simulated_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("utility_functions.R");
source("bulky_functions.R");
source("data_generation.R")
sourceCpp("wade.cpp")
Rcpp::sourceCpp('UpdateParamsGSL.cpp')

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

Simulated data

```
# Import simulated data
BaseMat = as.matrix(read_csv('BaseMat.csv'))
```

```
## Rows: 200 Columns: 40
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## dbl (40): bspl3.1, bspl3.2, bspl3.3, bspl3.4, bspl3.5, bspl3.6, bspl3.7, bsp...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
y_hat_true = as.matrix(read_csv('y_hat_true.csv'))
```

```
## Rows: 300 Columns: 200
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (200): V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
n <- dim(y_hat_true)[1]
r <- dim(y_hat_true)[2]
p <- dim(BaseMat)[2]
```

Initialization

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

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

# Fix tau_eps (squared)
tau_eps = 100

# Define the precision matrix K
K = rWishart(n = 1, df = p+10, Sigma = diag(p))
K = K[,,1]
# ACutils::ACheatmap(K,center_value = NULL, remove_diag = T)

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)  mettiamo insieme tutti i beta, verranno
Sdata = sum(diag(y_hat_true%*%t(y_hat_true))) # inefficient calculation ((2b + Sdata) è il b di tau_eps

# Set True binary flag used to update values
Update_Beta <- TRUE
Update_Mu <- TRUE
Update_Tau <- TRUE
a_tau_eps <- 2000
b_tau_eps <- 2
sigma_mu = 100

# Define variance of the Beta
beta_sig2 = 0.2
```

```r
# Compute graph density
graph_density = 0.3

# Set the number of iterations
niter <- 10000

# Create a list for chains
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)
)

simKG <- readRDS("simKG.rds")

# Initialization of the chains
chains$Beta[[1]] <- Beta
chains$mu[[1]] <- mu
chains$tau_eps[[1]] <- tau_eps
chains$K[[1]] <- simKG$Prec
chains$G[[1]] <- simKG$Graph
chains$z[[1]] <- c(rep(1,13), rep(2,13), rep(3,14))
chains$time <- 0
chains$rho[[1]] <- c(13,13,14)
sigma <-0.5
theta<-1
weights_a <- rep(1,p-1)
weights_d <- rep(1,p-1)
total_weights <- 0
total_K = simKG$Prec
total_graphs = simKG$Graph
graph_start = NULL
```

Gibbs sampler

```r
for(s in 2:10000) {

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

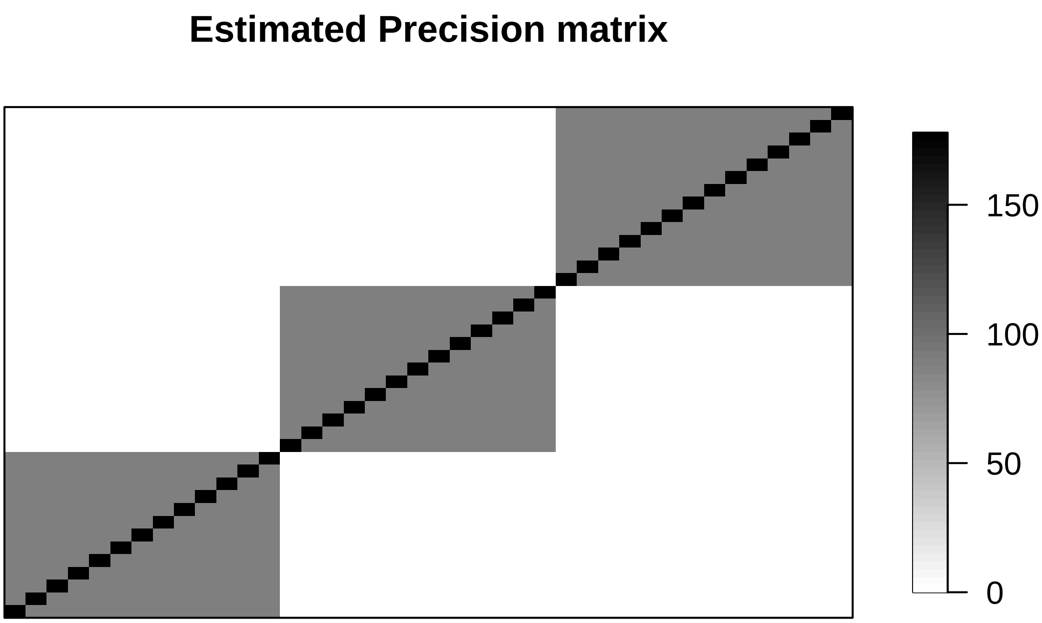
```
  a_tau_eps,
  b_tau_eps,
  sigma_mu,
  r,
  Update_Beta,
  Update_Mu,
  Update_Tau
  )

# 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=sigma,
  sigma_prior_parameters=list("a"=1,"b"=1,"c"=1,"d"=1),
  theta_prior_0=theta,
  theta_prior_parameters=list("c"=1,"d"=1),
  rho0=chains$rho[[s-1]],    # start with one group
  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,   # da modificare (expected value beta distr of the graph)
  beta_sig2=beta_sig2,     # da modificare (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=FALSE,
  update_theta_prior=FALSE,
  update_weights=FALSE,
  update_partition=FALSE,
  update_graph=FALSE,
  perform_shuffle=FALSE
  )

# 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 = 22111996,
```

```
    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]]   # primo valore NULL o identità

  # Save z
  chains$z[[s]] <- z      # primo valore NULL o mettiamo tutti 1

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

  # Update quantities for the next iteration
  # weights
  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$total_graphs[[1]]
  # graph
  graph_start = res$bdgraph_start
}
```

Final plots: Plot of the obtained K_fin

```
# Set the number burn in iterations
burn_in <- 1000  # TODO: decide which value to fix

# Update K as the sum of times*K/tot_time
sum <- 0
for(i in (burn_in+1):niter){
  sum <- sum + chains$time[[i]]*chains$K[[i]]
}
K_fin <- sum/sum(chains$time)

# Plot obtained K_fin
ACutils::ACheatmap(
  K_fin,
  use_x11_device = F,
  horizontal = F,
  main = "Estimated Precision matrix",
  center_value = NULL,
  col.upper = "black",
```

```
  col.center = "grey50",
  col.lower = "white"
)
```

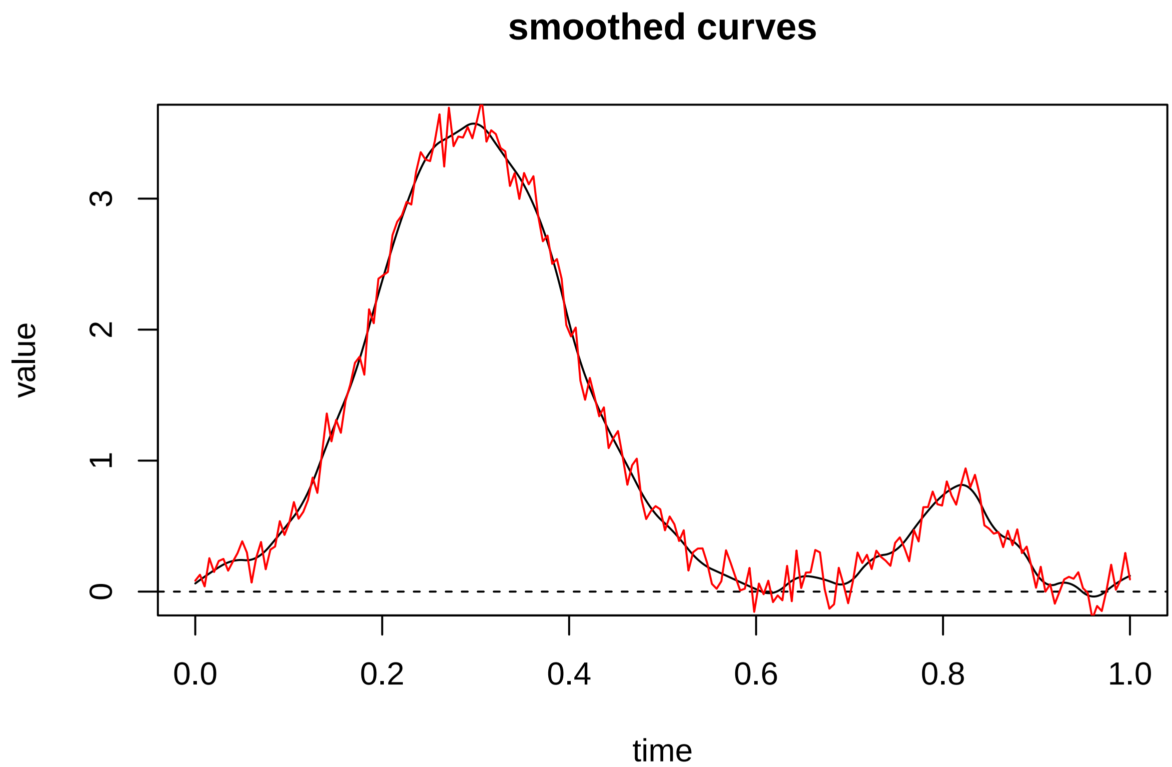


Useful plots: 1. Plot smoothed curves

```
# 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=40, norder=3)
data.fd <- Data2fd(y = data_post, argvals = x, basisobj = basis)

# Plot smoothed curves
plot.fd(data.fd[1,], main="smoothed curves")
```

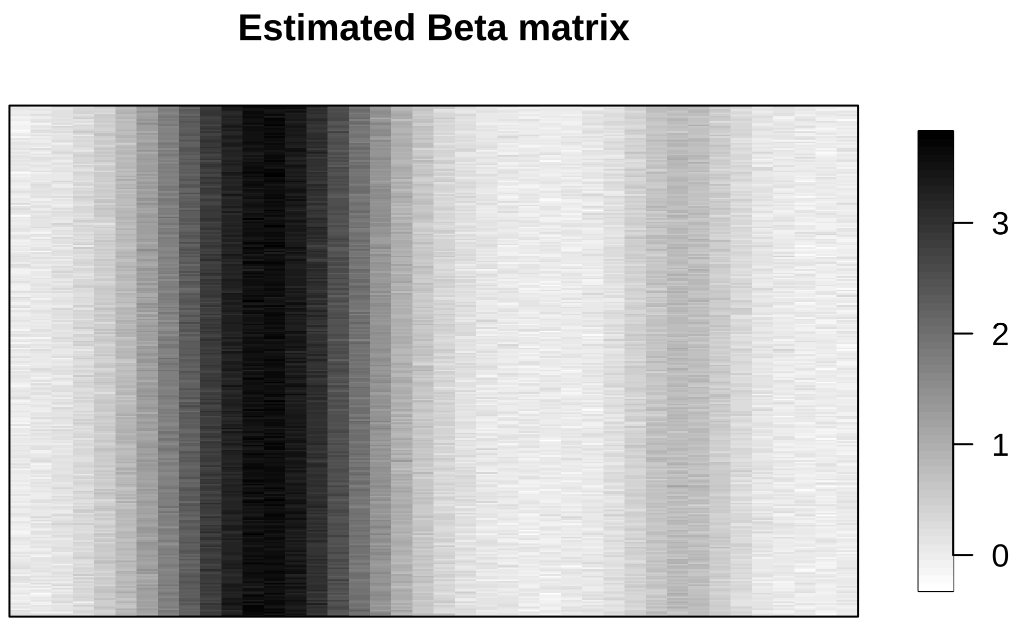
```
## [1] "done"
```

```
lines(x,y_hat_true[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)

```
library(coda)
```

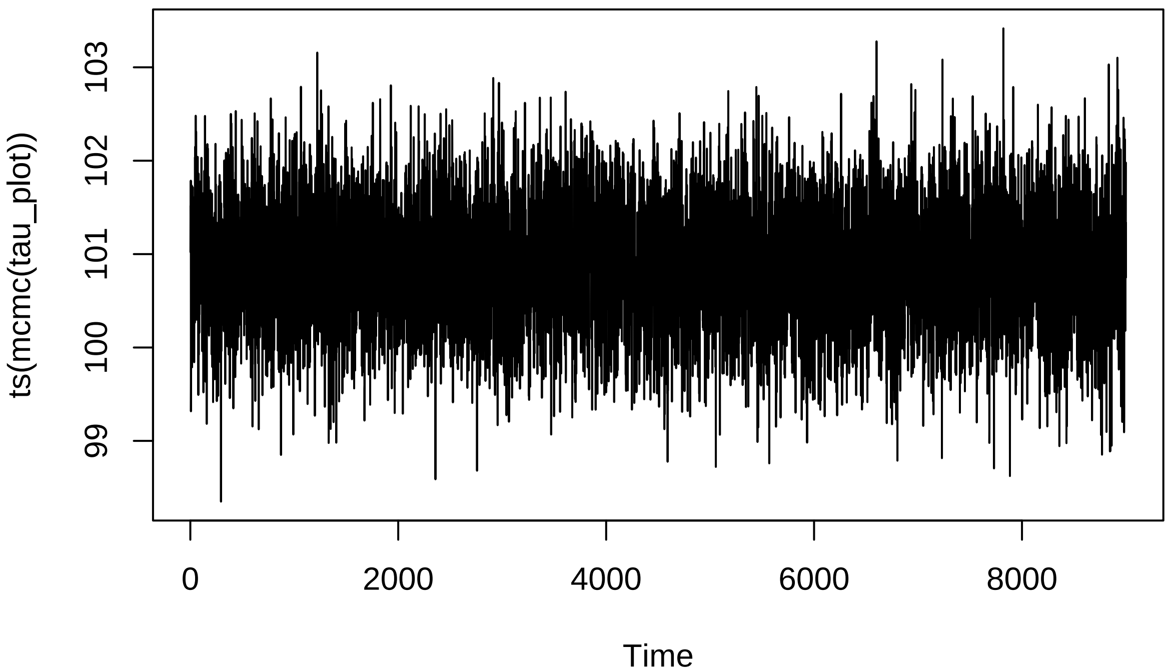
```
## Warning: il pacchetto 'coda' è stato creato con R versione 4.3.2
```

```
library(lattice)
```

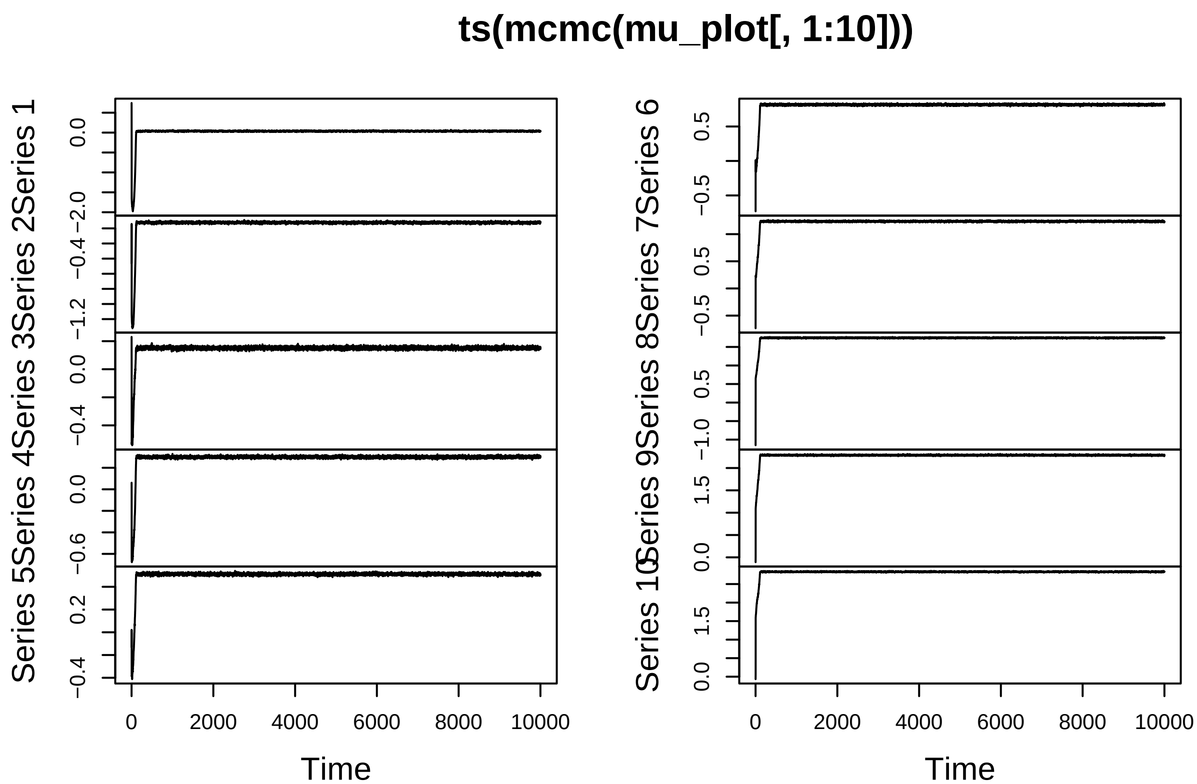
```
##
## Caricamento pacchetto: 'lattice'
```

```
## Il seguente oggetto è mascherato da 'package:fda':
##
##     melanoma
```

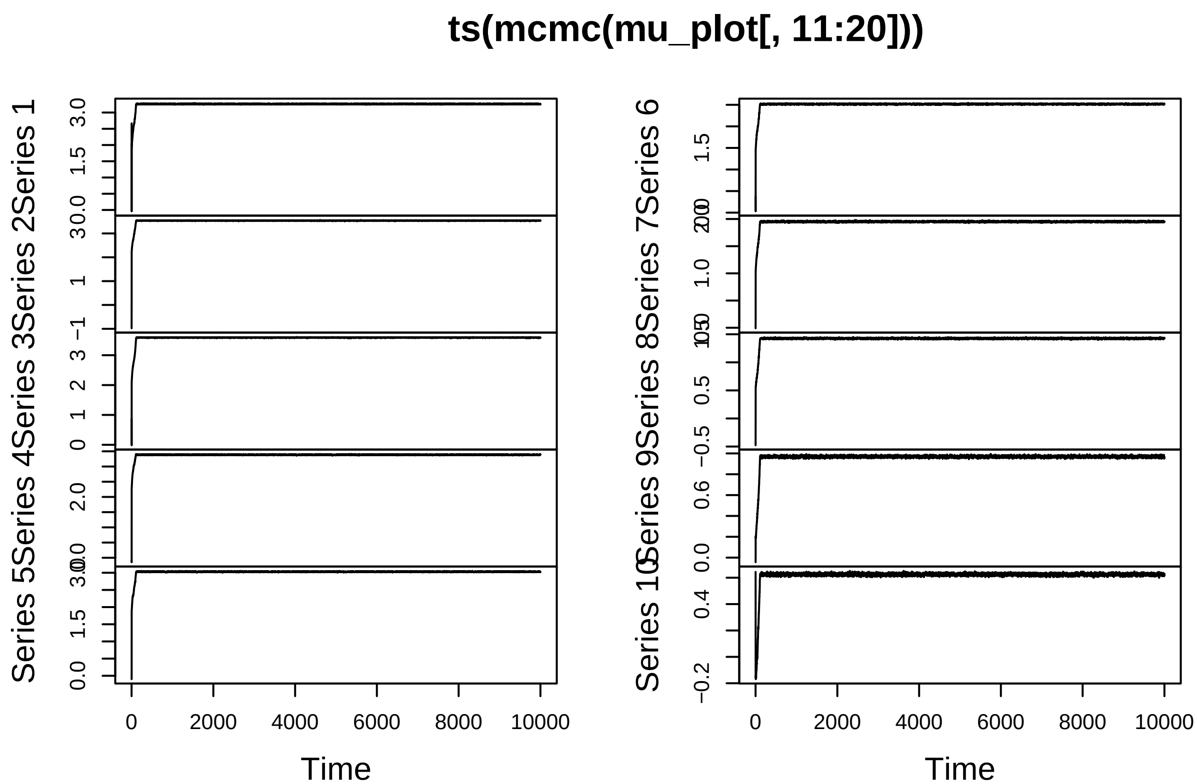
```
tau_plot <- as.vector(chains$tau_eps)
tau_plot <- tau_plot[(burn_in+1):10000]
plot(ts(mcmc(tau_plot)))
```



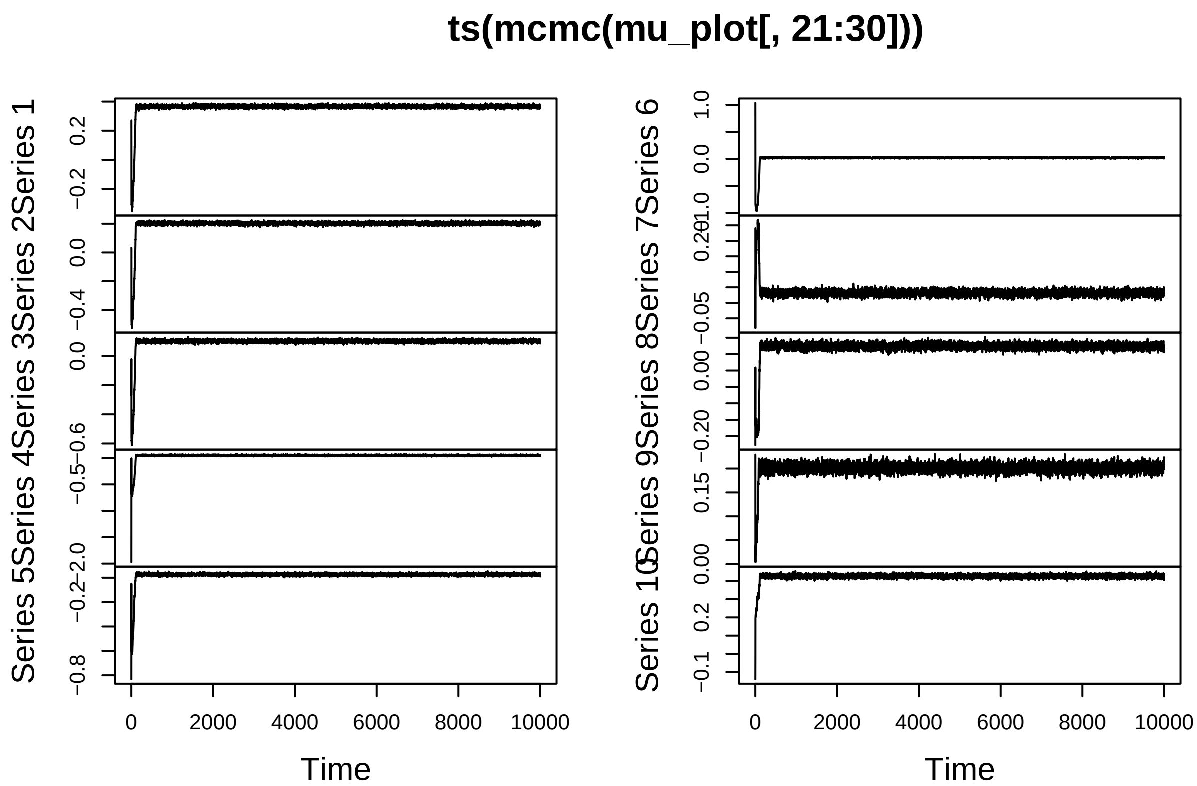

```r
mu_plot <- matrix(0, niter, p)
for(i in 1:niter){
  mu_plot[i, ] <- chains$mu[[i]]
}

plot(ts(mcmc(mu_plot[, 1:10])))
```



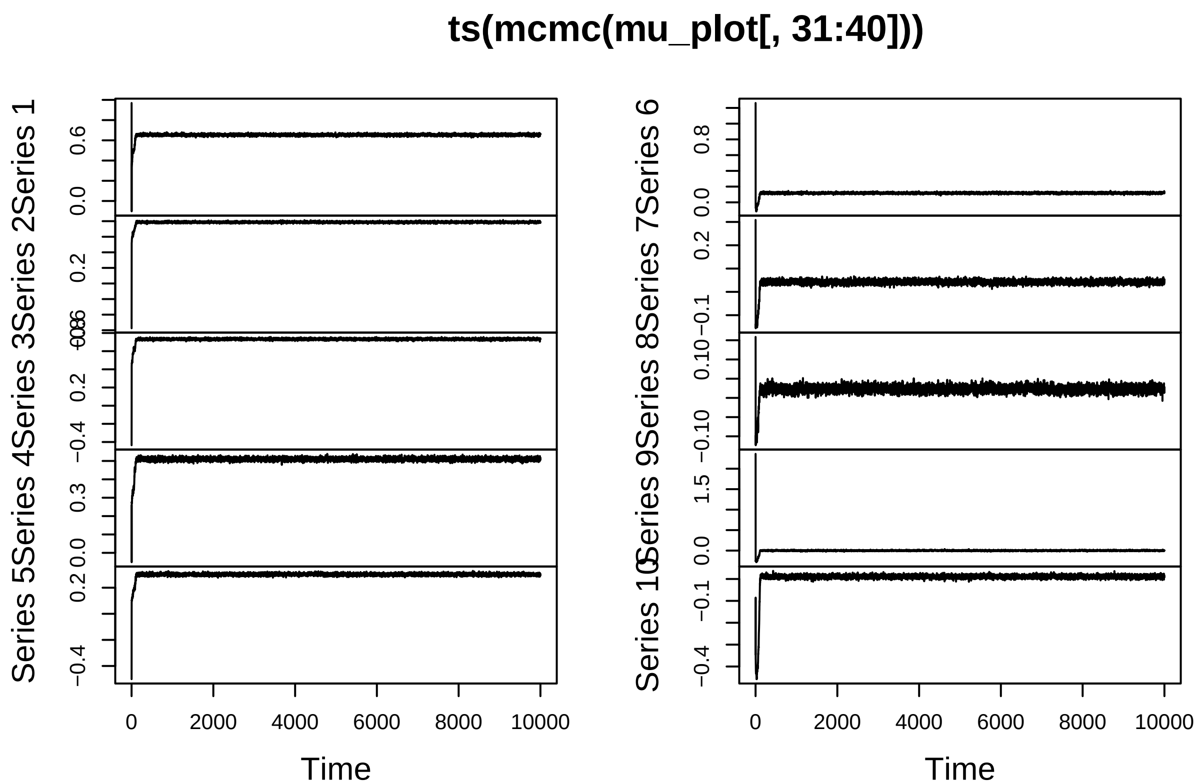

```r
plot(ts(mcmc(mu_plot[, 11:20])))
```




```r
plot(ts(mcmc(mu_plot[, 21:30])))
```

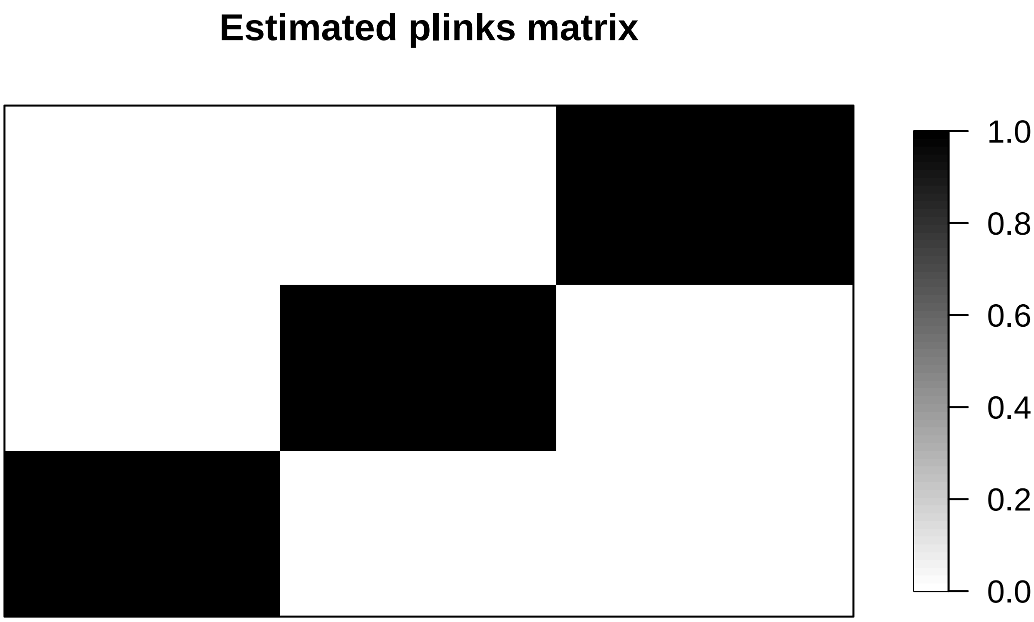



```r
plot(ts(mcmc(mu_plot[, 31:40])))
```

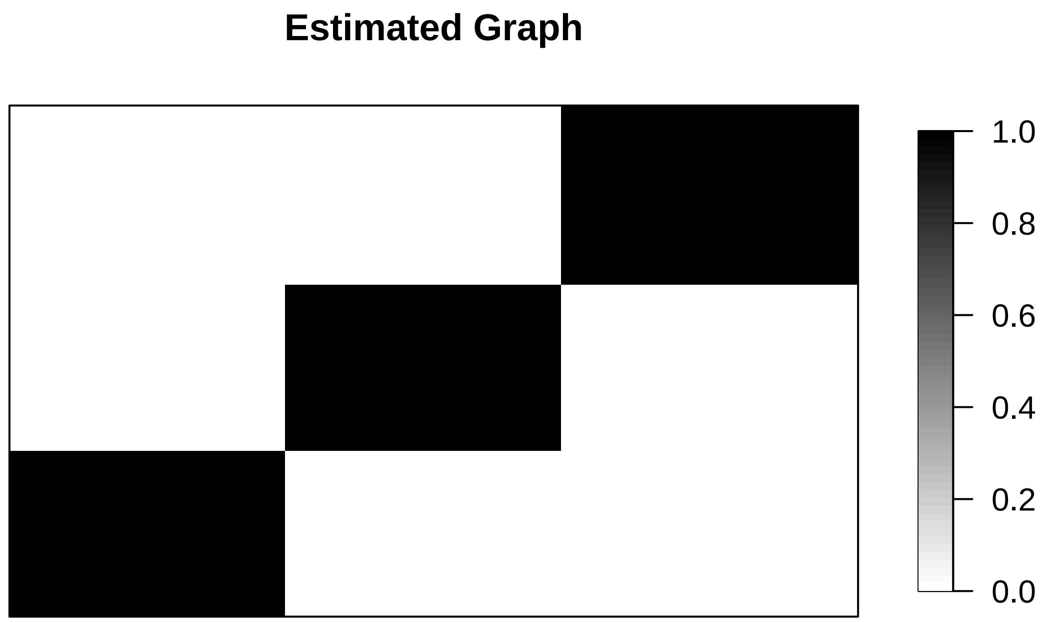


Useful plots: 3. Plot of the plinks matrix

```r
last_plinks = chains$G[[niter]]

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"
)
```




```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


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"
)
```




```r
# 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] 1
```

```r
G_est = bfdr_select$best_truncated_graph

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"
)
```