red')
```



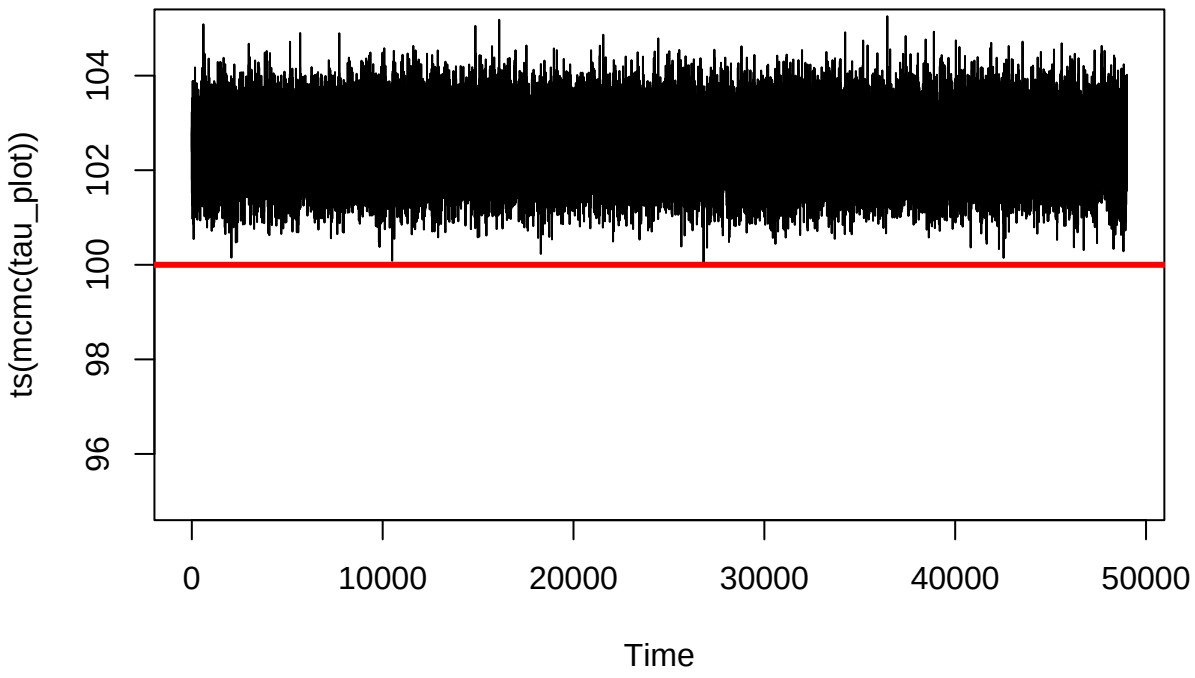
Plot of the final Beta matrix

```
ACutils::ACheatmap(
  chains$Beta[[niter]],
  use_x11_device = F,
  horizontal = F,
  main = "Estimated Beta matrix",
  center_value = NULL,
  col.upper = "black",
  col.center = "grey50",
  col.lower = "white"
)
```

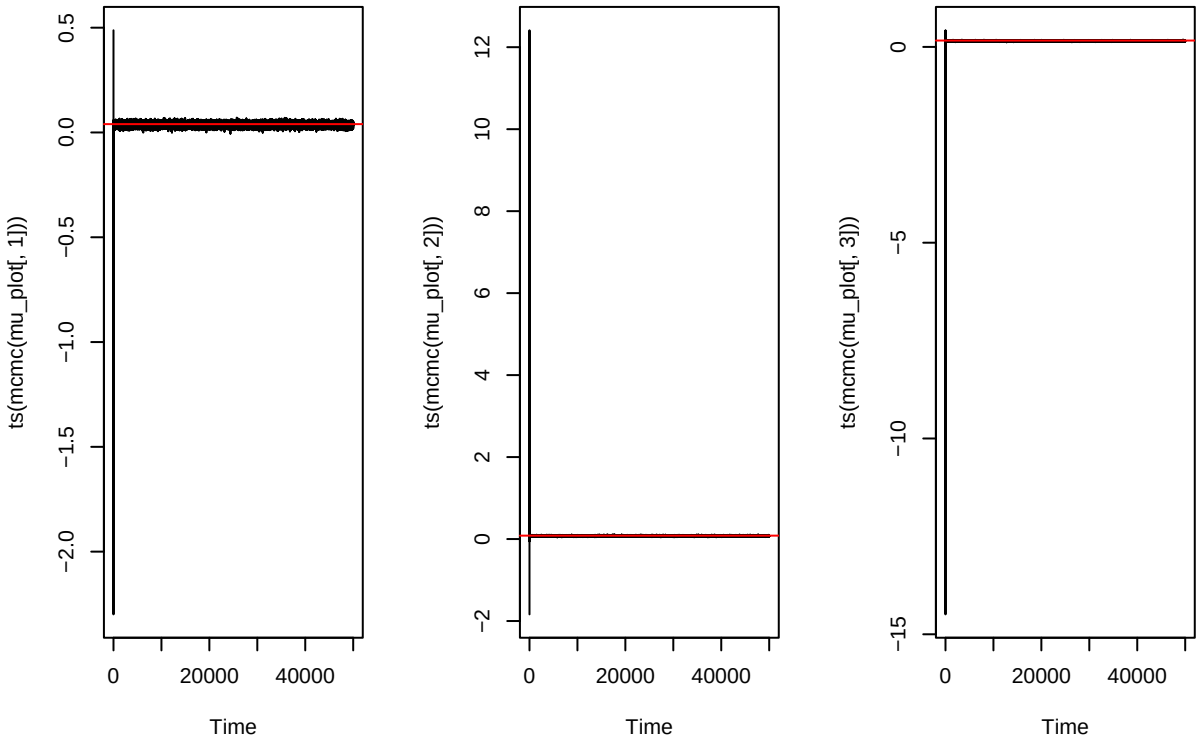


Traceplots (tau_eps, mu, beta)

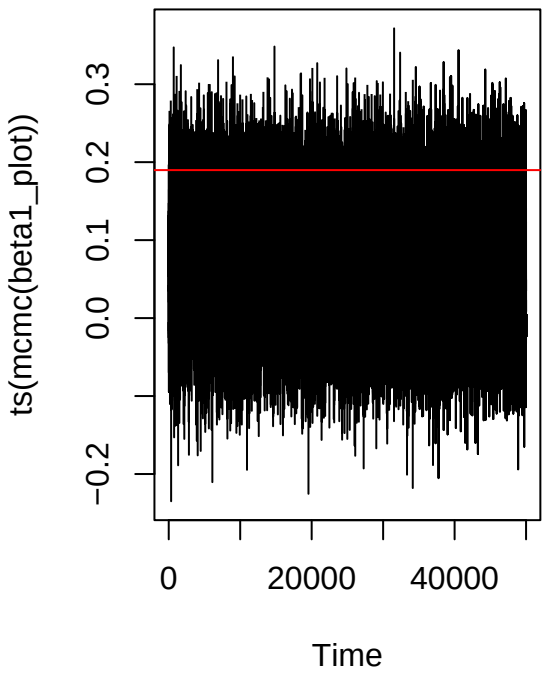
```
# tau_eps
tau_plot <- as.vector(chains$tau_eps)
tau_plot <- tau_plot[(burn_in+1):niter]
plot(ts(mcmc(tau_plot)), ylim=c(95,105))
abline(h=chains$tau_eps[[1]], col='red', lwd=3)    # initial value of tau_eps
```

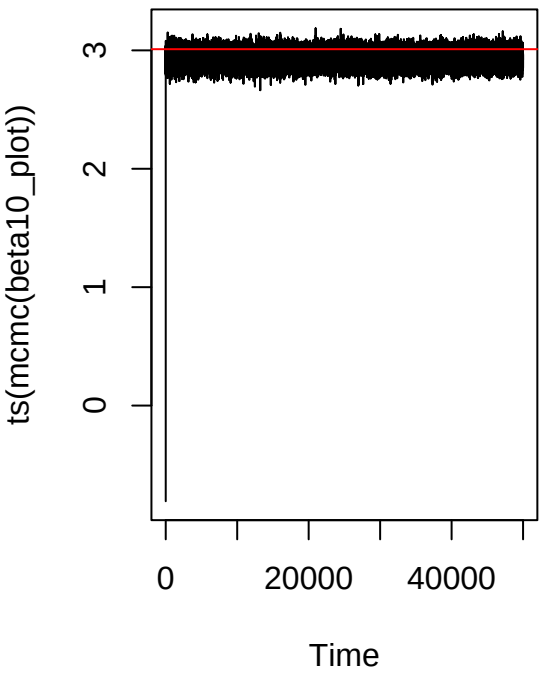



```
# mu
mu_plot <- matrix(0, niter, p)
for(i in 1:niter){
  mu_plot[i, ] <- chains$mu[[i]]
}
par(mfrow=c(1,3))
plot(ts(mcmc(mu_plot[, 1])))
abline(h=mu_true[1], col='red')
plot(ts(mcmc(mu_plot[, 2])))
abline(h=mu_true[2], col='red')
plot(ts(mcmc(mu_plot[, 3])))
abline(h=mu_true[3], col='red')
```



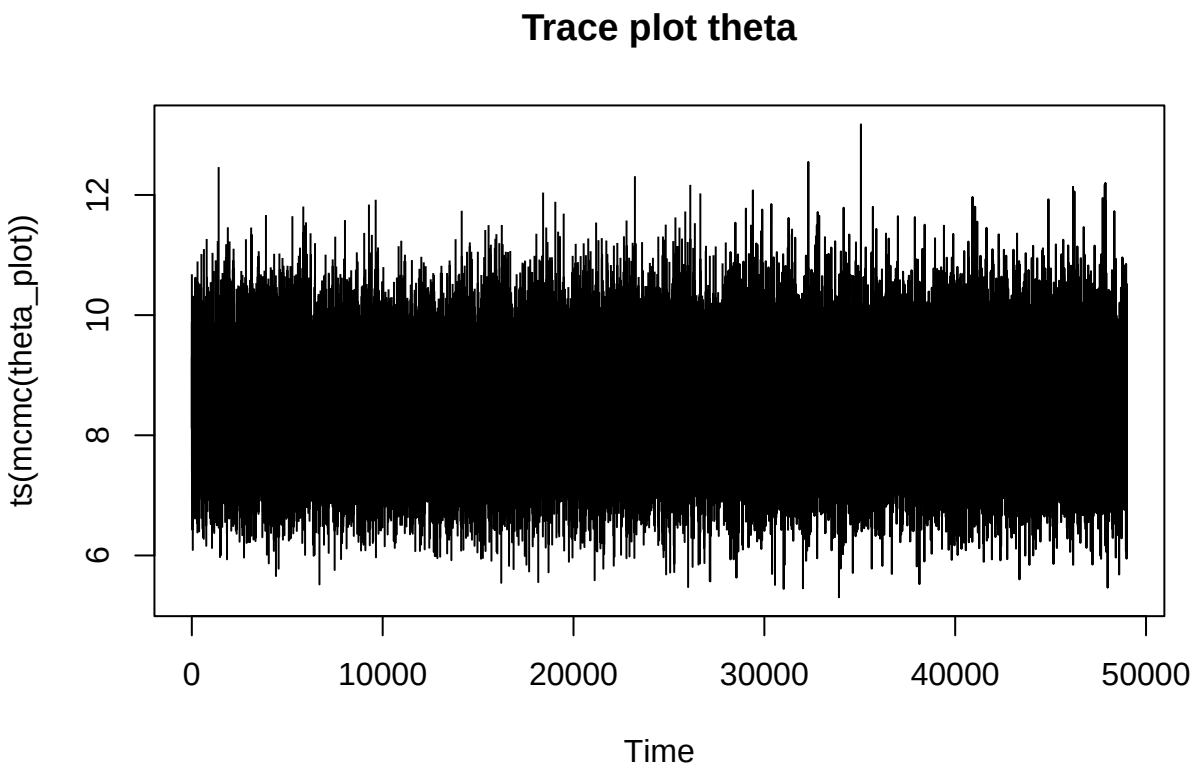

```r
# first element of first beta
beta1_plot <- rep(0, niter)
for(i in 1:niter){
  beta1_plot[i] <- chains$Beta[[i]][1,1]
}
# and 10th element of first beta
beta10_plot <- rep(0, niter)
for(i in 1:niter){
  beta10_plot[i] <- chains$Beta[[i]][10,1]
}

par(mfrow=c(1,2))
plot(ts(mcmc(beta1_plot)))
abline(h=beta_true[1,1], col='red')

plot(ts(mcmc(beta10_plot)))
abline(h=beta_true[1,10], col='red')
```



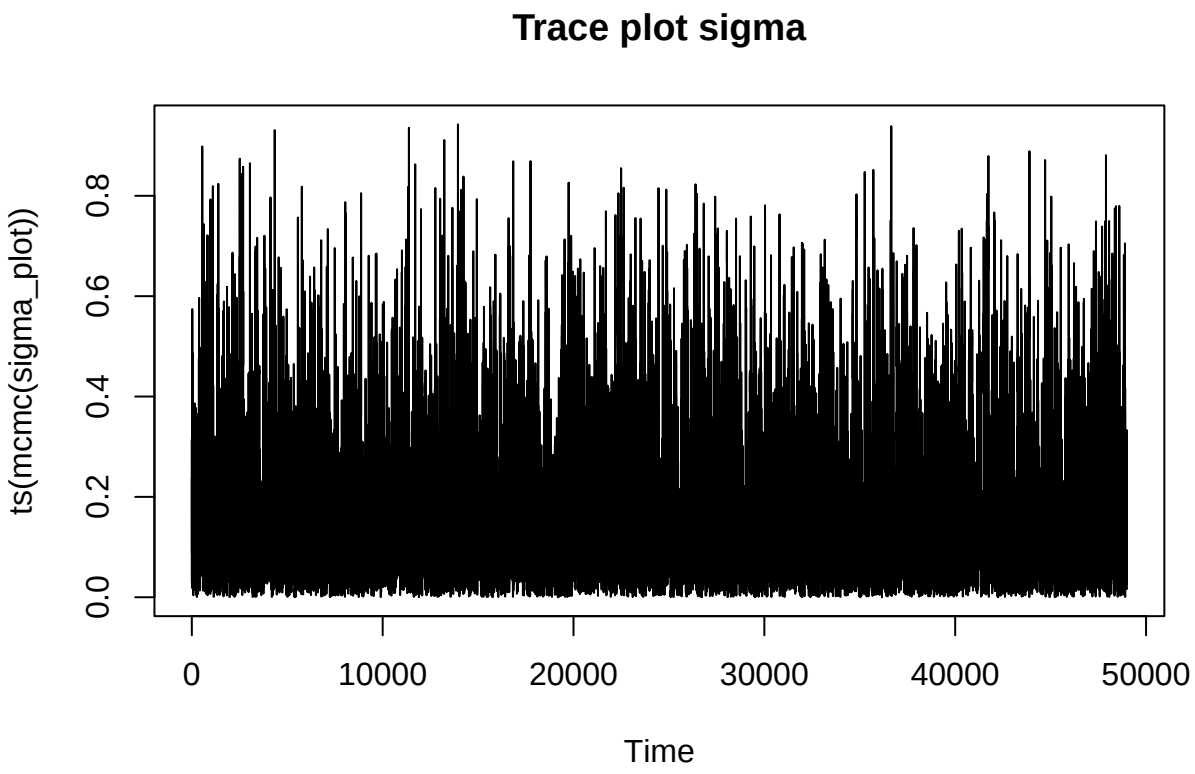


Theta e sigma plot

```
theta_plot <- as.vector(chains$theta)
theta_plot <- theta_plot[(burn_in+1):niter]
plot(ts(mcmc(theta_plot)), main='Trace plot theta')
```

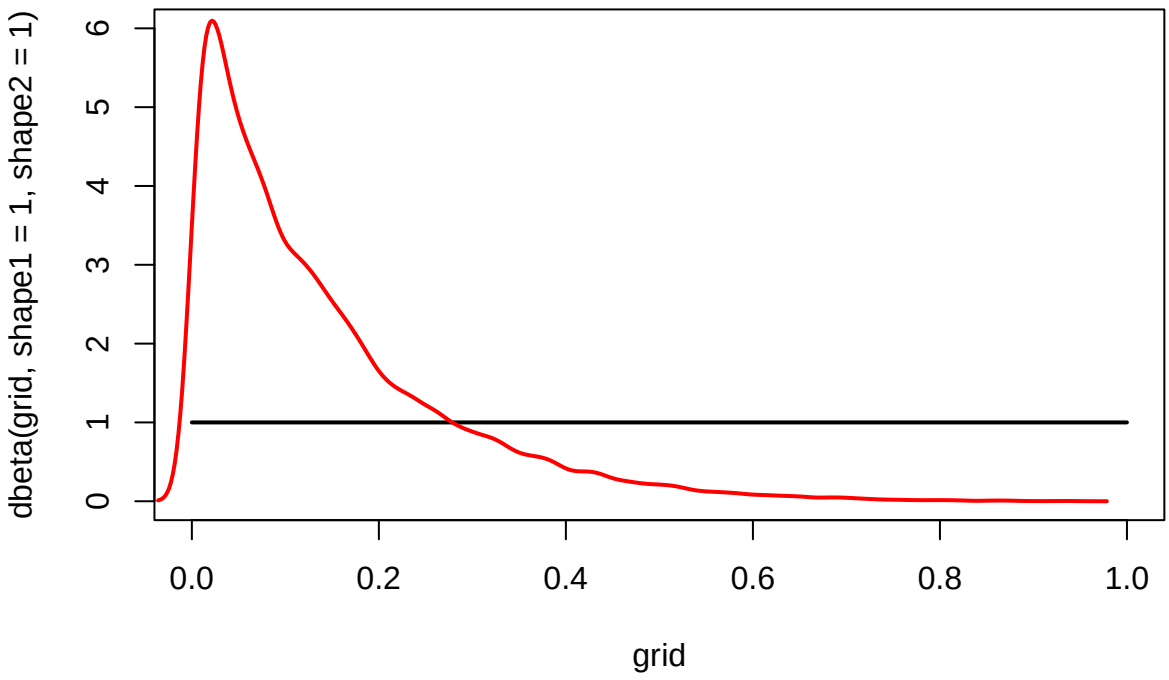



```
sigma_plot <- as.vector(chains$sigma)
sigma_plot <- sigma_plot[(burn_in+1):niter]
plot(ts(mcmc(sigma_plot)), main='Trace plot sigma')
```

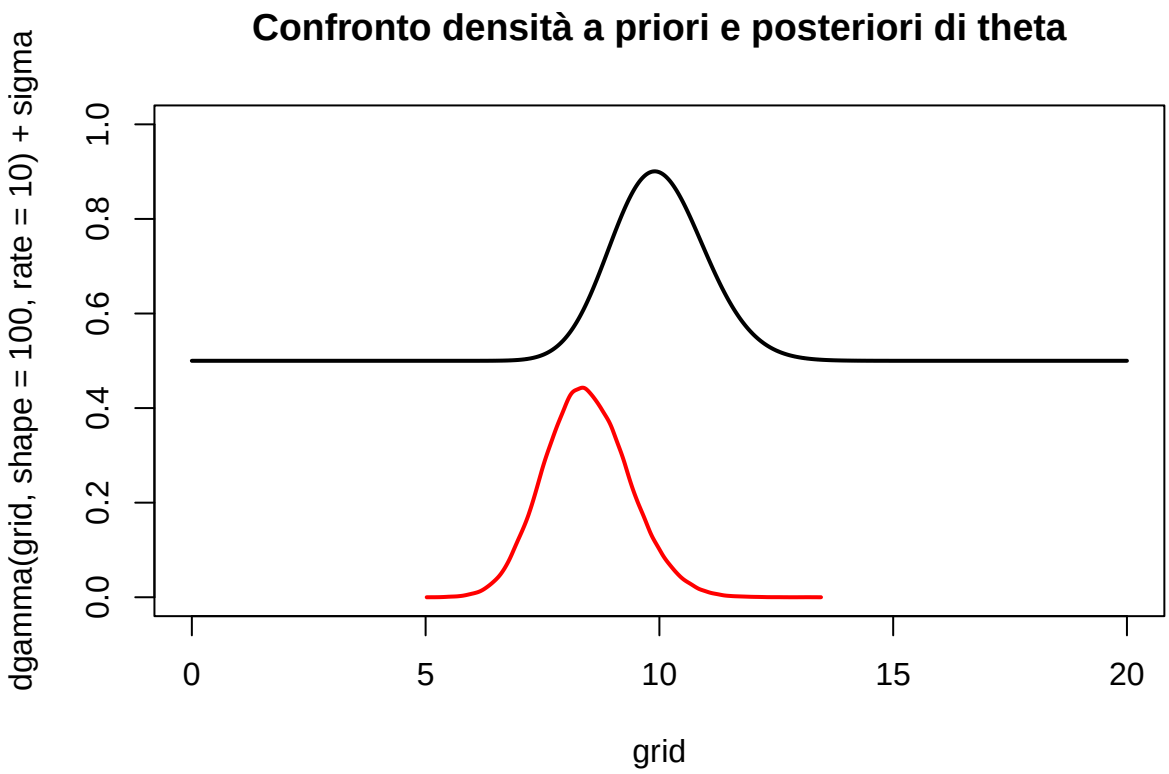


Confronto densità di sigma e theta

```
# confronto densità a priori
min_val <- 0
max_val <- 1
grid <- seq(min_val, max_val, length.out=10000)
plot(grid, dbeta(grid, shape1=1, shape2=1), type='l', lwd=2, ylim=c(0,6), main='Confronto densità a pri
lines(density(mcmc(sigma_plot))$x, density(mcmc(sigma_plot))$y, type='l', lwd=2, col='red') #posterior
```




```r
# confronto densità a priori
min_val <- 0
max_val <- 20
sigma <- 0.5
grid <- seq(min_val, max_val, length.out=10000)
plot(grid, dgamma(grid,shape=100, rate=10)+sigma, type='l', lwd=2, ylim=c(0,1), main='Confronto densità
lines(density(mcmc(theta_plot))$x, density(mcmc(theta_plot))$y, type='l', lwd=2, col='red') #posterior
```




```
# numero atteso di cluster, formula Martinez and Mena
theta_mean <- 10
n_mean_cluster <- lpochhammer(theta_mean + sigma,p, log=FALSE)/(sigma * lpochhammer(theta_mean + 1, p-1
n_mean_cluster
```

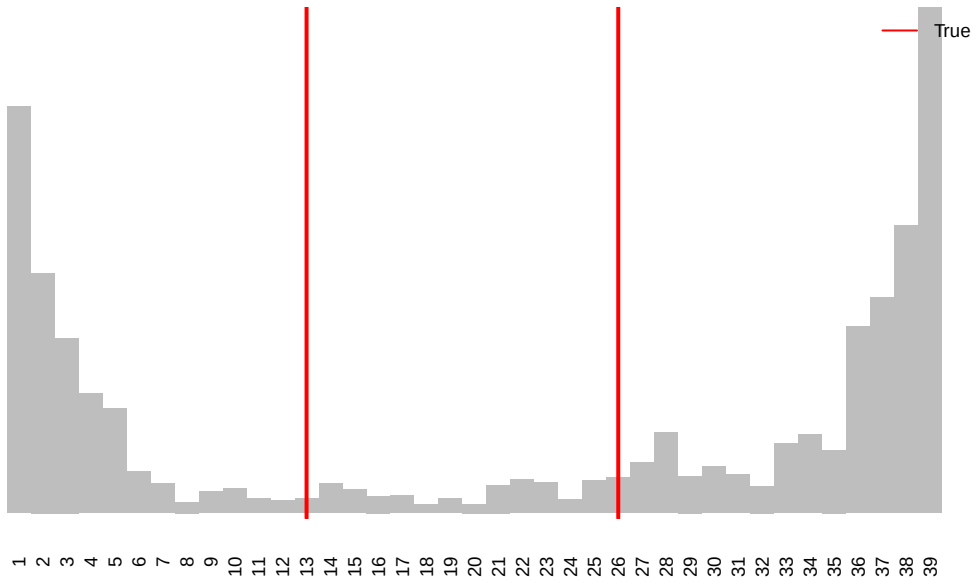
```
## [1] 25.17058
```

Posterior analysis

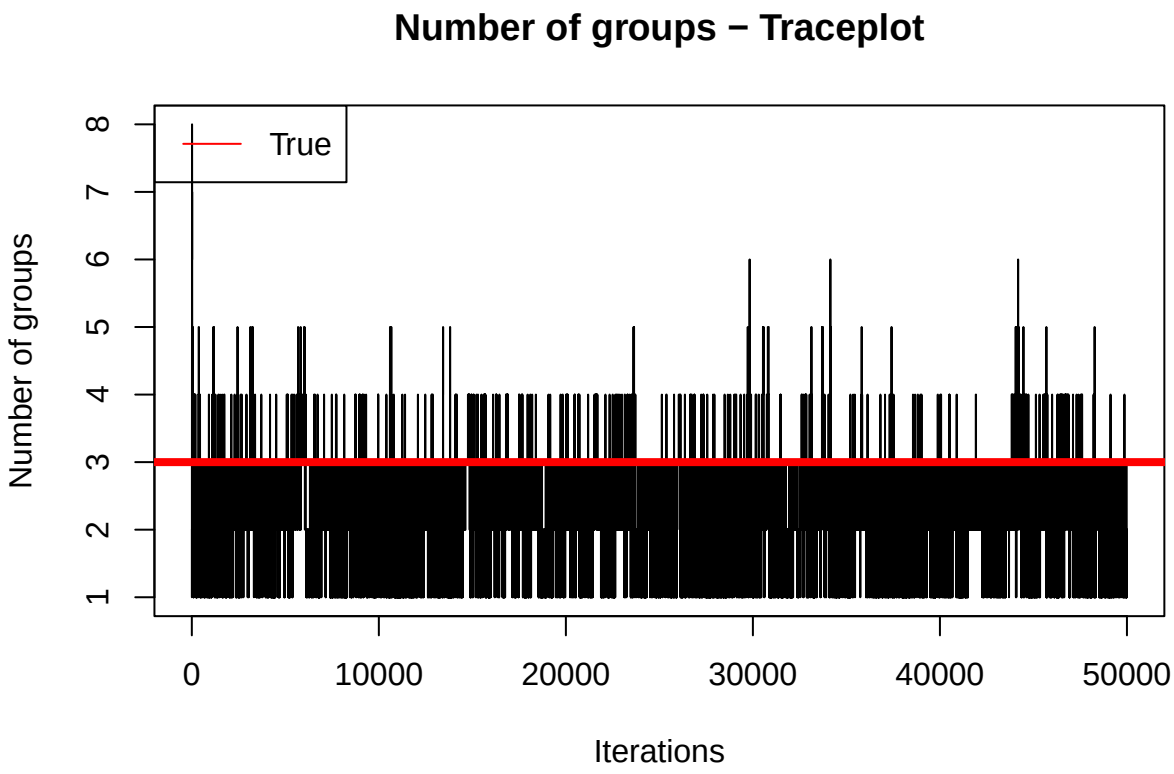
```
## Recomputing the partition in other forms and the number of groups

rho_true = c(13,13,14)
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 <- chains$rho
r = do.call(rbind, lapply(chains$rho, rho_to_r))
```

```
## Warning in (function (..., deparse.level = 1) : number of columns of result is
## not a multiple of vector length (arg 2)
```

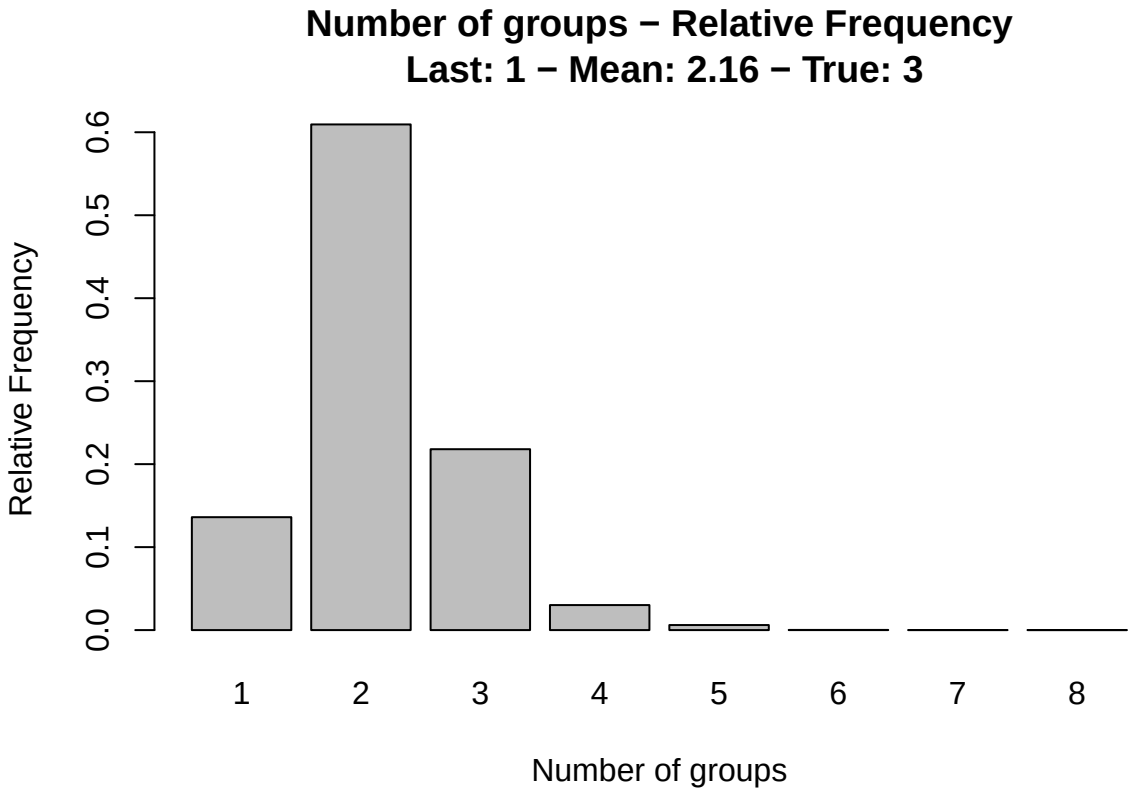
```r
z = do.call(rbind, lapply(chains$rho, rho_to_z))
num_clusters = do.call(rbind, lapply(chains$rho, length))
num_clusters = as.vector(num_clusters)



### 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[1:(p-1)],
  names = seq_along(bar_heights[1:(p-1)]),
  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)
```



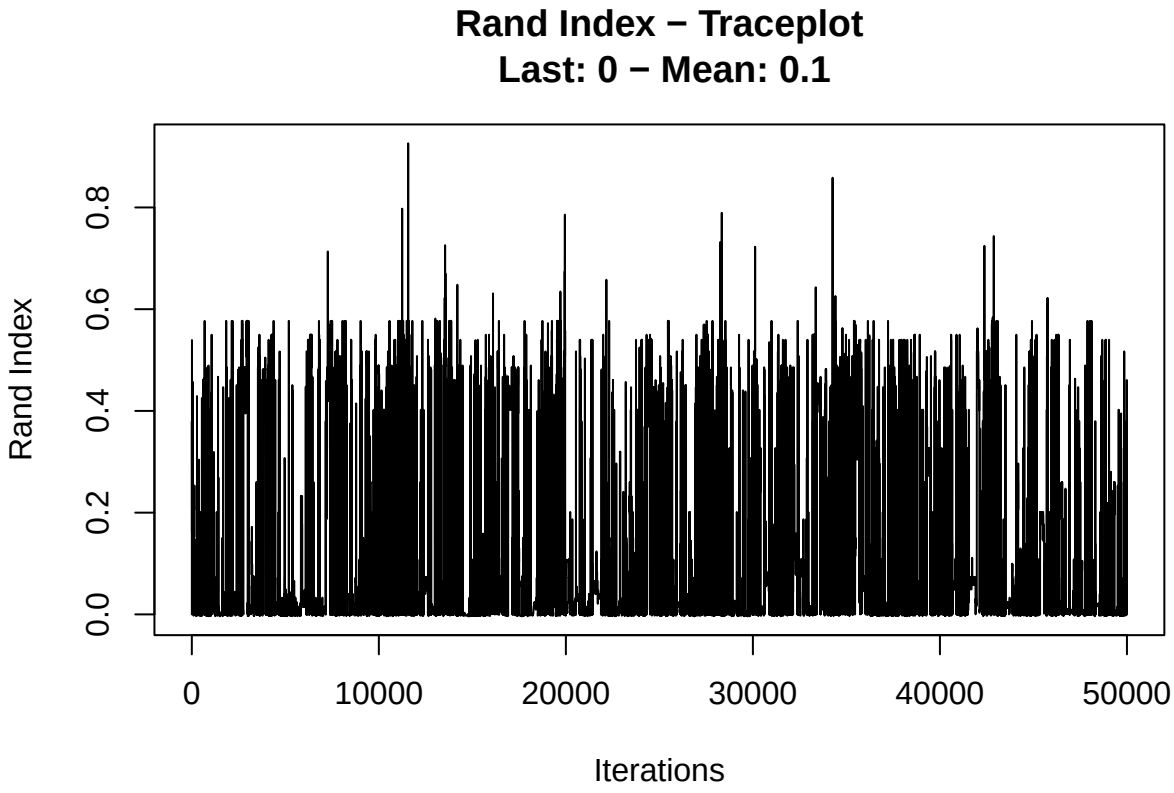

```
### Evolution of the number of clusters
plot(
  x = seq_along(num_clusters),
  y = num_clusters,
  type = "n",
  xlab = "Iterations",
  ylab = "Number of groups",
  main = "Number of groups - Traceplot"
)
lines(x = seq_along(num_clusters), y = num_clusters)
abline(h = length(z_to_rho(z_true)),
       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 = "Relative Frequency",
  main = paste(
    "Number of groups - Relative Frequency\n",
    "Last:",
    tail(num_clusters, n = 1),
    "- Mean:",
    round(mean(num_clusters), 2),
    "- True:",
    num_clusters_true
  )
)
```




```
### 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)
```




```
### Retrieving best partition using VI on visited ones (order is guaranteed here)
# 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] 0.3145863
```

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

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [39] 1 1
```

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

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

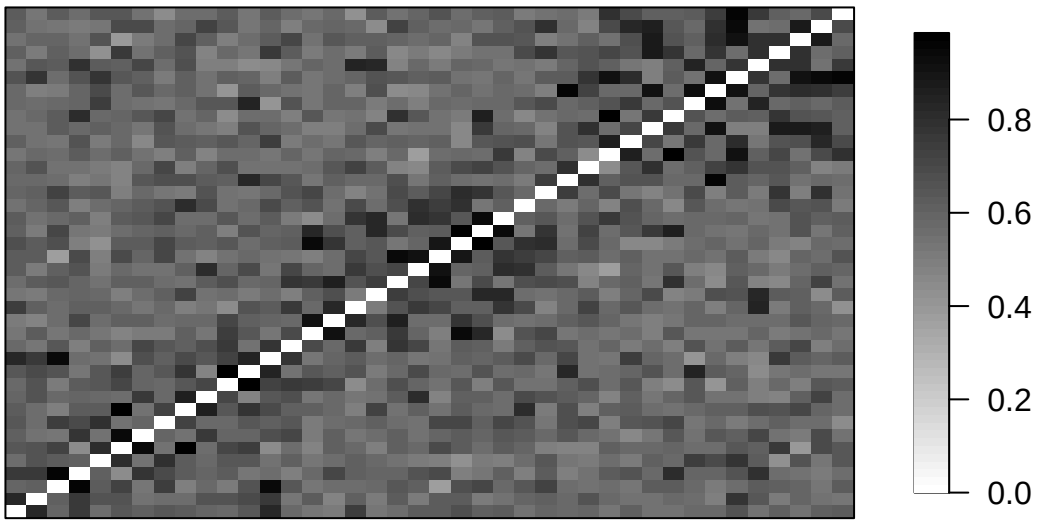
```r
## Graph

# Extract last plinks
last_plinks = tail(chains$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] 0.912
```

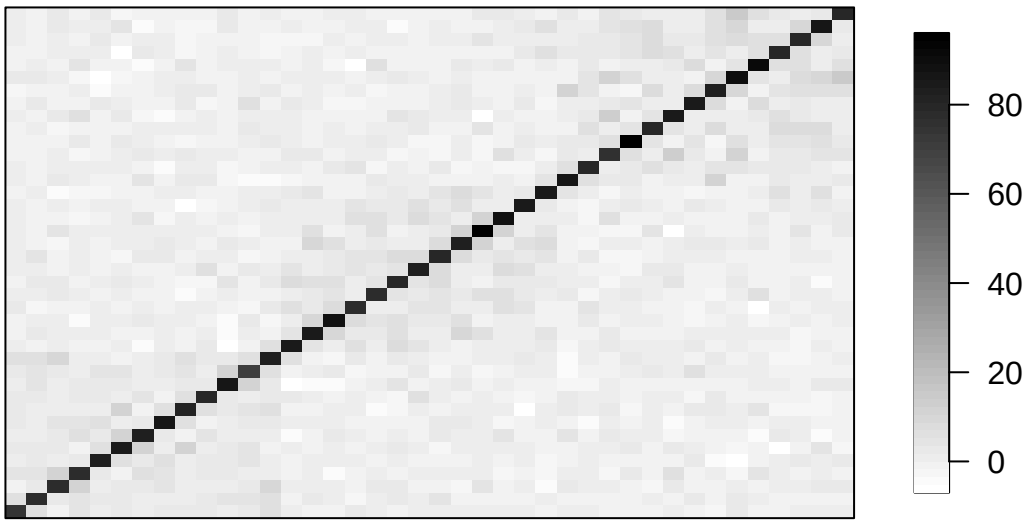
```r
G_est = bfdr_select$best_truncated_graph

### Standardized Hamming distance
SHD = sum(abs(simKG$Graph - G_est)) / (p^2 - p)
SHD
```

```
## [1] 0.3217949
```

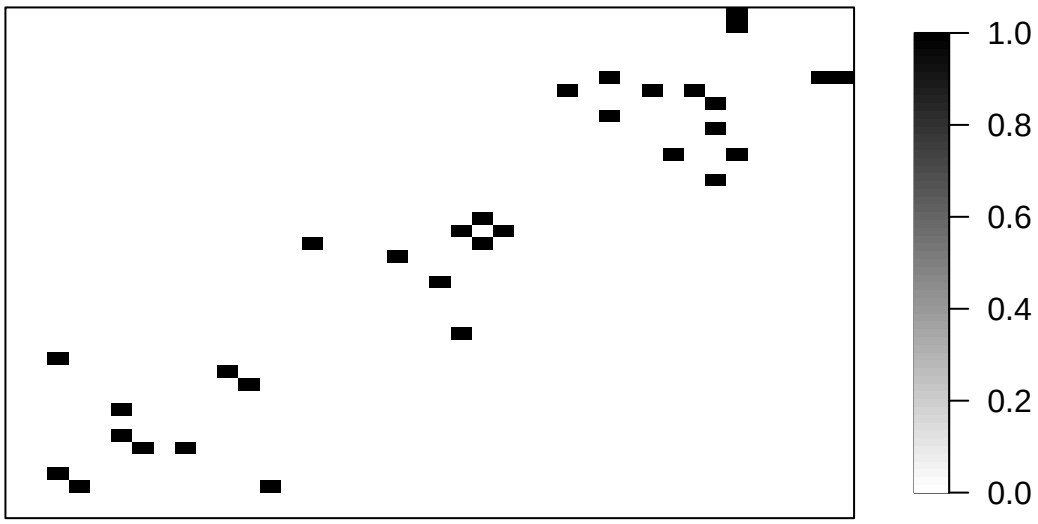
```r
### Plot estimated matrices
ACutils::ACheatmap(
  last_plinks,
  use_x11_device = F,
  horizontal = F,
  main = "Estimated plinks matrix",
  center_value = NULL,
  col.upper = "black",
  col.center = "grey50",
  col.lower = "white"
)
```



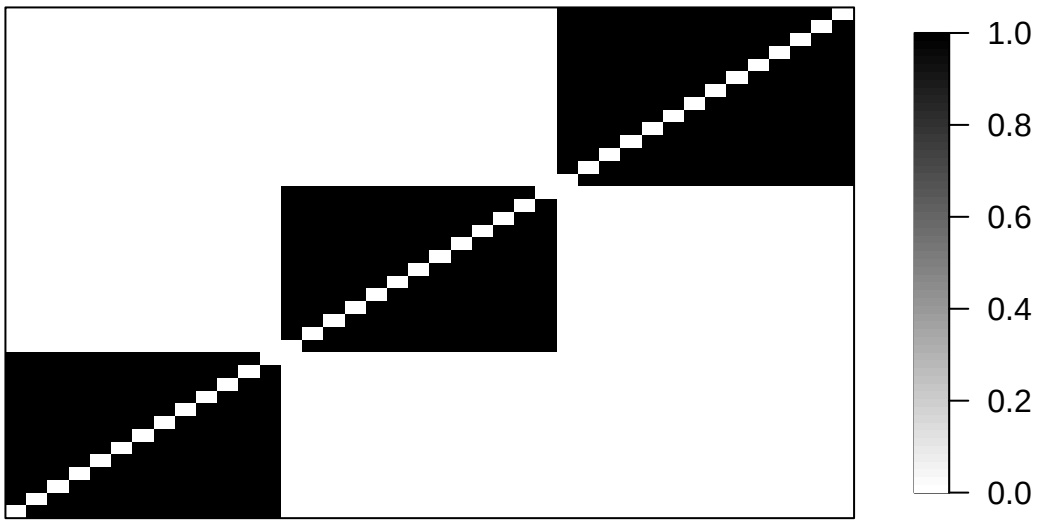

```
ACutils::ACheatmap(
  tail(chains$K,n=1)[[1]],
  use_x11_device = F,
  horizontal = F,
  main = "Estimated Precision matrix",
  center_value = NULL,
  col.upper = "black",
  col.center = "grey50",
  col.lower = "white"
)
```



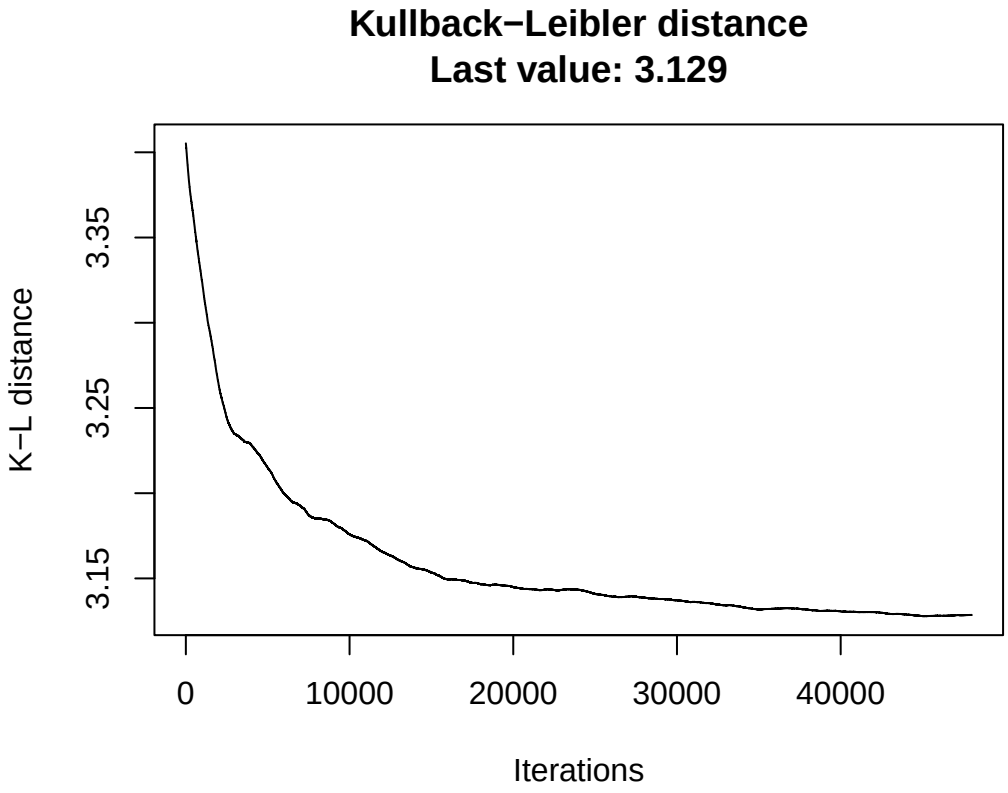

```
ACutils::ACheatmap(
  G_est,
  use_x11_device = F,
  horizontal = F,
  main = "Estimated Graph",
  center_value = NULL,
  col.upper = "black",
  col.center = "grey50",
  col.lower = "white"
)
```




```
# Plot the true graph
# remove self loops
simKG$Graph[col(simKG$Graph)==row(simKG$Graph)] = 0

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




```
### Evolution of the Kullback-Leibler
kl_dist = do.call(rbind, lapply(chains$K, function(k) {
  ACutils::KL_dist(simKG$Prec, k)
}))

last = round(tail(kl_dist, n=1), 3)
plot(
  x = seq_along(kl_dist[2000:length(kl_dist)]),
  y = kl_dist[2000:length(kl_dist)],
  type = "n",
  xlab = "Iterations",
  ylab = "K-L distance",
  main = paste("Kullback-Leibler distance\nLast value:", last)
)
lines(x = seq_along(kl_dist[2000:length(kl_dist)]), y = kl_dist[2000:length(kl_dist)])
```




```
# non c'è burn in!
```