Purees data

2024-01-13

Load libraries

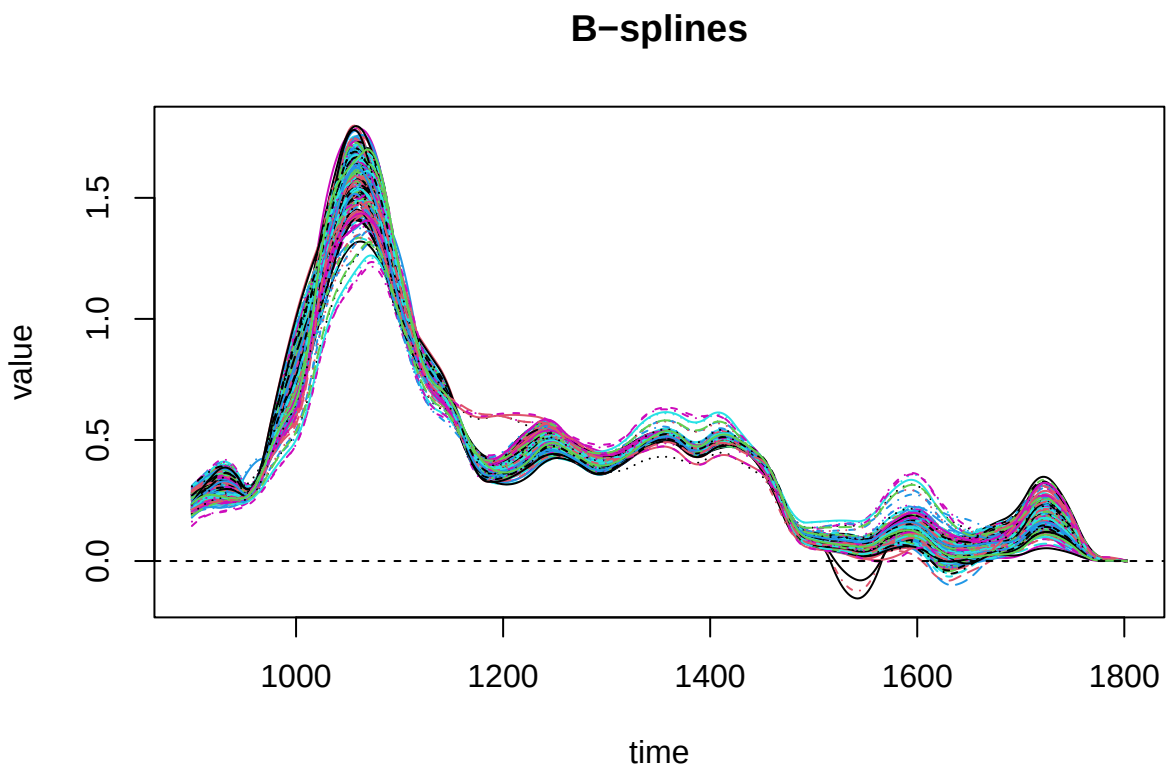
```
shhh = function(lib_name){ # It's a library, so shhh!
  suppressWarnings(suppressMessages(require(lib_name, character.only = TRUE)))
}
shhh("tidyverse")
shhh("ACutils")
shhh("mvtnorm")
shhh("salso")
shhh("FGM")
shhh("gmp")
shhh("mcclust")
shhh("mcclust.ext")
shhh("logr")
shhh("tidygraph")
shhh("ggraph")
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 =  40
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"
```

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

Initialization

```r
seed = 22111996
set.seed(seed)

# Define the starting Beta matrix
Beta = matrix(rnorm(n=p*n), nrow = p, ncol = n)

# Define the starting value of mu
mu = rnorm(n=p)

# Fix tau_eps (squared)
tau_eps = 100

# Define the starting precision matrix K and graph G
K = matrix(0,p,p)
G = matrix(0,p,p)

# Define the starting partition
rho = p
z = rep(1,p)

# 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 True binary flag used to update values
Update_Beta <- TRUE
Update_Mu <- TRUE
Update_Tau <- TRUE

# Define hyperparameters values
a_tau_eps <- 2000
b_tau_eps <- 2
sigma_mu <- 100

# Define variance of the Beta
beta_sig2 = 0.2

# Define graph density
graph_density = 0.3

# Set the number of iterations and burn-in
niter <- 10000
burn_in <- 1000


# Create a list for chains to save the values of each iteration
```

```
chains <- list(
  Beta = vector("list", length = niter),
  mu = vector("list", length = niter),
  tau_eps = vector("list", length = niter),
  K = vector("list", length = niter),
  G = vector("list", length = niter),
  z = vector("list", length = niter),
  rho = vector("list", length = niter),
  time = vector("list", length = niter),
  sigma_prior = vector("list", length = niter),
  theta_prior = vector("list", length = niter)
)


# Initialization of the chains
chains$Beta[[1]] <- Beta
chains$mu[[1]] <- mu
chains$tau_eps[[1]] <- tau_eps
chains$K[[1]] <- K
chains$G[[1]] <- G
chains$z[[1]] <- z
chains$rho[[1]] <- rho
chains$time <- 0          # execution time
chains$sigma <- 0.5       # parameter of the prior of the partition
chains$theta <- 1         # parameter of the prior of the partition

# initialization of parameters for set_options
weights_a <- rep(1,p-1)  # starting weights
weights_d <- rep(1,p-1)  # starting weights
total_weights <- 0       # sum of weights
total_K <- K
total_graphs <- G
graph_start <- NULL
```

Gibbs sampler

```
for(s in 2:niter) {

  fit = UpdateParamsGSL(
    chains$Beta[[s-1]],
    chains$mu[[s-1]],
    chains$tau_eps[[s-1]],
    chains$K[[s-1]],
    tbase_base,
    tbase_data,
    Sdata,
    a_tau_eps,
    b_tau_eps,
    sigma_mu,
    r,
    Update_Beta,
```

```
  Update_Mu,
  Update_Tau
  )

# Save Beta
chains$Beta[[s]] <- fit$Beta

# Save mu
chains$mu[[s]] <- fit$mu

# Save tau
chains$tau_eps[[s]] <- fit$tau_eps

# Set options for a single iteration of the Gibbs_sampler
options = set_options(
  sigma_prior_0=chains$sigma[[s-1]],
  sigma_prior_parameters=list("a"=1,"b"=1,"c"=1,"d"=1),
  theta_prior_0=chains$theta[[s-1]],
  theta_prior_parameters=list("c"=1,"d"=1),
  rho0=chains$rho[[s-1]],
  weights_a0=weights_a,
  weights_d0=weights_d,
  total_weights0=total_weights,
  total_K0 = total_K,
  total_graphs0 = total_graphs,
  graph = graph_start,
  alpha_target=0.234,
  beta_mu=graph_density,    # expected value beta distr of the graph
  beta_sig2=beta_sig2,      # var beta distr del grafo, fra 0 e 0.25
  d=3,                      # param della G wishart (default 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
  )

# Run an iteration of the Gibbs Sampler
res <- Gibbs_sampler(
  data = t(fit$Beta - fit$mu),
  niter = 1, # niter finali, già tolto il burn in
  nburn = 0,
  thin = 1,
  options = options,
  seed = 123456,
  print = FALSE
)

z = do.call(rbind, lapply(res$rho, rho_to_z))
```

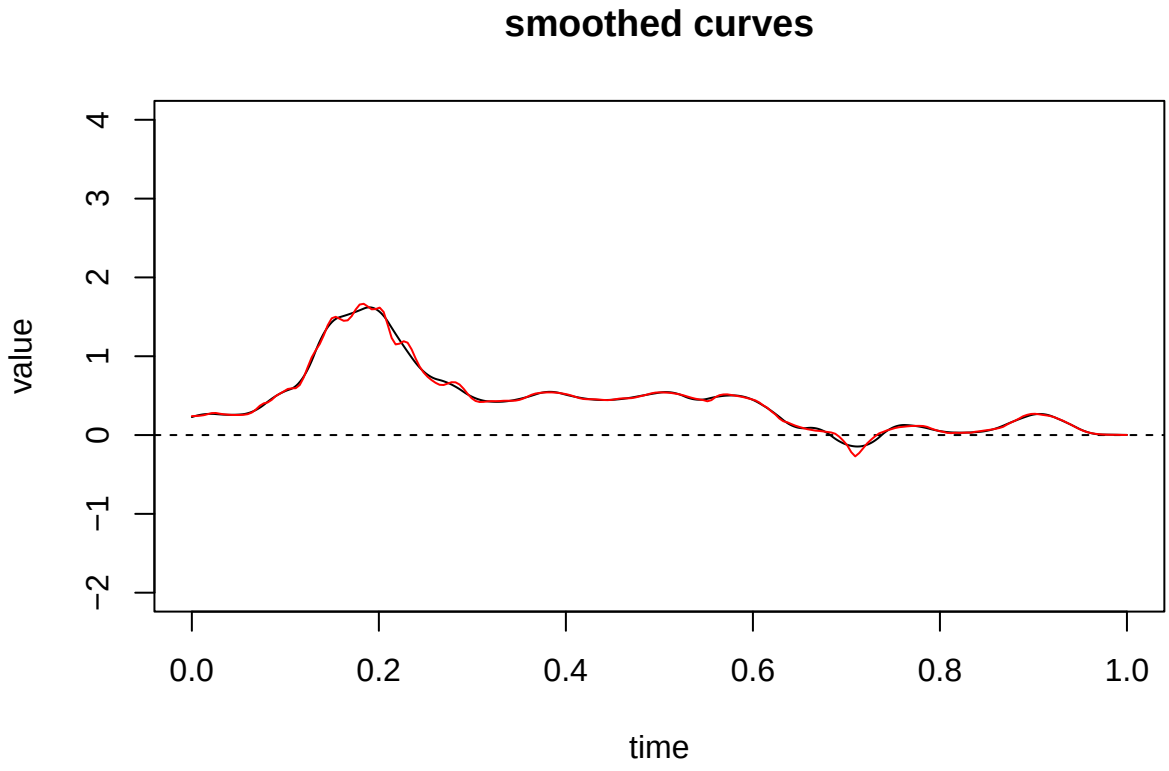
```
  # Save rho
  chains$rho[[s]] <- res$rho[[1]]

  # Save K
  chains$K[[s]] <- res$K [[1]]

  # Save G
  chains$G[[s]] <- res$G [[1]]

  # Save z
  chains$z[[s]] <- z

  # Save times for each K
  chains$time[[s]] <- res$execution_time

  # Save sigma amd theta
  chains$sigma[[s]] <- res$sigma[[1]]
  chains$theta[[s]] <- res$theta[[1]]

  # Update quantities for the next iteration
  weights_a <- res$weights_a[[1]]
  weights_d <- res$weights_d[[1]]
  total_weights <- res$total_weights
  total_K <- res$total_K[[1]]
  total_graphs <- res$total_graphs[[1]]
  graph_start <- res$bdgraph_start

}
```

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(-2,4))
```

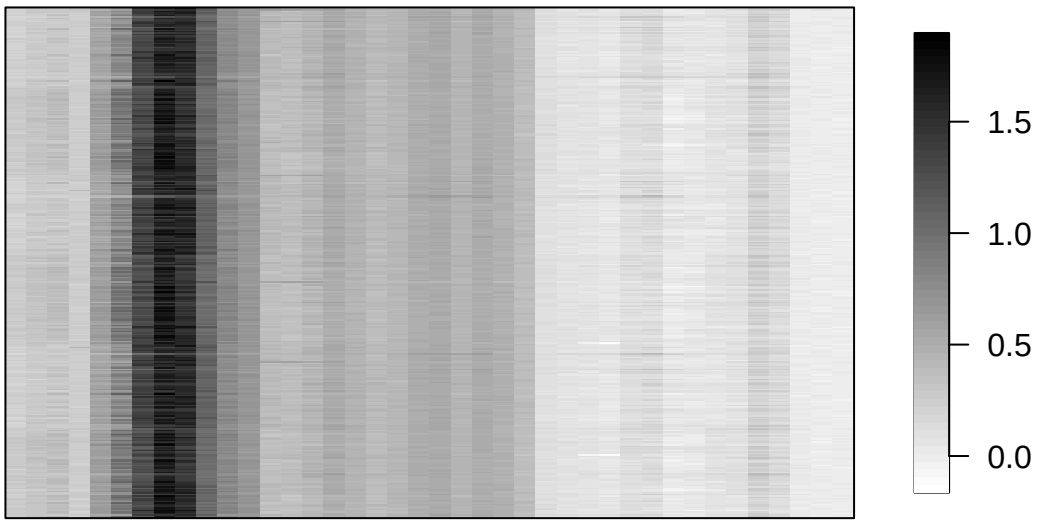
```
## [1] "done"
```

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



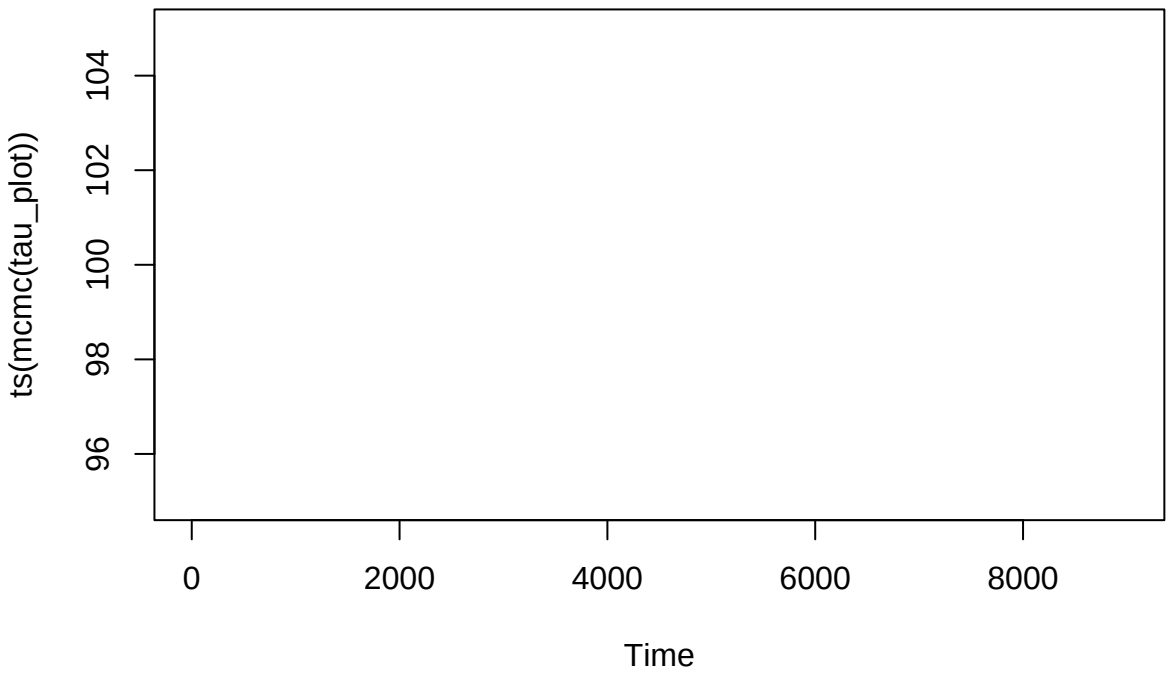
Useful plots: 2. Plot of the final Beta matrix

```r
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"
)
```

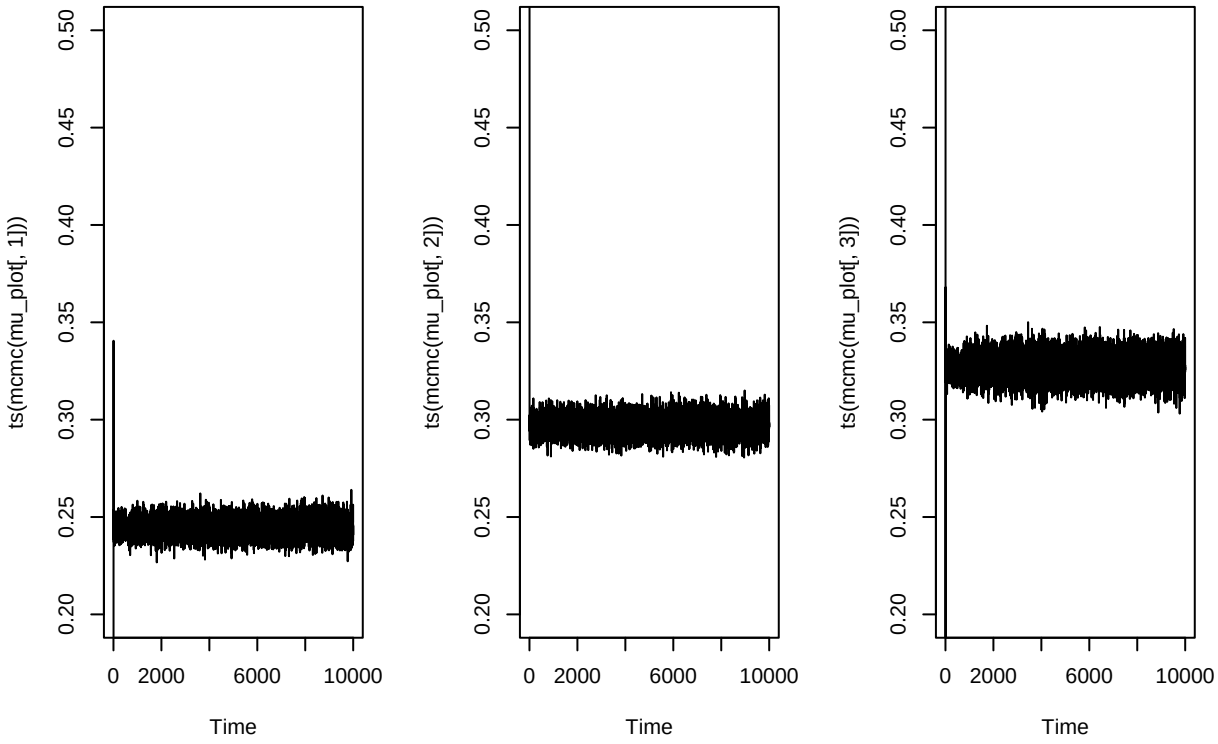


Useful plots: 2. Traceplots (tau_eps, mu, beta)

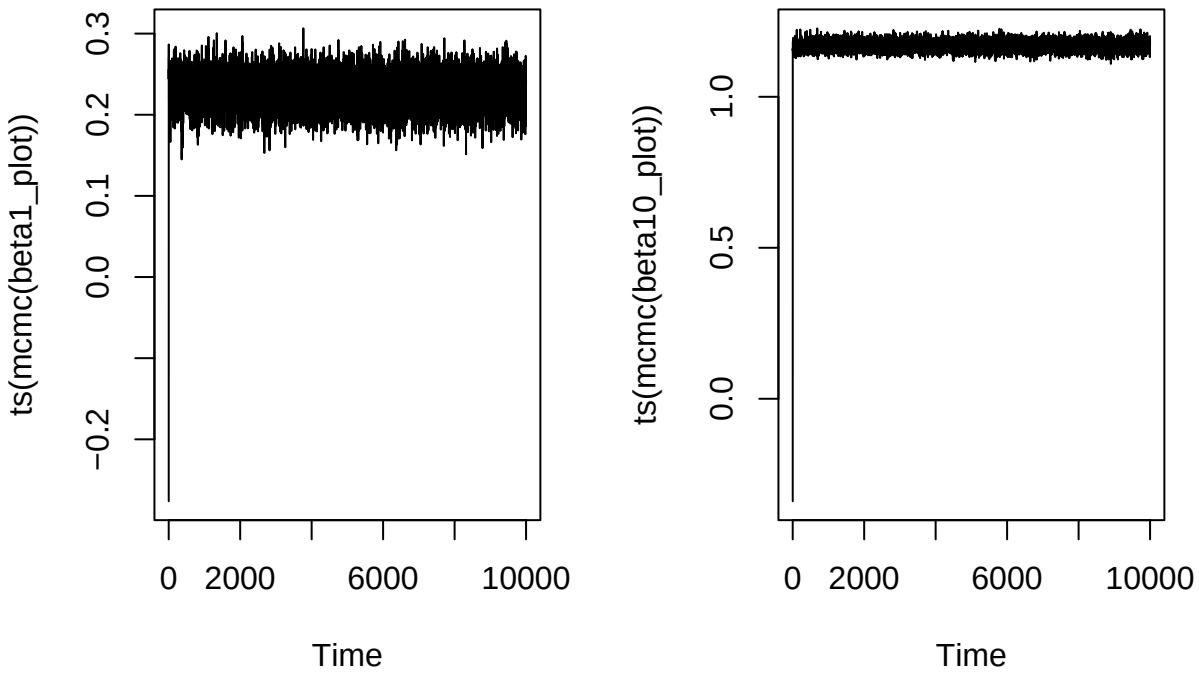
```
# tau_eps
tau_plot <- as.vector(chains$tau_eps)
tau_plot <- tau_plot[(burn_in+1):niter]
plot(ts(mcmc(tau_plot)), ylim=c(95,105))
```




```
# 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)))
```

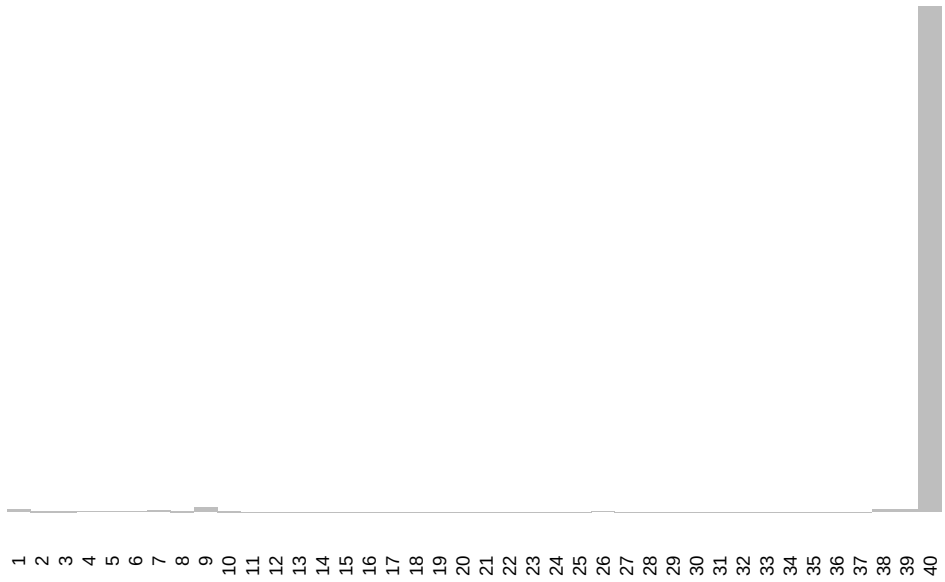



```
# Posterior analysis

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

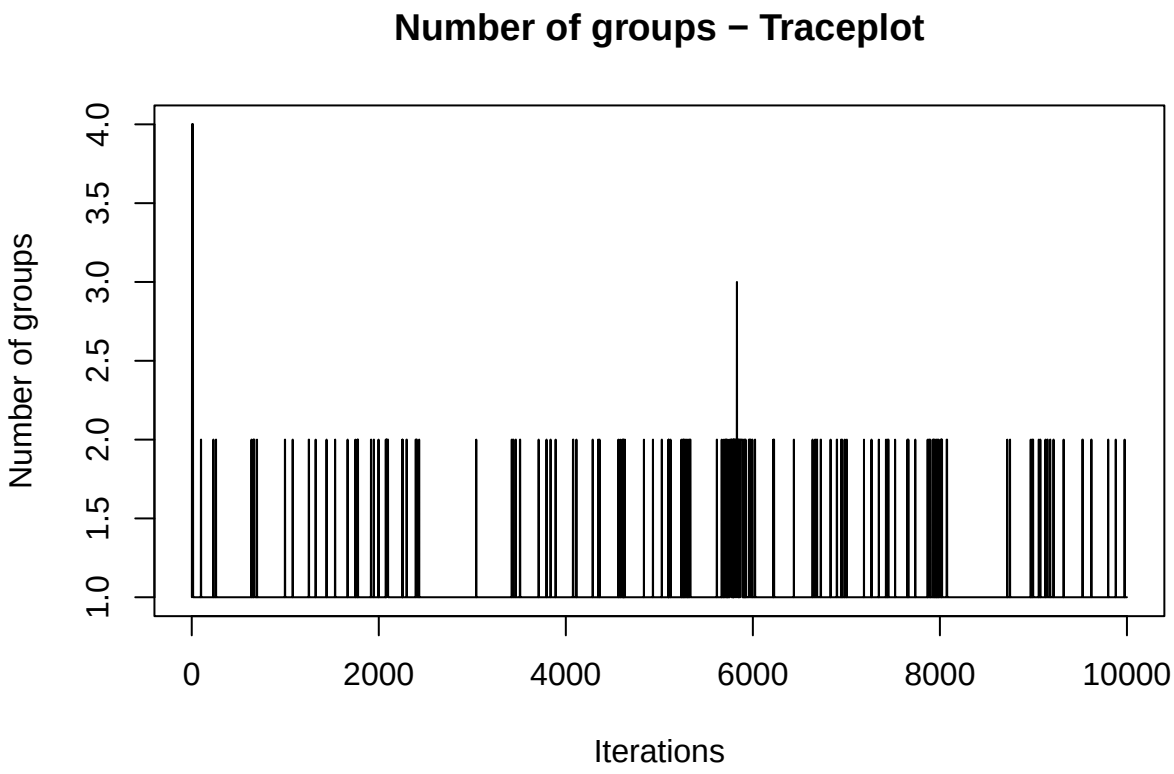
```
## Warning in (function (..., deparse.level = 1) : number of columns of result is
## not a multiple of vector length (arg 2)
```

```
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,
  names = seq_along(bar_heights),
  border = "NA",
  space = 0,
  yaxt = "n",
  main="Changepoint frequency distribution",
```

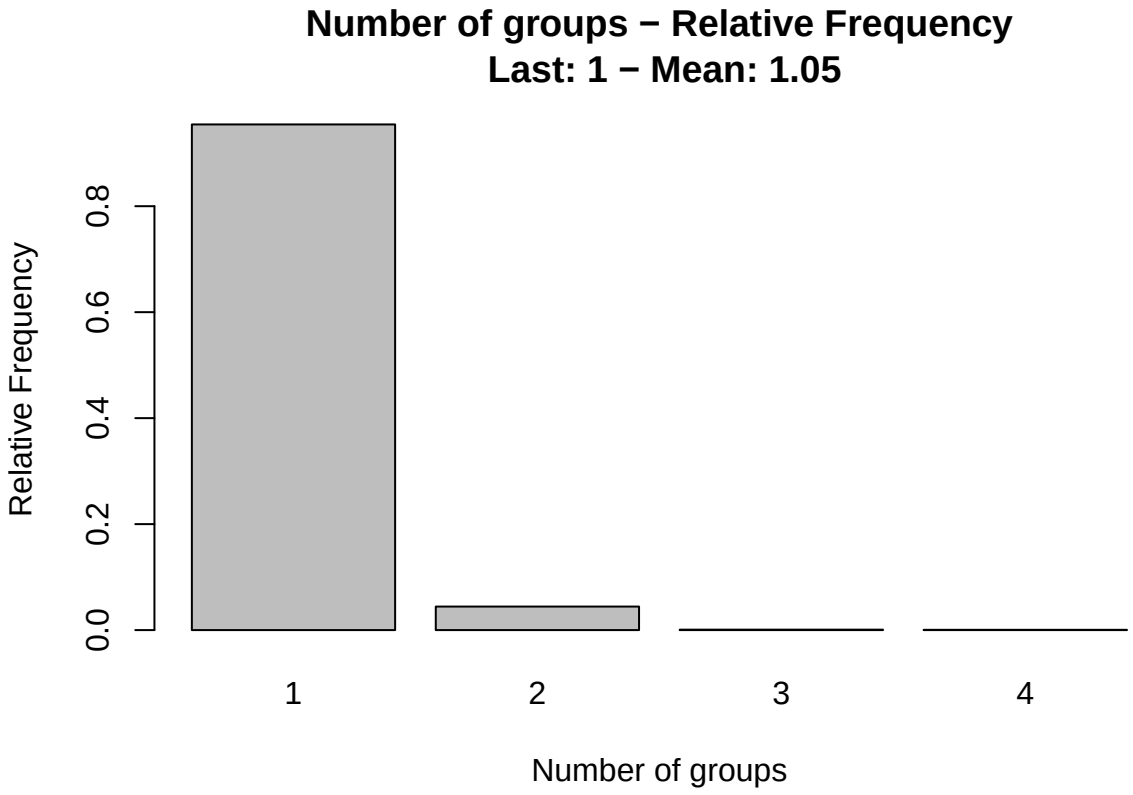
```
  #col = color,
  cex.names=.6,
  las=2
)
```




```
### 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)
```




```
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] 0.01452645
```

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

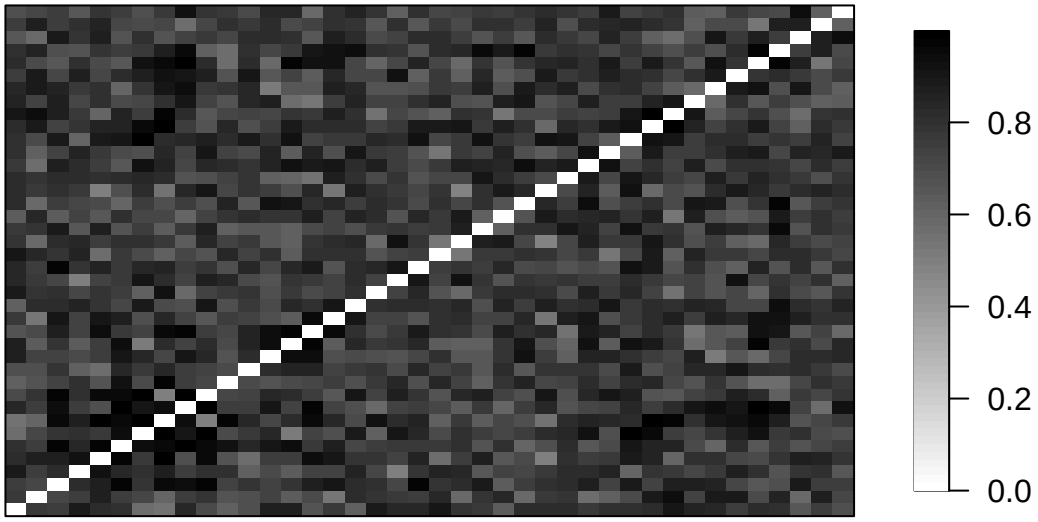
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [39] 1 1
```

```
## Graph

# Extract last plinks
```

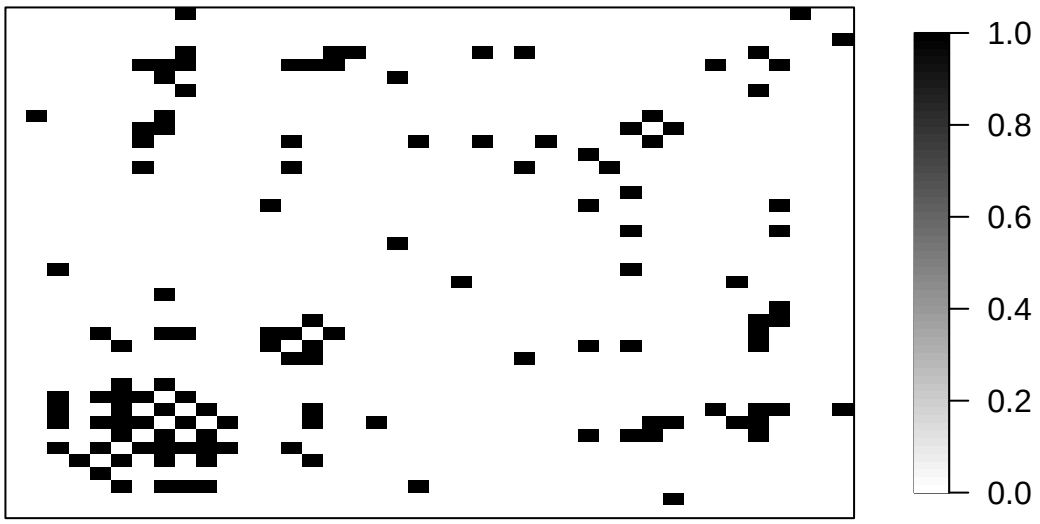
```r
last_plinks = tail(chains$G, n=1)[[1]]

# 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] 0.914
```

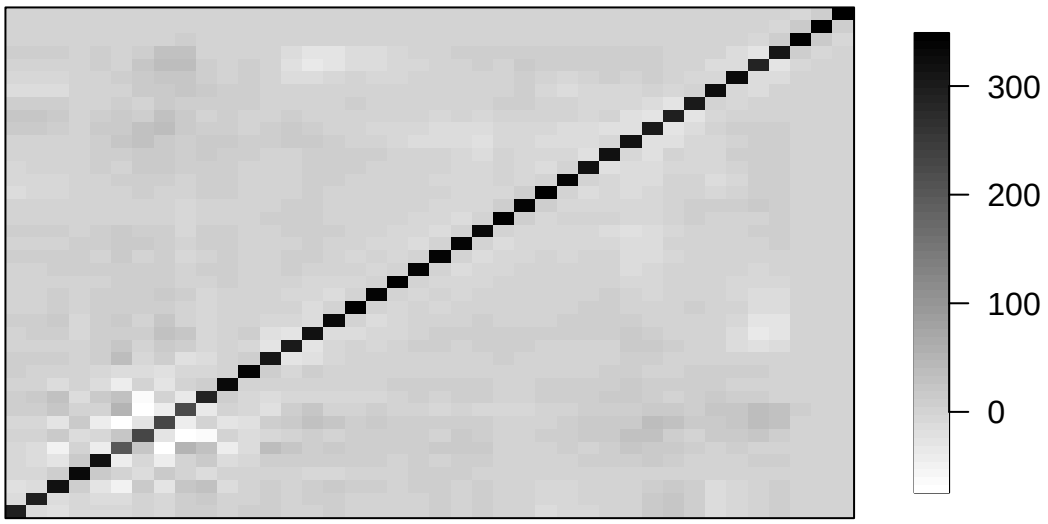
```r
G_est = bfdr_select$best_truncated_graph


### 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
200
100
0