Step-by-step guide to estimating the multivariate CCC-GARCH(1,1) SSM

Zayd Omar

2024-06-08

Introduction

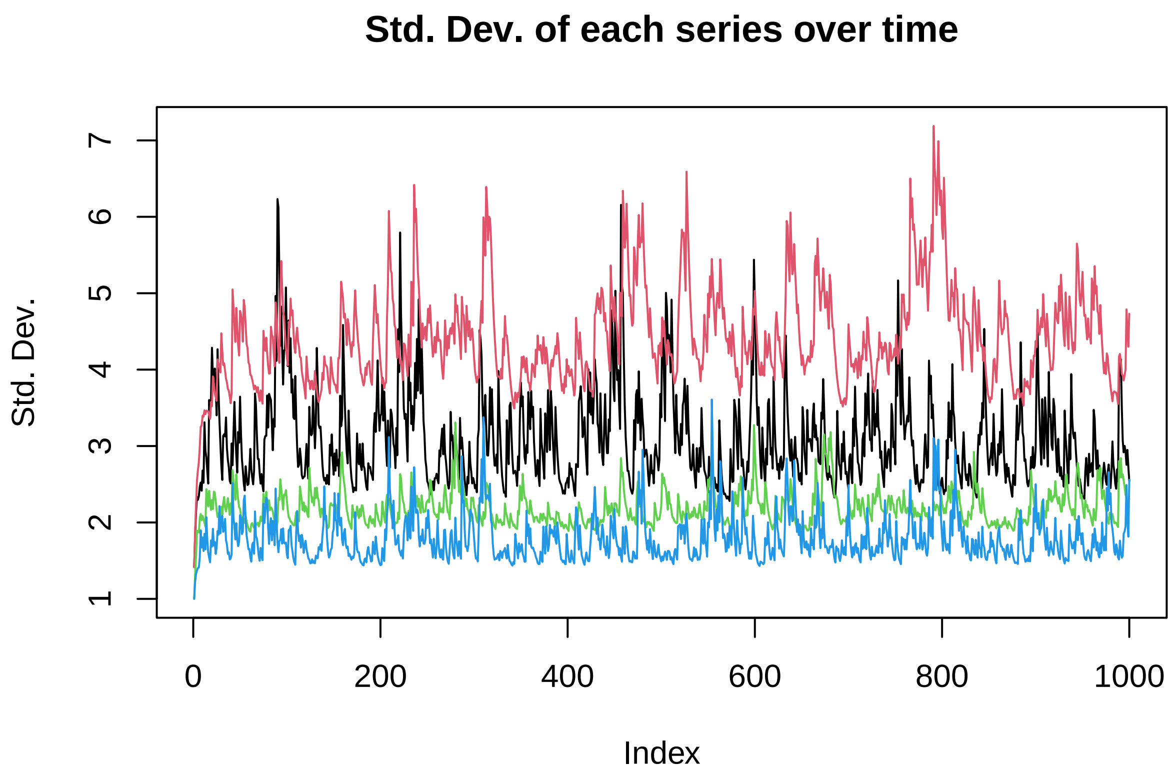
In this R-markdown file we provide a guide to implement the model proposed by Omar et al. in the work "*A Bayesian Non-Stationary Heteroskedastic Time Series Model for Multivariate Critical Care Data*". In particular we will generate data and estimate the 4-D GARCH(1,1) state-space model.

Step-1 Data Generation

We generate the data based on the parameters used in the simulation study. The plots below show the dynamic nature of the variance and the non stationarity of the series.

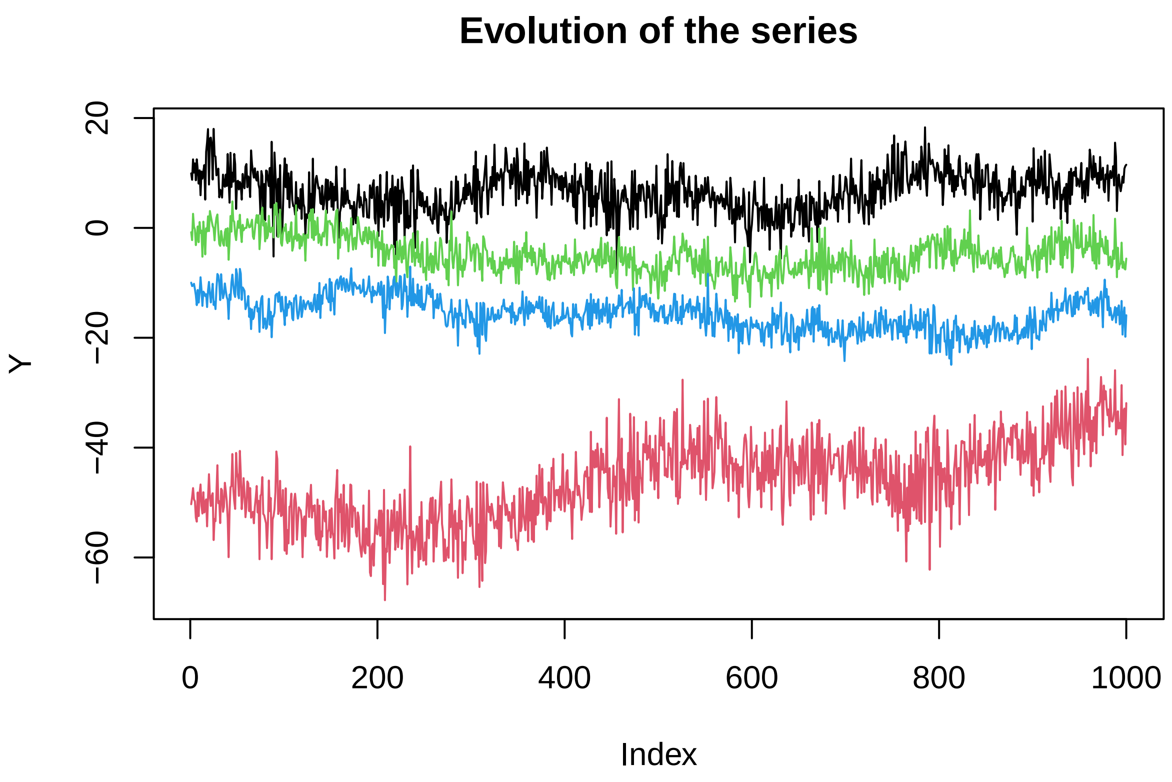
```r
library(mvtnorm)
set.seed(20240606)
n = 1000
p = 4
FF = diag(1, p)  # Observation level matrix
GG = diag(1, p)  # State evolution matrix
m0 = rep(0, p)
C0 = diag(10, p)


model_par = rbind(c(2, 0.2, 0.6, 0.1), c(2, 0.1, 0.8, 0.1), c(1,
    0.1, 0.7, 0.1), c(1, 0.2, 0.5, 0.1))




u1 = c(runif(p, -1, 1))
u2 = c(0, runif(p - 1, 0, 1))
u3 = c(0, 0, runif(p - 2, 0, 1))
u4 = c(0, 0, 0, runif(p - 3, 0, 1))

u1 = u1/norm(u1, "2")
u2 = u2/norm(u2, "2")
u3 = u3/norm(u3, "2")
u4 = u4/norm(u4, "2")




U = rbind(u1, u2, u3, u4)
V = tcrossprod(U)
```

```r
v = matrix(rmvnorm(1, rep(0, p), diag(1, p)), nc = p, nr = n,
    byr = T)
z.t = matrix(model_par[, 1], nrow = n, nc = p, byr = T)
ST = list(diag(c(model_par[1:p, 1])))


for (i in 2:n) {
    z.t[i, ] = model_par[1:p, 1] + model_par[1:p, 2] * v[i -
        1, ]^2 + model_par[1:p, 3] * z.t[i - 1, ]
    ST[[i]] = diag(sqrt(z.t[i - 1, ])) %*% V %*% diag(sqrt(z.t[i -
        1, ]))
    v[i, ] = t(chol(ST[[i]])) %*% rnorm(p, 0, 1)

}

par(mfrow = c(1, 1))
plot(sqrt(z.t[, 1]), type = "l", ylim = c(min(sqrt(z.t)), max(sqrt(z.t))),
    main = "Std. Dev. of each series over time", ylab = "Std. Dev.")
lines(sqrt(z.t[, 2]), type = "l", col = 2)
lines(sqrt(z.t[, 3]), type = "l", col = 3)
lines(sqrt(z.t[, 4]), type = "l", col = 4)
```



```r
state = matrix(c(10.6, -50.5, 0, -10), nc = p, nr = n + 1, byrow = T)
Y = matrix(0, nr = n, nc = p)
for (i in 1:n) {
    state[i + 1, ] = t(GG %*% as.vector(state[i, ]) + as.vector(chol(diag(model_par[1:p,
        4])) %*% rnorm(p)))
    Y[i, ] = FF %*% as.vector(state[i, ]) + v[i, ]
}
```

```r
plot(Y[, 1], type = "l", ylim = c(min(Y, na.rm = T), max(Y, na.rm = T)),
    main = "Evolution of the series", ylab = "Y")
lines(Y[, 2], type = "l", col = 2)
lines(Y[, 3], type = "l", col = 3)
lines(Y[, 4], type = "l", col = 4)
```



Step-2 Bayesian Estimation of the model

Next, we estimate the model. We need to assign initial values of for the MCMC to start. We also need to call the *Rcpp* functions for the FFBS and Kalman Filter. Since they are written in *Rcpp* we will need to load the library *Rcpp* and compile the source code using the function *sourcecpp*

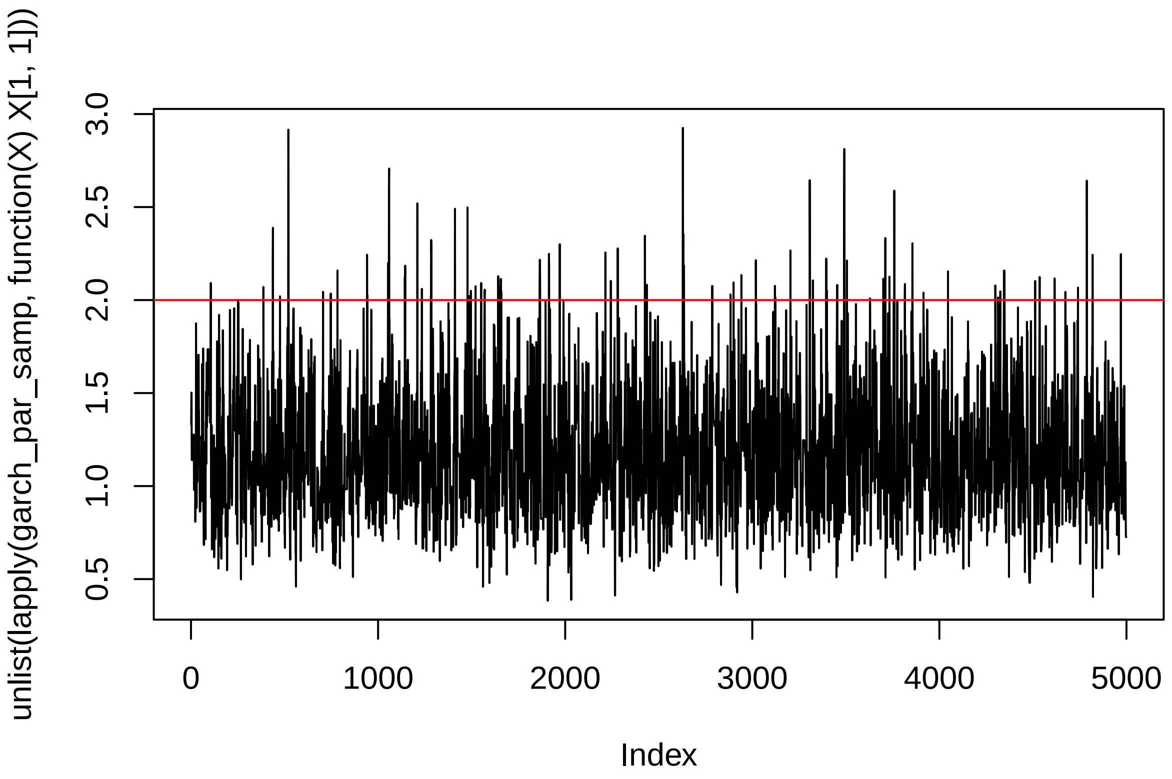
```r
## MH-MCMC estimation

library("MCMCpack", "Rcpp", "truncnorm")

Rcpp::sourceCpp("./Functions/MvtKalFiltGarchC.cpp")
Rcpp::sourceCpp("./Functions/FFBS.cpp")
Rcpp::sourceCpp("./Functions/MvtLkhd.cpp")
Rcpp::sourceCpp("./Functions/diwishart.cpp")


L1 = chol(diag(c(0.5, 0.01, 0.01)))  ## Step size control for the random walk M-H step
S.t = diag(1, dim(GG)[1])
n.t = 10

burn = 50000
thin = 50
```

```
sims = (5000) * thin + burn

## Store posterior samples
S.t = diag(1, dim(GG)[1])
n.t = 10


S_t.samp = list()

predicted = list()
sampled_theta = list()


garch_par_samp = list(matrix(rep(c(1, 0, 0), 4), nr = 4, byr = T))  ## initializing the garch_parameter
old_param = garch_par_samp[[1]]

W.samp = list()
W.old = diag(1, dim(GG)[2])


cor.samp = list()
if (ncol(Y) == 2) {
    cor.old = 0
    V.old = matrix(0)
    cor.samp[[1]] = cor.old
} else {
    cor.old = diag(1, ncol(Y))
    V.old = tcrossprod(cor.old/apply(cor.old, 1, norm, "2"))
    cor.samp[[1]] = cor.old
}




out = MVTKalFiltGarch(Y = Y, GG = GG, FF = FF, W = W.old, m0 = m0,
    C0 = C0, garch = old_param, cor = V.old)
theta = do.call(rbind, lapply(ffbsC(Y, out)$theta, t))




## Start MCMC
for (it in 1:sims) {
    if (it%%100 == 0) {
        print(it)
    }
    if (it%%100 == 0) {
        print(old_param)
    }
    if (it%%100 == 0) {
        print(cor.old)
    }
    if (it%%100 == 0) {
        print(W.old)
    }

    n = nrow(Y)

    for (j in 1:ncol(Y)) {
        prop = t(c(old_param[j, 1], old_param[j, 2], old_param[j,
            3]) + L1 %*% rnorm(3, 0, 1))
        while (sum(prop[2], prop[3]) > 1 || any(prop < 0)) {
            prop = t(c(old_param[j, 1], old_param[j, 2], old_param[j,
                3]) + L1 %*% rnorm(3, 0, 1))
        }

        new_param = old_param
        new_param[j, 1:3] = c(prop[1], prop[2], prop[3])
        rho = MVT_Lkhd(par = new_param, cor = V.old, Y = Y, W = W.old,
            theta = theta, FF = FF, GG = GG) - MVT_Lkhd(par = old_param,
            cor = V.old, Y = Y, W = W.old, theta = theta, FF = FF,
            GG = GG) + sum(dcauchy(prop, log = T)) - sum(dcauchy(old_param[j,
            ], log = T))

        if (log(runif(1)) < rho) {
            old_param[j, 1:3] = c(prop[1], prop[2], prop[3])
            if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin >
                0) {
                garch_par_samp[[(it - burn - 1)/thin]] = old_param
            }
        } else {
            if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin >
                1) {
                garch_par_samp[[(it - burn - 1)/thin]] = old_param
            }
        }
    }

    if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin > 0) {
        garch_par_samp[[(it - burn - 1)/thin]] = old_param
    }

    mu_theta = t(GG %*% t(theta[-n, ]))
    SS.t = crossprod(theta[-1, ] - mu_theta) + S.t
    df.t = n + n.t
    W.old = riwish(df.t, SS.t)
    if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin > 0) {
        W.samp[[round((it - burn - 1)/thin)]] = W.old
    }

    if (dim(Y)[2] > 2) {

        # Use this reparametrization only when the
        # dimension of Y is greater than 2

        j = sample(p - 1, 1)
        k = sample(j:p, 1)
        cor.new = cor.old
        cor.new[j, k] = round(cor.old[j, k] + rnorm(1, 0, 0.1),
            5)
        cor.new[j, j:p] = cor.new[j, j:p]/norm(cor.new[j, j:p],
            "2")

        d1 = d2 = 0
        if (j == k) {
            # positivity constraint
            cor.new[j, k] = rtruncnorm(1, a = 0, mean = cor.old[j,
                k], sd = 0.1)
            cor.new[j, j:p] = cor.new[j, j:p]/norm(cor.new[j,
                j:p], "2")

            d1 = dtruncnorm(cor.new[j, k], a = 0, mean = cor.old[j,
                k], sd = 0.1)
            d2 = dtruncnorm(cor.old[j, k], a = 0, mean = cor.new[j,
                k], sd = 0.1)
        }

        V.new = tcrossprod(round(cor.new, 5))
        rho = MVT_Lkhd(par = old_param, cor = V.new, Y = Y, W = W.old,
            theta = theta, FF = FF, GG = GG) + d2 - MVT_Lkhd(par = old_param,
            cor = V.old, Y = Y, W = W.old, theta = theta, FF = FF,
            GG = GG) - d1

        if (log(runif(1)) < rho) {
            cor.old = cor.new
            V.old = V.new
            if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin >
                0) {
                cor.samp[[(it - burn - 1)/thin]] = cor.old
            }
        } else {
            if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin >
                1) {
                cor.samp[[(it - burn - 1)/thin]] = cor.old
            }
        }
    } else {
        # When we have the 2-d case the correlation
        # parameter, V, is pretty simple
        V.new = matrix(V.old + runif(1, -0.1, 0.1))
        while (V.new > 1 || V.new < -1) {
            V.new = matrix(V.old + runif(1, -0.1, 0.1))
        }
        rho = MVT_Lkhd(par = old_param, cor = V.new, Y = Y, W = W.old,
            theta = theta, FF = FF, GG = GG) - MVT_Lkhd(par = old_param,
            cor = V.old, Y = Y, W = W.old, theta = theta, FF = FF,
            GG = GG)

        if (log(runif(1)) < rho) {
            cor.old = V.new
            V.old = V.new
            if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin >
                0) {
                cor.samp[[(it - burn - 1)/thin]] = cor.old
            }
        } else {
            if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin >
                1) {
                cor.samp[[(it - burn - 1)/thin]] = cor.old
            }
        }

    }

    out = MVTKalFiltGarch(Y = Y, GG = GG, FF = FF, W = W.old,
        m0 = m0, C0 = C0, garch = old_param, cor = V.old)
    theta = do.call(rbind, lapply(ffbsC(Y, out)$theta, t))

    if ((it - burn - 1)%%thin == 0 & (it - burn - 1)/thin > 0) {
        S_t.samp[[(it - burn - 1)/thin]] = out$S_t
        predicted[[(it - burn - 1)/thin]] = do.call(rbind, lapply(out$y_t_1,
            t))
        sampled_theta[[(it - burn - 1)/thin]] = theta
    }


}
```

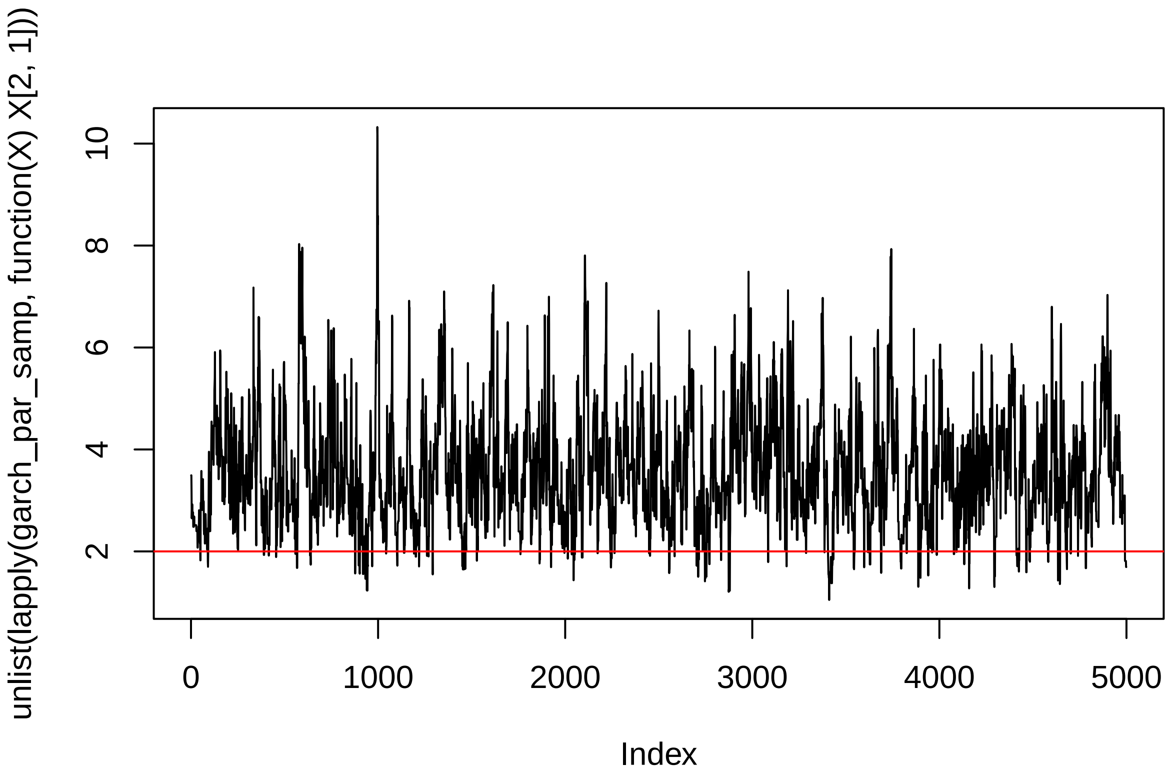
Step-3 Posterior Summary

The following can be used to evaluate the samples from the posterior distribution. We can use them to create the credible intervals for the state vectors, obtain estimates for time varying variance, assess convergence of the GARCH parameters and obtain posterior point estimates.

```
## Summary Posterior






## a0 param
par(mfrow = c(1, 1))
plot(unlist(lapply(garch_par_samp, function(X) X[1, 1])), type = "l")
abline(h = model_par[1, 1], col = "red")
```

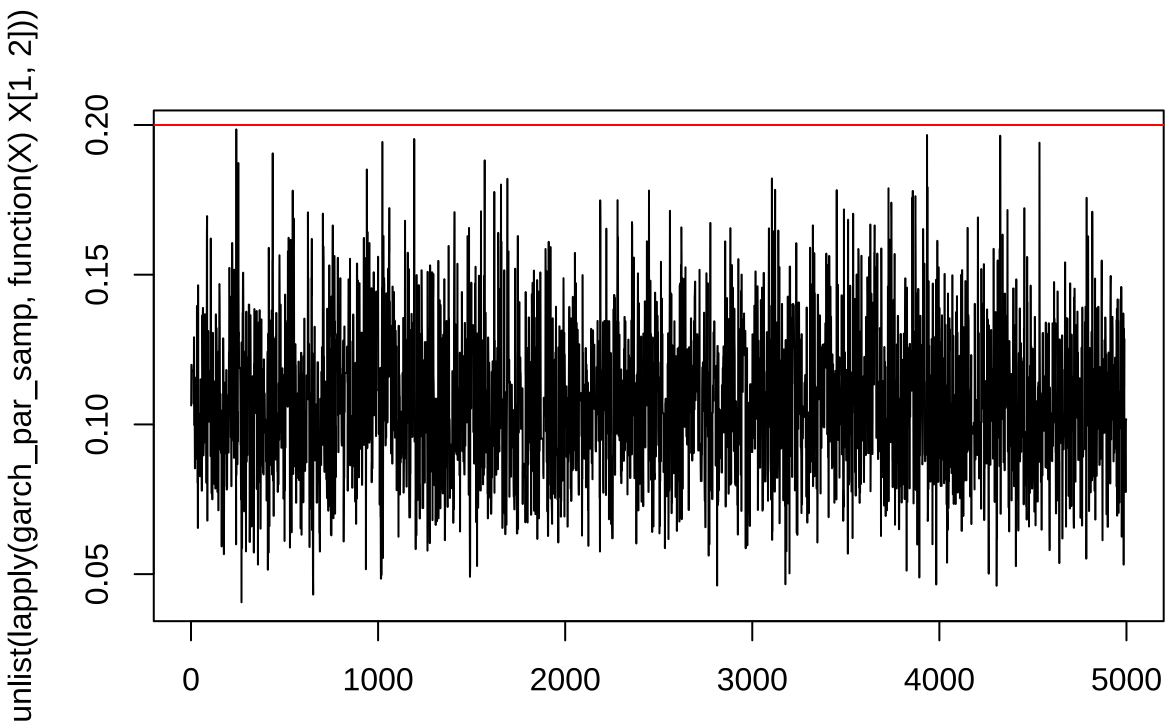


```
plot(unlist(lapply(garch_par_samp, function(X) X[2, 1])), type = "l")
abline(h = model_par[2, 1], col = "red")
```

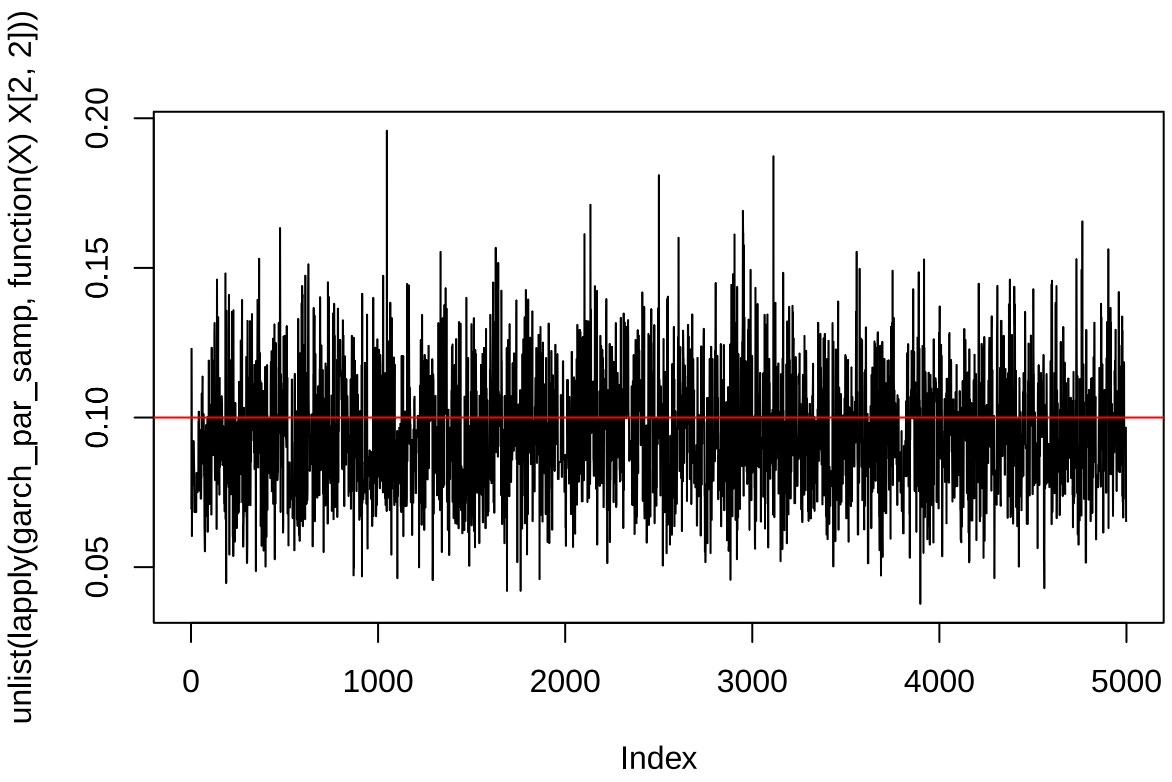


```
## a1 param
plot(unlist(lapply(garch_par_samp, function(X) X[1, 2])), type = "l")
```

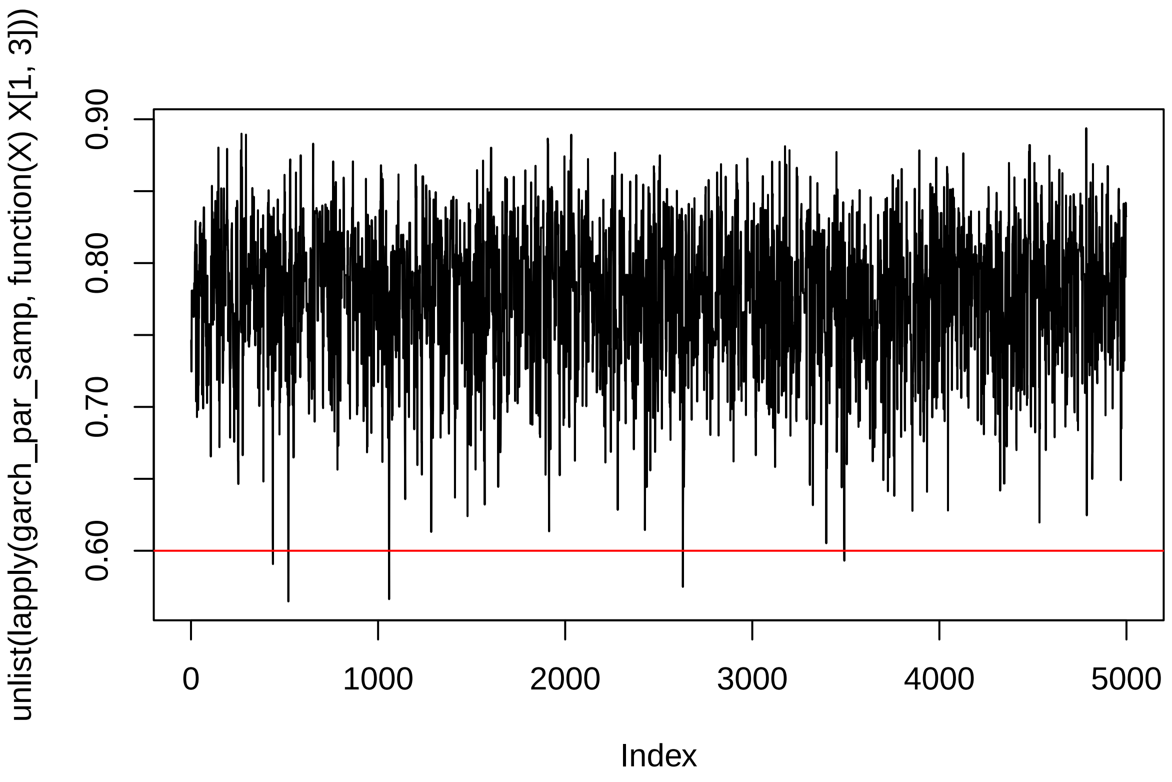
```
abline(h = model_par[1, 2], col = "red")
```



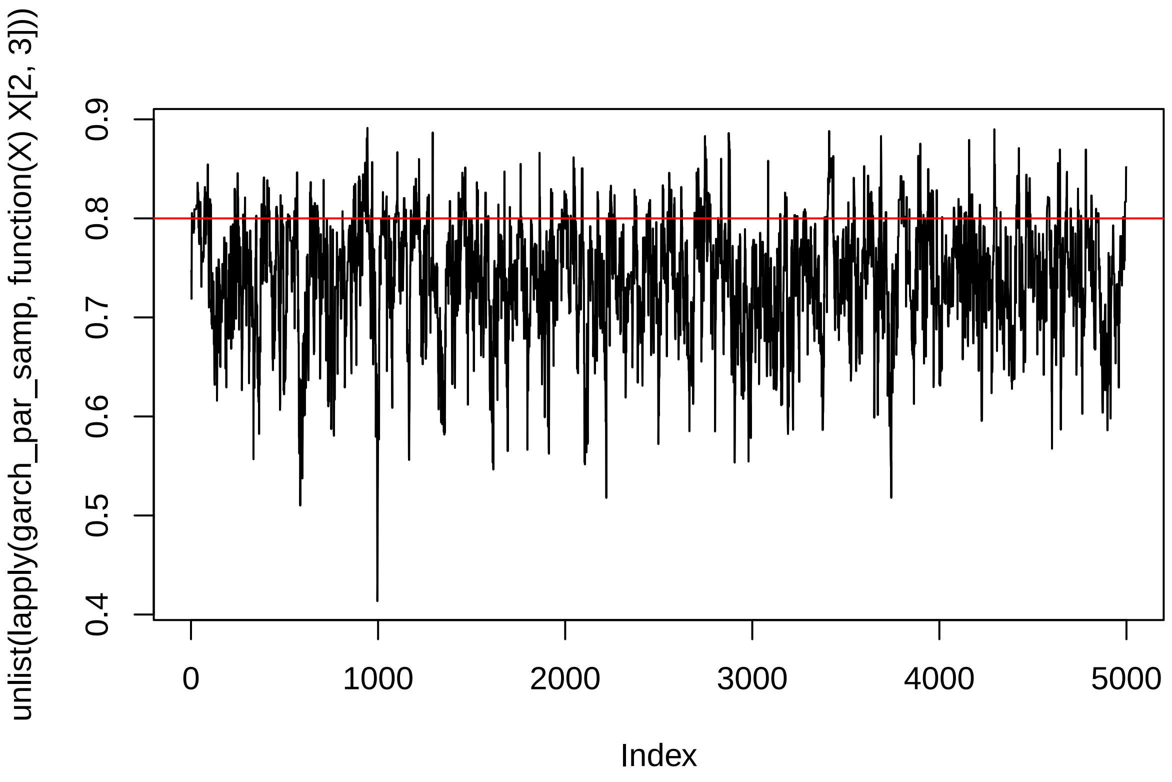
```
plot(unlist(lapply(garch_par_samp, function(X) X[2, 2])), type = "l")
abline(h = model_par[2, 2], col = "red")
```



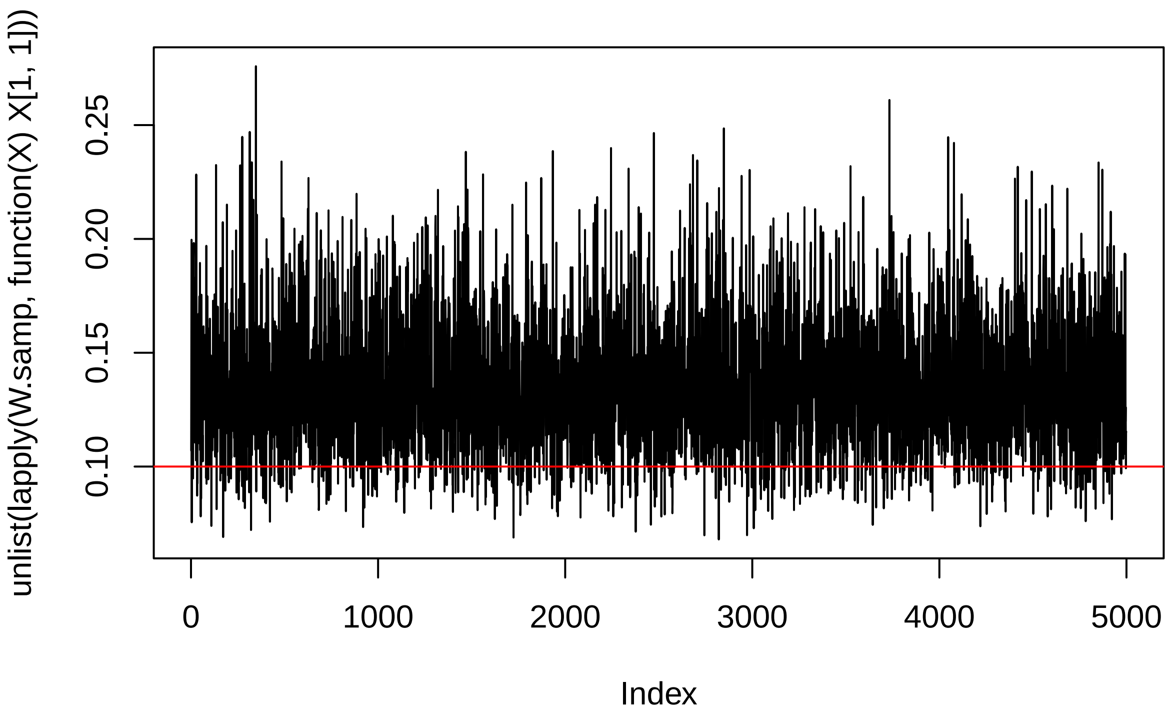
```r
## b1 param
plot(unlist(lapply(garch_par_samp, function(X) X[1, 3])), type = "l")
abline(h = model_par[1, 3], col = "red")
```



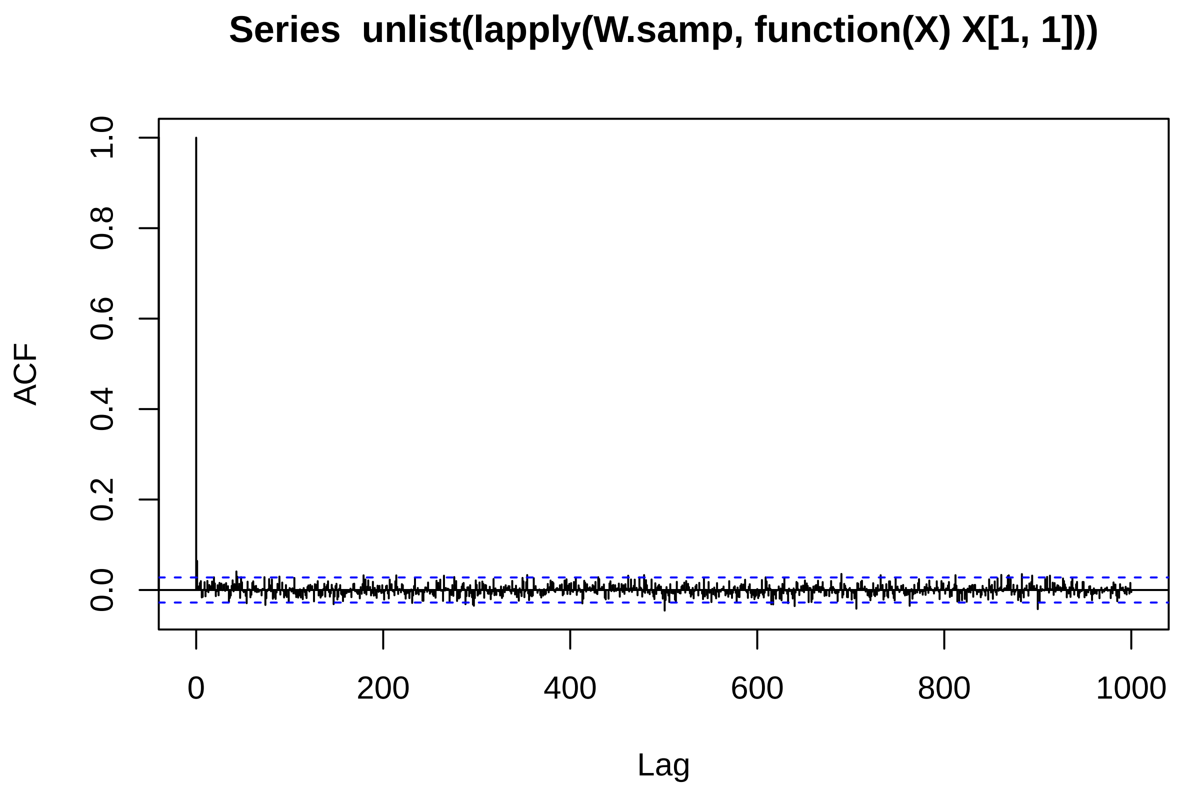
```r
plot(unlist(lapply(garch_par_samp, function(X) X[2, 3])), type = "l")
abline(h = model_par[2, 3], col = "red")
```



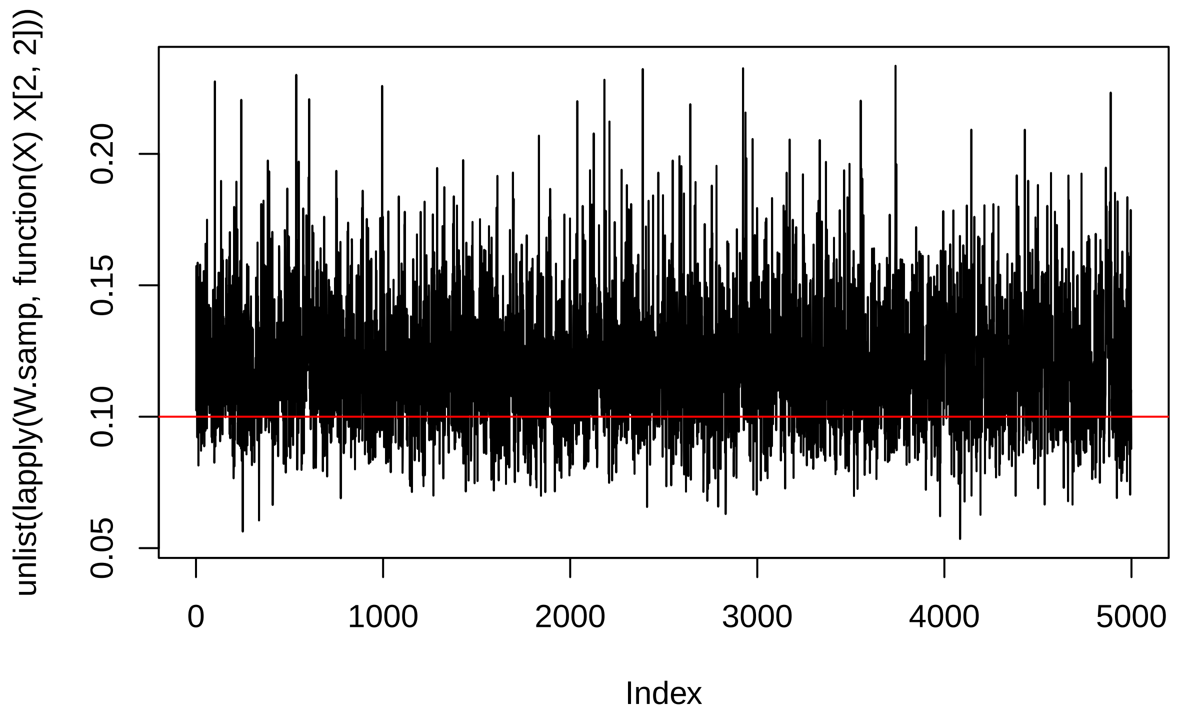
```r
## Plotting W
W.post = Reduce("+", W.samp)/length(W.samp)

plot(unlist(lapply(W.samp, function(X) X[1, 1])), type = "l")
abline(h = model_par[1, 4], col = "red")
```



```r
acf(unlist(lapply(W.samp, function(X) X[1, 1])), lag.max = 1000)
```

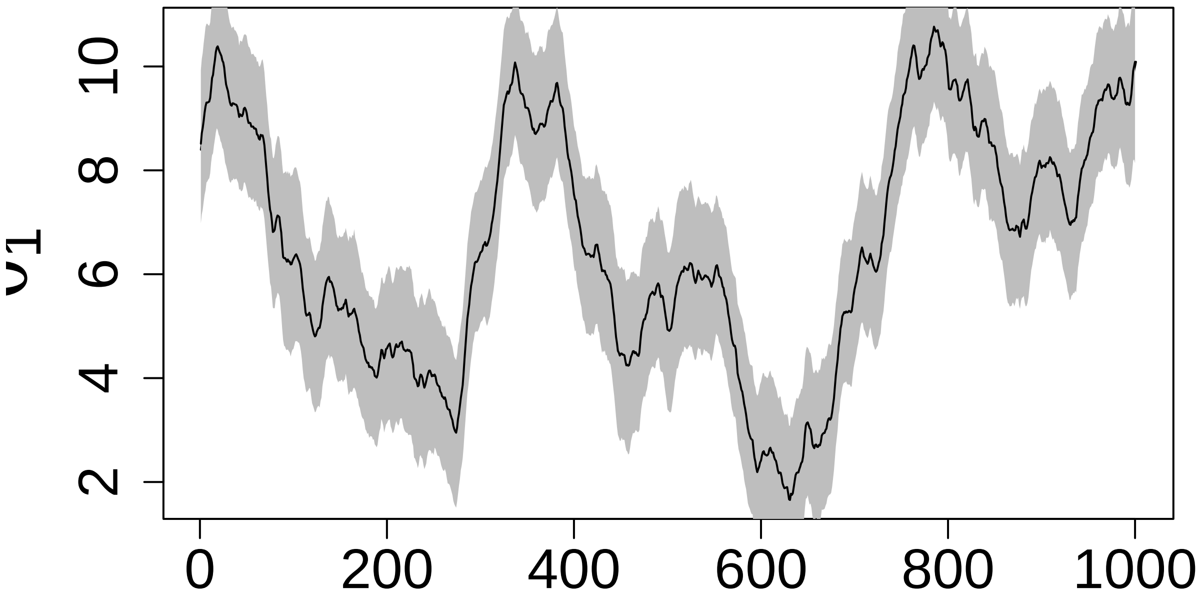


```r
plot(unlist(lapply(W.samp, function(X) X[2, 2])), type = "l")
abline(h = model_par[2, 4], col = "red")
```

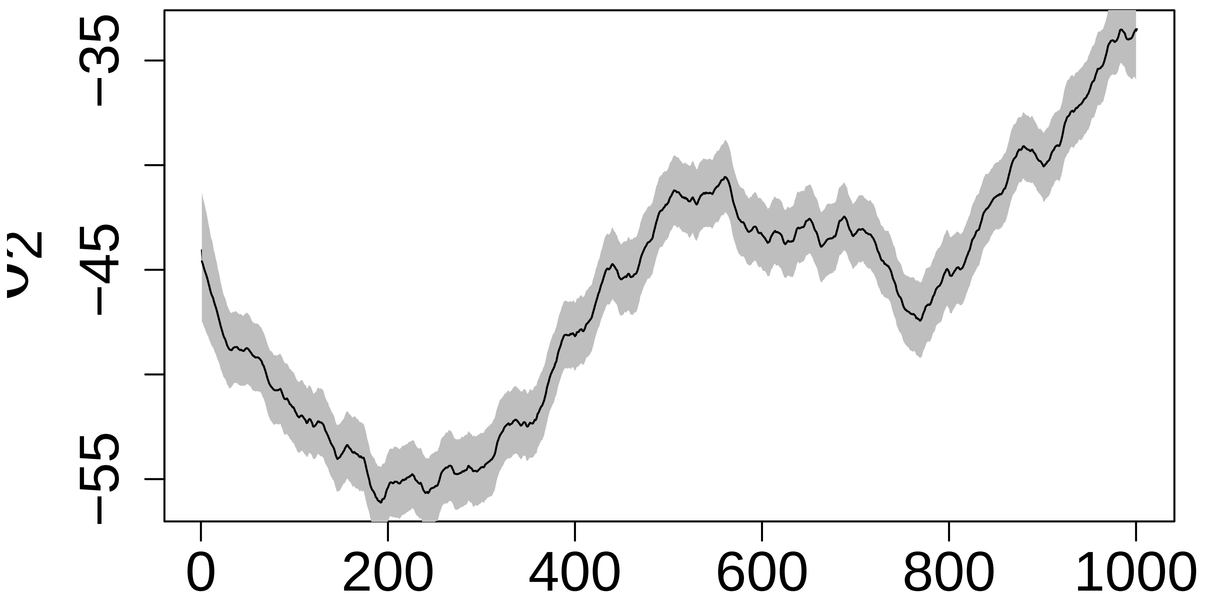


```r
## Plotting theta_t
Time = 1:n
```

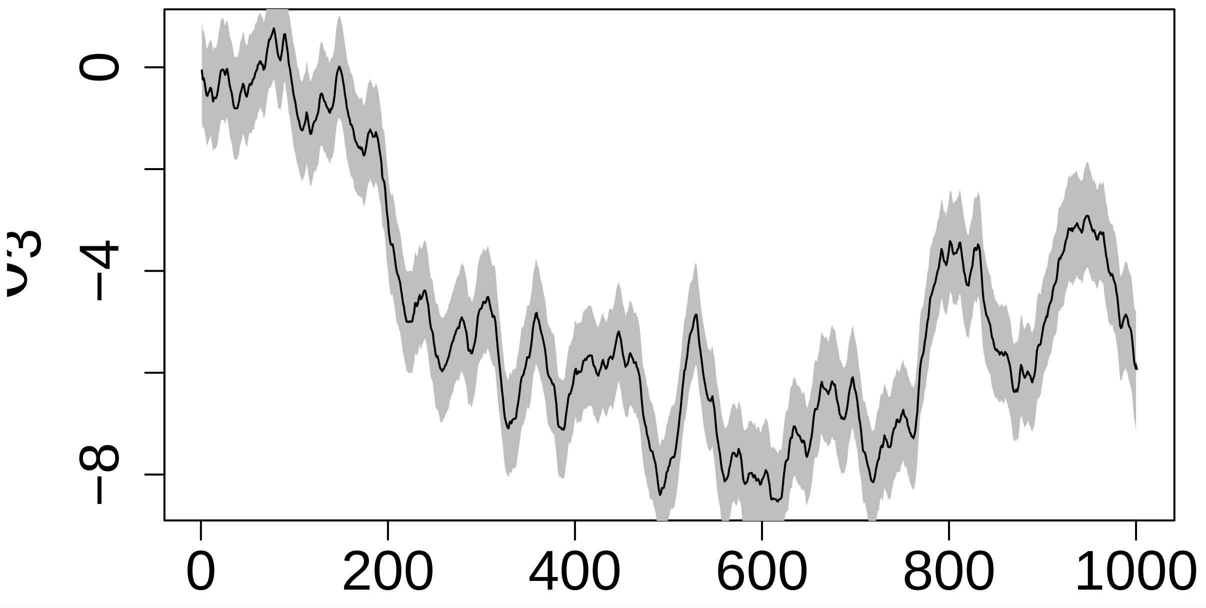
```
theta.post = Reduce("+", sampled_theta)/length(sampled_theta)

plot(theta.post[, 1], type = "l", ylab = expression(theta[1]),
    xlab = "", main = "", cex.lab = 2.5, cex.axis = 1.75)
LB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    1]))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]
UB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    1]))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(theta.post[-1, 1])
```



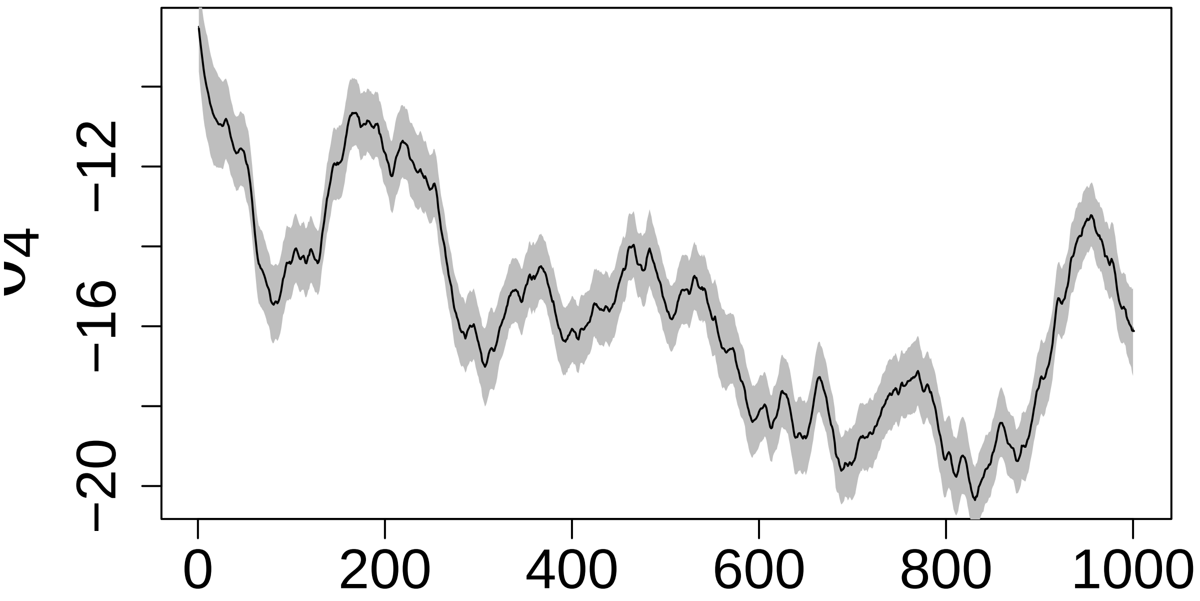
```
plot(theta.post[, 2], type = "l", ylab = expression(theta[2]),
    xlab = "", main = "", cex.lab = 2.5, cex.axis = 1.75)
LB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    2]))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]
UB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    2]))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(theta.post[-1, 2])
```



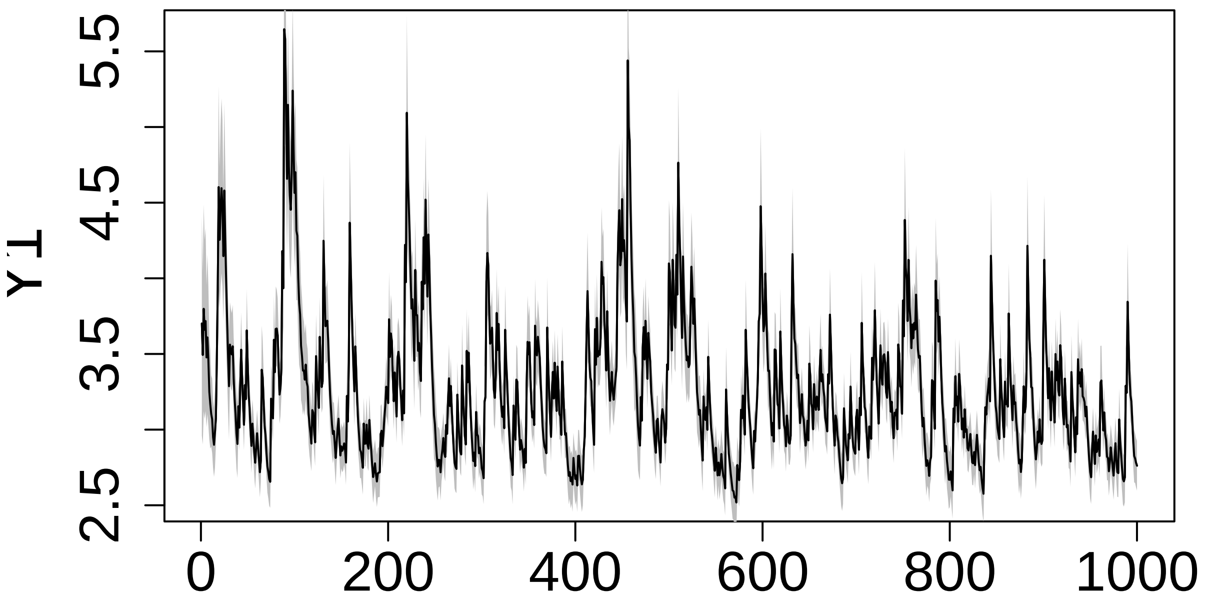
```
plot(theta.post[, 3], type = "l", ylab = expression(theta[3]),
    xlab = "", main = "", cex.lab = 2.5, cex.axis = 1.75)
LB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    3]))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]
UB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    3]))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(theta.post[-1, 3])
```



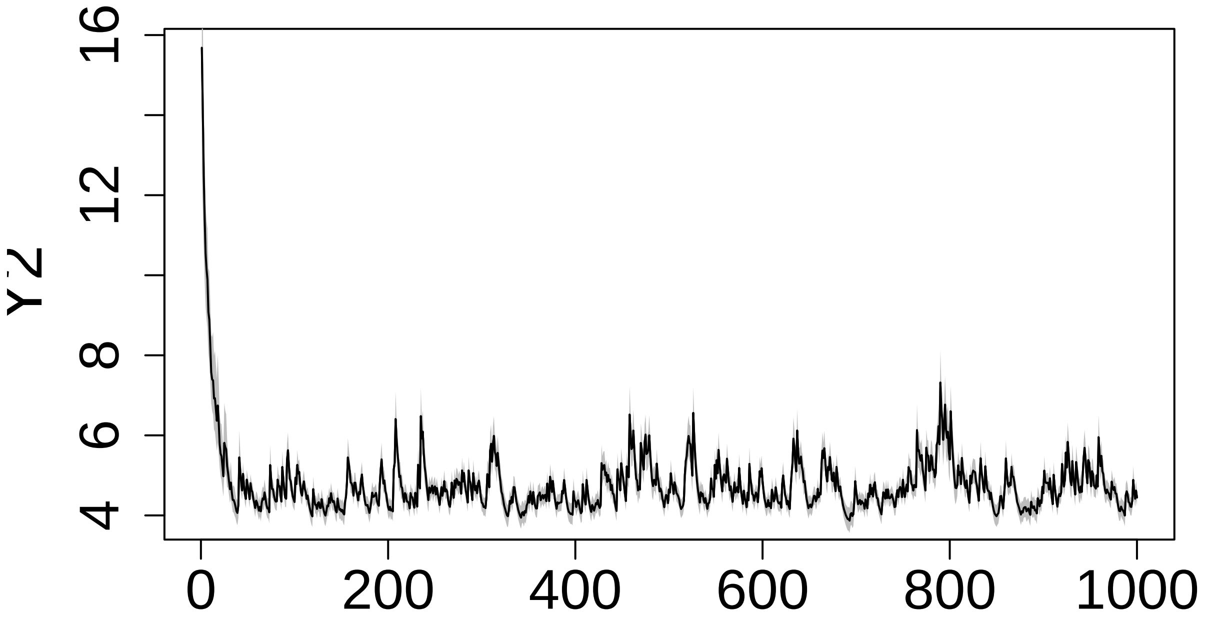
```
plot(theta.post[, 4], type = "l", ylab = expression(theta[4]),
    xlab = "", main = "", cex.lab = 2.5, cex.axis = 1.75)
LB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    4]))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]
UB = apply(t(do.call(rbind, lapply(sampled_theta, function(x) x[,
    4]))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(theta.post[-1, 4])
```



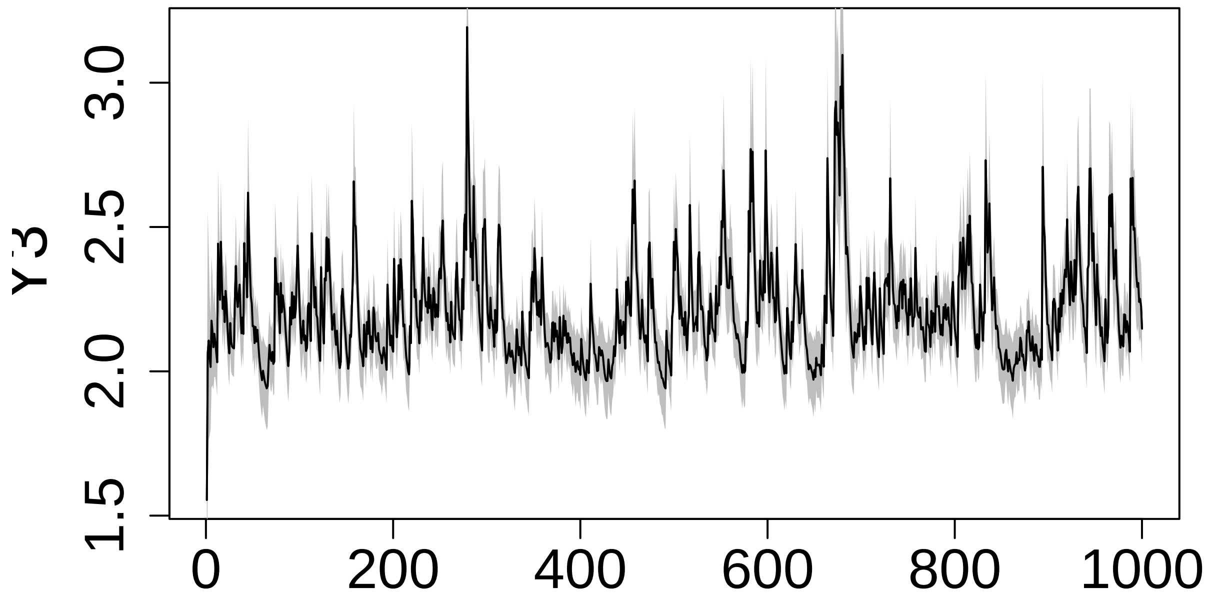
```
## Plotting S_t
S_t.post = Reduce("+", S_t.samp)/length(S_t.samp)

# S1
UB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    1])))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
LB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    1])))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]

plot(sqrt(S_t.post[-1, 1]), type = "l", ylab = "Y1", xlab = "",
    main = "", col = "black", cex.lab = 2, cex.axis = 1.75)
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(sqrt(S_t.post[-1, 1]))
```



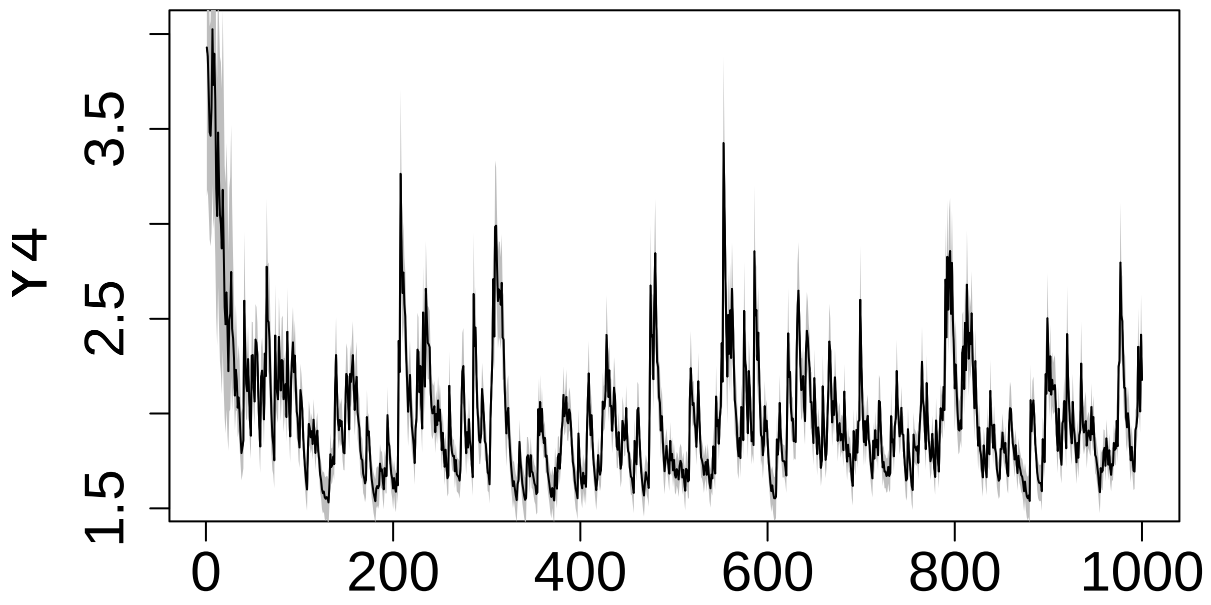
```r
# S2
UB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    2])))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
LB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    2])))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]

plot(sqrt(S_t.post[-1, 2]), type = "l", ylab = "Y2", main = "",
    xlab = "", col = "black", cex.lab = 2, cex.axis = 1.75)
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(sqrt(S_t.post[-1, 2]))
```



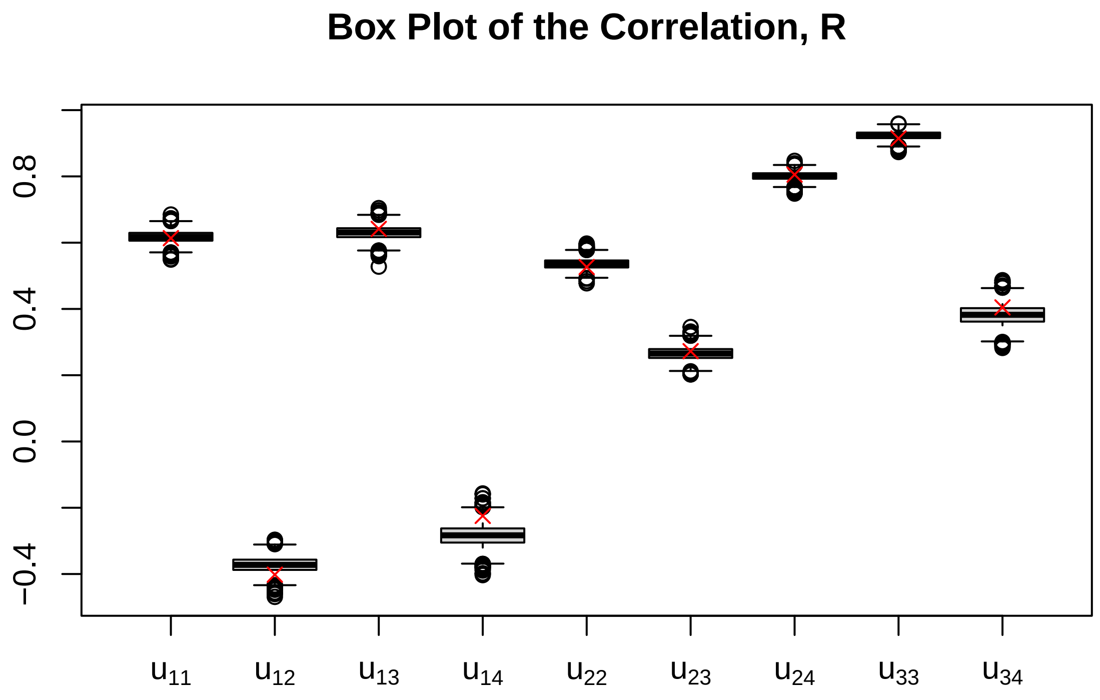
```r
# S3
UB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    3])))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
LB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    3])))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]

plot(sqrt(S_t.post[-1, 3]), type = "l", ylab = "Y3", main = "",
    xlab = "", col = "black", cex.lab = 2, cex.axis = 1.75)
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(sqrt(S_t.post[-1, 3]))
```



```r
# S4
UB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    4])))), 1, function(x) {
    quantile(x, c(0.975))
})[-1]
LB = apply(t(do.call(rbind, lapply(S_t.samp, function(x) sqrt(x[,
    4])))), 1, function(x) {
    quantile(x, c(0.025))
})[-1]

plot(sqrt(S_t.post[-1, 4]), type = "l", ylab = "Y4", main = "",
    xlab = "", col = "black", cex.lab = 2, cex.axis = 1.75)
polygon(c(Time, rev(Time)), c(UB, rev(LB)), col = "grey", border = NA)
lines(sqrt(S_t.post[-1, 4]))
```



```r
# Estimates for R, the correlation matrix
par(mfrow = c(1, 1))
V.chol = gmat::uchol(V)
boxplot(do.call(rbind, lapply(cor.samp, function(x) c(x[1, 1:4],
    x[2, 2:4], x[3, 3:4]))), main = "Box Plot of the Correlation, R",
    names = expression(u[1][1], u[1][2], u[1][3], u[1][4], u[2][2],
        u[2][3], u[2][4], u[3][3], u[3][4]))
points(c(V.chol[1, 1:4], V.chol[2, 2:4], V.chol[3, 3:4]), pch = 4,
    col = "red")
```



Step-4 Model Evaluation and Residual Analysis

WAIC

```r
## WAIC for MVT mod

Rcpp::sourceCpp("./Functions/waic1Garch.cpp")

V.samp = lapply(cor.samp, function(x) tcrossprod(x))
waic_g = MVT_Garch_waic1(par = garch_par_samp, cor = V.samp,
    W = W.samp, Y, FF = FF, GG = GG, m0 = m0)
```

```
## 4999
## GJR
```

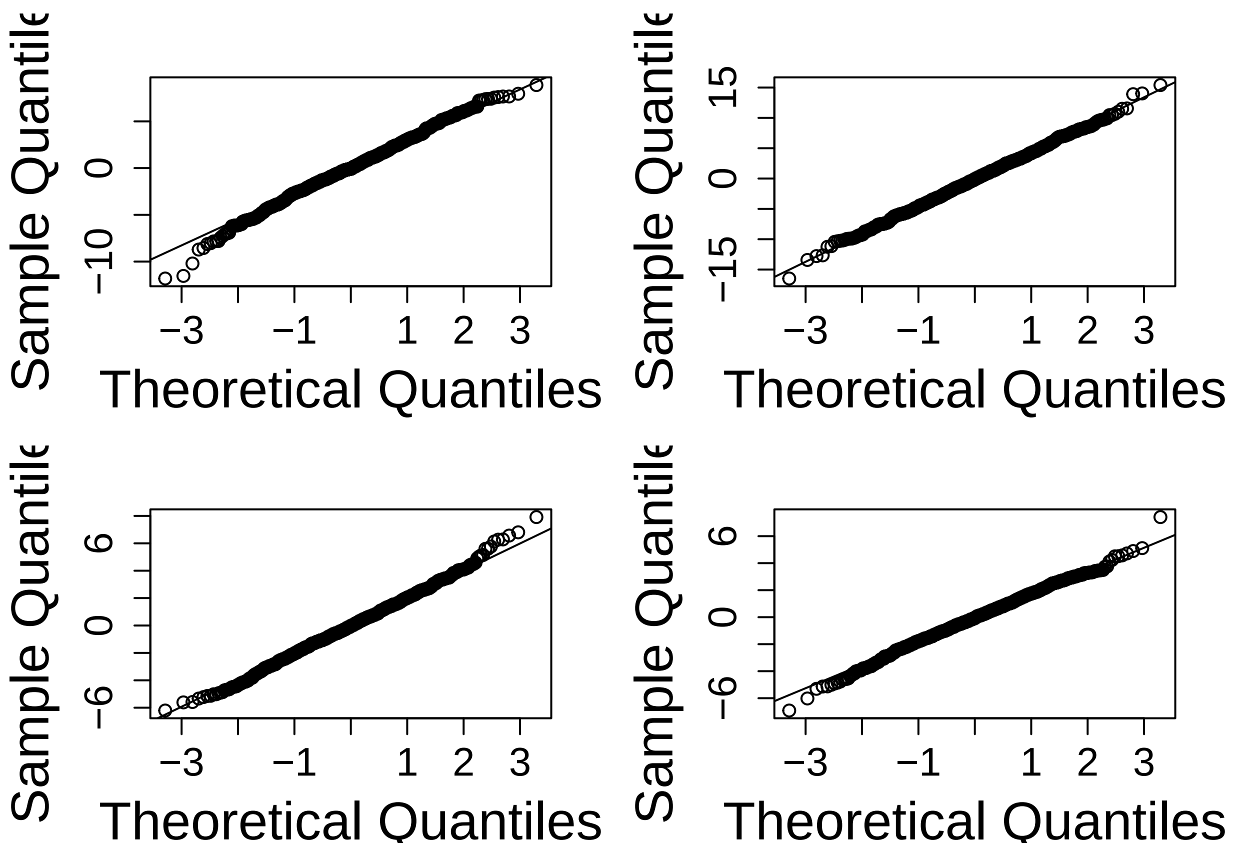
```r
waic_g
```

```
## $waic
## [1] -18032.82
```

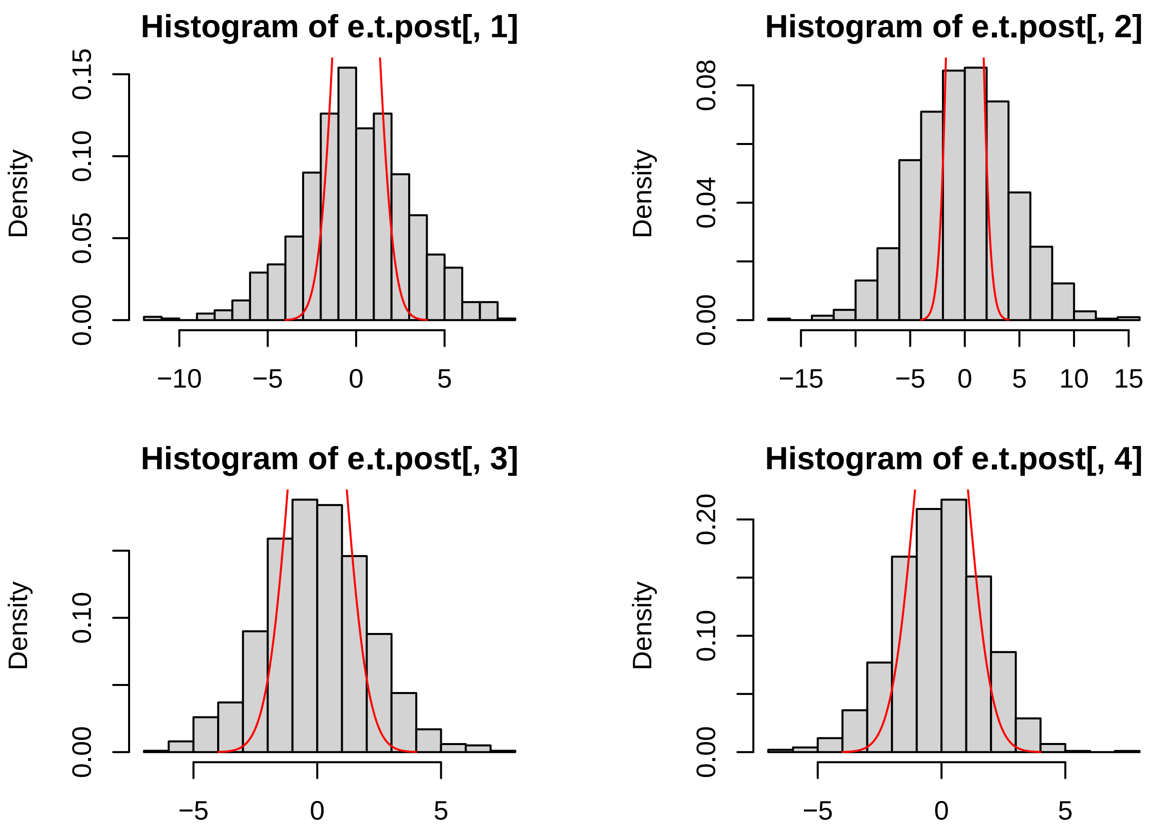
Residual Analysis

```r
garch_par_post = rbind(apply(do.call(rbind, lapply(garch_par_samp,
    function(x) x[1, ])), 2, median)[1:4999], apply(do.call(rbind,
    lapply(garch_par_samp, function(x) x[2, ])), 2, median)[1:4999],
    apply(do.call(rbind, lapply(garch_par_samp, function(x) x[3,
        ])), 2, median)[1:4999], apply(do.call(rbind, lapply(garch_par_samp,
        function(x) x[4, ])), 2, median)[1:4999])

S_t.post = Reduce("+", S_t.samp[1:4999])/4999


## Posterior estimate of the correlation matrix
R.post = lapply(cor.samp, function(x) {
    tcrossprod(x)
})[1:4999]
R.post = Reduce("+", R.post)/length(R.post)


## Posterior estimate of the time varying observation
## variance matrix
V.post = list()
for (i in 1:n) {
    V.post[[i]] = diag(sqrt(S_t.post[i, ])) %*% R.post %*% diag(sqrt(S_t.post[i,
        ]))
}

## Posterior estimates of the innovations
e.t.post = Y - theta.post[-1, ]

# Normalized
par(mfrow = c(2, 2))
par(mar = c(5, 5, 2, 2))
qqnorm(e.t.post[, 1], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.post[, 1])
```

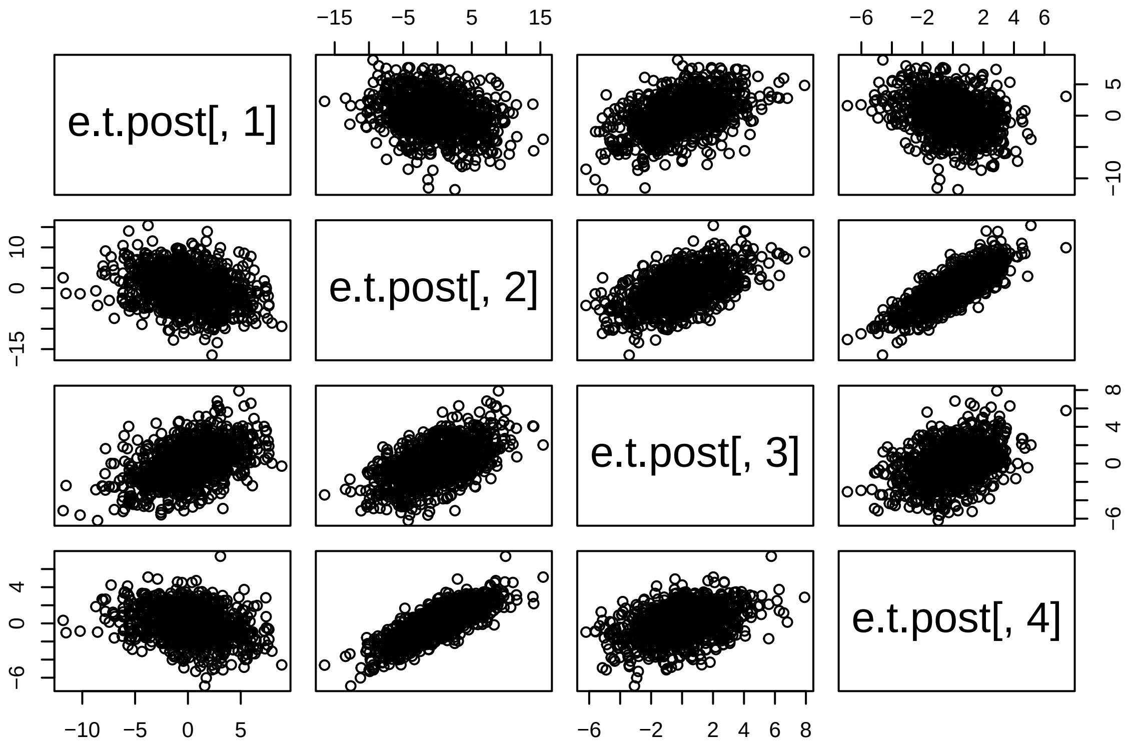
```r
qqnorm(e.t.post[, 2], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.post[, 2])

qqnorm(e.t.post[, 3], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.post[, 3])

qqnorm(e.t.post[, 4], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.post[, 4])
```



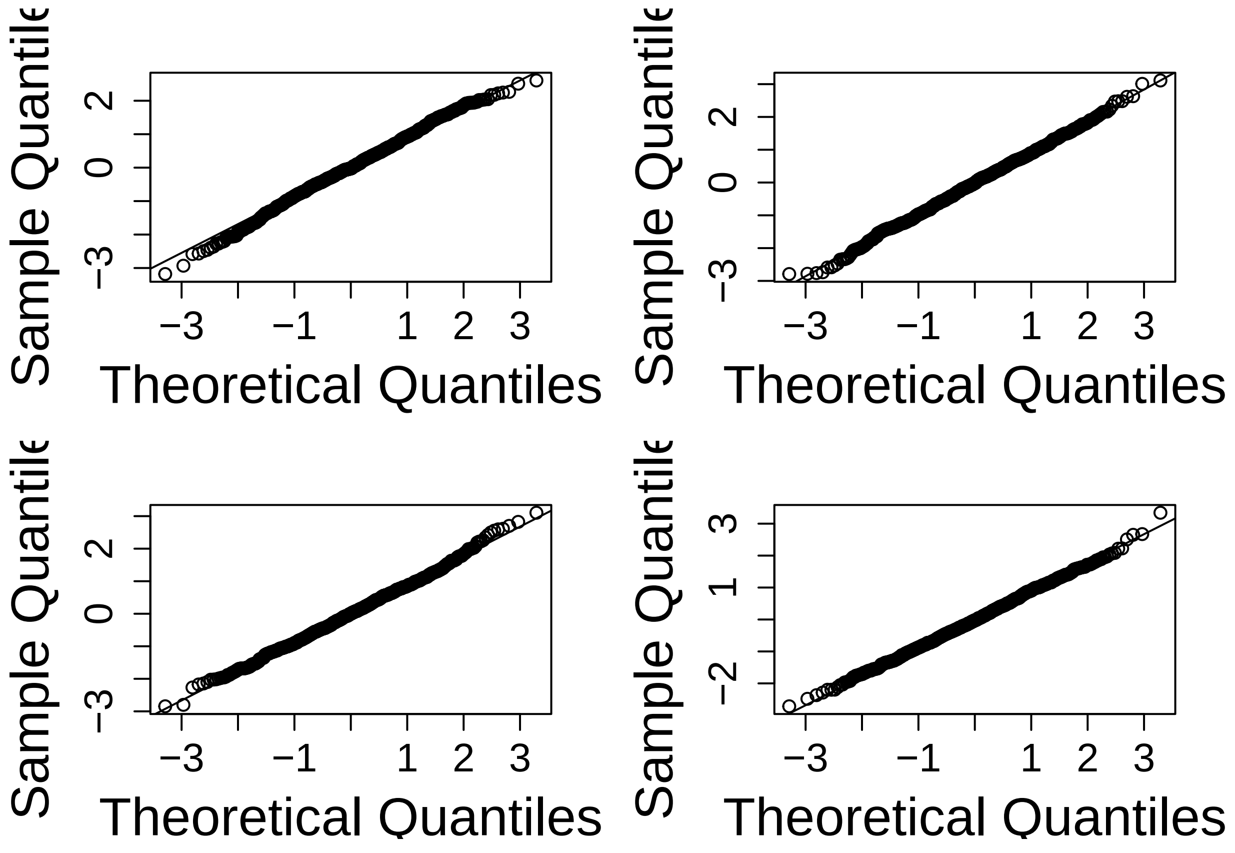
```r
# Hist Unnormalized
par(mfrow = c(2, 2))
par(mar = c(3, 5, 2, 2))
hist(e.t.post[, 1], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")

hist(e.t.post[, 2], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")

hist(e.t.post[, 3], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")

hist(e.t.post[, 4], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")
```



```r
par(mfrow = c(1, 1))
pairs(~e.t.post[, 1] + e.t.post[, 2] + e.t.post[, 3] + e.t.post[,
    4])
```

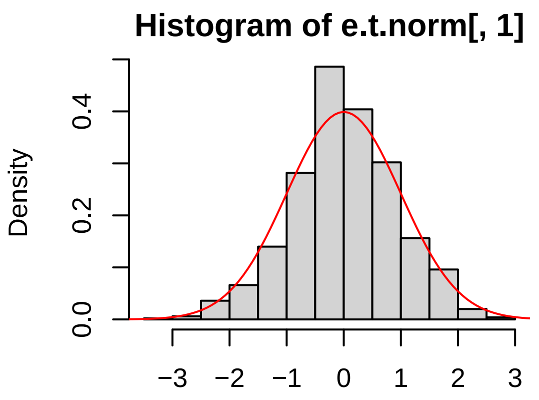


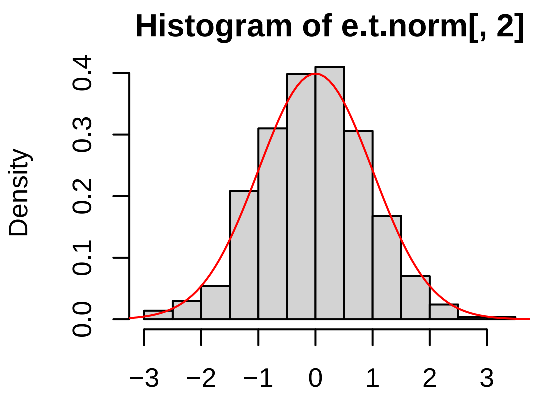
```r
## Normalize the estimated innovations
e.t.norm = matrix(nrow = n, ncol = p)
for (i in 1:n) {
    e.t.norm[i, ] = t(solve(t(chol(V.post[[i]]))) %*% e.t.post[i,
        ])
}


par(mfrow = c(2, 2))
par(mar = c(5, 5, 2, 2))
qqnorm(e.t.norm[, 1], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.norm[, 1])

qqnorm(e.t.norm[, 2], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.norm[, 2])

qqnorm(e.t.norm[, 3], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.norm[, 3])

qqnorm(e.t.norm[, 4], main = " ", cex.lab = 2, cex.axis = 1.5)
qqline(e.t.norm[, 4])
```

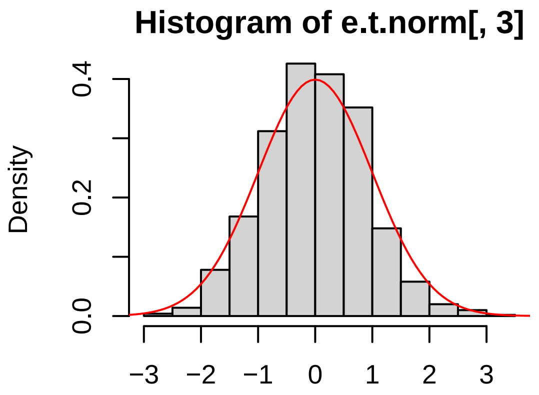


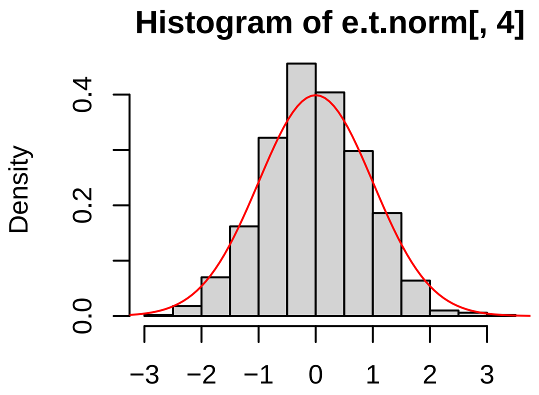
```r
# Hist normalized
par(mfrow = c(2, 2))
par(mar = c(3, 5, 2, 2))
hist(e.t.norm[, 1], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")
```

```r
hist(e.t.norm[, 2], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")

hist(e.t.norm[, 3], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")

hist(e.t.norm[, 4], freq = F, breaks = 15)
curve(dnorm(x, 0, 1), -4, 4, add = T, col = "red")
```

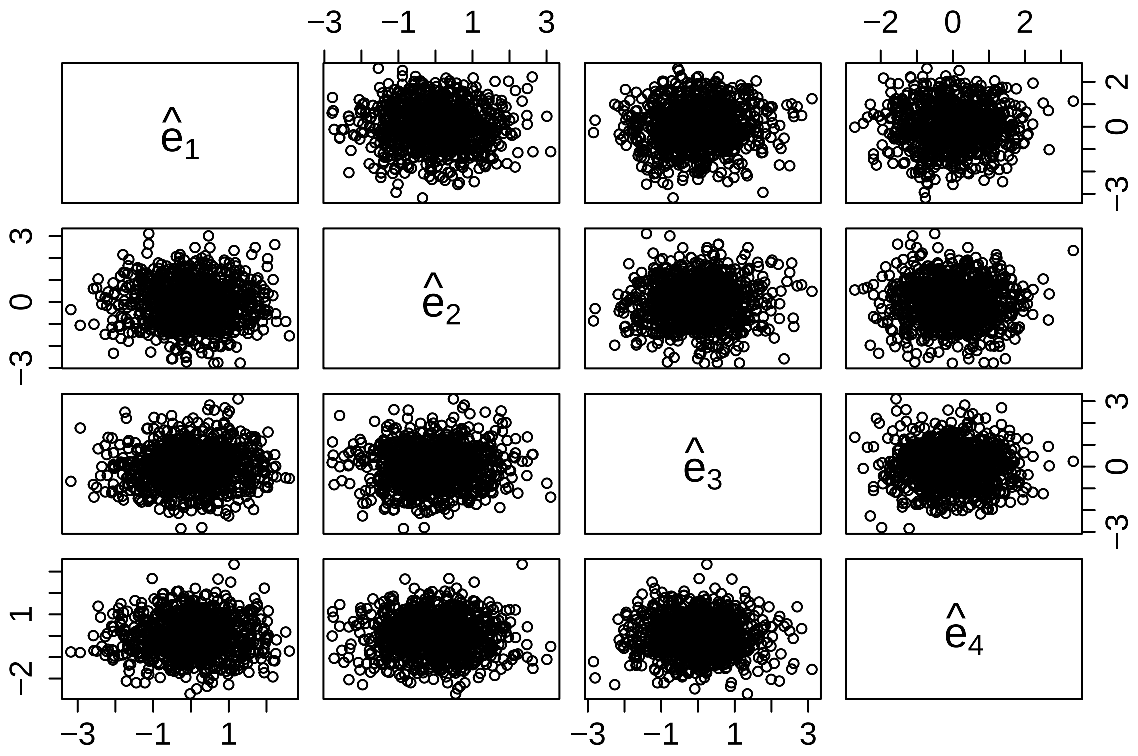








```r
# Scatter plots of the normalized residuals
par(mfrow = c(1, 1))
par(mar = c(5, 5, 2, 2))
pairs(~e.t.norm[, 1] + e.t.norm[, 2] + e.t.norm[, 3] + e.t.norm[,
    4], cex.lab = 2, cex.axis = 1.5, labels = c(expression(hat(e)[1],
    hat(e)[2], hat(e)[3], hat(e)[4])))
```



$\hat{e}_1$
$\hat{e}_2$
$\hat{e}_3$
$\hat{e}_4$
−3 −1 1 3
−2 0 2