Purees data

2024-01-13

Load libraries

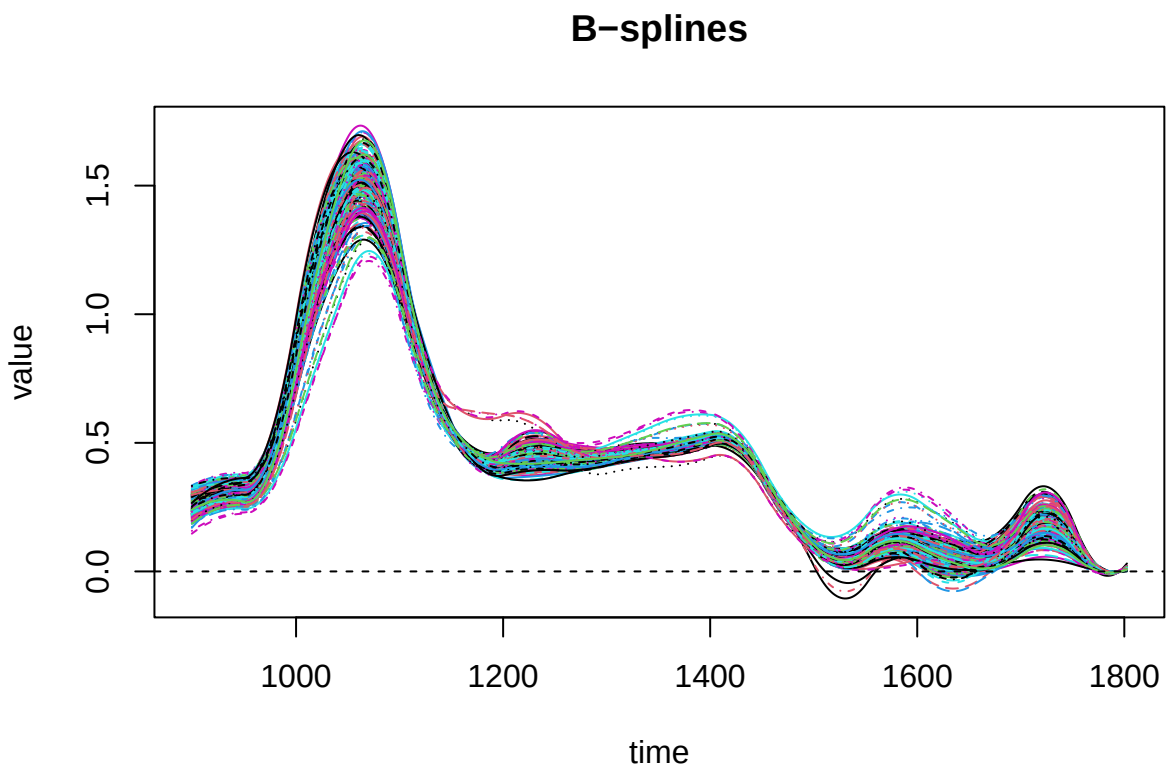
```
shhh = function(lib_name){ # It's a library, so shhh!
  suppressWarnings(suppressMessages(require(lib_name, character.only = TRUE)))
}
shhh("tidyverse")
shhh("ACutils")
shhh("mvtnorm")
shhh("salso")
shhh("FGM")
shhh("gmp")
shhh("mcclust")
shhh("mcclust.ext")
shhh("logr")
shhh("tidygraph")
shhh("ggraph")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

Purees data

```
load("purees.Rdat")
curves = purees$data
wavelengths = purees$wavelengths
strawberry <- curves[which(curves$Group == "Strawberry"), ]
data = strawberry[,-1]*100
data = as.matrix(data)    # n x r

#Generate basis
p =  20
n = dim(data)[1]
r = dim(data)[2]
range_x = range(wavelengths)

data_W <- t(as.matrix(data))
basis <- create.bspline.basis(rangeval=range(wavelengths), nbasis=p, norder=3)
data.fd <- Data2fd(y = data_W, argvals = wavelengths, basisobj = basis)
plot.fd(data.fd, main="B-splines")
```




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

```
BaseMat <- eval.basis(wavelengths,basis) # matrix r x p
```

Initialization

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

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       = 10,
    d_theta       = 1,
    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 = 0.3,
    beta_sig2     = 0.2,
```

```
    d             = 3
)
```

Gibbs sampler

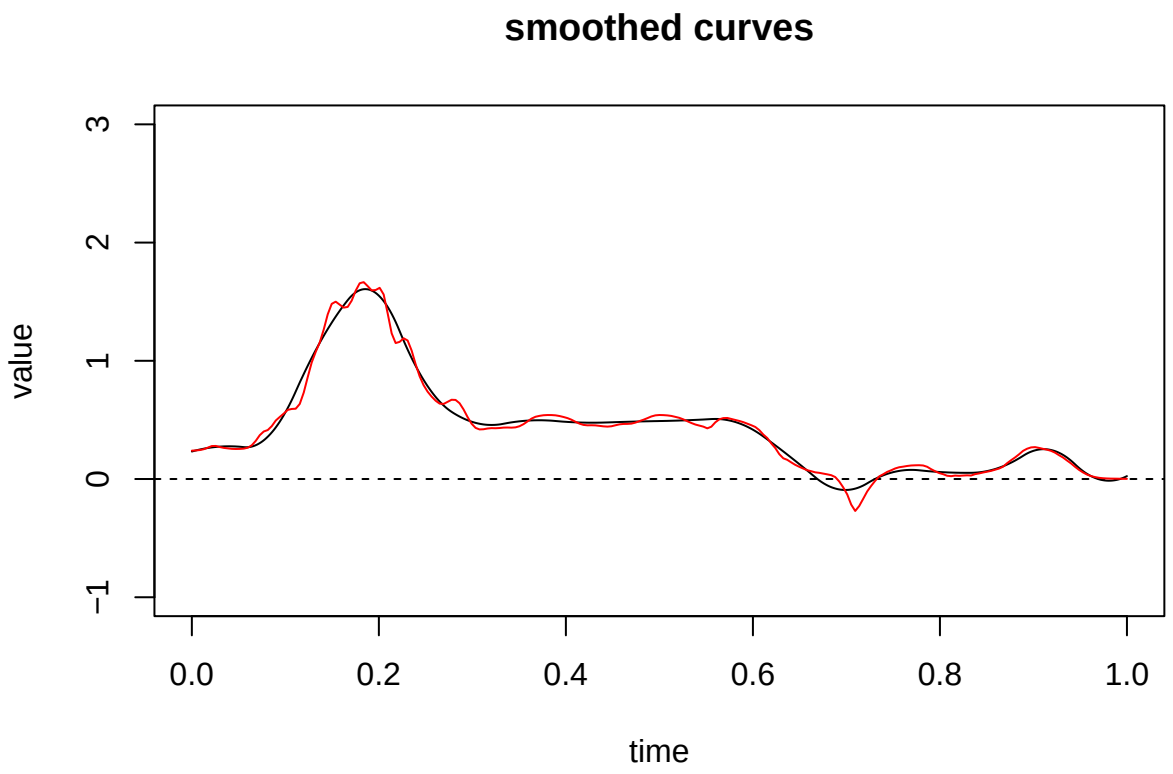
```
chains = Gibbs_sampler_update(
  set_UpdateParamsGSL_list,
  niter,
  initialization_values,
  alpha_target           = 0.234,
  alpha_add              = 0.5,
  adaptation_step        = 1/(p*1000),
  seed                   = 22111996,
  update_sigma_prior     = TRUE,
  update_theta_prior     = TRUE,
  update_weights         = TRUE,
  update_partition       = TRUE,
  update_graph           = TRUE,
  perform_shuffle        = TRUE
)
```

Useful plots: 1. 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
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(-1,3))
```

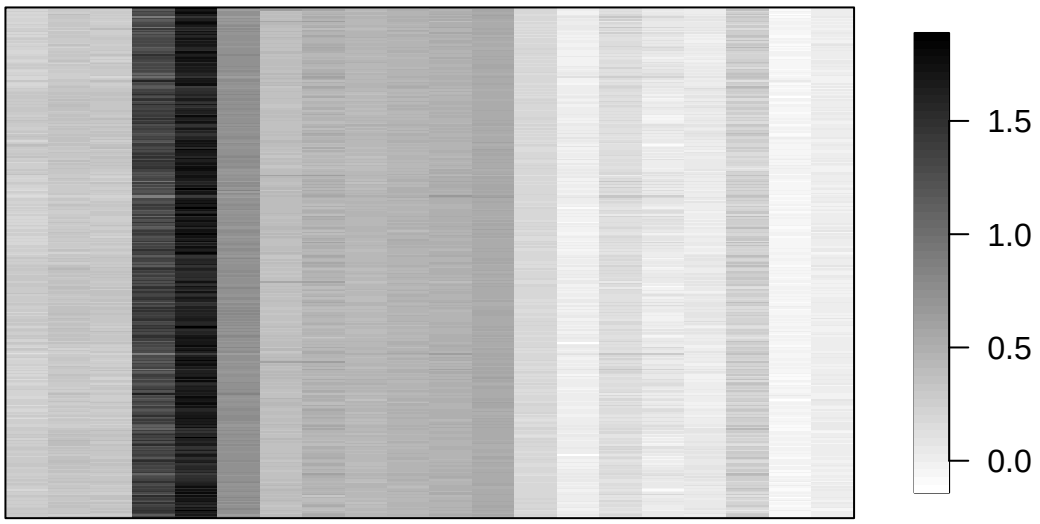
```
## [1] "done"
```

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



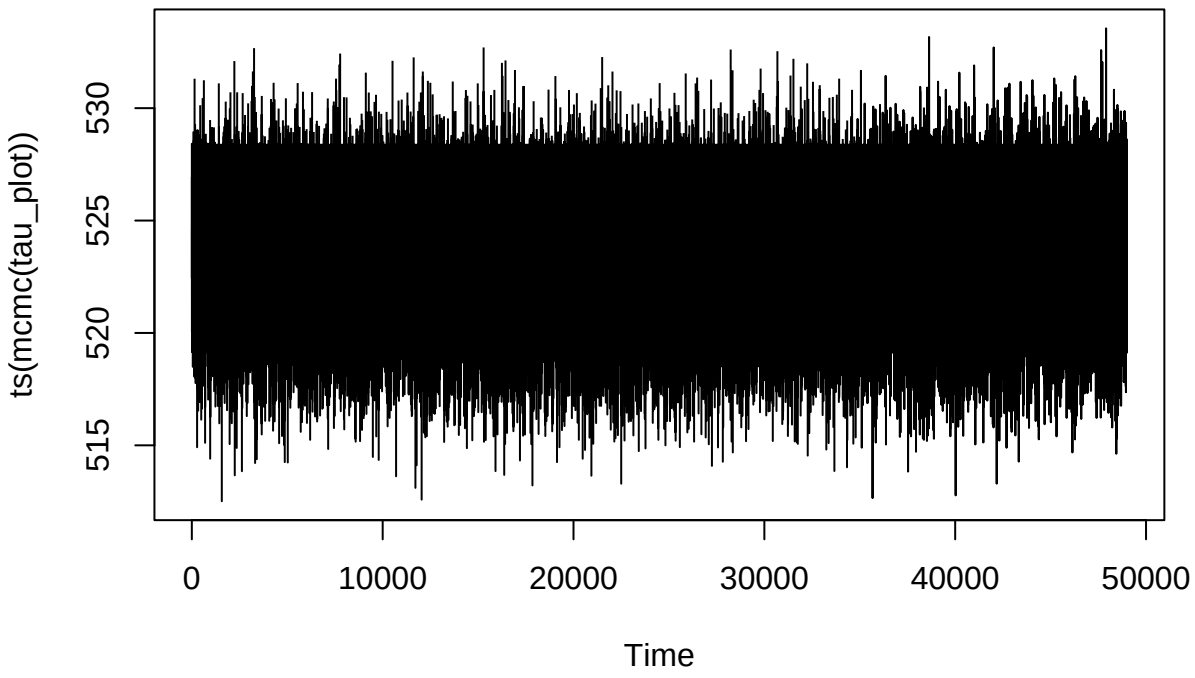
Useful plots: 2. Plot of the final Beta matrix

```
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"
)
```

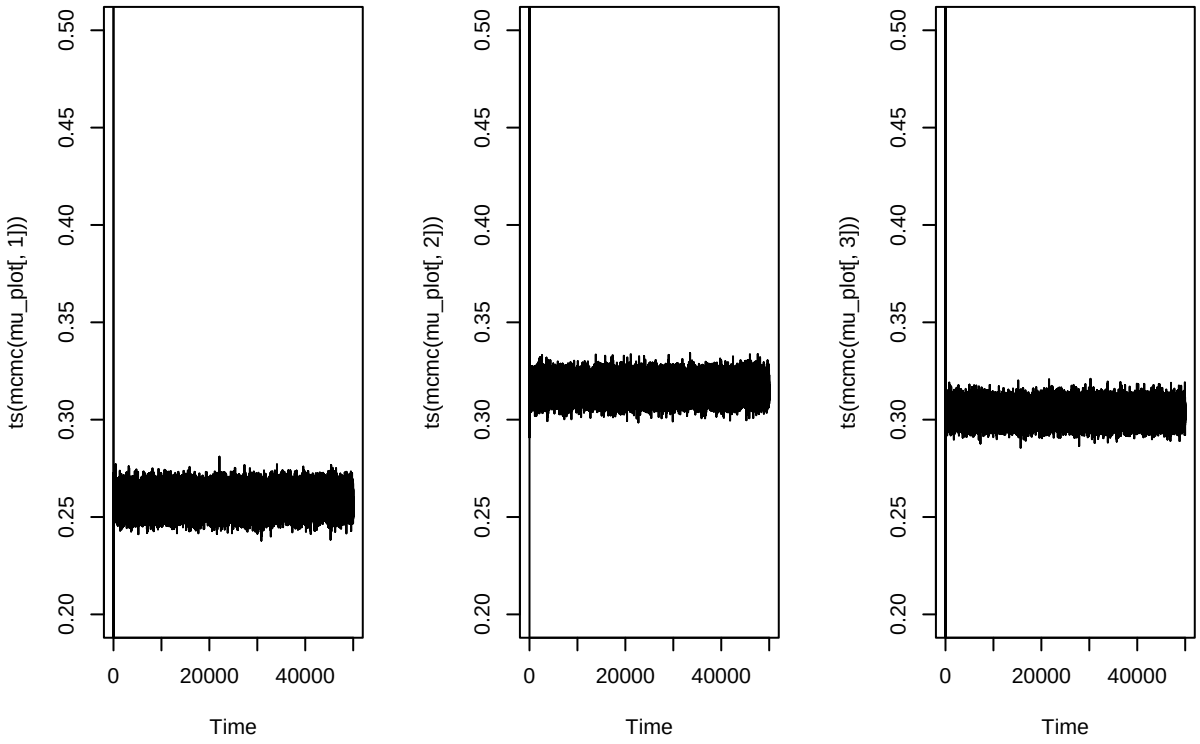


Useful plots: 2. Traceplots (tau_eps, mu, beta)

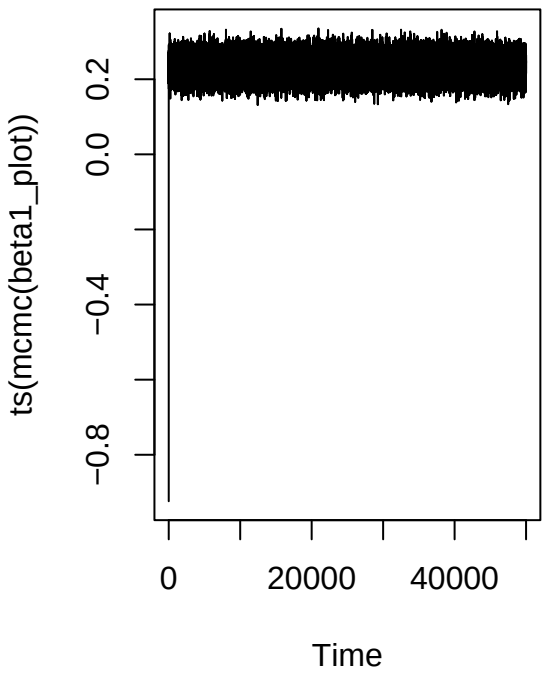
```
# tau_eps
tau_plot <- as.vector(chains$tau_eps)
tau_plot <- tau_plot[(burn_in+1):niter]
plot(ts(mcmc(tau_plot)))
```

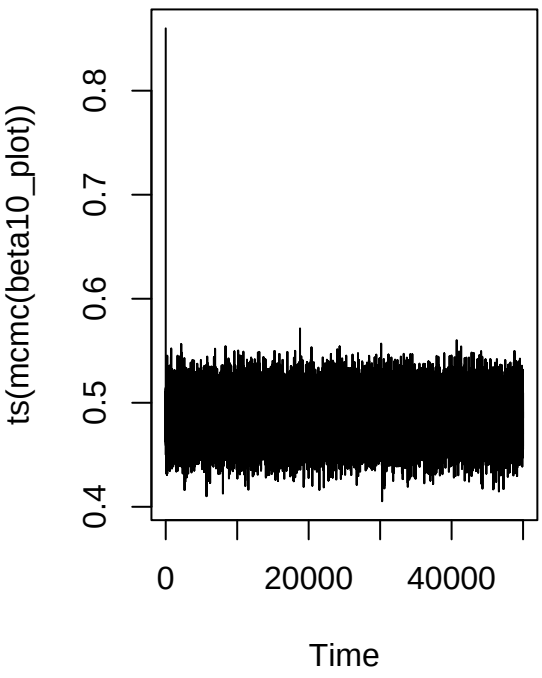



```
# mu
mu_plot <- matrix(0, niter, p)
for(i in 1:niter){
  mu_plot[i, ] <- chains$mu[[i]]
}
par(mfrow=c(1,3))
plot(ts(mcmc(mu_plot[, 1])), ylim=c(0.2,0.5))
plot(ts(mcmc(mu_plot[, 2])), ylim=c(0.2,0.5))
plot(ts(mcmc(mu_plot[, 3])), ylim=c(0.2,0.5))
```



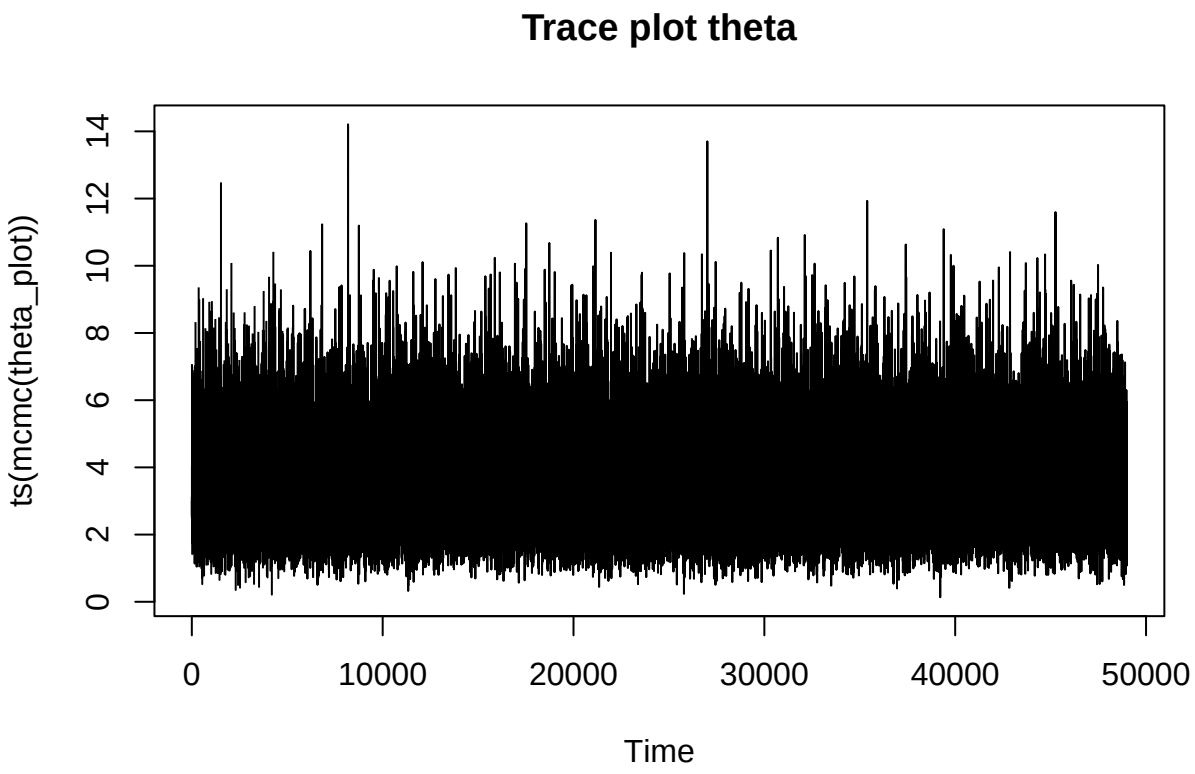

```
# first element of first beta
beta1_plot <- rep(0, niter)
for(i in 1:niter){
  beta1_plot[i] <- chains$Beta[[i]][1,1]
}
# and 10th element of first beta
beta10_plot <- rep(0, niter)
for(i in 1:niter){
  beta10_plot[i] <- chains$Beta[[i]][10,1]
}

par(mfrow=c(1,2))
plot(ts(mcmc(beta1_plot)))

plot(ts(mcmc(beta10_plot)))
```



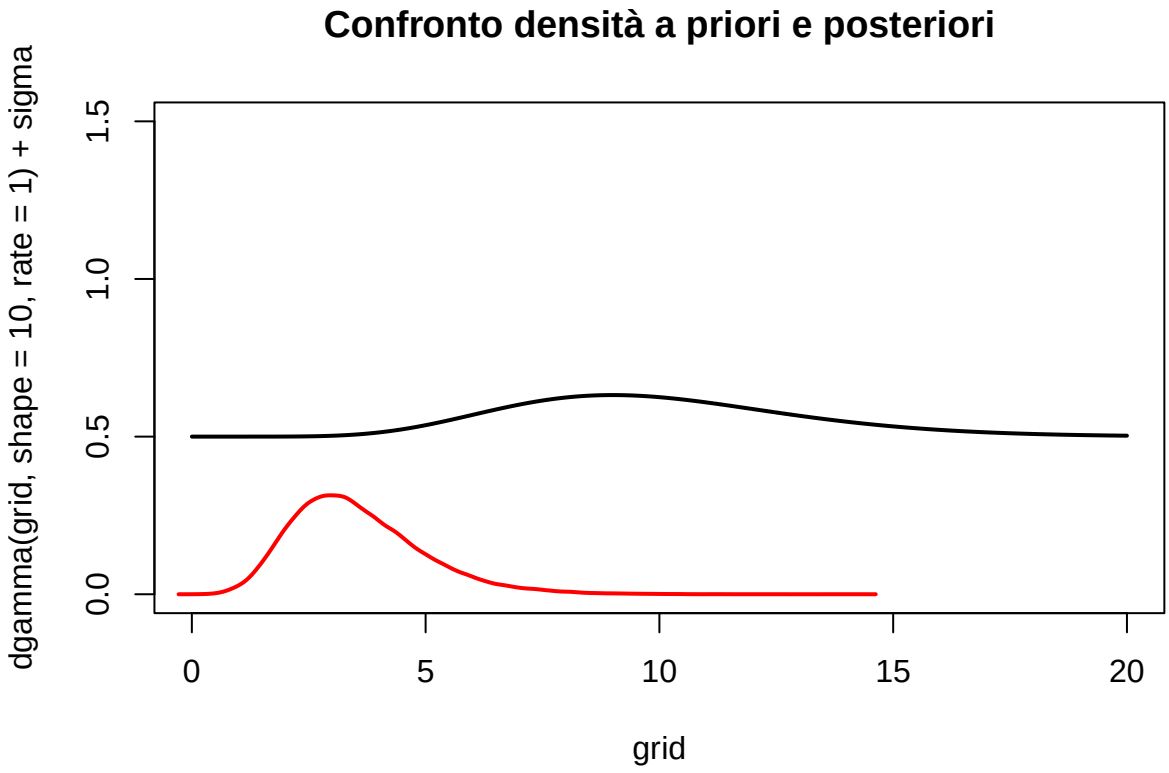


Posterior analysis

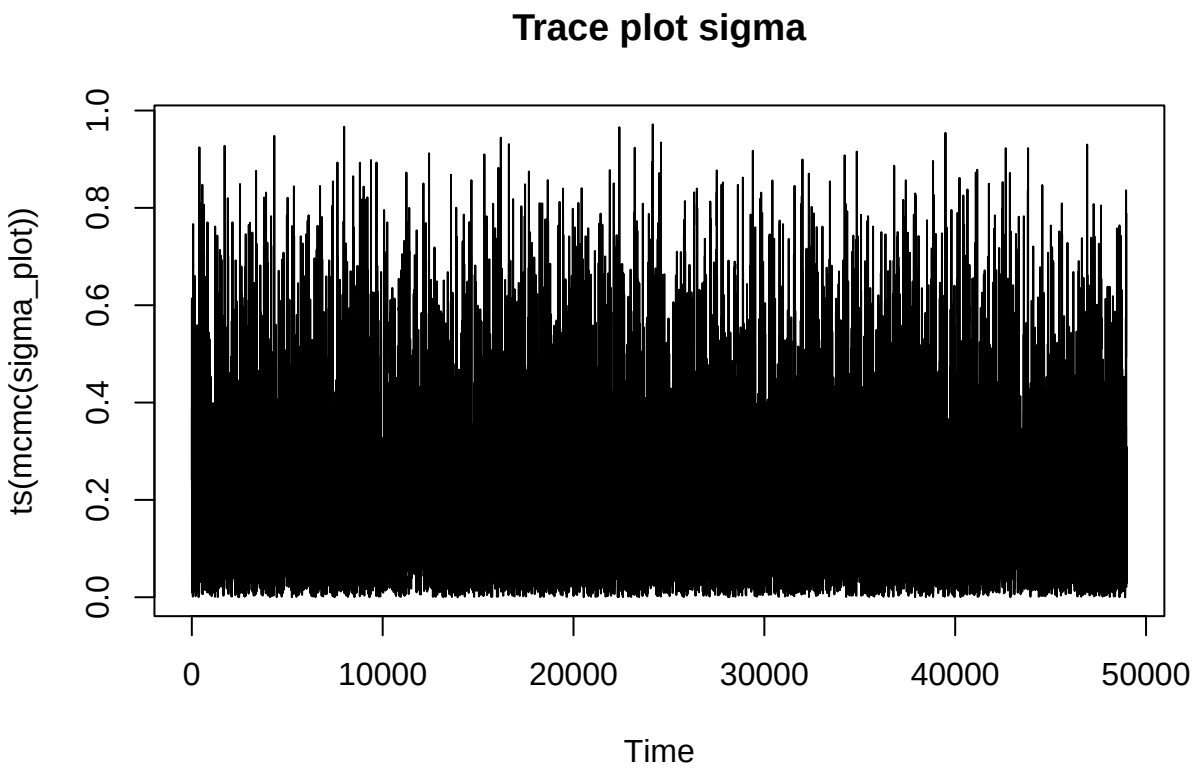
```
#prior and posterior of theta
#traceplot theta
theta_plot <- as.vector(chains$theta)
theta_plot <- theta_plot[(burn_in+1):niter]
plot(ts(mcmc(theta_plot)), main='Trace plot theta')
```



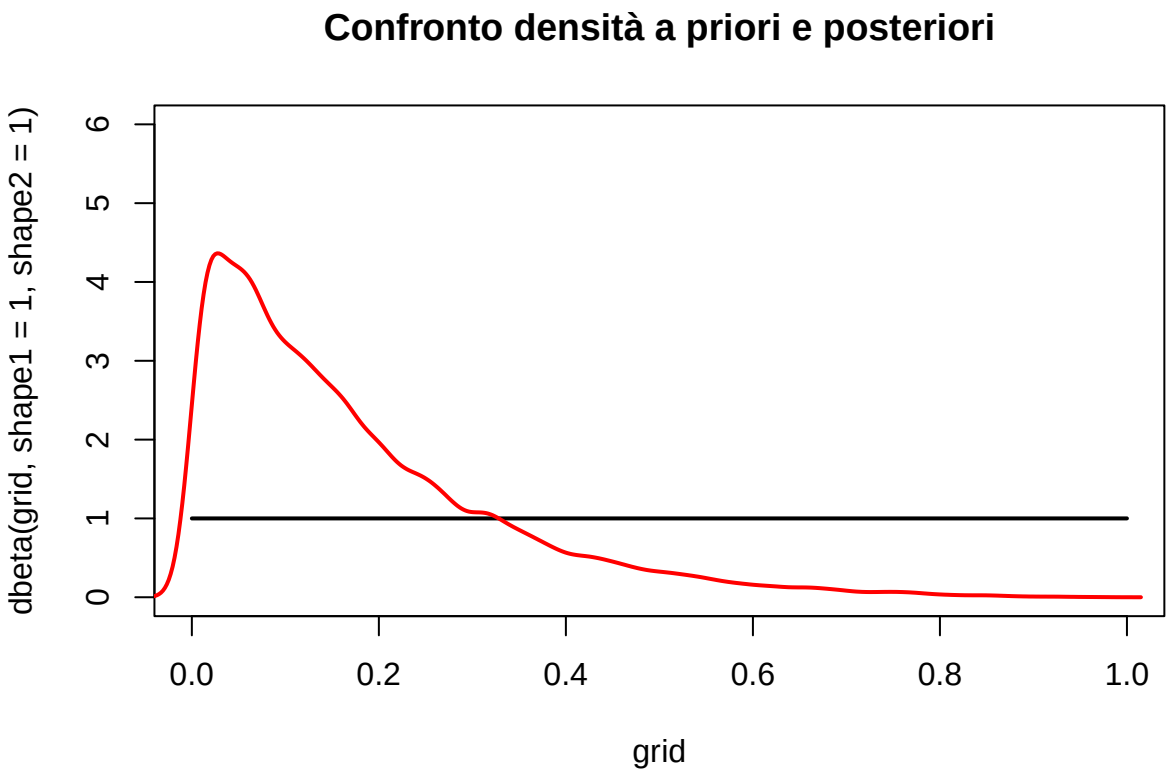

```r
# confronto densità a priori e posteriori
min_val <- 0
max_val <- 20
sigma <- 0.5
grid <- seq(min_val, max_val, length.out=10000)
plot(grid, dgamma(grid,shape=10, rate=1)+sigma, type='l', lwd=2, ylim=c(0,1.5), main='Confronto densità
lines(density(mcmc(theta_plot))$x, density(mcmc(theta_plot))$y, type='l', lwd=2, col='red') #posterior
```




```
#prior and posterior of sigma

#traceplot sigma
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 a priori e posteriori
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 prio
lines(density(mcmc(sigma_plot))$x, density(mcmc(sigma_plot))$y, type='l', lwd=2, col='red') #posterior
```

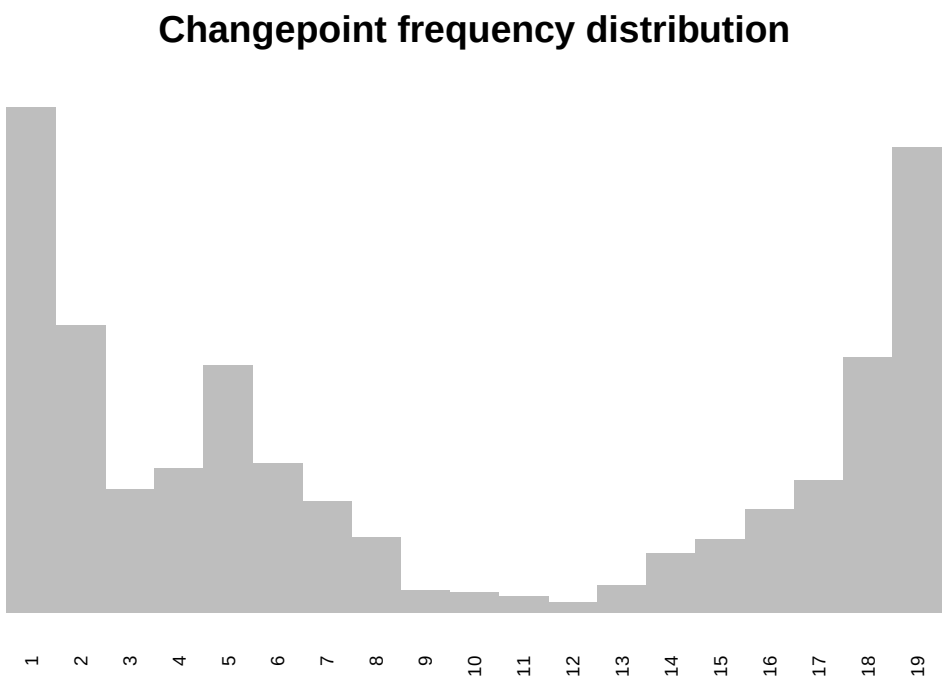



```
## Recomputing the partition in other forms and the number of groups
rho <- chains$rho
r = do.call(rbind, lapply(chains$rho, rho_to_r))
```

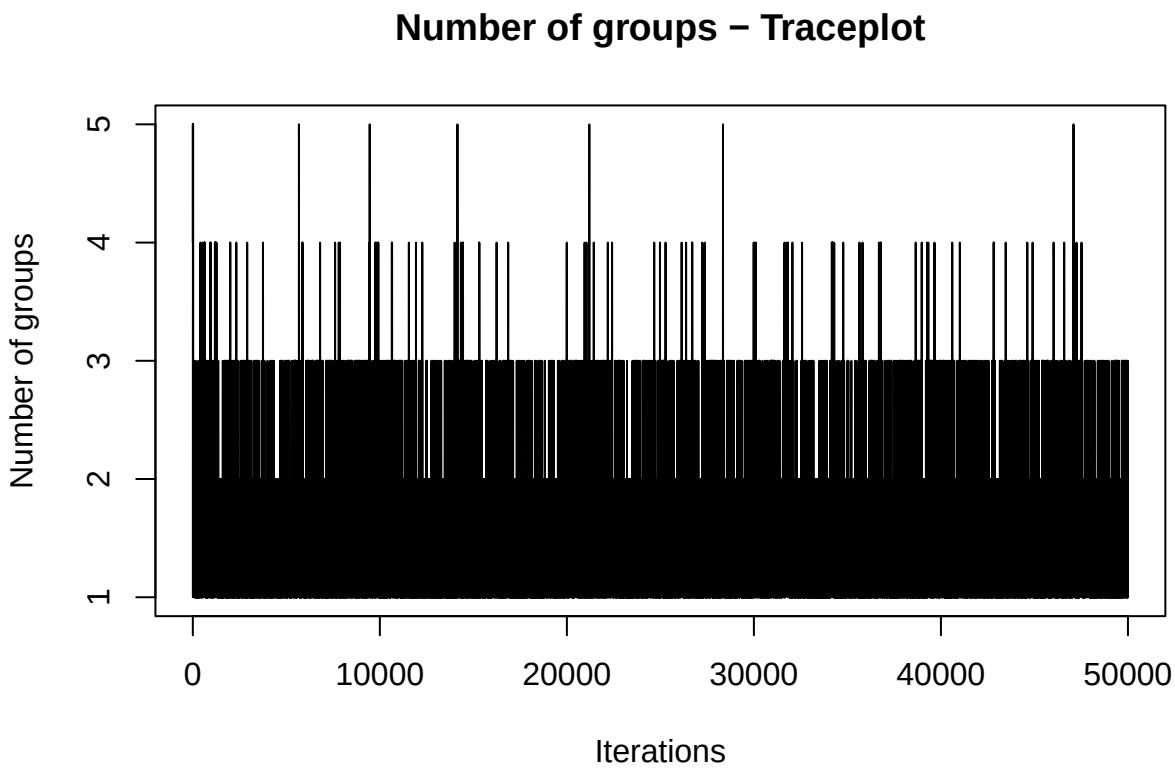
```
## Warning in (function (..., deparse.level = 1) : number of columns of result is
## not a multiple of vector length (arg 2)
```

```
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)

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,
```

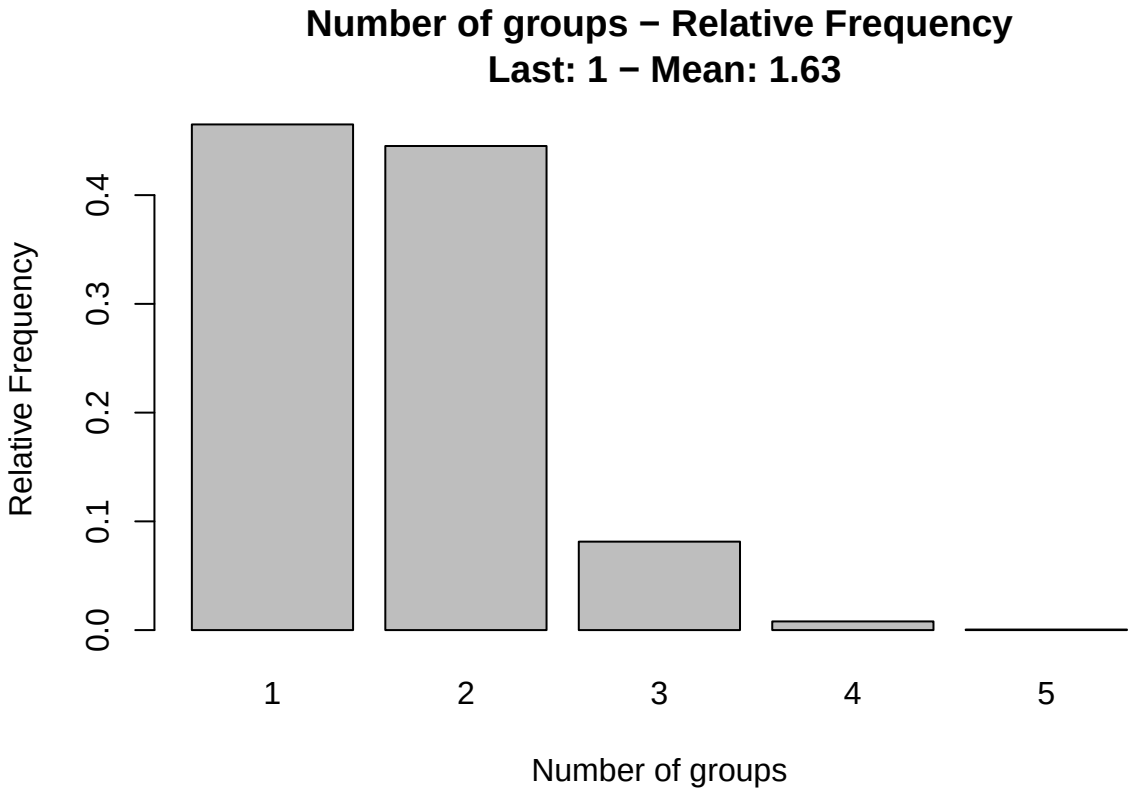
```
  las=2
)
```




```
### 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)
```




```r
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)

  )
)
```




```
### 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] 0.2172285
```

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

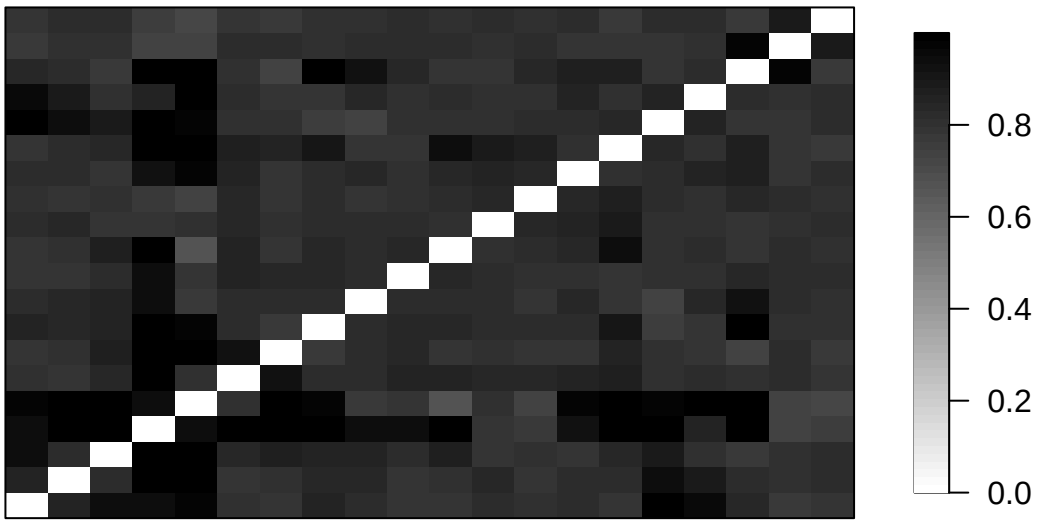
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
## Graph

# Extract last plinks
last_plinks = tail(chains$G, n=1)[[1]]
```

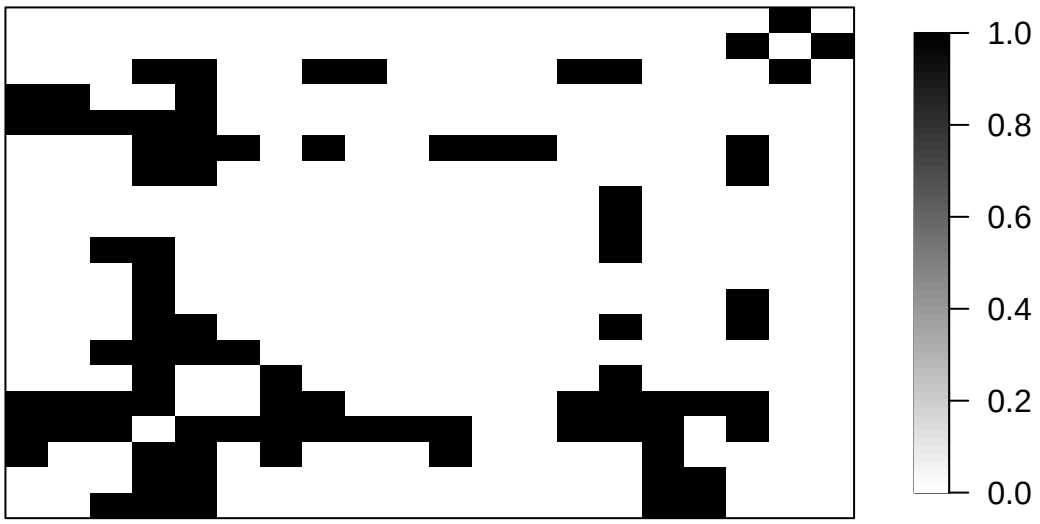
```r
# Criterion 1 to select the threshold (should not work very well) and assign final graph
threshold = 0.5
G_est <- matrix(0,p,p)
G_est[which(last_plinks>threshold)] = 1

#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] 0.863
```

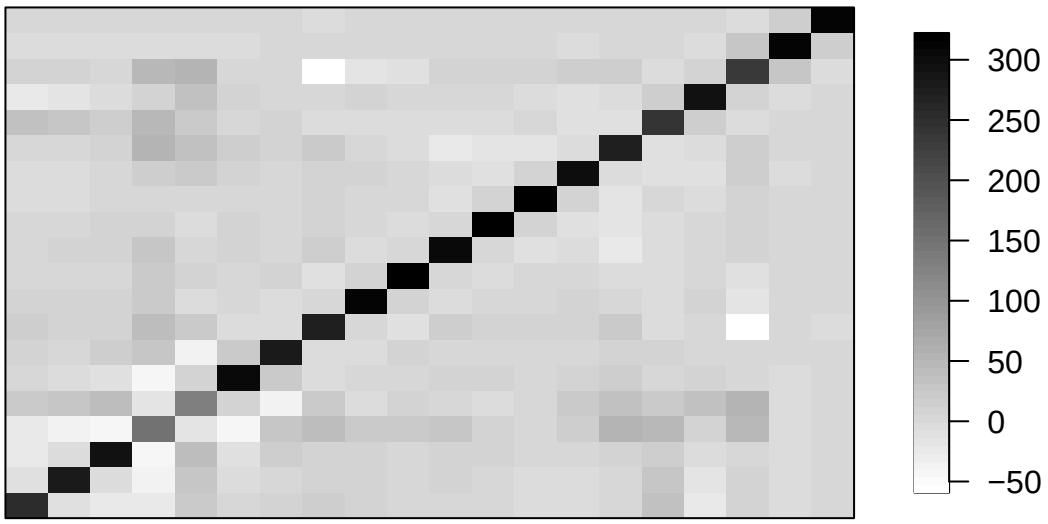
```r
G_est = bfdr_select$best_truncated_graph


### 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(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"
)
```


Estimated Precision matrix
300
250
200
150
100
50
0
−50