Esercitazione 7 Recupero lezione

December 1, 2024

1 Serie Storiche multivariate

Riprendiamo il dataset della scorsa esercitazione

[ ]:
```
#install.packages("gapminder")
set.seed(123)
library(ggplot2)
library(tidyverse)
library(magrittr)
library(gridExtra)
setwd("/Users/gianlucamastrantonio/Dropbox (Politecnico di Torino Staff)/
↪Didattica/statistica computazionale/esercizi")
load("dataset_clima_long.RData")
summary(dataset_clima)
dataset_clima <- as.data.frame(dataset_clima)

nstaz <- 2496
ntempi <- dim(dataset_clima)[1] / nstaz
ntempi
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse
2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v lubridate 1.9.3     v tibble    3.2.1
v purrr     1.0.2     v tidyr     1.3.1
-- Conflicts ------------------------------------------
tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package
(<http://conflicted.r-lib.org/>) to force all conflicts to
become errors

Caricamento pacchetto: 'magrittr'


Il seguente oggetto `e mascherato da 'package:purrr':
```

```
    set_names


Il seguente oggetto `e mascherato da 'package:tidyr':

    extract



Caricamento pacchetto: 'gridExtra'


Il seguente oggetto `e mascherato da 'package:dplyr':

    combine
```

```
 sea_level_pressure mean_global_radiation precipitation_sum mean_temperature
 Min.   : 821.2     Min.   :   0.0        Min.   :  0.0     Min.   :-40.36
 1st Qu.:1009.9     1st Qu.:  51.0        1st Qu.:  0.0     1st Qu.:  3.70
 Median :1015.5     Median : 124.0        Median :  0.0     Median :  9.86
 Mean   :1014.9     Mean   : 135.1        Mean   :  2.1     Mean   :  9.44
 3rd Qu.:1020.9     3rd Qu.: 210.0        3rd Qu.:  2.2     3rd Qu.: 15.70
 Max.   :1069.0     Max.   :2777.0        Max.   :228.5     Max.   : 35.79
 NA's   :227204     NA's   :357129        NA's   :270469    NA's   :120845
 minimum_temperature maximum_temperature    humidity         longitude
 Min.   :-41.52      Min.   :-39.14      Min.   :12.1      Min.   :-9.875
 1st Qu.:  0.20      1st Qu.:  7.22      1st Qu.:68.2      1st Qu.: 2.125
 Median :  5.73      Median : 14.29      Median :78.0      Median :11.625
 Mean   :  5.22      Mean   : 14.02      Mean   :75.6      Mean   :10.344
 3rd Qu.: 11.04      3rd Qu.: 21.08      3rd Qu.:85.3      3rd Qu.:18.625
 Max.   : 30.49      Max.   : 46.14      Max.   :94.5      Max.   :24.625
 NA's   :65821       NA's   :55112       NA's   :1142233
    latitude          time         external
 Min.   :32.38   Min.   :   0   Min.   :  24
 1st Qu.:43.38   1st Qu.: 912   1st Qu.:3650
 Median :49.38   Median :1824   Median :3650
 Mean   :50.36   Mean   :1824   Mean   :3602
 3rd Qu.:56.88   3rd Qu.:2737   3rd Qu.:3650
 Max.   :70.88   Max.   :3649   Max.   :3650
```

3650

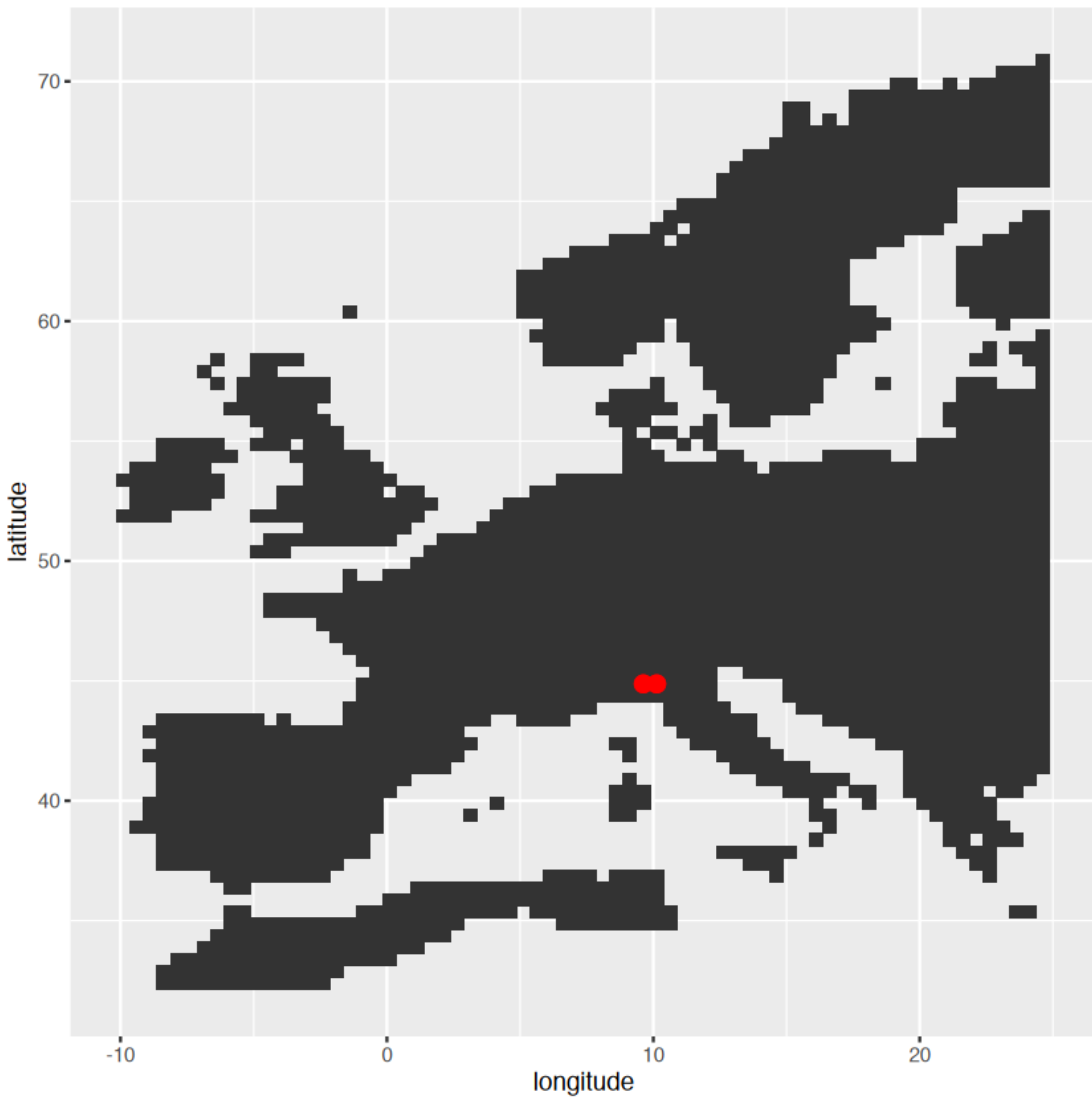
Quello che vogliamo fare è di prendere 4 serie qualsiasi, e di fare una modellizzazione congiunta. Definiamo

$$\mathbf{y}_t = (y_{t,1}, y_{t,2}, y_{t,3}, y_{t,4})^T$$

e definiamo il modello come

$$f(\mathbf{y}_1)f(\mathbf{y}_2|\mathbf{y}_1)f(\mathbf{y}_3|\mathbf{y}_2,\mathbf{y}_1)\dots f(\mathbf{y}_n|\mathbf{y}_{n-1},\dots\mathbf{y}_2,\mathbf{y}_1)$$

[ ]:
```
index_staz <- c(745, 746, 745, 746)
dat1 <- dataset_clima[seq(index_staz[1], nrow(dataset_clima), by = nstaz),␣
↪"minimum_temperature"]
dat2 <- dataset_clima[seq(index_staz[2], nrow(dataset_clima), by = nstaz),␣
↪"minimum_temperature"]
dat3 <- dataset_clima[seq(index_staz[3], nrow(dataset_clima), by = nstaz),␣
↪"maximum_temperature"]
dat4 <- dataset_clima[seq(index_staz[4], nrow(dataset_clima), by = nstaz),␣
↪"maximum_temperature"]
data_small <- data.frame(dat1, dat2, dat3, dat4)
data_coords <- data.frame(x = dataset_clima$longitude[index_staz], y =␣
↪dataset_clima$latitude[index_staz])

dataset_clima %>%
  slice(1:nstaz) %>%
  ggplot(aes(x = longitude, y = latitude)) +
  geom_tile()+ geom_point(data = data_coords, aes(x = x, y = y), col = "red",␣
↪size = 3)
```



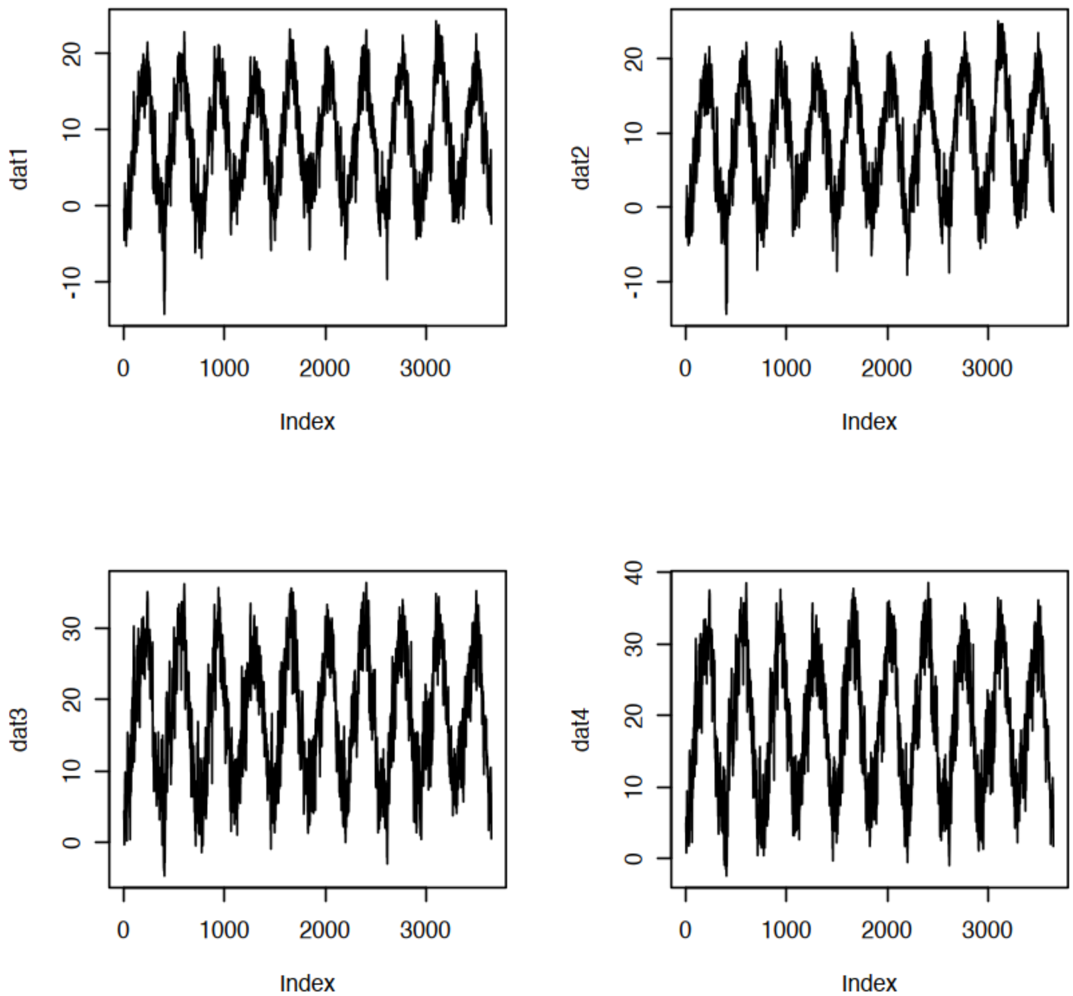
Vediamo le serie scelte

[ ]:

```

```

[48]:

```
par(mfrow=c(2,2))
plot(dat1, type="l")
plot(dat2, type = "l")
plot(dat3, type = "l")
plot(dat4, type = "l")
par(mfrow = c(1, 1))
```



Essendo queste delle serie temporali, vogliamo medellizzarle mettendo delle dispendenza temporale. Utilizziamo una modellizzazione autoregressiva multivariata, del tipo

$$\mathbf{x}_t = (1-B)\mathbf{y}_t = diag(\alpha) + \mathbf{w}_t$$

dove

$$\mathbf{w}_t \sim N(0, \Sigma)$$

$$\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)^T$$

e con $(1-B)\mathbf{y}_t$ intendo

$$(1-B)\mathbf{y}_t = ((1-B)y_{t,1}, (1-B)y_{t,2}, (1-B)y_{t,3}, (1-B)y_{t,4})^T$$

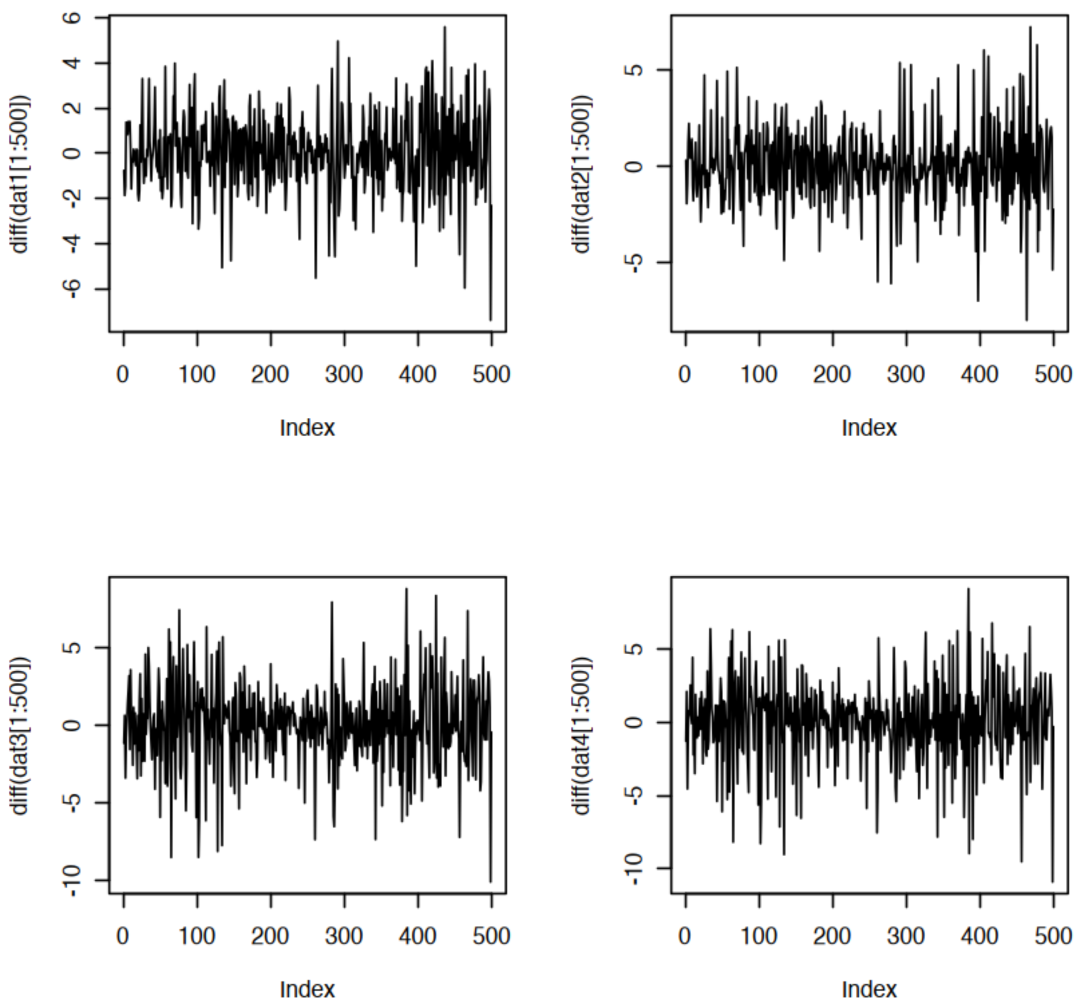
Visto che abbiamo che marginalmente ogni componente è AR(1)

$$x_{t,j} = \alpha_j x_{t-1,j} + w_{t,j}$$

Allora 1uesto si può vedere come un modello $ARMA(1, 1, 0)$ multivariato.

Le serie delle differenze sono mostrate sotto

[9]:
```
par(mfrow=c(2,2))
plot(diff(dat1[1:500]), type="l")
plot(diff(dat2[1:500]), type = "l")
plot(diff(dat3[1:500]), type = "l")
plot(diff(dat4[1:500]), type = "l")
par(mfrow = c(1, 1))
```



Domande

1. Scrivete un MCMC assumendo le componenti di $\mathbf{w}_t$ come indipendenti.
2. Simulate dei dati dal modello e verificate se il codice funziona.

3. Stimare il modello sui dati
4. Scrivere e stimare un modello con una Σ generica, con prior InverseWIshart, assumendo che esista un vettore $\mathbf{y}_0$, su cui dovete mettere una prior

Per prima cose scrivo il codice per la stima del modello, assumendo

$$\alpha_j \sim U(-1, 1)$$

e

$$\sigma_j^2 \sim IG(a, b)$$

Per il campionamento di α_j bisogna usare un passo Metropolis, e io utilizo anche una proposta adattiva. Per fare questo bisogna fare un cambio di variabili

$$\eta_j = \log\left(\frac{\alpha_j + 1}{1 - \alpha_j}\right) \in \mathbb{R}$$

che ha prior

$$f_{\eta_j}(\eta_j) = f_{\alpha_j}(\alpha_j)|\frac{d\alpha_j}{d\eta_j}|$$

d dove

$$\alpha_j = \frac{-1 + 1\exp(\eta_j)}{1 + \exp(\eta_j)}$$

e

$$|\frac{d\alpha_j}{d\eta_j}| \propto \frac{\exp(\eta_j)}{(1 + \exp(\eta_j))^2}$$

[13]:
```
model <- function(dataset, prior_sigma2_a = 1, prior_sigma2_b, iterations =
 ↪1000, adapt_batch = 50, adapt_a = 2000, adapt_b = 1000, adapt_alpha = 0.234,
 ↪adapt_stop = 1000) {
  # Dimensioni del dataset
  p <- ncol(dataset) # Numero di colonne (variabili)
  n <- nrow(dataset) # Numero di righe (osservazioni)

  # Matrici per memorizzare i risultati delle iterazioni
  alpha_out <- matrix(NA, nrow = iterations, ncol = p)
  sigma2_out <- matrix(NA, nrow = iterations, ncol = p)

  # Inizializzazione dei valori per alpha e sigma2
  alpha_mcmc <- rep(0, p) # Valori iniziali per alpha
  sigma2_mcmc <- rep(1, p) # Valori iniziali per sigma^2

  # Inizializzazione della deviazione standard per la proposta e del conteggio
 ↪delle accettazioni
  sd_prop_alpha <- rep(0.1, p) # Deviazione standard iniziale per alpha
  acc_alpha <- rep(0, p) # Contatore per il tasso di accettazione

  # Ciclo principale per le iterazioni MCMC
  for (iter in 1:iterations)
```

```
{
  # Aggiornamento dei parametri alpha
  for (ip in 1:p)
  {
    # Trasformazione logistica per alpha (da [-1, 1] a tutto R)
    alpha_trans_mcmc <- log((alpha_mcmc[ip] + 1) / (1 - alpha_mcmc[ip]))
    alpha_trans_prop <- rnorm(1, alpha_trans_mcmc, sd_prop_alpha[ip]) # Proposta per alpha
    alpha_prop <- (-1 + exp(alpha_trans_prop)) / (1 + exp(alpha_trans_prop)) # Ritorno nello spazio originale

    log_MH_ratio <- 0 # Log della probabilità del rapporto Metropolis-Hastings

    # Likelihood: contributo della verosimiglianza per alpha
    log_MH_ratio <- log_MH_ratio + dnorm(dataset[1, ip], 0, (sigma2_mcmc[ip] / (1 - alpha_prop^2))^0.5, log = TRUE)
    log_MH_ratio <- log_MH_ratio - dnorm(dataset[1, ip], 0, (sigma2_mcmc[ip] / (1 - alpha_mcmc[ip]^2))^0.5, log = TRUE)
    for (i in 2:n)
    {
      log_MH_ratio <- log_MH_ratio + dnorm(dataset[i, ip], alpha_prop * dataset[i - 1, ip], sigma2_mcmc[ip]^0.5, log = TRUE)
      log_MH_ratio <- log_MH_ratio - dnorm(dataset[i, ip], alpha_mcmc[ip] * dataset[i - 1, ip], sigma2_mcmc[ip]^0.5, log = TRUE)
    }

    # Prior: contributo della prior per alpha
    log_MH_ratio <- log_MH_ratio + (alpha_trans_prop - 2 * log(1 + exp(alpha_trans_prop)))
    log_MH_ratio <- log_MH_ratio - (alpha_trans_mcmc - 2 * log(1 + exp(alpha_trans_mcmc)))

    # Aggiornamento del parametro alpha
    acc_alpha[ip] <- acc_alpha[ip] + min(1, exp(log_MH_ratio)) # Aggiornamento del tasso di accettazione
    if (runif(1) < min(1, exp(log_MH_ratio))) {
      alpha_mcmc[ip] <- alpha_prop # Accetta la proposta
    }
  }

  # Aggiornamento dei parametri sigma^2
  for (ip in 1:p)
  {
    # Calcolo dei parametri aggiornati per la distribuzione a posteriori di sigma^2
    a_p <- prior_sigma2_a + n / 2
```

```
    b_p <- prior_sigma2_b + 0.5 * (1 - alpha_mcmc[ip]^2) * dataset[1, ip]^2
    b_p <- prior_sigma2_b + 0.5 * sum((dataset[-1, ip] - alpha_mcmc[ip] *␣
↪dataset[-n, ip])^2)

    # Aggiornamento di sigma^2 da una distribuzione Gamma inversa
    sigma2_mcmc[ip] <- 1 / rgamma(1, shape = a_p, rate = b_p)
  }

  # Adattamento della varianza della proposta
  if (iter %% adapt_batch == 0) {
    if (iter < adapt_stop) {
      # Aggiornamento della deviazione standard per alpha basato sul tasso di␣
↪accettazione
      for (ip in 1:p)
      {
        acc_alpha[ip] <- acc_alpha[ip] / adapt_batch # Calcolo del tasso␣
↪medio di accettazione
        sd_prop_alpha[ip] <- exp(log(sd_prop_alpha[ip]) + adapt_a / (adapt_b␣
↪+ iter) * (acc_alpha[ip] - adapt_alpha))
        acc_alpha[ip] <- 0 # Resetta il contatore
      }
    }
  }

  # Memorizzazione dei risultati delle iterazioni
  alpha_out[iter, ] <- alpha_mcmc
  sigma2_out[iter, ] <- sigma2_mcmc
 }

 # Restituzione dei risultati finali
 return(list(alpha_out = alpha_out, sigma2_out = sigma2_out))
}
```

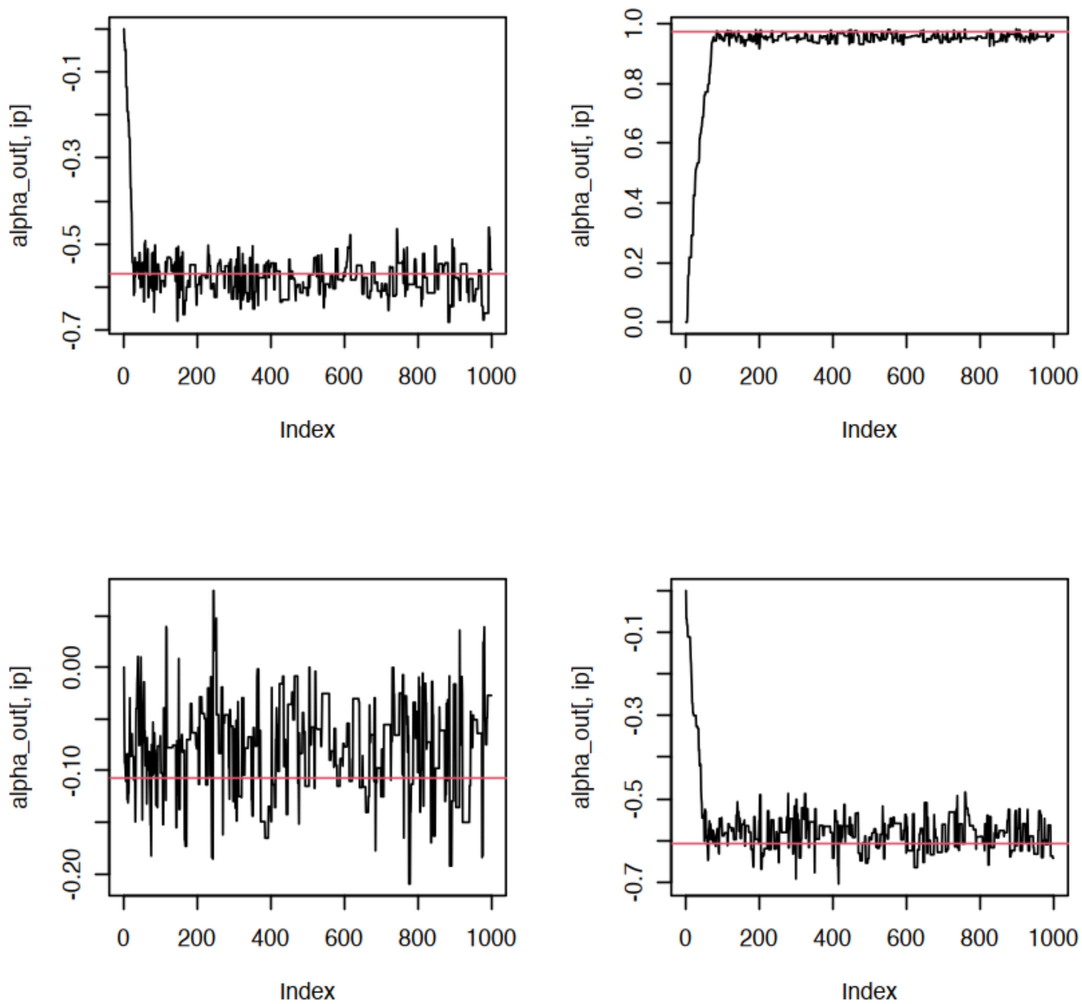
[3]:
```
## Simuliamo dei dati e testiamo l'algoritmo

# Definizione dei parametri di simulazione
p <- 4 # Numero di variabili
n <- 500 # Numero di osservazioni
alpha <- runif(p, -1, 1) # Coefficienti alpha generati casualmente␣
↪nell'intervallo [-1, 1]
sigma2 <- runif(p, 0.5, 2) # Varianze sigma^2 generate casualmente␣
↪nell'intervallo [0.5, 2]
x_sim <- matrix(NA, nrow = n, ncol = p) # Matrice per contenere i dati simulati

# Simulazione della prima riga del dataset
for (ip in 1:p)
{
```

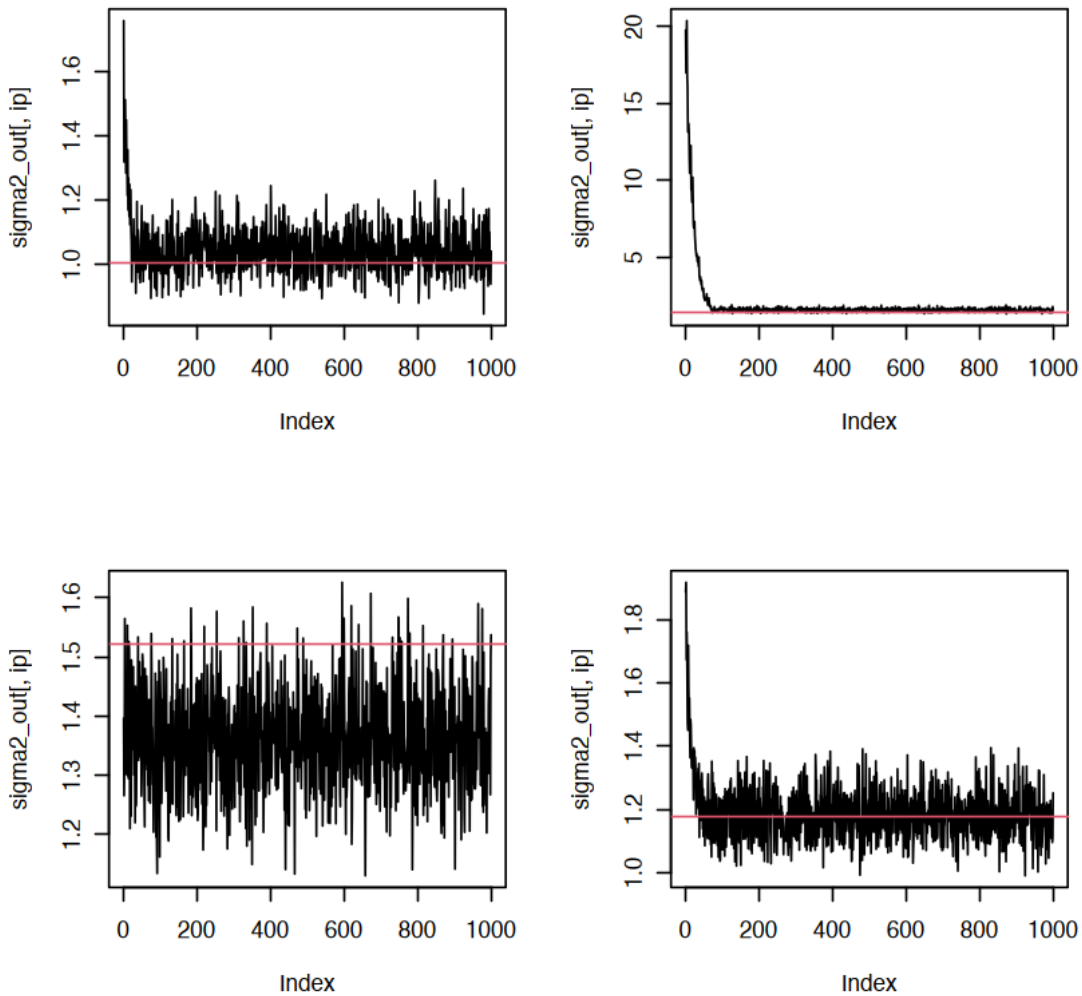
```
  # La prima osservazione di ogni variabile segue una distribuzione normale
  # con media 0 e varianza sigma2 / (1 - alpha^2) per garantire la stazionarietà
  x_sim[1, ip] <- rnorm(1, 0, (sigma2[ip] / (1 - alpha[ip]^2))^0.5)
}

# Simulazione delle osservazioni successive
for (i in 2:n)
{
  for (ip in 1:p)
  {
    # Le osservazioni successive seguono un modello autoregressivo di ordine 1␣
↪(AR(1))
    # con media alpha[ip] * valore precedente e deviazione standard␣
↪sqrt(sigma2[ip])
    x_sim[i, ip] <- rnorm(1, alpha[ip] * x_sim[i - 1, ip], sigma2[ip]^0.5)
  }
}

# Esecuzione del modello sull'insieme di dati simulati
sim_out <- model(
  x_sim, # Dataset simulato
  prior_sigma2_a = 1, # Parametro della prior di sigma^2
  prior_sigma2_b = 1, # Parametro della prior di sigma^2
  iterations = 1000, # Numero di iterazioni MCMC
  adapt_batch = 50, # Intervallo per l'adattamento della proposta
  adapt_a = 2000, # Parametro di adattamento
  adapt_b = 1000, # Parametro di adattamento
  adapt_alpha = 0.234, # Tasso di accettazione target
  adapt_stop = 1000 # Iterazione in cui si ferma l'adattamento
)

# Il risultato, `sim_out`, contiene i campioni MCMC per alpha e sigma^2
```

Vediamo dei plot dei risultati e indichiamo con una linea rossa il vero valore

[14]:
```
alpha_out = sim_out$alpha_out
sigma2_out = sim_out$sigma2_out

par(mfrow=c(2,2))
for(ip in 1:p)
{
  plot(alpha_out[,ip], type="l")
  abline(h = alpha[ip], col=2)
}
for(ip in 1:p)
{
  plot(sigma2_out[,ip], type="l")
```

```r
  abline(h = sigma2[ip], col=2)
}
par(mfrow = c(1, 1))
```



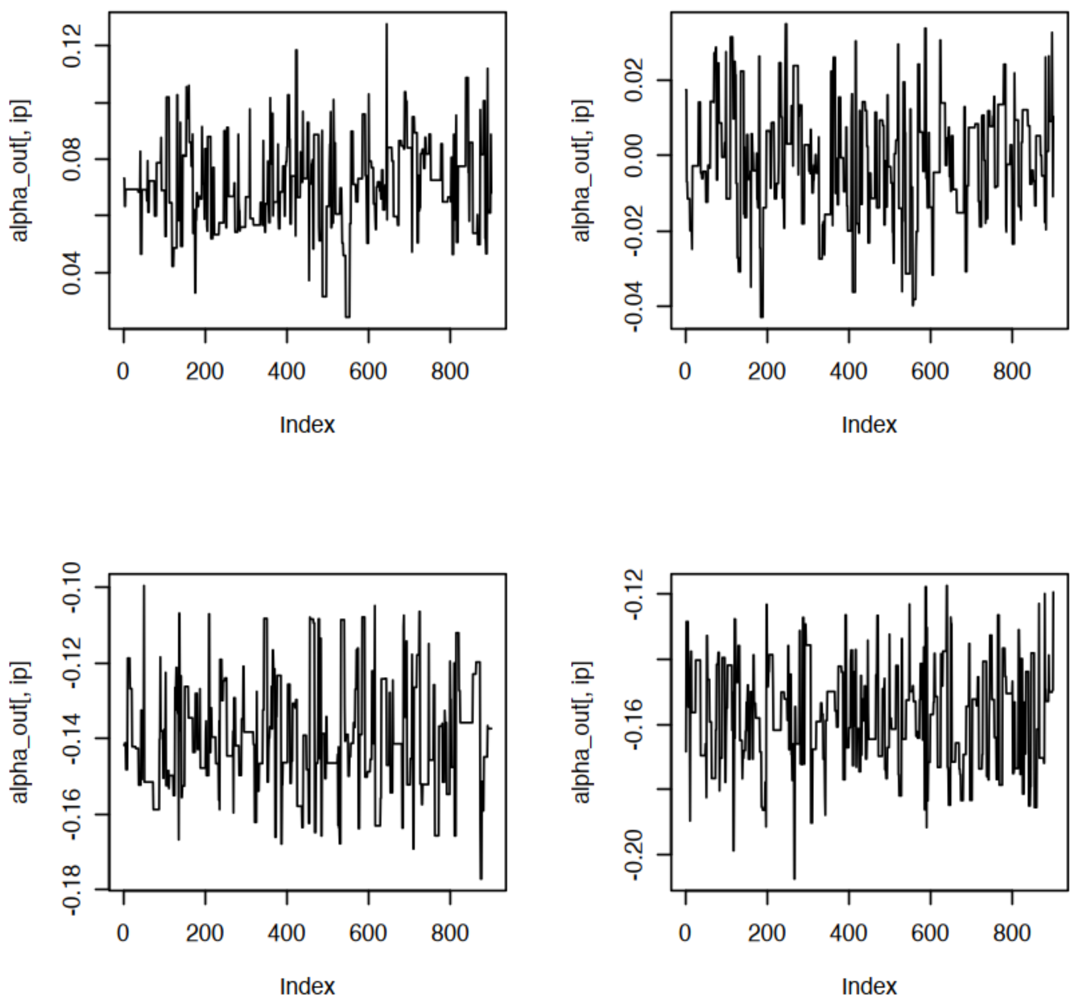


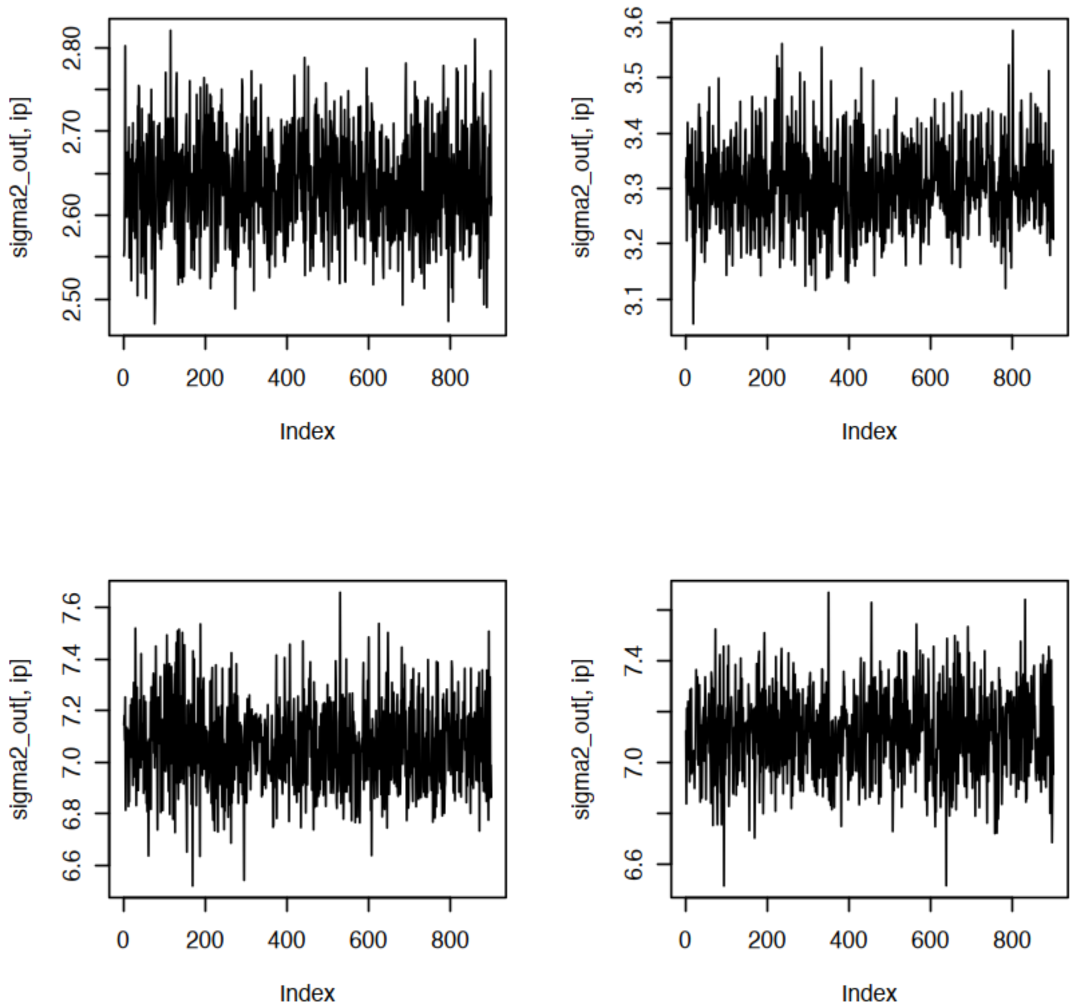
Dovremmo togliere il burnin, ma direi che il modello funziona. Adesso applichiamo il modello alla serie differenziata

[16]:
```
# applico alla serie
data_small_diff = apply(data_small, 2, diff)

data_out <- model(data_small_diff, prior_sigma2_a = 1, prior_sigma2_b = 1, iterations = 1000, adapt_batch = 50, adapt_a = 2000, adapt_b = 1000, adapt_alpha = 0.234, adapt_stop = 1000)
```

vediamo qualche risultato, eliminando un po' di burnin

[19]:
```
alpha_out = data_out$alpha_out[-c(1:100), ]
sigma2_out = data_out$sigma2_out[-c(1:100), ]

par(mfrow=c(2,2))
for(ip in 1:p)
{
  plot(alpha_out[,ip], type="l")
}
for(ip in 1:p)
{
  plot(sigma2_out[,ip], type="l")
}
par(mfrow = c(1, 1))
```





[55]:
```
library(coda)
summary(as.mcmc(data_out$alpha_out))
summary(as.mcmc(data_out$sigma2_out))
```

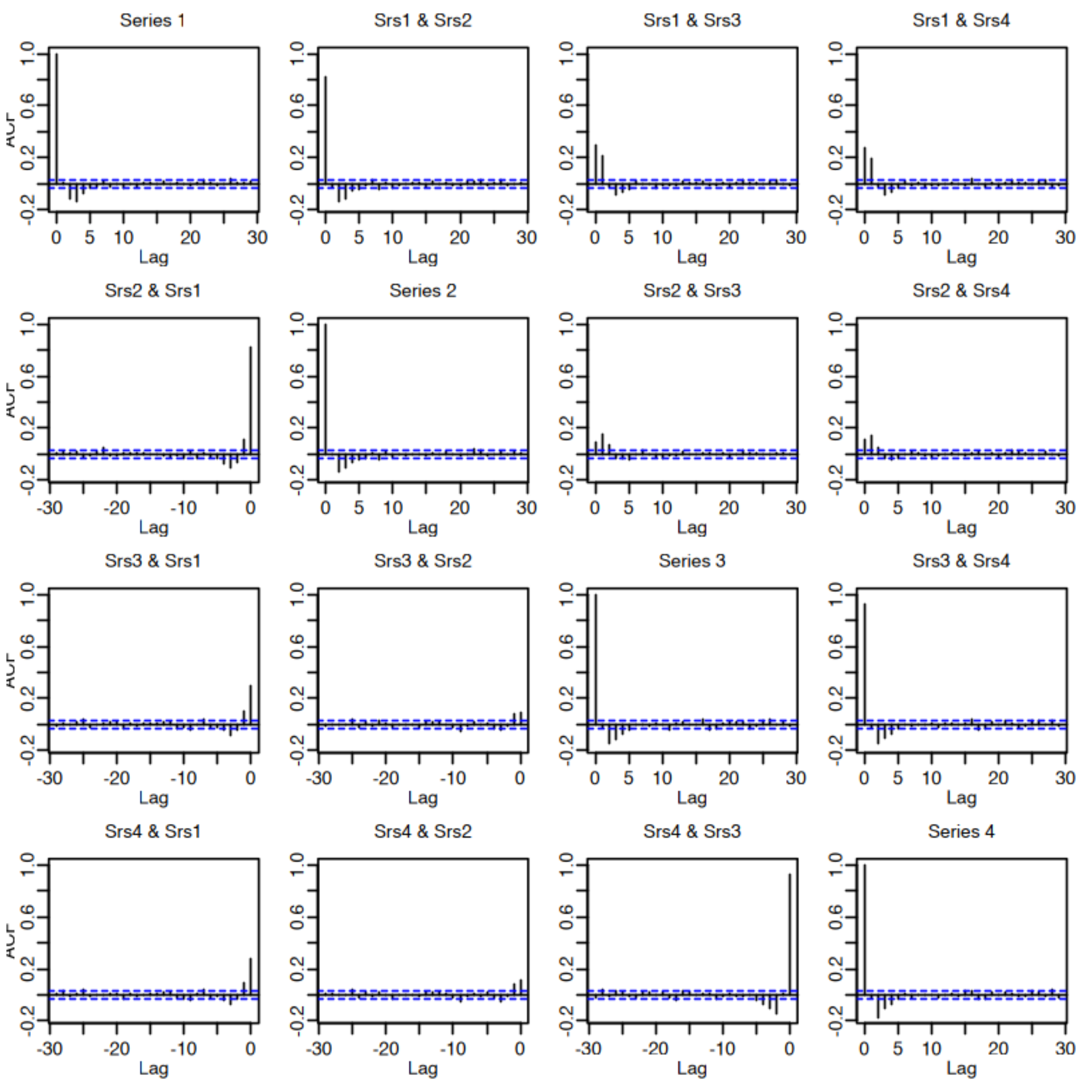
```
Iterations = 1:1000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
```

```
            Mean      SD  Naive SE Time-series SE
[1,]  0.0708180 0.01509 0.0004771       0.001104
[2,] -0.0006247 0.01649 0.0005214       0.001246
[3,] -0.1352222 0.01946 0.0006153       0.001819
[4,] -0.1574864 0.01782 0.0005636       0.001352

2. Quantiles for each variable:

         2.5%      25%       50%      75%    97.5%
var1  0.04113  0.06144  0.070897  0.08075  0.09945
var2 -0.03283 -0.01142 -0.001757  0.01060  0.03162
var3 -0.16987 -0.14463 -0.133867 -0.12530 -0.10598
var4 -0.19326 -0.17023 -0.157274 -0.14718 -0.12142



Iterations = 1:1000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

      Mean      SD Naive SE Time-series SE
[1,] 2.635 0.06191 0.001958       0.001958
[2,] 3.309 0.07943 0.002512       0.002512
[3,] 7.060 0.16656 0.005267       0.005267
[4,] 7.117 0.16879 0.005338       0.004290

2. Quantiles for each variable:

      2.5%   25%   50%   75% 97.5%
var1 2.515 2.595 2.636 2.677 2.759
var2 3.151 3.256 3.308 3.360 3.458
var3 6.749 6.951 7.053 7.177 7.389
var4 6.803 7.003 7.119 7.226 7.458
```

dagli intervalli vediamo che la seconda variabile è probabilemente un rumore bianco, visto che il CI di α_2 contiene lo zero. fate solo attenzione che questo non significa che la serie ha componenti indipendenti, perchè stiamo modellizando una differenza finita. Possiamo calcolare i residui e vedere se sono dei rumori bianchi

[56]:
```
# calcolimao i residui
nsample = nrow(alpha_out)
residui <- array(NA,c(nrow(data_small_diff), p, nsample))
for(isim in 1:nsample)
```

```
{
  residui[1, , isim] <- data_small_diff[1, ]
  for (ip in 1:p)
  {
    residui[-1, ip, isim] = data_small_diff[-1, ip] - alpha_out[isim, ip] *
    data_small_diff[-nrow(data_small_diff), ip]
  }

}

mean_residui <- apply(residui,c(1,2),mean)

acf(mean_residui)
```



dai grafici sembrerebbe che ci sia ancora della dipendenza temporale. Questo indica che il modello probabilmente non è appropriato

Implmentiamo il secondo modello con

$$\Sigma \sim IW(\nu, \Psi)$$

dove

$$\nu > k-1$$

dove k è la dimensione di Σ e Ψ è una matrice delle stesse dimensioni di Σ, simmetrica e definita positiva. Abbiamo che

$$E(\Sigma) = \frac{\Psi}{\nu - k - 1}$$

e

$$f(\Sigma) \propto |\Sigma|^{-(\nu+k+1)/2} \exp\left(-\frac{1}{2} tr(\Psi \Sigma^{-1})\right)$$

Si può vedere che questa è coniuggata con i dati visto che

$$f(\mathbf{x}) = f(\mathbf{x}_1|\mathbf{x}_0) f(\mathbf{x}_2|\mathbf{x}_1) f(\mathbf{x}_3|\mathbf{x}_2) \dots f(\mathbf{x}_n|\mathbf{x}_{n-1}) = \prod_{t=1}^{n} f(\mathbf{x}_t|\mathbf{x}_{t-1}) \propto$$

$$\prod_{t=1}^{n} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T \Sigma^{-1} (\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})\right) =$$

$$|\Sigma|^{-\frac{n}{2}} \exp\left(-\frac{1}{2}\sum_{t=1}^{n}(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T \Sigma^{-1} (\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})\right)$$

Ma visto che

$$(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T \Sigma^{-1} (\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})$$

è uno scalare, allora è uguale a

$$tr((\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T \Sigma^{-1} (\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1}))$$

e usando le proprietù della traccia abbiamo che è uguale a

$$tr((\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T \Sigma^{-1})$$

In definitiva, la full conditional di Σ è proporzionale a

$$|\Sigma|^{-(\nu+k+1)/2} \exp\left(-\frac{1}{2} tr(\Psi \Sigma^{-1})\right) |\Sigma|^{-\frac{n}{2}} \exp\left(-\frac{1}{2}\sum_{t=1}^{n} tr((\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T \Sigma^{-1})\right) =$$

$$|\Sigma|^{-(\nu+n+k+1)/2} \exp\left(-\frac{1}{2} tr(\Psi \Sigma^{-1} + \sum_{t=1}^{n}(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T \Sigma^{-1})\right) =$$

$$|\Sigma|^{-(\nu+n+k+1)/2} \exp\left(-\frac{1}{2} tr((\Psi + \sum_{t=1}^{n}(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T)\Sigma^{-1})\right) =$$

che è il kernel di

$$IW\left(\nu + n, \Psi + \sum_{t=1}^{n}(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})(\mathbf{x}_t - diag(\alpha)\mathbf{x}_{t-1})^T\right)$$

Rispetto al codice precedente abbiamo anche la variabile $\mathbf{x}_0$ su cui mettiamo la prior

$$\mathbf{x}_0 \sim N(\mathbf{0}, 10000 I_4)$$

abbiamo qundi che la sua full conditional è proporzionale a

$$\exp\left(-\frac{1}{2 \times 10000}\mathbf{x}_0^T\mathbf{x}_0\right)\exp\left(-\frac{1}{2}(\mathbf{x}_1 - diag(\alpha)\mathbf{x}_0)^T\Sigma^{-1}(\mathbf{x}_1 - diag(\alpha)\mathbf{x}_0)\right) =$$

$$\exp\left(-\frac{1}{2}\left(\mathbf{x}_0^T\left(diag(\alpha)\Sigma^{-1}diag(\alpha) + 10000^{-1}I\right)\mathbf{x}_0 - 2\mathbf{x}_0^T diag(\alpha)\Sigma^{-1}\mathbf{x}_1\right)\right)$$

Questo ci dice che

$$\mathbf{x}_0|\ldots \sim N(\left(diag(\alpha)\Sigma^{-1}diag(\alpha) + 10000^{-1}I\right)diag(\alpha)\Sigma^{-1}\mathbf{x}_1, \left(diag(\alpha)\Sigma^{-1}diag(\alpha) + 10000^{-1}I\right))$$

[57]:
```
library(MCMCpack) # Per generare campioni dalla distribuzione Inversa-Wishart␣
↪(riwish)
library(MASS) # Per funzionalità statistiche aggiuntive

# Definizione di una funzione per calcolare la densità di una normale␣
↪multivariata
dmnorm <- function(x, mean = rep(0, d), varcov, log = FALSE) {
  d <- if (is.matrix(varcov)) {
    ncol(varcov) # Dimensione del vettore di media
  } else {
    1
  }
  if (d > 1 & is.vector(x)) {
    x <- matrix(x, 1, d) # Assicura che x sia una matrice
  }
  n <- if (d == 1) {
    length(x) # Lunghezza per variabili unidimensionali
  } else {
    nrow(x) # Numero di righe
  }
  # Calcolo del termine di scarto (X - mean)
  X <- t(matrix(x, nrow = n, ncol = d)) - mean
  # Calcolo del quadrato della distanza pesata
  Q <- apply((solve(varcov) %*% X) * X, 2, sum)
  # Calcolo del logaritmo del determinante di varcov
  logDet <- sum(logb(abs(diag(qr(varcov)[[1]]))))
  # Calcolo della densità logaritmica
  logPDF <- as.vector(Q + d * logb(2 * pi) + logDet) / (-2)
```

```r
  if (log) {
    logPDF # Restituisce la densità in scala logaritmica
  } else {
    exp(logPDF) # Restituisce la densità
  }
}

# Funzione principale per il modello MCMC
model_v2 <- function(dataset_dataframe, prior_sigma2_nu, prior_sigma2_psi,
 ↪iterations = 1000, adapt_batch = 50, adapt_a = 2000, adapt_b = 1000,
 ↪adapt_alpha = 0.234, adapt_stop = 1000) {
  # Conversione del dataset in matrice
  dataset <- as.matrix(dataset_dataframe)
  p <- ncol(dataset) # Numero di variabili
  n <- nrow(dataset) # Numero di osservazioni

  # Inizializzazione degli output
  alpha_out <- matrix(NA, nrow = iterations, ncol = p) # Tracciato di alpha
  sigma2_out <- array(NA, c(iterations, p, p)) # Tracciato di sigma^2

  # Inizializzazione dei parametri
  alpha_mcmc <- rep(0, p) # Inizializzazione di alpha
  sigma2_mcmc <- diag(1, p) # Matrice di varianza-covarianza iniziale
  y0 <- rep(0, p) # Valori iniziali di y per il modello autoregressivo

  # Inizializzazione per la proposta di alpha
  sd_prop_alpha <- rep(0.1, p) # Deviazioni standard della proposta
  acc_alpha <- rep(0, p) # Contatori di accettazione per alpha

  for (iter in 1:iterations) {
    # Calcolo dell'inversa della matrice di varianza-covarianza
    inv_sigma_mcmc <- solve(sigma2_mcmc)

    ## campiono yo
    var_p <- solve(diag(alpha_mcmc,p) %*% inv_sigma_mcmc %*% diag(alpha_mcmc,p)
 ↪+ diag(1 / 10000,p))
    mean_p <- matrix(var_p %*% diag(alpha_mcmc, p) %*% inv_sigma_mcmc %*%
 ↪matrix(dataset[1, ], ncol = 1),ncol=1)
    y0 <- mean_p + t(chol(var_p)) %*% matrix(rnorm(4, 0, 1), ncol = 1)
    # Aggiornamento di alpha
    for (ip in 1:p) {
      # Trasformazione logistica per garantire alpha in (-1, 1)
      alpha_trans_mcmc <- log((alpha_mcmc[ip] + 1) / (1 - alpha_mcmc[ip]))
      alpha_trans_prop <- rnorm(1, alpha_trans_mcmc, sd_prop_alpha[ip])
      alpha_prop <- (-1 + exp(alpha_trans_prop)) / (1 + exp(alpha_trans_prop))

      # Aggiorna il vettore alpha con la proposta
```

```
    alpha_vec_mcmc <- alpha_mcmc
    alpha_vec_prop <- alpha_mcmc
    alpha_vec_prop[ip] <- alpha_prop

    # Calcolo del rapporto di Metropolis-Hastings
    log_MH_ratio <- 0
    # Likelihood
    log_MH_ratio <- log_MH_ratio +
      (-0.5 * t(matrix(dataset[1, ] - alpha_vec_prop * y0, ncol = 1)) %*% inv_sigma_mcmc %*%
        (matrix(dataset[1, ] - alpha_vec_prop * y0, ncol = 1))) -
      (-0.5 * t(matrix(dataset[1, ] - alpha_vec_mcmc * y0, ncol = 1)) %*% inv_sigma_mcmc %*%
        (matrix(dataset[1, ] - alpha_vec_mcmc * y0, ncol = 1)))

    for (i in 2:n) {
      log_MH_ratio <- log_MH_ratio +
        (-0.5 * t(matrix(dataset[i, ] - alpha_vec_prop * dataset[i - 1, ], ncol = 1)) %*% inv_sigma_mcmc %*%
          (matrix(dataset[i, ] - alpha_vec_prop * dataset[i - 1, ], ncol = 1))) -
        (-0.5 * t(matrix(dataset[i, ] - alpha_vec_mcmc * dataset[i - 1, ], ncol = 1)) %*% inv_sigma_mcmc %*%
          (matrix(dataset[i, ] - alpha_vec_mcmc * dataset[i - 1, ], ncol = 1)))
    }

    # Prior per alpha
    log_MH_ratio <- log_MH_ratio + (alpha_trans_prop - 2 * log(1 + exp(alpha_trans_prop)))
    log_MH_ratio <- log_MH_ratio - (alpha_trans_mcmc - 2 * log(1 + exp(alpha_trans_mcmc)))

    # Aggiornamento di alpha
    acc_alpha[ip] <- acc_alpha[ip] + min(1, exp(log_MH_ratio))
    if (runif(1) < min(1, exp(log_MH_ratio))) {
      alpha_mcmc[ip] <- alpha_prop
    }
  }

  # Aggiornamento di sigma^2 (matrice Inversa-Wishart)
  nu_p <- prior_sigma2_nu + n # Aggiornamento del parametro di grado di libertà
  psi_p <- prior_sigma2_psi +
    matrix(dataset[1, ] - alpha_mcmc * y0, ncol = 1) %*%
    matrix(dataset[1, ] - alpha_mcmc * y0, nrow = 1)
```

```
    for (i in 2:n) {
      psi_p <- psi_p +
        matrix(dataset[i, ] - alpha_mcmc * dataset[i - 1, ], ncol = 1) %*%
        matrix(dataset[i, ] - alpha_mcmc * dataset[i - 1, ], nrow = 1)
    }
    sigma2_mcmc <- riwish(nu_p, psi_p)

    # Adattamento della varianza della proposta
    if (iter %% adapt_batch == 0 && iter < adapt_stop) {
      for (ip in 1:p) {
        acc_alpha[ip] <- acc_alpha[ip] / adapt_batch
        sd_prop_alpha[ip] <- exp(log(sd_prop_alpha[ip]) + adapt_a / (adapt_b +
 ↪iter) * (acc_alpha[ip] - adapt_alpha))
        acc_alpha[ip] <- 0
      }
    }

    # Salvataggio degli output
    alpha_out[iter, ] <- alpha_mcmc
    sigma2_out[iter, , ] <- sigma2_mcmc
  }

  return(list(alpha_out = alpha_out, sigma2_out = sigma2_out))
}
```

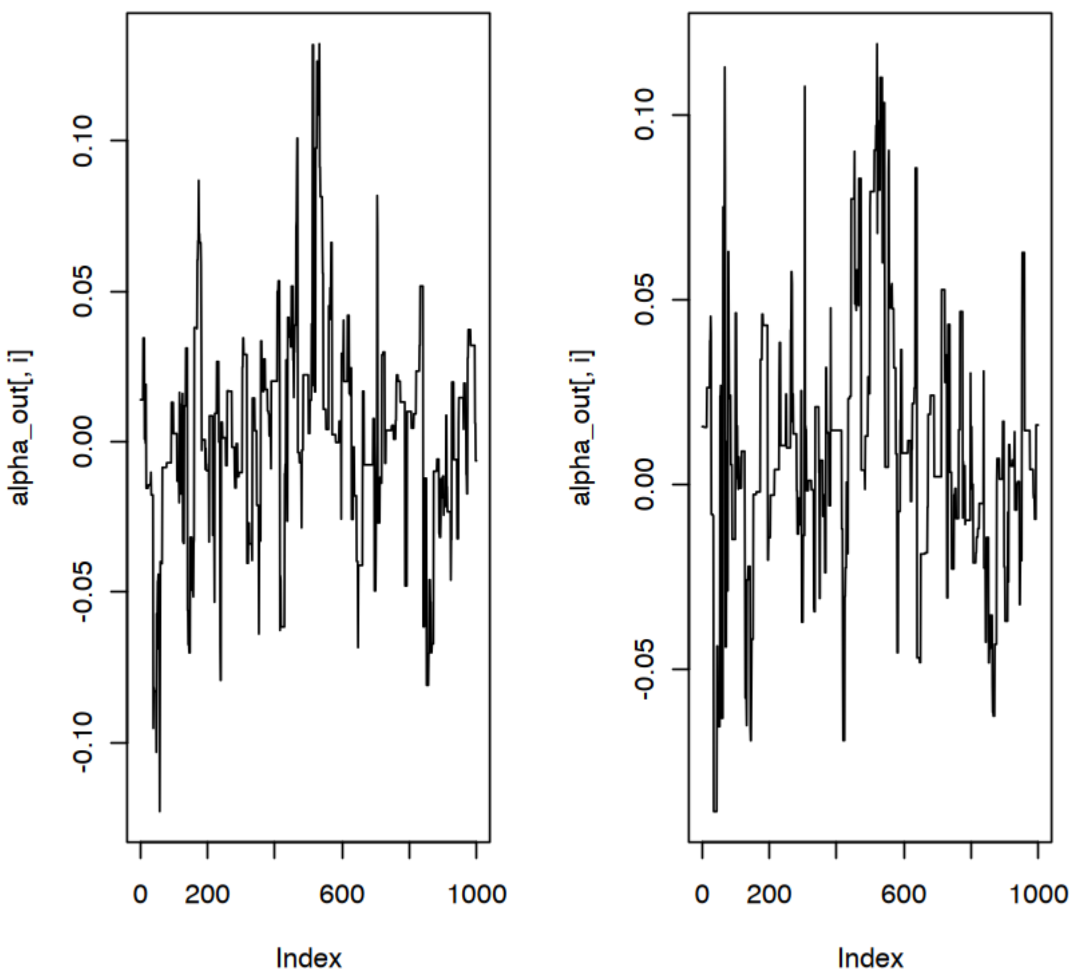
Il codice ci mette un po' e quindi lo applico a una porzione dei dati

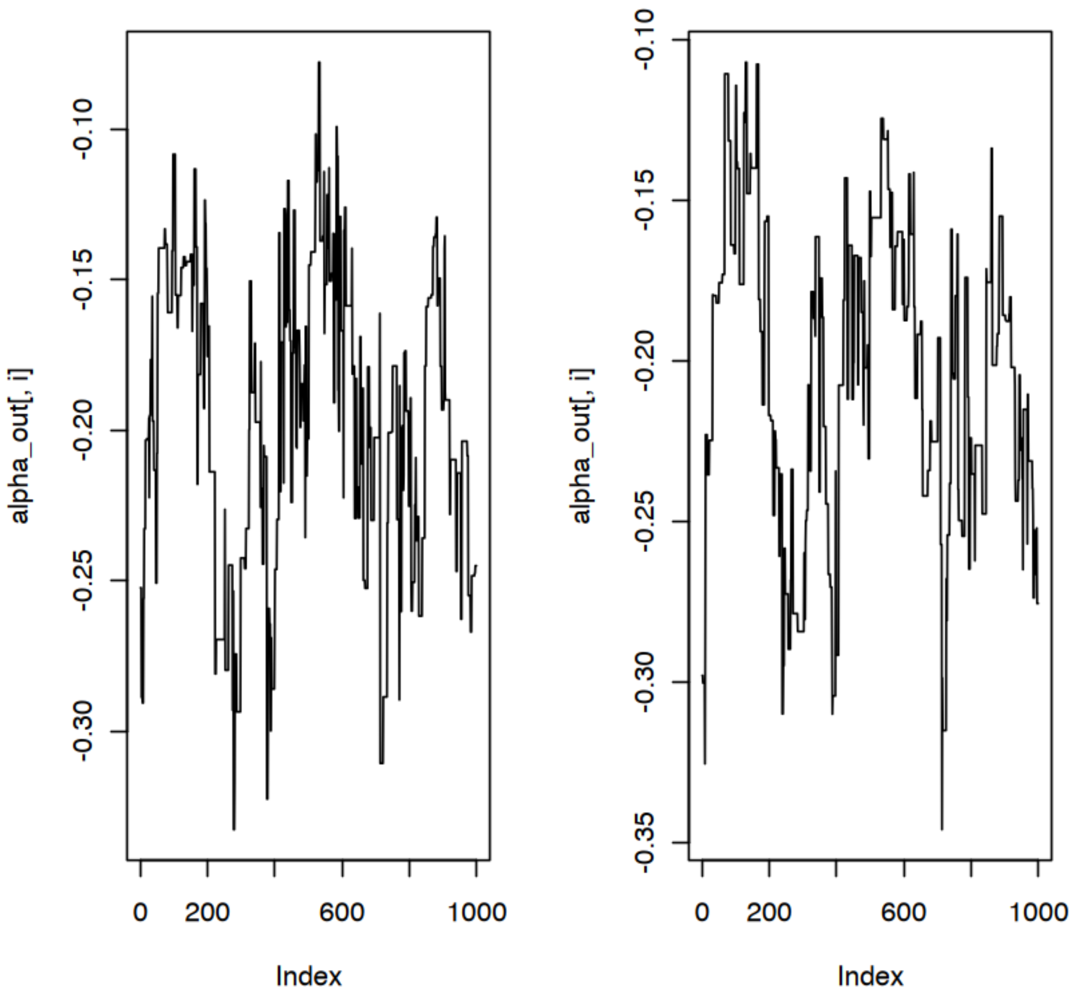
[58]:
```
res_out <- model_v2(data_small_diff[1:200,], prior_sigma2_nu = 6,
 ↪prior_sigma2_psi = diag(1,4), iterations = 2000, adapt_batch = 50, adapt_a =
 ↪2000, adapt_b = 1000, adapt_alpha = 0.234, adapt_stop = 1000)
```

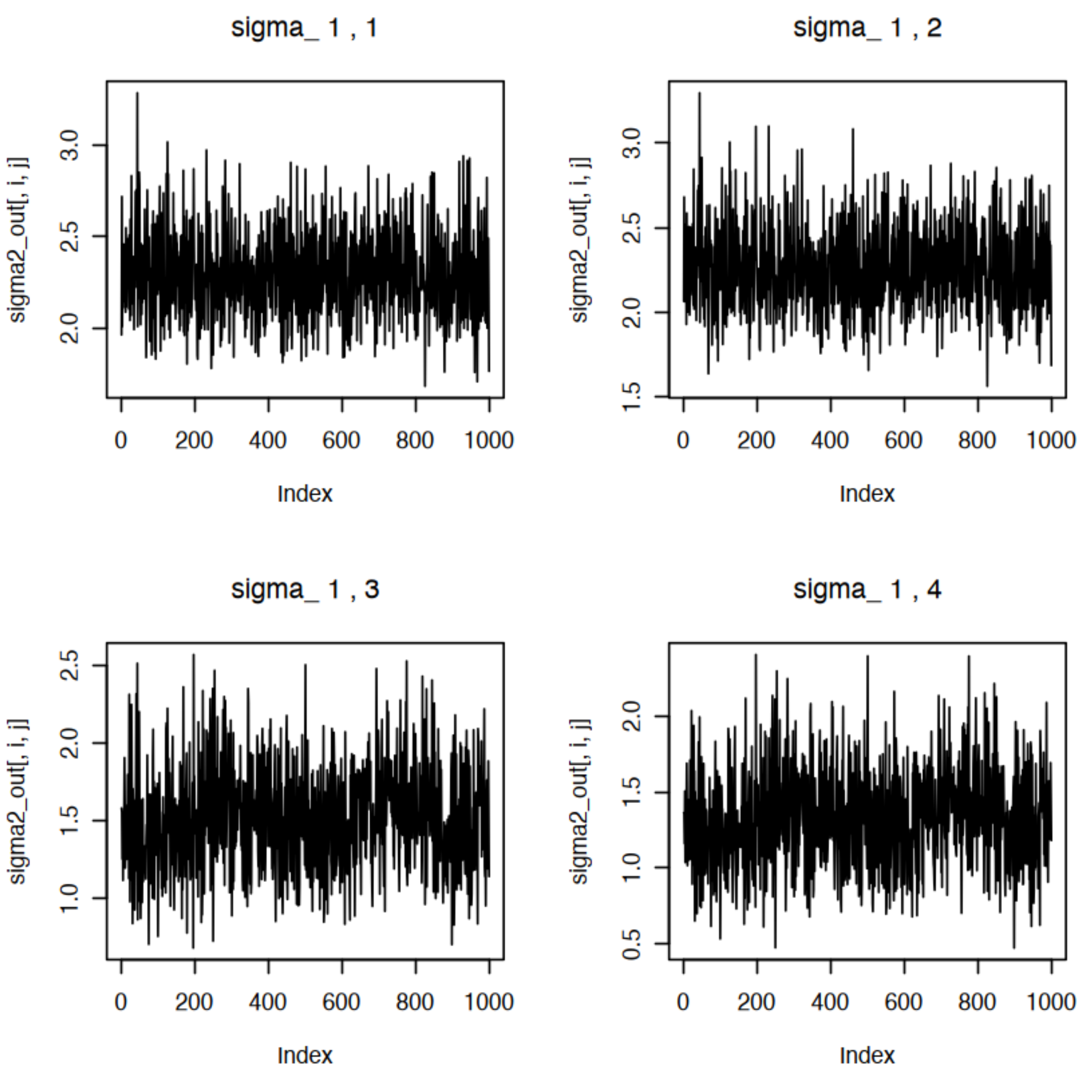
[59]:
```
sigma2_out <- res_out$sigma2_out[-c(1:1000), , ]
alpha_out <- res_out$alpha_out[-c(1:1000), ]
```

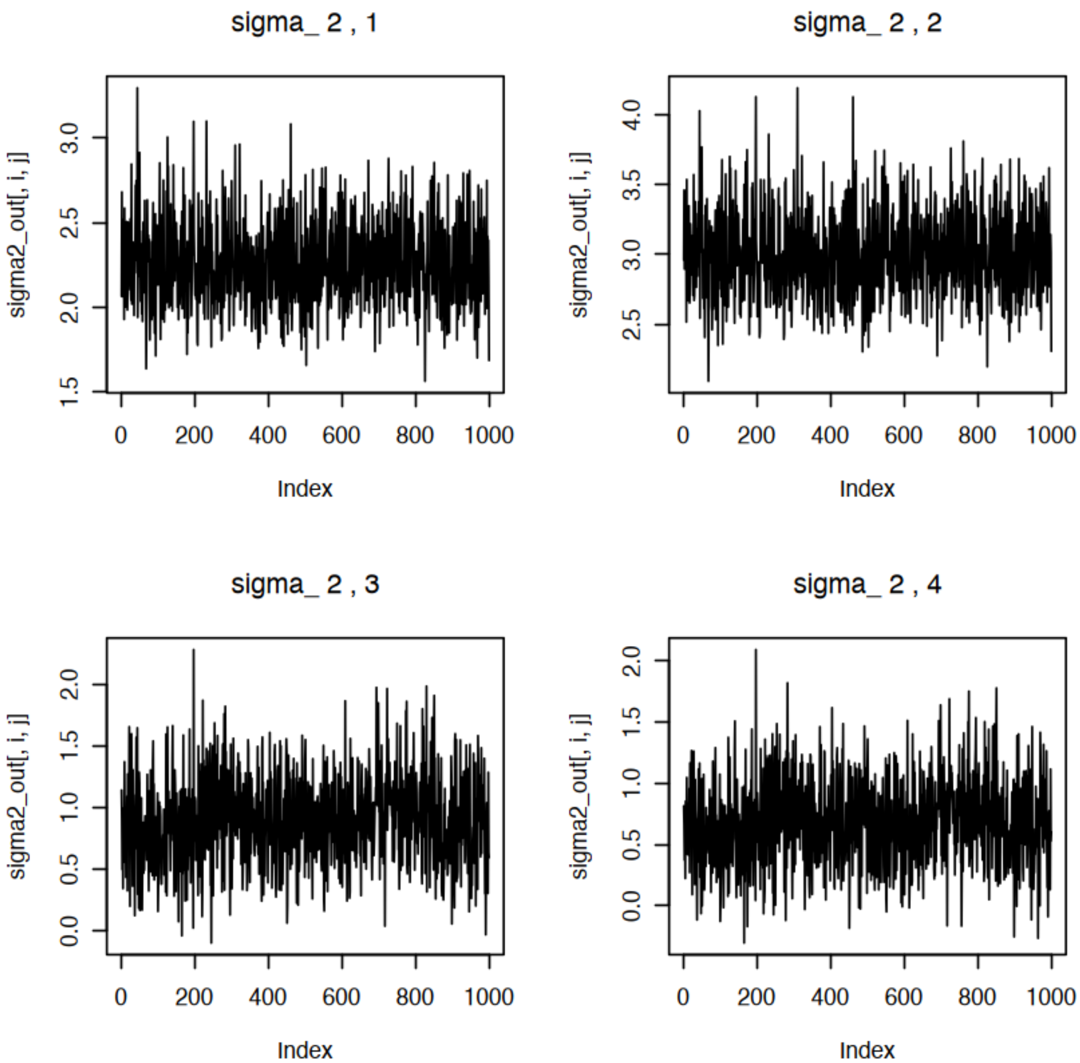
[60]:
```
par(mfrow = c(1, 2))
for(i in 1:p)
{
  plot(alpha_out[, i], type = "l")
}
par(mfrow = c(1, 1))

par(mfrow = c(2, 2))
for (i in 1:p)
{
  for(j in 1:p)
  {
    plot(sigma2_out[, i,j], type = "l", main = paste("sigma_",i,",",j))
```

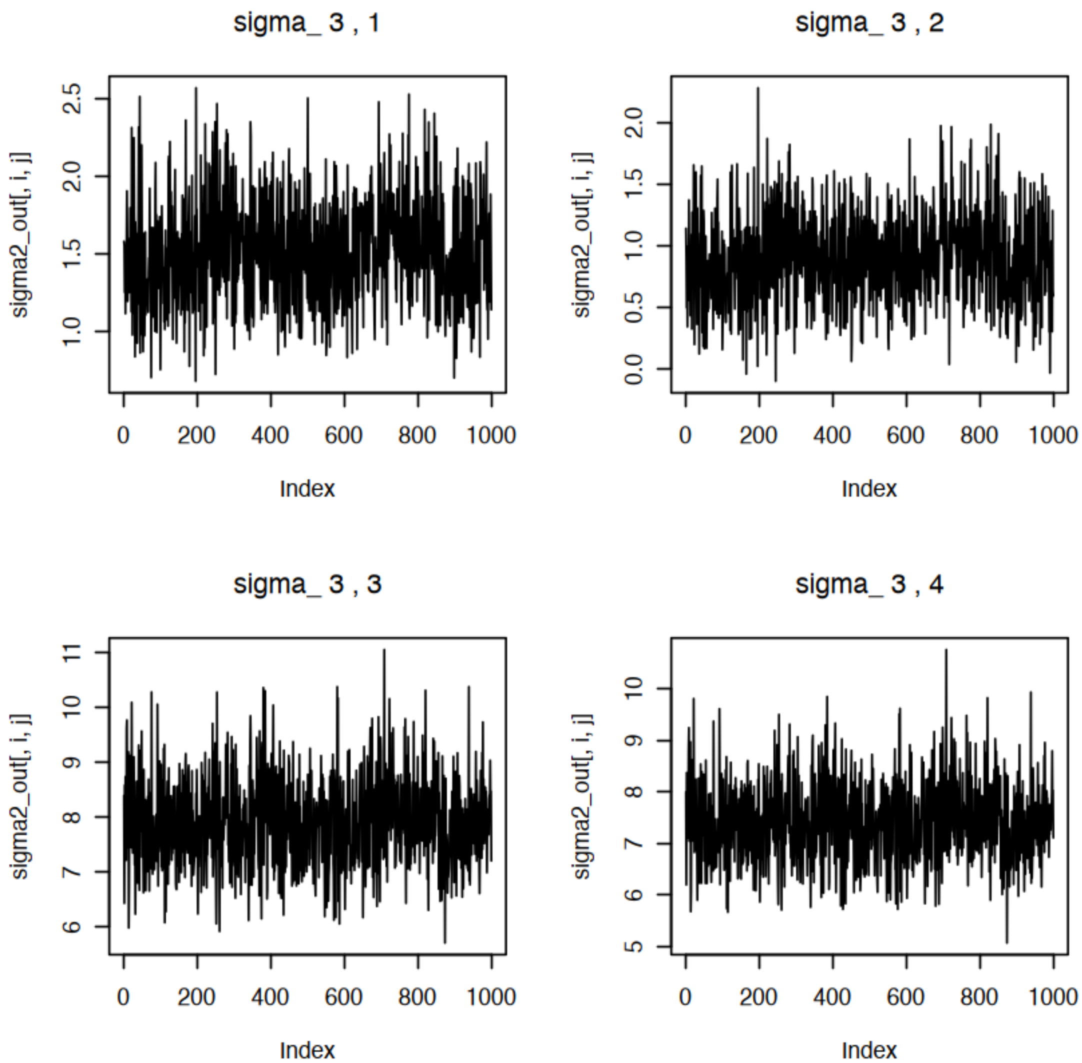
```
  }

}
par(mfrow = c(1, 1))
```

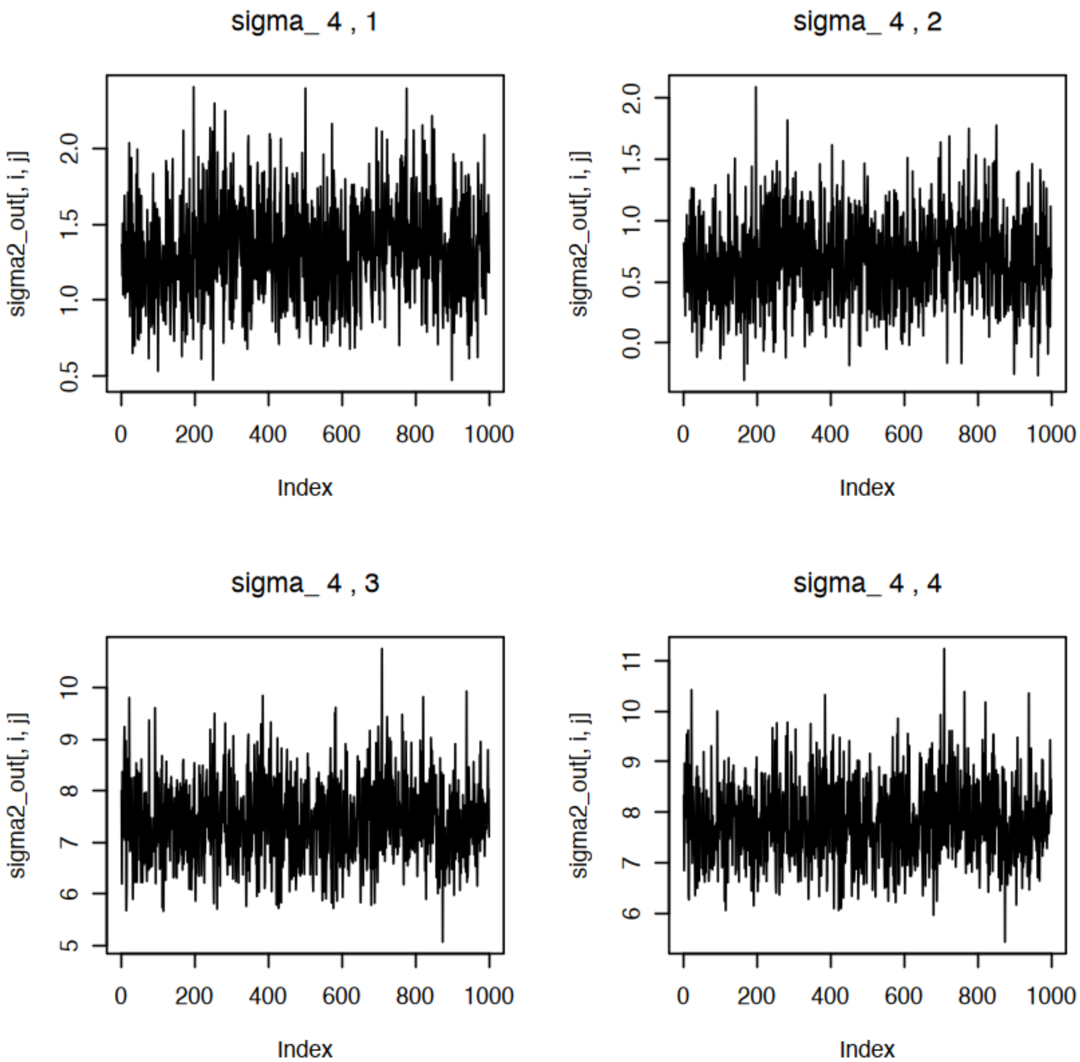



alpha_out[, i]
Index
-0.10
-0.15
-0.20
-0.25
-0.30
-0.35
0
200
600
1000


sigma_ 1 , 1
sigma_ 1 , 2
sigma_ 1 , 3
sigma_ 1 , 4
sigma2_out[, i, j]
Index
0
200
400
600
800
1000


sigma_ 2 , 1
sigma2_out[, i, j]
Index
sigma_ 2 , 2
sigma2_out[, i, j]
Index
sigma_ 2 , 3
sigma2_out[, i, j]
Index
sigma_ 2 , 4
sigma2_out[, i, j]
Index


sigma_ 3 , 1
sigma_ 3 , 2
sigma_ 3 , 3
sigma_ 3 , 4
sigma2_out[, i, j]
Index
0
200
400
600
800
1000



In questo nuovo modello la serie 1 e 2 non hanno componente temporale marginale. In questo caso può essere interessante anche valutare la stima della matrice di varianza (almeno il suo valor medio a posteriori)

[ ]:

```

```

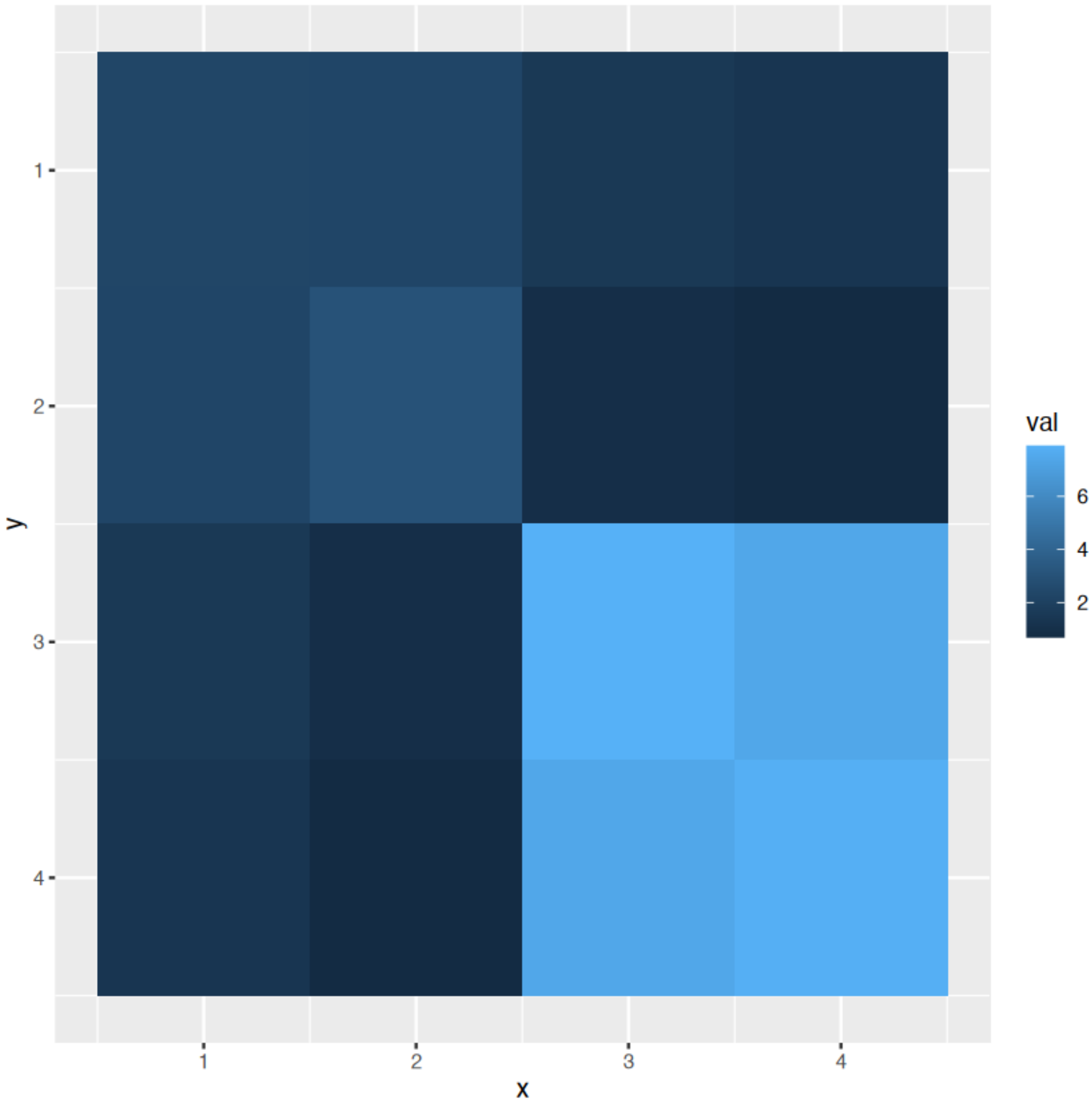
[61]:

```
mean_sigma = apply(sigma2_out,c(2,3), mean)
mean_sigma
data_mean_sigma <- data.frame(val = c(mean_sigma), x = rep(1:4, 4), y = rep(1:
 ↪4, each = 4))


data_mean_sigma %>% ggplot(aes(x = x, y = y)) +
```

```
  geom_tile(aes(fill = val)) + scale_y_reverse()
```

A matrix: 4 x 4 of type dbl

2.288878	2.2591600	1.518066	1.3264144
2.259160	2.9959888	0.899745	0.6858177
1.518066	0.8997450	7.904664	7.4028973
1.326414	0.6858177	7.402897	7.8027749



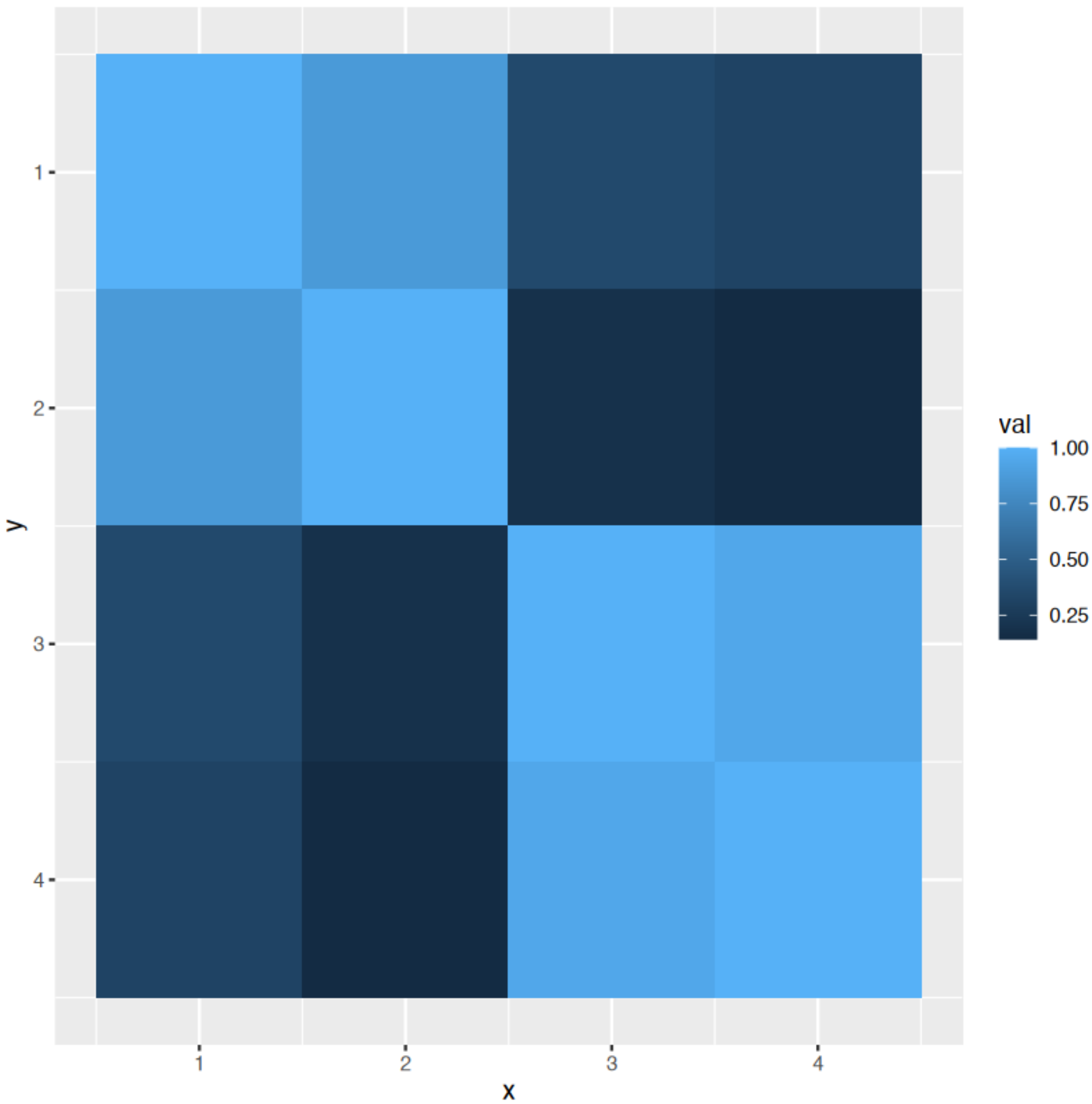
Visto che le varianze sono differenti, anche le scale delle covarianze lo sono. Per avere un'idea più chiara conviene studiare le correlazioni

[62]:
```
cor_out = sigma2_out
for(i in 1:dim(sigma2_out)[1])
{
```

```
  cor_out[i,,] = cov2cor(sigma2_out[i,,])
}

mean_cor <- apply(cor_out, c(2, 3), mean)
mean_cor
data_mean_cor <- data.frame(val = c(mean_cor), x = rep(1:4, 4), y = rep(1:4,
 ↪each = 4))
data_mean_cor %>% ggplot(aes(x = x, y = y)) +
  geom_tile(aes(fill = val)) +
  scale_y_reverse()
```

A matrix: 4 x 4 of type dbl

1.0000000	0.8621516	0.3561486	0.3131326
0.8621516	1.0000000	0.1844719	0.1415241
0.3561486	0.1844719	1.0000000	0.9422975
0.3131326	0.1415241	0.9422975	1.0000000



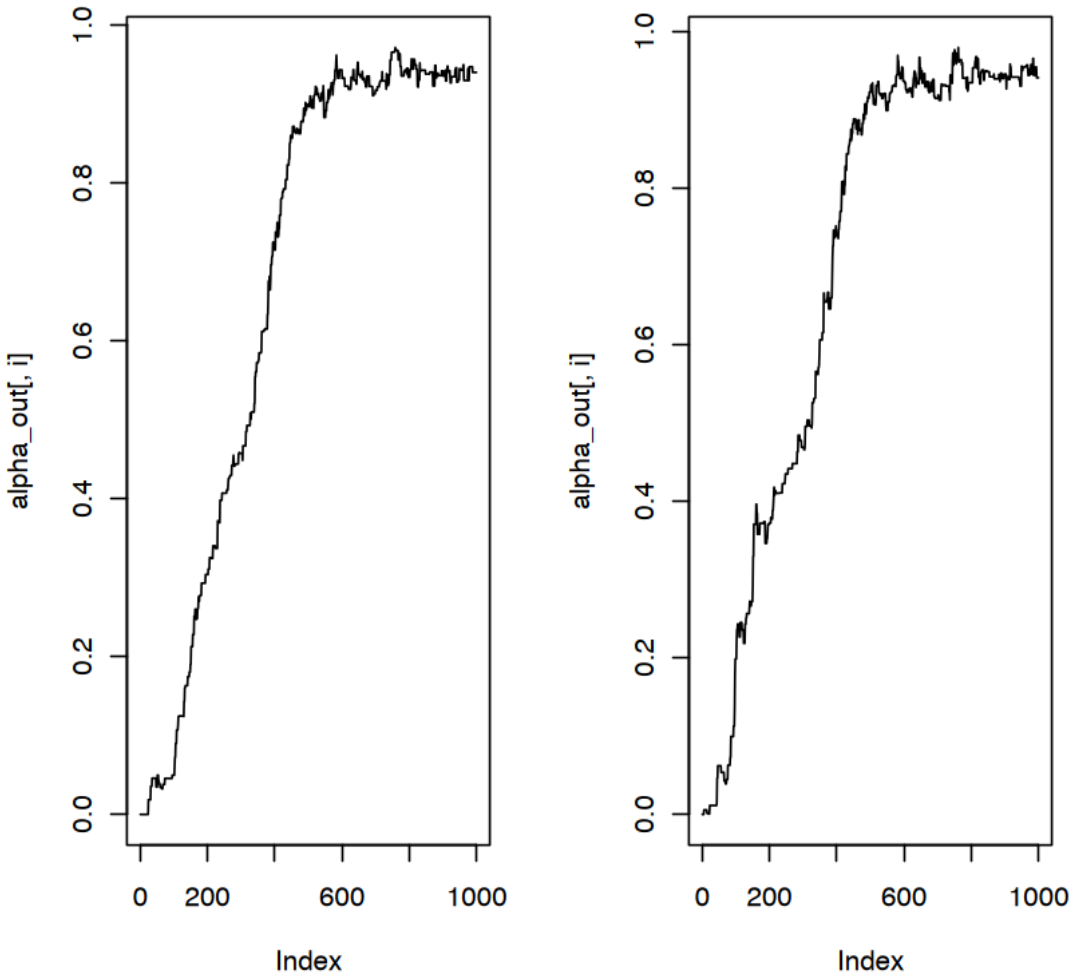
questo ci suggerisce che la dipendenza spaziale tra le stazioni è più forte della dipendenza tra le due temperature.

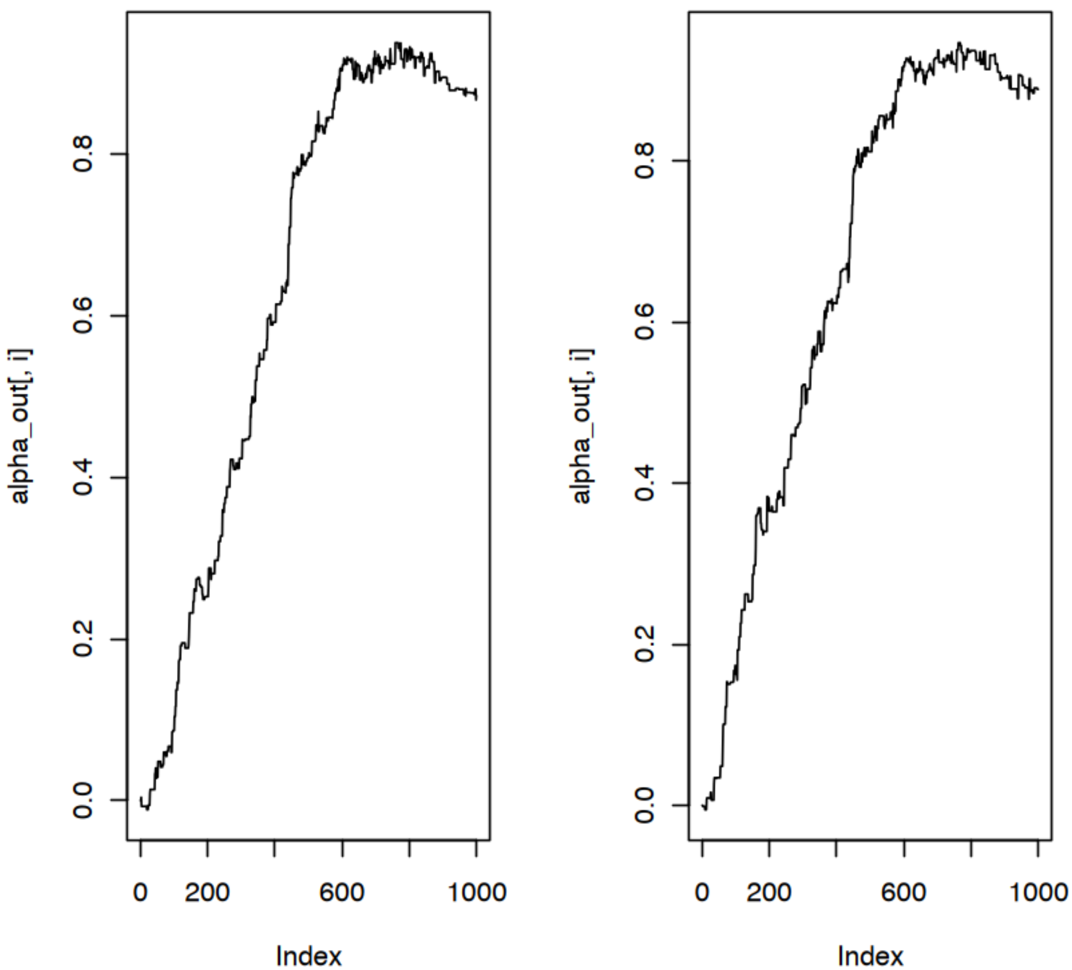
Come punto extra, provo a applicare il modello sulla serie originale, però centrata su zero (non abbiamo un parametro che modella la media)

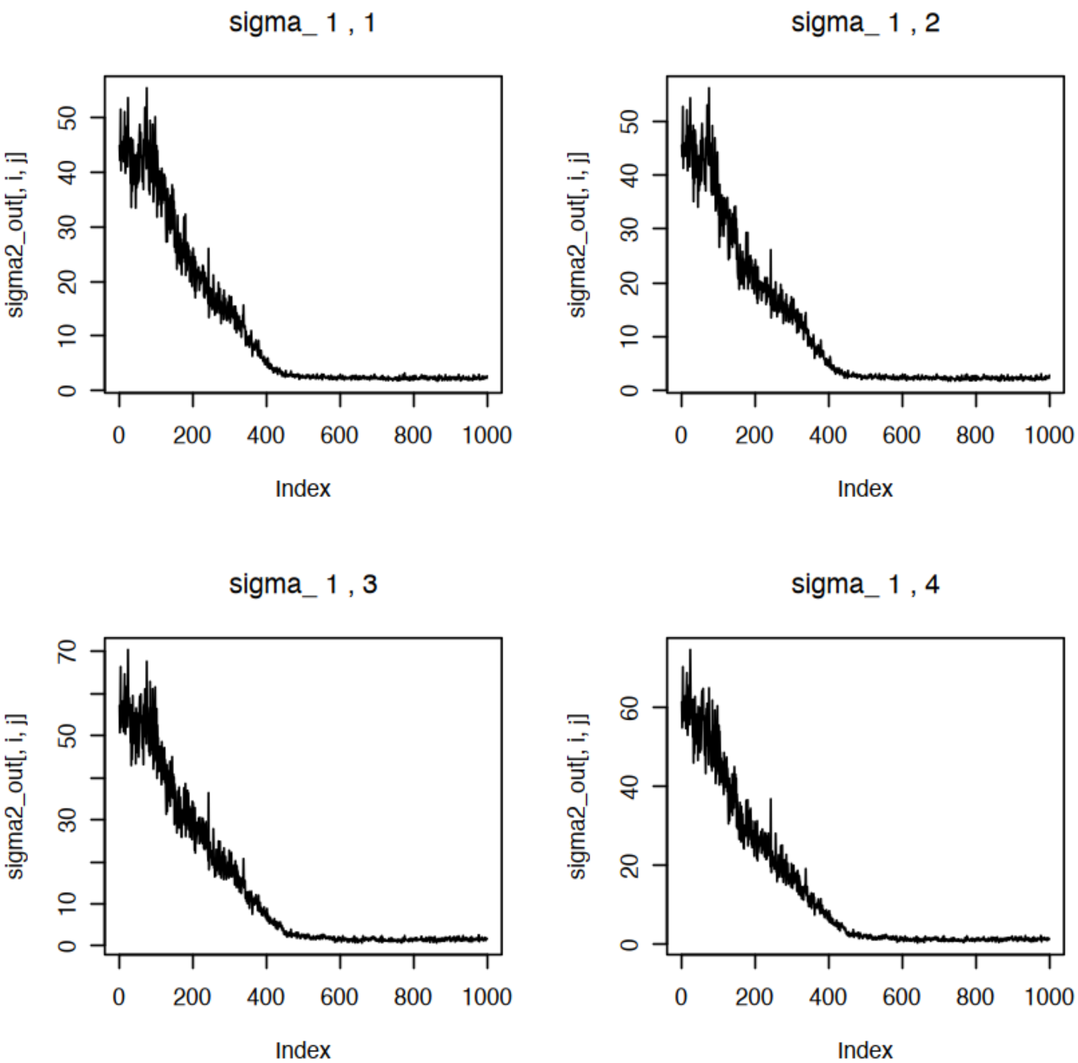
[63]:
```
data_app <- data_small[1:200, ]
for(i in 1:dim(data_app)[2])
{
  data_app[, i] = data_app[, i] - mean(data_app[, i])
}
res_out <- model_v2(dataset = data_app, prior_sigma2_nu = 6, prior_sigma2_psi =␣
↪diag(1, 4), iterations = 1000, adapt_batch = 50, adapt_a = 2000, adapt_b =␣
↪1000, adapt_alpha = 0.234, adapt_stop = 1000)
```

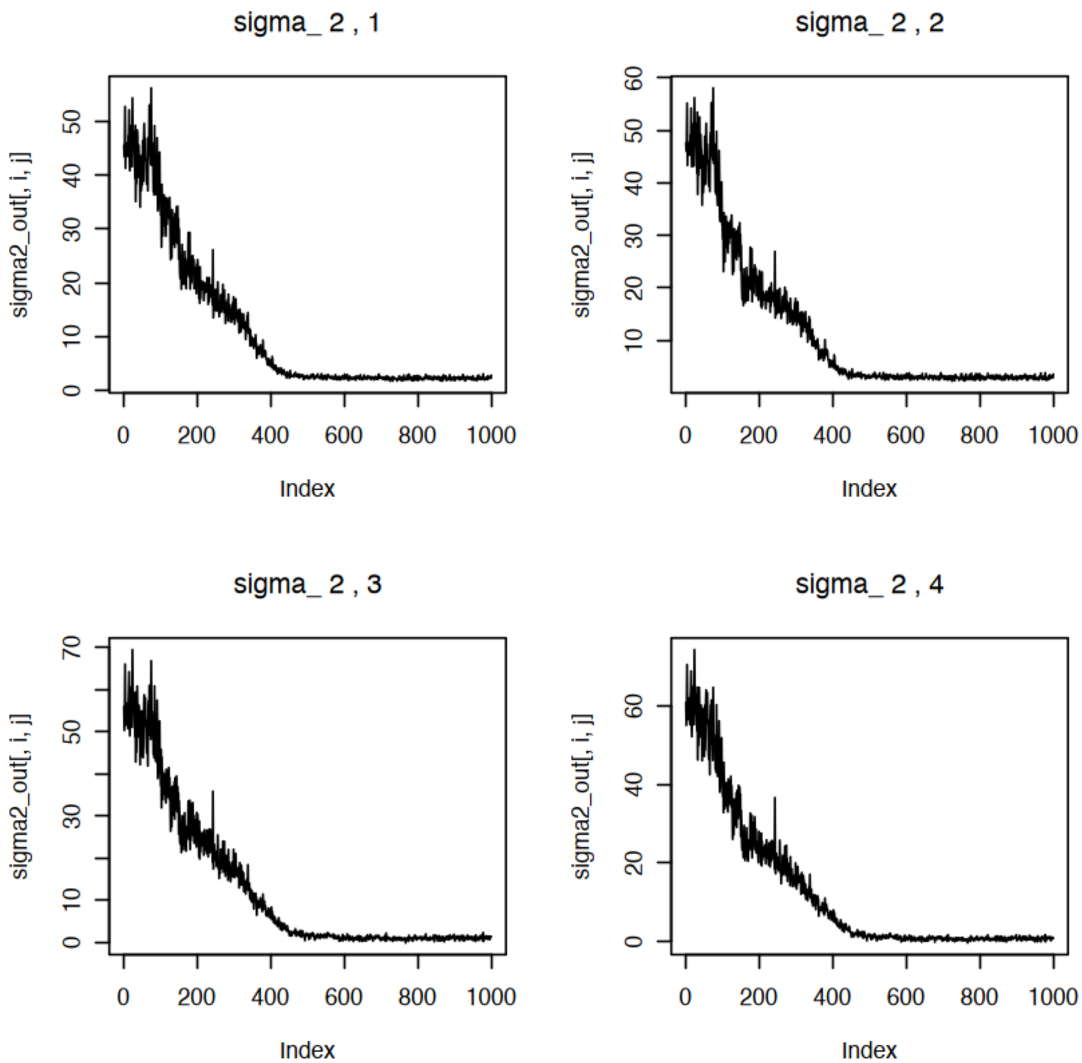
dai risultati vediamo che servono più iterazioni, altrimenti facendo burnin abbiamo veramente pochi campioni

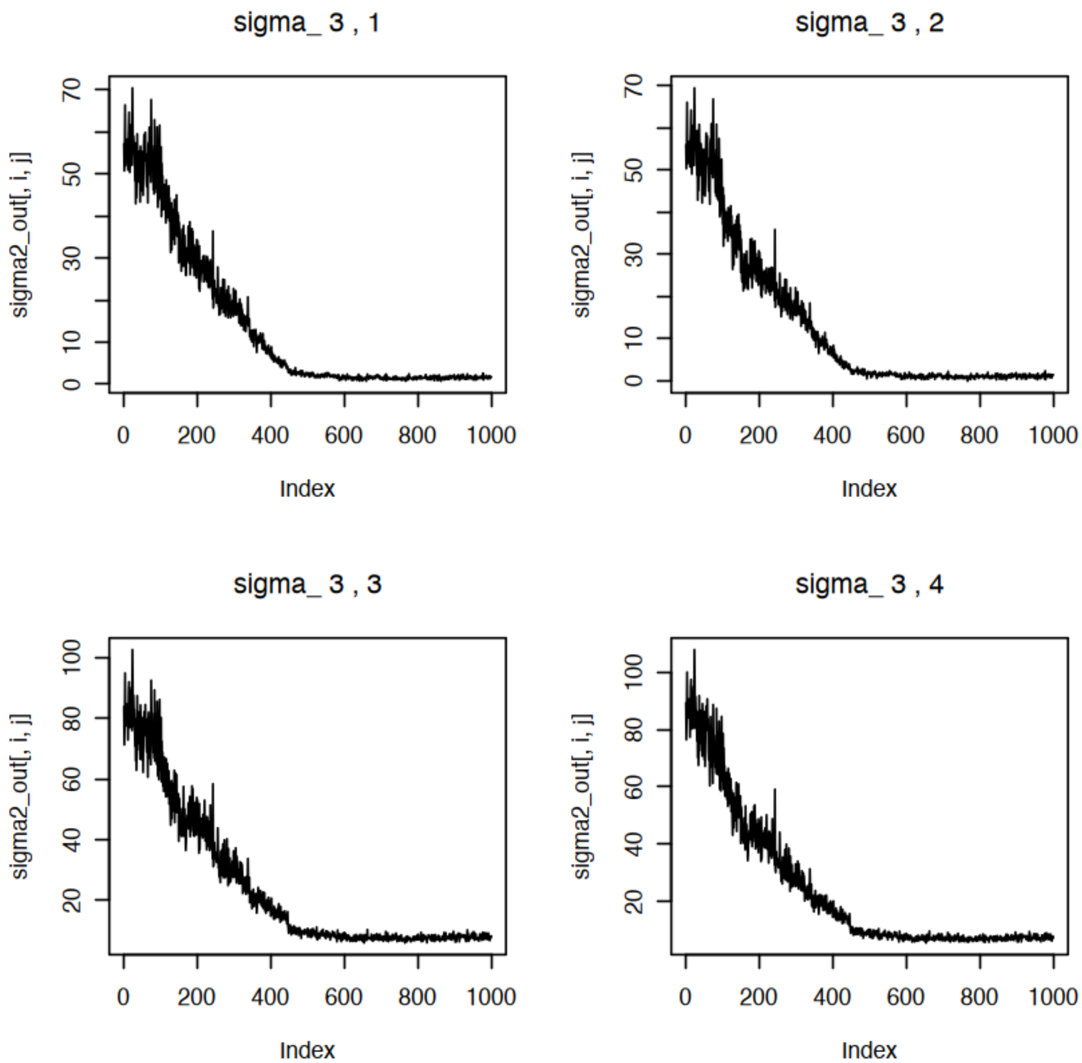
[64]:
```
sigma2_out <- res_out$sigma2_out
alpha_out <- res_out$alpha_out
par(mfrow = c(1, 2))
for (i in 1:p)
{
  plot(alpha_out[, i], type = "l")
}
par(mfrow = c(1, 1))

par(mfrow = c(2, 2))
for (i in 1:p)
{
  for (j in 1:p)
  {
    plot(sigma2_out[, i, j], type = "l", main = paste("sigma_", i, ",", j))
  }
}
par(mfrow = c(1, 1))
```

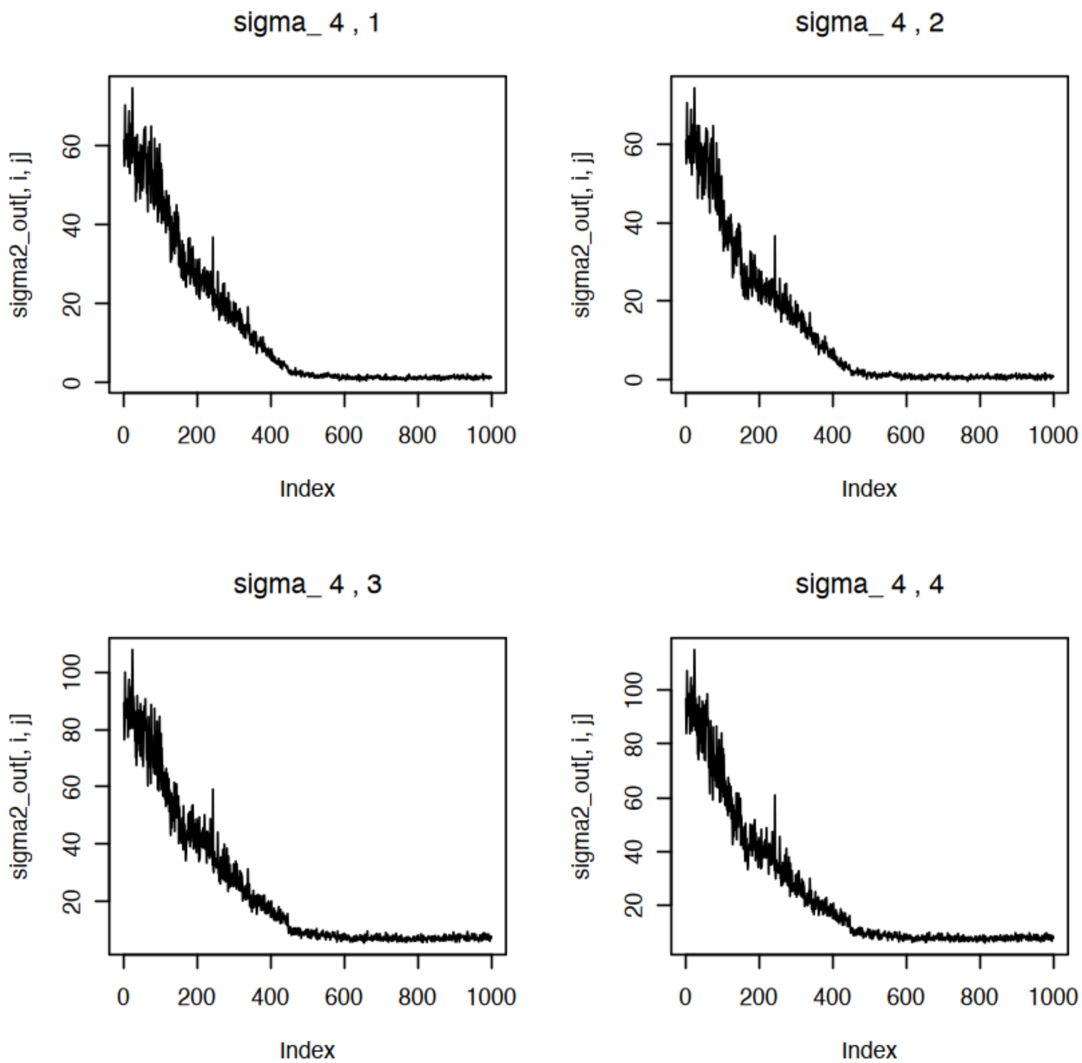

alpha_out[, i]
0.0
0.2
0.4
0.6
0.8
1.0
0
200
600
1000
Index
alpha_out[, i]
0.0
0.2
0.4
0.6
0.8
1.0
0
200
600
1000
Index


alpha_out[, i]
0.0
0.2
0.4
0.6
0.8
0
200
600
1000
Index
alpha_out[, i]
0.0
0.2
0.4
0.6
0.8
0
200
600
1000
Index


sigma_ 1 , 1
sigma_ 1 , 2
sigma_ 1 , 3
sigma_ 1 , 4
sigma2_out[, i, j]
Index
0
200
400
600
800
1000
0
10
20
30
40
50
60
70


sigma_ 2 , 1
sigma_ 2 , 2
sigma_ 2 , 3
sigma_ 2 , 4
sigma2_out[, i, j]
Index
0
200
400
600
800
1000
0
10
20
30
40
50
60
70

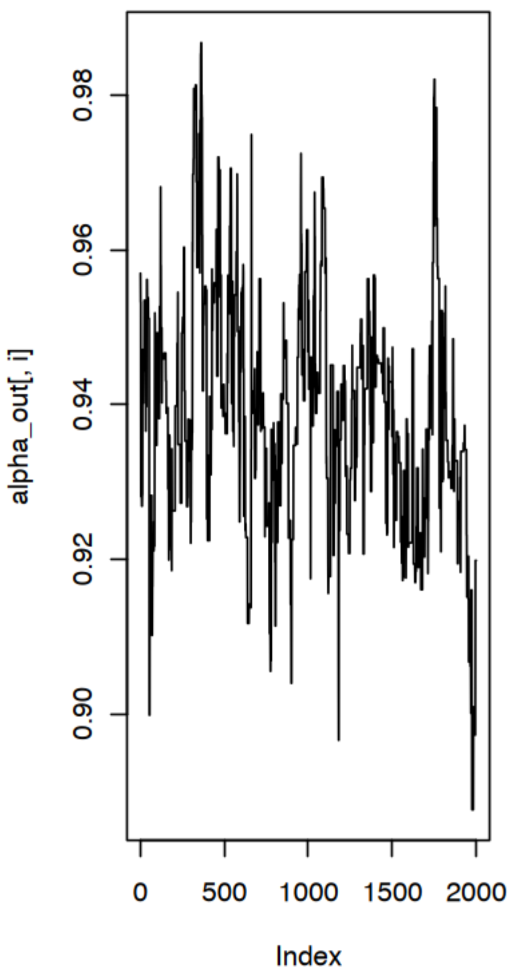

sigma_ 3 , 1
sigma_ 3 , 2
sigma_ 3 , 3
sigma_ 3 , 4
sigma2_out[, i, j]
Index

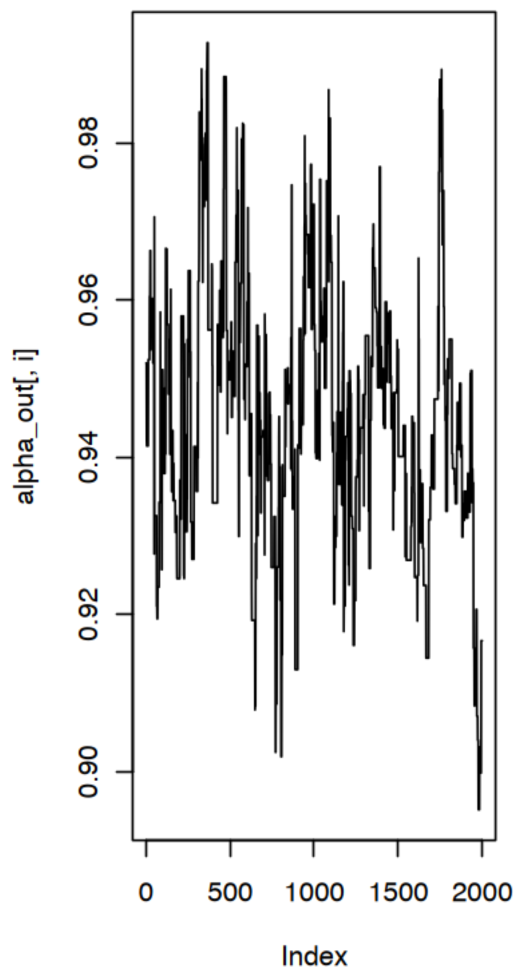


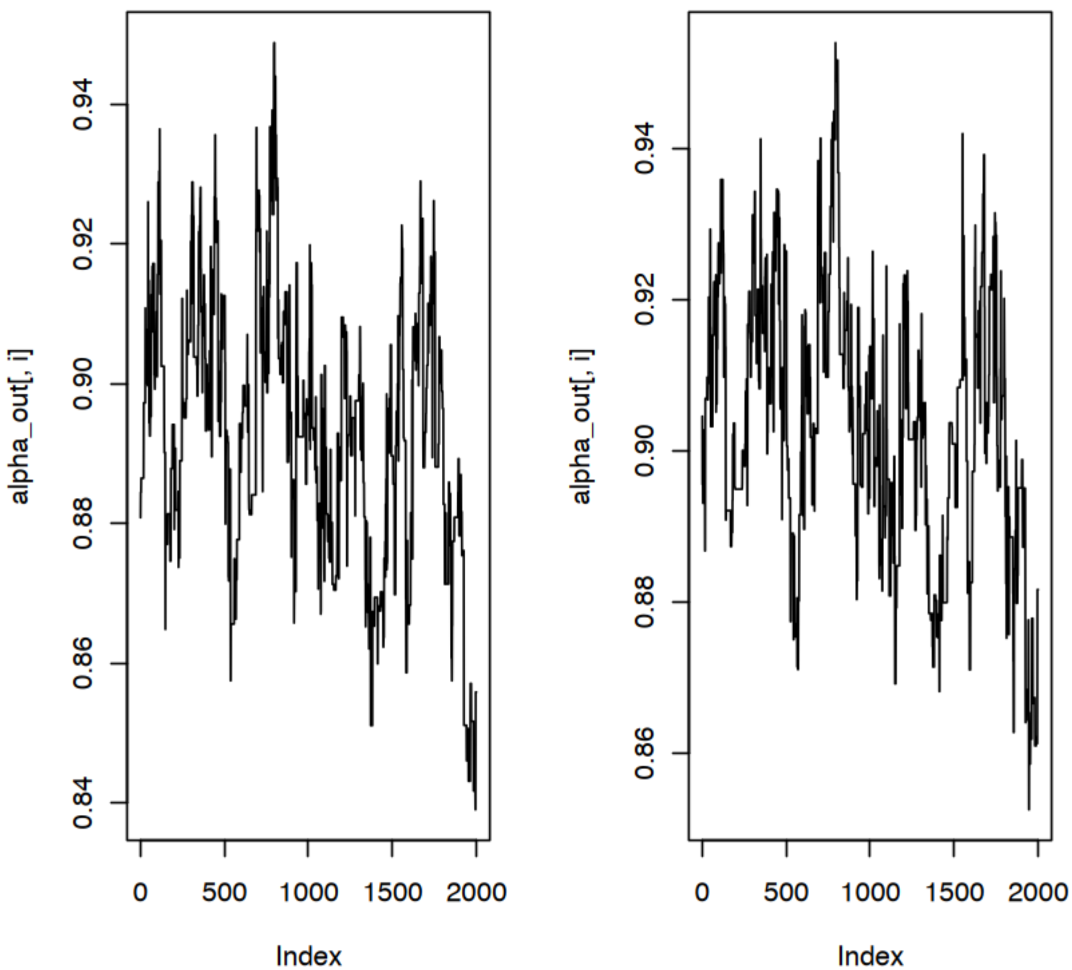
[65]:
```
res_out <- model_v2(dataset = data_app, prior_sigma2_nu = 6, prior_sigma2_psi =␣
↪diag(1, 4), iterations = 5000, adapt_batch = 50, adapt_a = 2000, adapt_b =␣
↪1000, adapt_alpha = 0.234, adapt_stop = 4000)
```

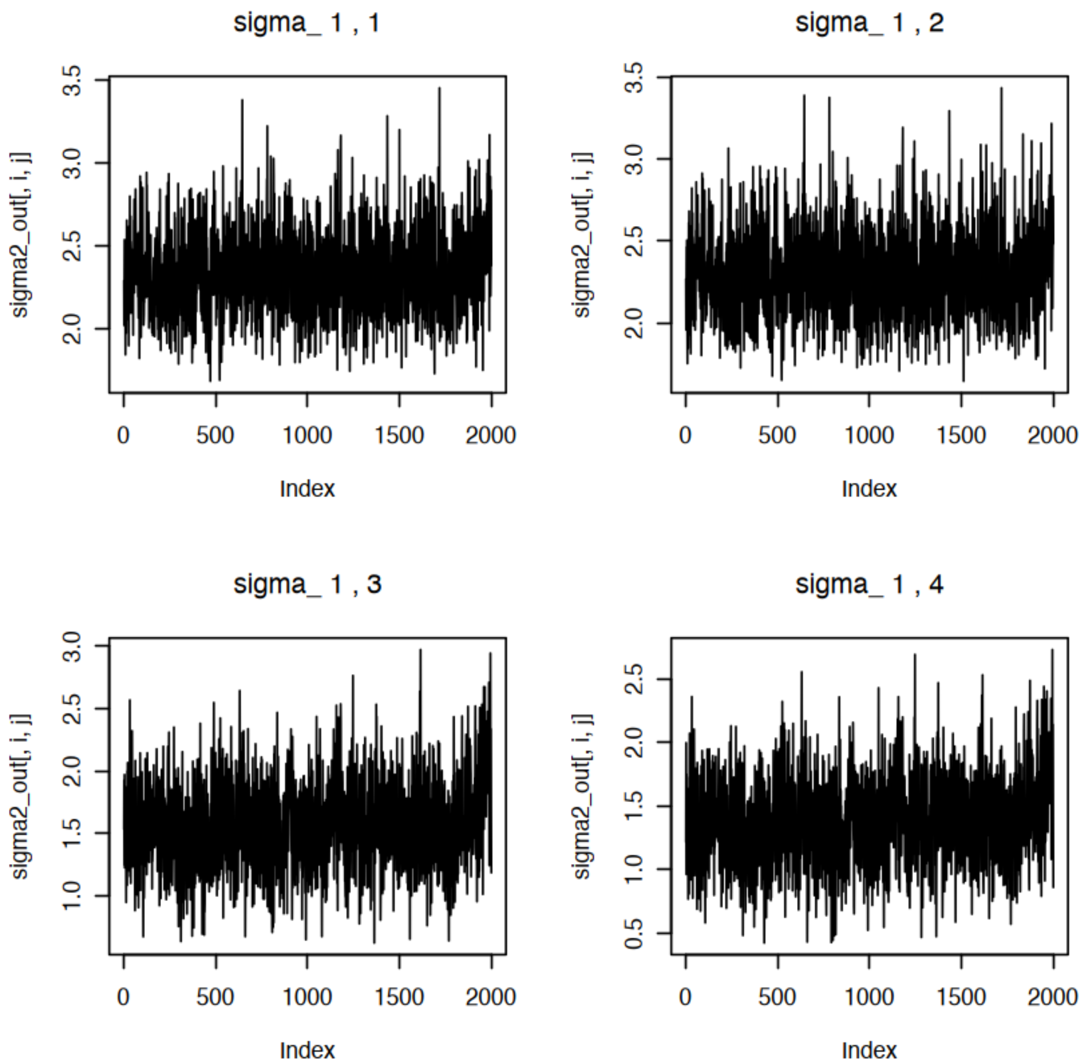
[66]:
```
sigma2_out <- res_out$sigma2_out[-c(1:3000), , ]
alpha_out <- res_out$alpha_out[-c(1:3000), ]
par(mfrow = c(1, 2))
for (i in 1:p)
{
  plot(alpha_out[, i], type = "l")
}
par(mfrow = c(1, 1))
```

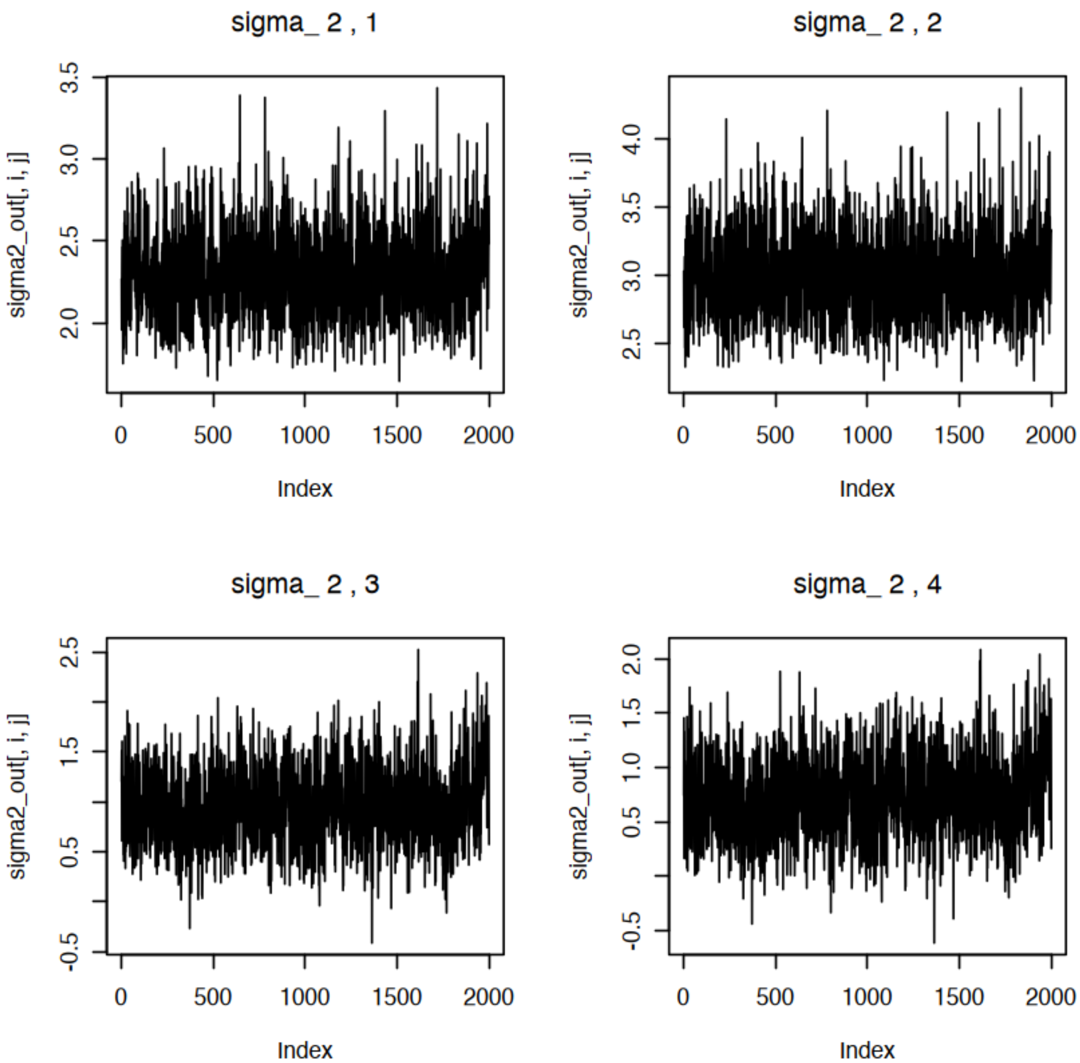
```r
par(mfrow = c(2, 2))
for (i in 1:p)
{
  for (j in 1:p)
  {
    plot(sigma2_out[, i, j], type = "l", main = paste("sigma_", i, ",", j))
  }
}
par(mfrow = c(1, 1))
```

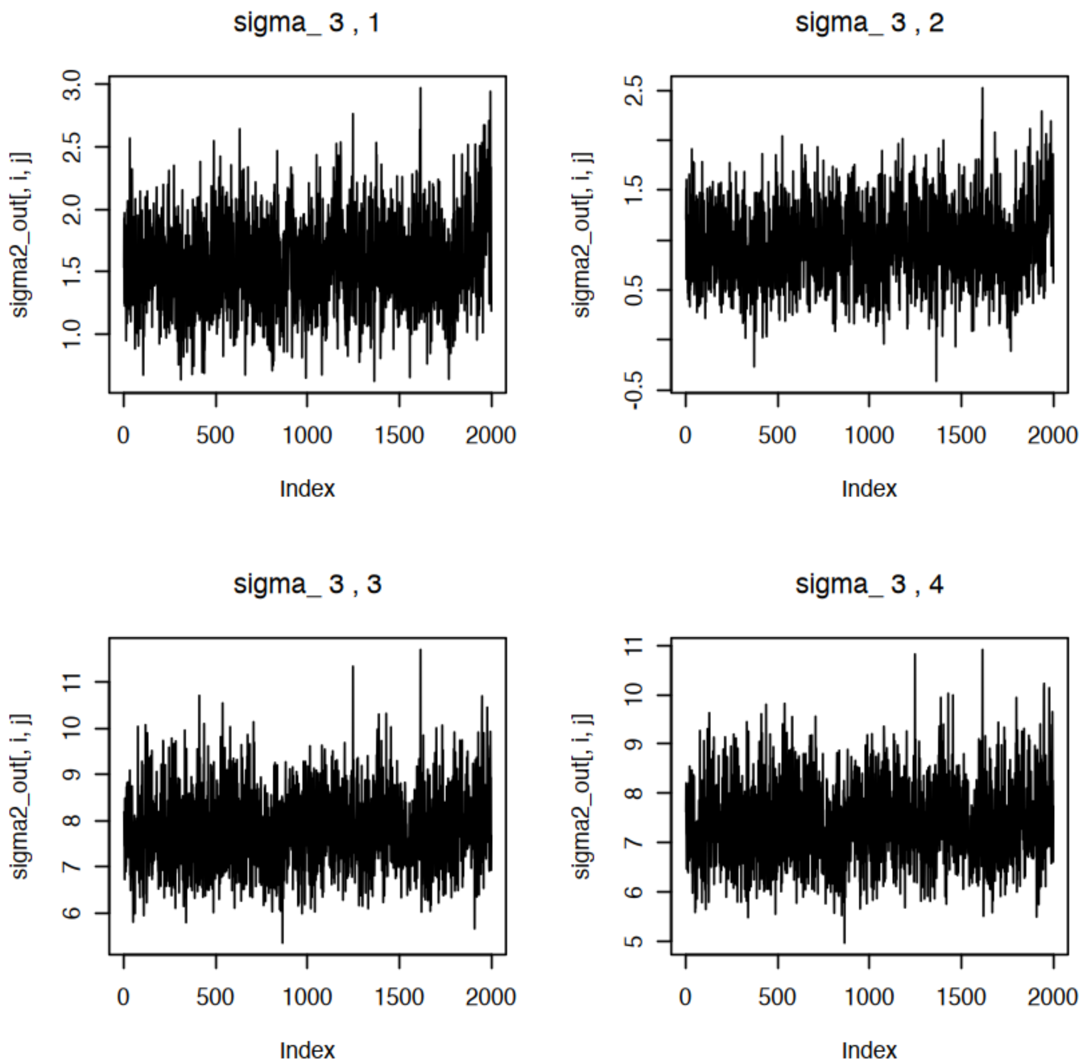


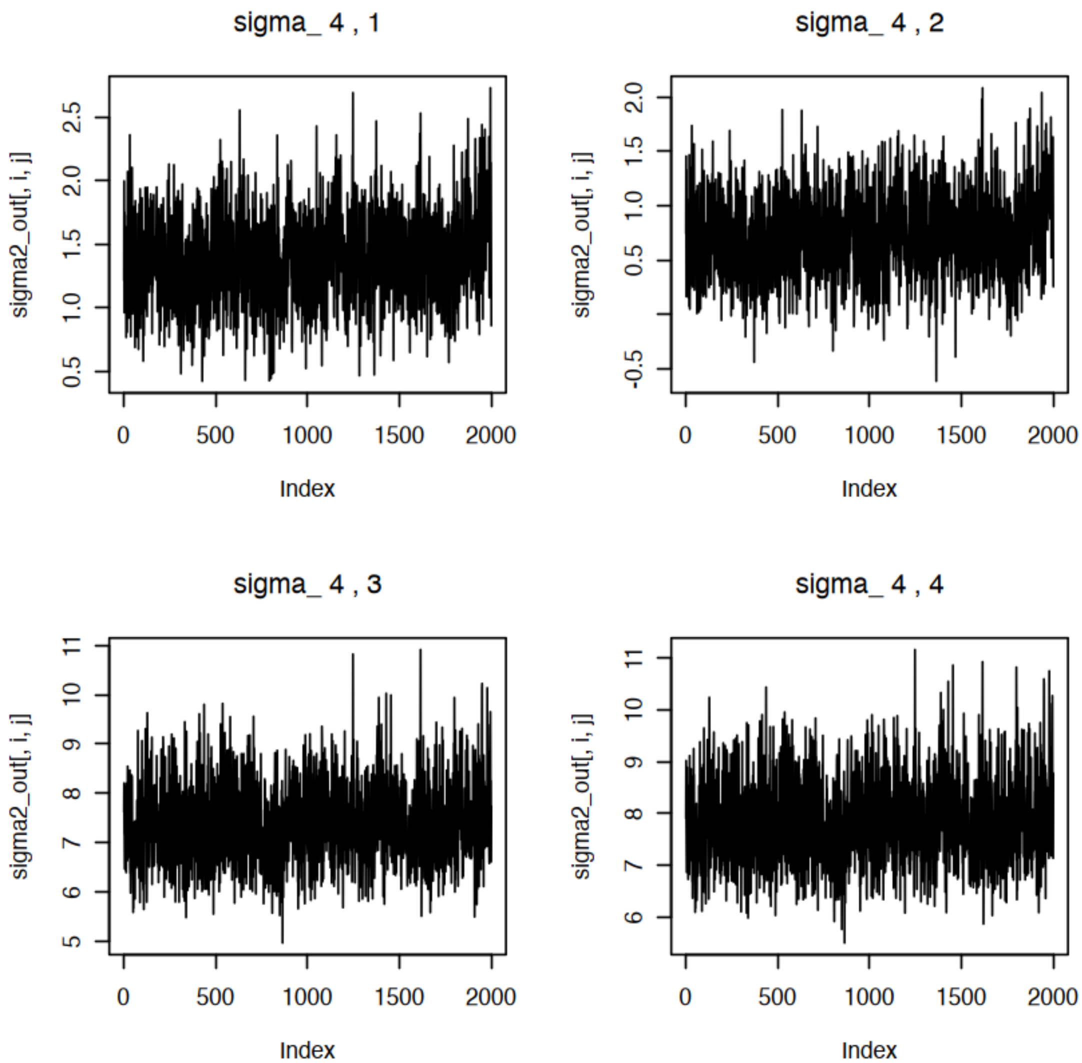



alpha_out[, i]
0.84
0.86
0.88
0.90
0.92
0.94
0
500
1000
1500
2000
Index
alpha_out[, i]
0.86
0.88
0.90
0.92
0.94
0
500
1000
1500
2000
Index


sigma_ 1 , 1
sigma2_out[, i, j]
2.0 2.5 3.0 3.5
0 500 1000 1500 2000
Index
sigma_ 1 , 2
sigma2_out[, i, j]
2.0 2.5 3.0 3.5
0 500 1000 1500 2000
Index
sigma_ 1 , 3
sigma2_out[, i, j]
1.0 1.5 2.0 2.5 3.0
0 500 1000 1500 2000
Index
sigma_ 1 , 4
sigma2_out[, i, j]
0.5 1.0 1.5 2.0 2.5
0 500 1000 1500 2000
Index


sigma_ 2 , 1
sigma_ 2 , 2
sigma_ 2 , 3
sigma_ 2 , 4
sigma2_out[, i, j]
Index
0
500
1000
1500
2000
2.0
2.5
3.0
3.5
4.0
-0.5
0.5
1.0
1.5


sigma_ 3 , 1
sigma_ 3 , 2
sigma_ 3 , 3
sigma_ 3 , 4
sigma2_out[, i, j]
Index



[67]:

```
cor_out = sigma2_out
for (i in 1:dim(sigma2_out)[1])
{
  cor_out[i, , ] <- cov2cor(sigma2_out[i, , ])
}

mean_cor <- apply(cor_out, c(2, 3), mean)
mean_cor
data_mean_cor <- data.frame(val = c(mean_cor), x = rep(1:4, 4), y = rep(1:4, each = 4))
data_mean_cor %>% ggplot(aes(x = x, y = y)) +
  geom_tile(aes(fill = val)) +
  scale_y_reverse()
```

A matrix: 4 x 4 of type dbl

1.0000000	0.8686719	0.3678173	0.3226314
0.8686719	1.0000000	0.1988998	0.1502542
0.3678173	0.1988998	1.0000000	0.9443162
0.3226314	0.1502542	0.9443162	1.0000000