pruebas iid

Eduardo Rubio M

28-04-2024

Primero estamos probando generar series IID para ver como se comporta el estadistico X__n(r,d)

```r
# Generar las series IID
set.seed(125)
N <- 1000  # Longitud de la serie
data <- rnorm(N)
```

Generamos la función para calcular la Corelación integral según nuestra definición que sería una dependiente de 3 inputs, la serie "data" con su tamaño, la distancia fijada r y la dimensión d.

```r
# Definir la función para calcular la Corelación integral
correlation_integral <- function(data, r, d) {
  N <- length(data)
  count <- 0

  # Iterar sobre todos los pares (i, j) tal que 1 <= i < j <= N-d+1
  for (i in 1:(N-d)) {
    for (j in (i+1):(N-d+1)) {
      # Crear los vectores u_i^d y u_j^d
      u_i_d <- data[i:(i+d-1)]
      u_j_d <- data[j:(j+d-1)]

      # Calcular la distancia euclidiana entre u_i^d y u_j^d
      distance <- sqrt(sum((u_i_d - u_j_d)^2))

      # Verificar si la distancia es menor que r
      if (distance < r) {
        count <- count + 1
      }
    }
  }

  # Calcular C_N(r, d)
  C_N <- (2 / (N^2)) * count
  return(C_N)
}
```

Probamos para distintos casos de d y r como funciona la función.

```r
# Parámetros
r <- 0.5  # Umbral para la norma
```

```r
d <- 2    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.059494
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 3    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.011054
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 4    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.001888
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 5    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.000284
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 7    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 4e-06
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 10    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0
```

```r
# Parámetros
r <- 0.1  # Umbral para la norma
d <- 2    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.002414
```

```r
# Parámetros
r <- 1  # Umbral para la norma
d <- 2    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.218298
```

```r
# Parámetros
r <- 5  # Umbral para la norma
d <- 2    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.994614
```

```r
# Parámetros
r <- 10  # Umbral para la norma
d <- 2    # Dimensión del espacio fase

# Calcular la correlación integral
C_N_value <- correlation_integral(data, r, d)
C_N_value
```

```
## [1] 0.997002
```

Verificamos que la función se comporta según lo esperado y para r muy grande tiende a acercarse a 1. También mientras se aumenta d la correlación integral es afectada inversamente. para d>8 ya empiezan a solo sumarse 0s.

Dejemos de ejemplo para d=2 y r=0.5

Ahora generamos la función para el X_n(r,d) utilizando la función de correlación integral. Por la definición de esta para d=1 se tiene el caso particular y a partir de ahi tiene su definición con respecto a correlaciones integrales para d-1 y d+1

```r
# Definir la función X_N(r, d)
X_N <- function(data, r, d) {
  C_N_d <- correlation_integral(data, r, d)
  C_N_d_minus_1 <- correlation_integral(data, r, d-1)
  C_N_d_plus_1 <- correlation_integral(data, r, d+1)

  X_N_value <- (C_N_d^2) / (C_N_d_minus_1 * C_N_d_plus_1)
  return(X_N_value)
}

# Caso especial para d = 1
X_N_d_1 <- function(data, r) {
  C_N_1 <- correlation_integral(data, r, 1)
  C_N_2 <- correlation_integral(data, r, 2)

  X_N_value <- (C_N_1^2) / C_N_2
  return(X_N_value)
}
```

Probemos calcular para d=2 y d=1

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 2    # Dimensión del espacio fase

# Calcular X_N(r, d)
X_N_value <- X_N(data, r, d)
X_N_value
```

```
## [1] 1.169806
```

```r
# Calcular X_N(r, 1)
X_N_value_d_1 <- X_N_d_1(data, r)
X_N_value_d_1
```

```
## [1] 1.259368
```

probemos como va con distintos d y r

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 3    # Dimensión del espacio fase
```

```r
# Calcular X_N(r, d)
X_N_value <- X_N(data, r, d)
X_N_value
```

```
## [1] 1.087837
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 4    # Dimensión del espacio fase

# Calcular X_N(r, d)
X_N_value <- X_N(data, r, d)
X_N_value
```

```
## [1] 1.135445
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 5    # Dimensión del espacio fase

# Calcular X_N(r, d)
X_N_value <- X_N(data, r, d)
X_N_value
```

```
## [1] 1.068008
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 6    # Dimensión del espacio fase

# Calcular X_N(r, d)
X_N_value <- X_N(data, r, d)
X_N_value
```

```
## [1] 1.408451
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d <- 10    # Dimensión del espacio fase

# Calcular X_N(r, d)
X_N_value <- X_N(data, r, d)
X_N_value
```

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

dado que la funcion correlación integral sobre el d=8 genera 0s, esta función que divide por correlaciones previas y siguientes desde el d=7 genera problemas. (Por revisar)

Veamos como funciona para multiples series IID

```r
# Generar múltiples series IID
set.seed(125)
n_series <- 100  # Número de series
length_series <- 1000  # Longitud de cada serie

# Crear una matriz donde cada fila es una serie IID
data_matrix <- matrix(rnorm(n_series * length_series), nrow = n_series, ncol = length_series)
```

```r
# Definir parámetros
r <- 0.5  # Umbral para la norma
d <- 2    # Dimensión del espacio fase

# Inicializar vectores para almacenar los resultados
X_N_values <- numeric(n_series)
X_N_values_d_1 <- numeric(n_series)

# Calcular X_{N}(r,d) para cada serie
for (i in 1:n_series) {
  series <- data_matrix[i, ]
  X_N_values[i] <- X_N(series, r, d)
  X_N_values_d_1[i] <- X_N_d_1(series, r)
}

# Ver los resultados
X_N_values
```

```
##   [1] 1.181169 1.167130 1.170547 1.185432 1.177401 1.153439 1.208369 1.186084
##   [9] 1.156712 1.159597 1.138112 1.171050 1.162195 1.182815 1.173059 1.194506
##  [17] 1.147803 1.178639 1.198537 1.170228 1.191433 1.222673 1.199569 1.171237
##  [25] 1.185215 1.178151 1.215053 1.172980 1.180195 1.197459 1.184045 1.176353
##  [33] 1.165736 1.175397 1.180943 1.159417 1.178222 1.141139 1.194178 1.171103
##  [41] 1.176637 1.133649 1.186982 1.176775 1.160479 1.159702 1.161862 1.179293
##  [49] 1.198824 1.169903 1.195558 1.184801 1.161502 1.179882 1.192829 1.165009
##  [57] 1.184510 1.166827 1.159194 1.152142 1.186000 1.183238 1.141893 1.201110
##  [65] 1.205457 1.189317 1.162830 1.152324 1.174431 1.132083 1.176698 1.160196
##  [73] 1.176646 1.185737 1.195158 1.190191 1.165975 1.165774 1.183544 1.183722
##  [81] 1.154712 1.173892 1.170254 1.183857 1.177527 1.165031 1.179284 1.196087
##  [89] 1.117626 1.181243 1.166420 1.202396 1.182430 1.165970 1.168928 1.198818
##  [97] 1.195391 1.196811 1.182072 1.203117
```

```r
X_N_values_d_1
```

```
##   [1] 1.223207 1.258024 1.258963 1.268799 1.268469 1.269435 1.256971 1.290304
##   [9] 1.235536 1.260435 1.270915 1.277441 1.232070 1.279731 1.260068 1.258417
##  [17] 1.241870 1.261970 1.261575 1.262450 1.262135 1.267108 1.266177 1.265875
##  [25] 1.274315 1.264648 1.260368 1.274086 1.260509 1.249471 1.253267 1.254430
##  [33] 1.276405 1.245739 1.268487 1.251752 1.248219 1.241521 1.261942 1.250276
##  [41] 1.277958 1.295423 1.250850 1.261984 1.261040 1.249601 1.275189 1.276216
##  [49] 1.258175 1.255955 1.288889 1.274566 1.252817 1.258422 1.253199 1.262185
##  [57] 1.251268 1.260348 1.258656 1.236048 1.266253 1.261052 1.262919 1.274302
##  [65] 1.281775 1.262764 1.271487 1.238988 1.258093 1.225886 1.260887 1.263421
##  [73] 1.258403 1.264195 1.278287 1.251172 1.262927 1.288294 1.270446 1.261439
```

```
##  [81] 1.265368 1.253989 1.259130 1.278974 1.258699 1.276673 1.261131 1.255033
##  [89] 1.252642 1.247345 1.288938 1.263536 1.273479 1.265242 1.270198 1.271319
##  [97] 1.260510 1.272554 1.256068 1.256815
```

```r
# Parámetros
r <- 0.5  # Umbral para la norma
d_values <- 1:5  # Valores de d

# Inicializar listas para almacenar los resultados
X_N_results <- matrix(0, nrow = n_series, ncol = length(d_values))
colnames(X_N_results) <- paste("d =", d_values)

# Calcular X_{N}(r,d) para cada serie y para cada valor de d
for (i in 1:n_series) {
  series <- data_matrix[i, ]

  # Caso especial para d = 1
  X_N_results[i, 1] <- X_N_d_1(series, r)

  # Calcular X_{N}(r,d) para d = 2, 3, 4, 5
  for (d in 2:length(d_values)) {
    X_N_results[i, d] <- X_N(series, r, d_values[d])
  }
}

# Ver los resultados
head(X_N_results)
```

```
##         d = 1    d = 2    d = 3    d = 4     d = 5
## [1,] 1.223207 1.181169 1.103162 1.084939 0.9755604
## [2,] 1.258024 1.167130 1.109663 1.208229 1.3017553
## [3,] 1.258963 1.170547 1.137877 1.273275 1.1535256
## [4,] 1.268799 1.185432 1.133412 1.046204 0.9598881
## [5,] 1.268469 1.177401 1.165448 1.040907 1.2465908
## [6,] 1.269435 1.153439 1.112630 1.158928 0.9291116
```

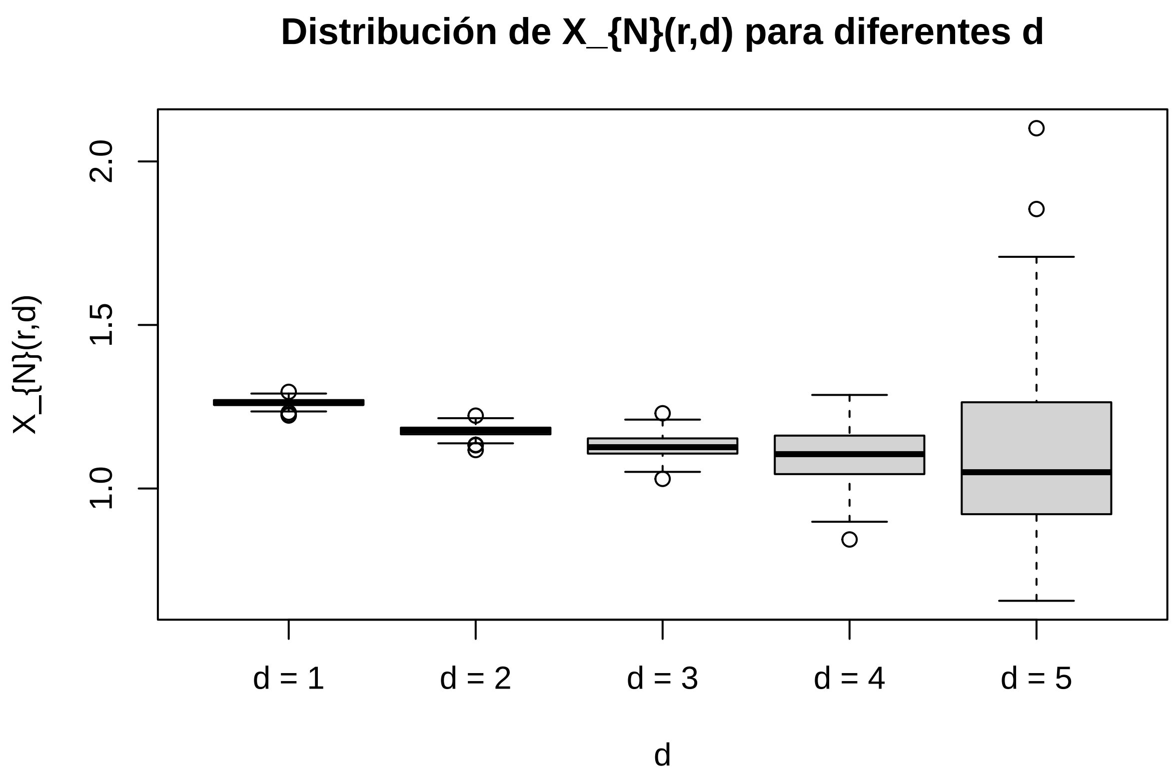
Veamos como se comporta X_{N}(r,d) para este caso

```r
# Resumen de los resultados
summary(X_N_results)
```

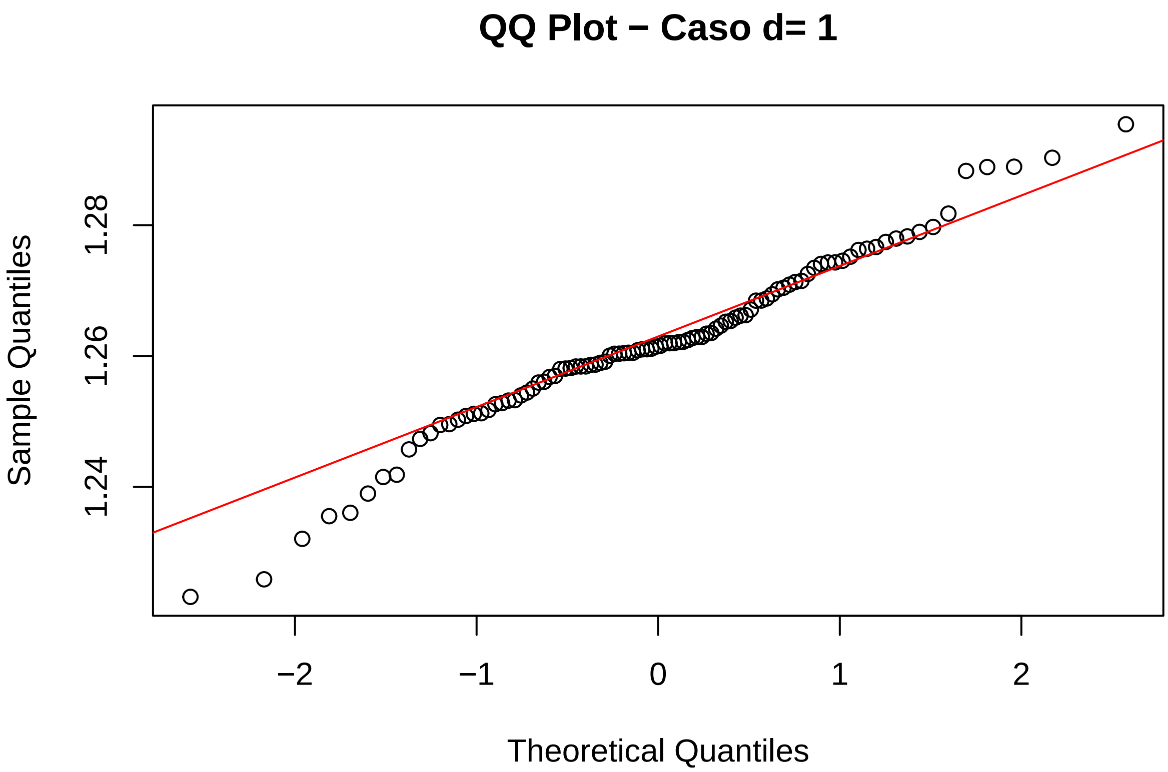
```
##      d = 1           d = 2           d = 3           d = 4       
##  Min.   :1.223   Min.   :1.118   Min.   :1.030   Min.   :0.8442  
##  1st Qu.:1.256   1st Qu.:1.166   1st Qu.:1.107   1st Qu.:1.0451  
##  Median :1.262   Median :1.177   Median :1.126   Median :1.1050  
##  Mean   :1.262   Mean   :1.176   Mean   :1.128   Mean   :1.1072  
##  3rd Qu.:1.270   3rd Qu.:1.186   3rd Qu.:1.152   3rd Qu.:1.1604  
##  Max.   :1.295   Max.   :1.223   Max.   :1.230   Max.   :1.2861  
##      d = 5       
##  Min.   :0.6567  
##  1st Qu.:0.9220  
##  Median :1.0498  
##  Mean   :1.1050  
```

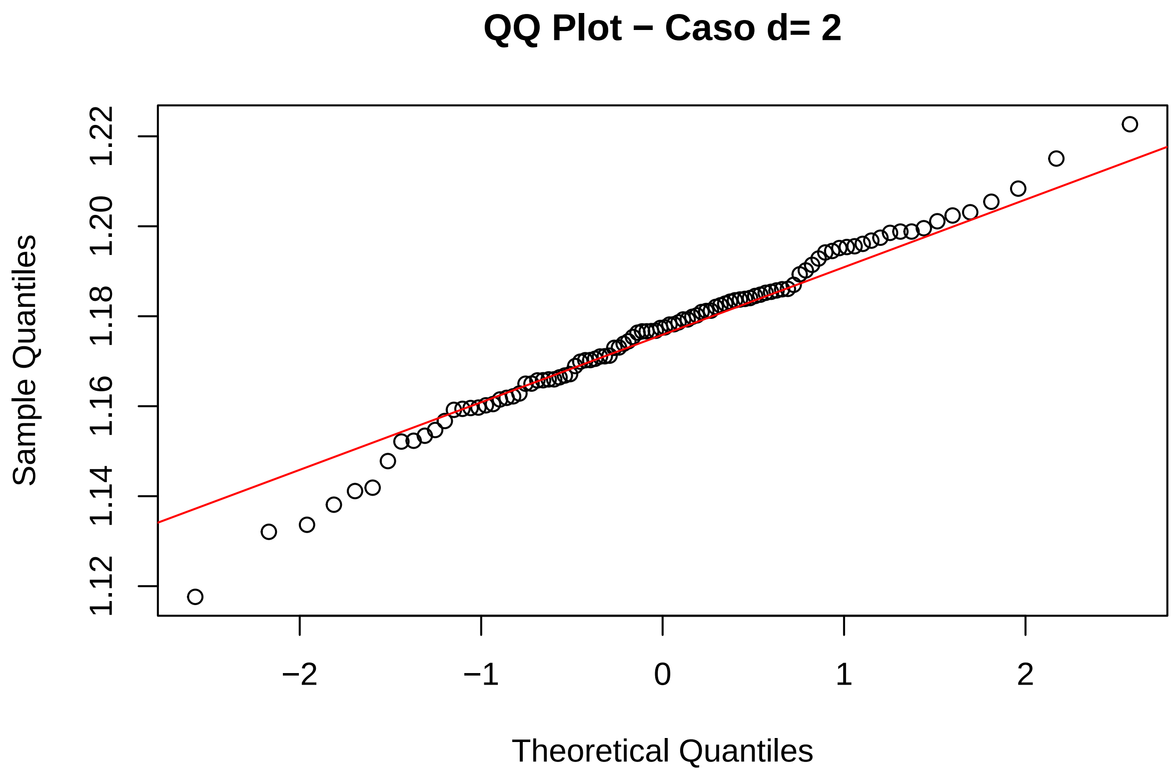
```
##   3rd Qu.:1.2575
##   Max.   :2.1015
```

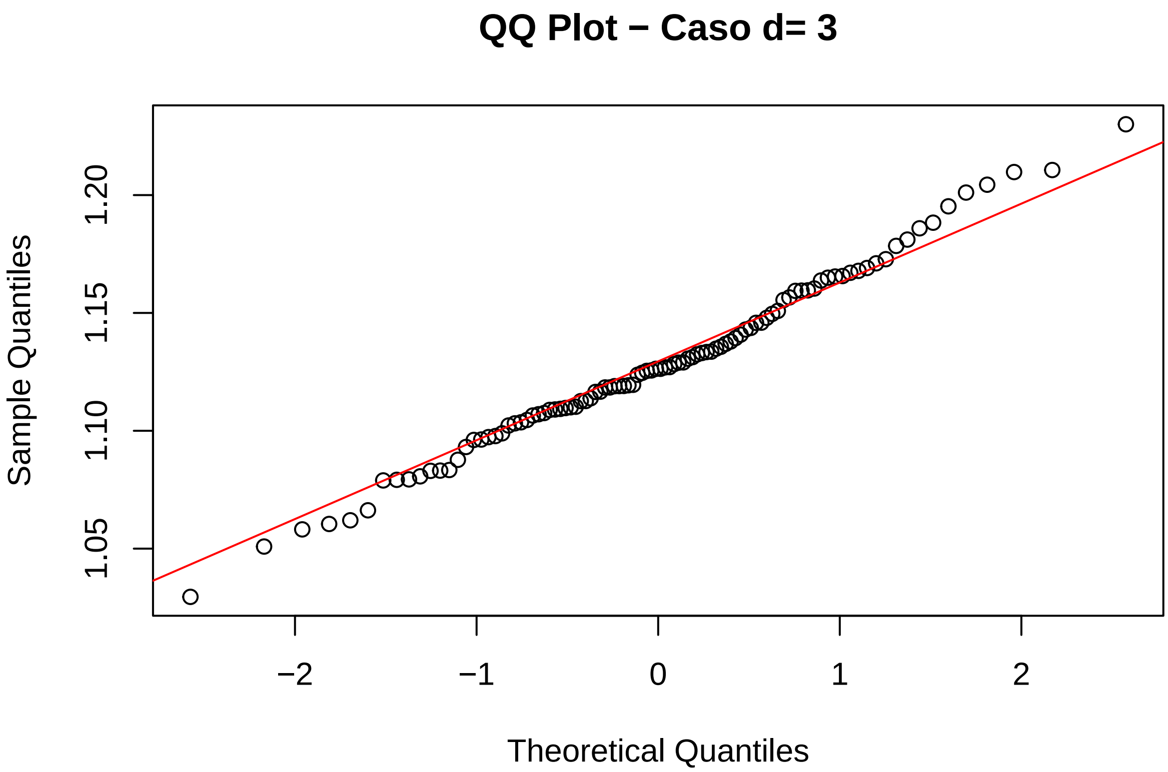
```r
# Graficar los resultados
boxplot(X_N_results, main="Distribución de X_{N}(r,d) para diferentes d", ylab="X_{N}(r,d)", xlab="d")
```

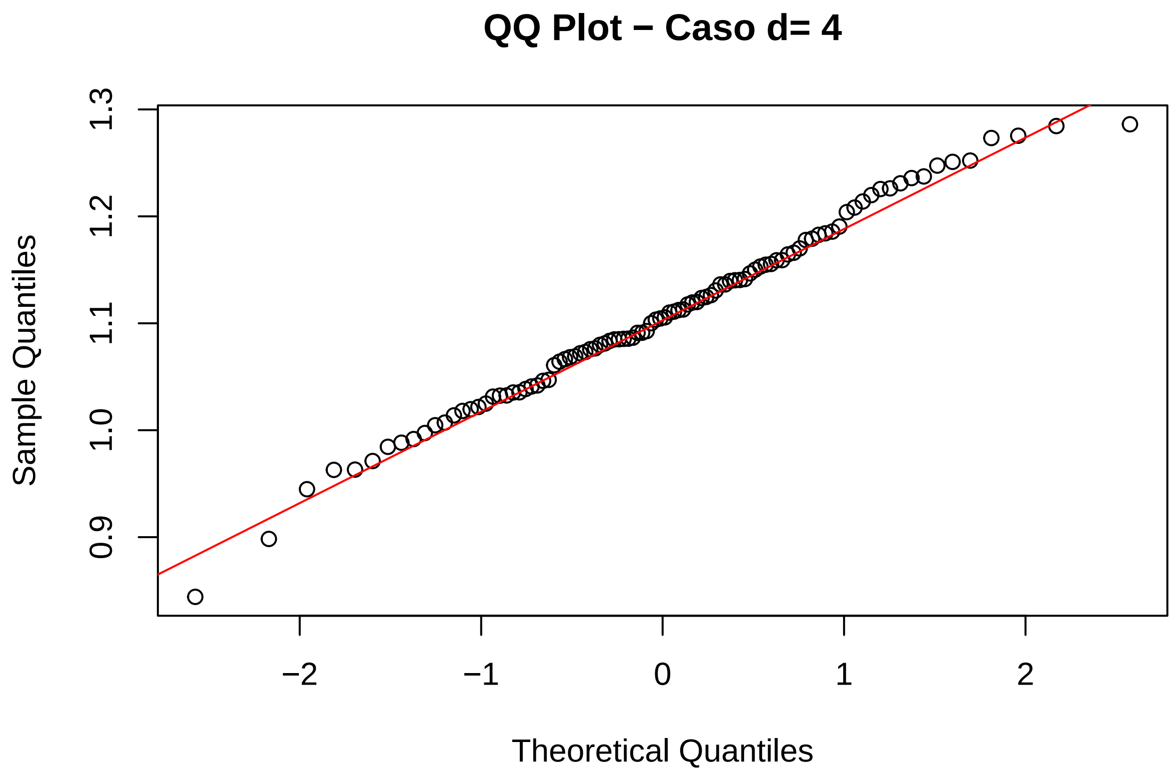


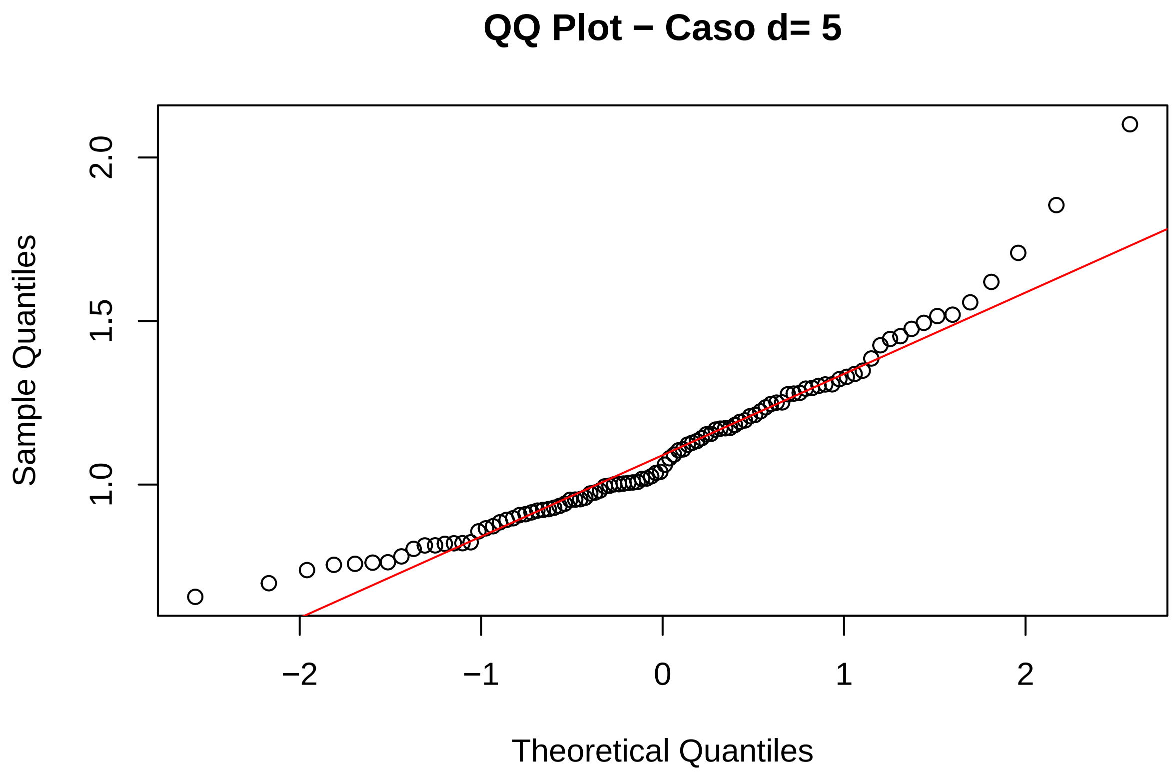

```r
# Crear el QQ plot

for (i in 1:ncol(X_N_results)) {
  qqnorm(X_N_results[, i], main = paste("QQ Plot - Caso d=", i))
  qqline(X_N_results[, i], col = "red")
}
```


QQ Plot − Caso d= 1
Sample Quantiles
1.24
1.26
1.28
−2
−1
0
1
2
Theoretical Quantiles


QQ Plot − Caso d= 2
Sample Quantiles
1.12
1.14
1.16
1.18
1.20
1.22
−2
−1
0
1
2
Theoretical Quantiles


QQ Plot − Caso d= 3
Sample Quantiles
1.05
1.10
1.15
1.20
−2
−1
0
1
2
Theoretical Quantiles


QQ Plot − Caso d= 4
Sample Quantiles
Theoretical Quantiles
0.9
1.0
1.1
1.2
1.3
−2
−1
0
1
2



Ya estando mas cerca del estadistico completo calculamos $\sqrt{N} ln(X_N(r,d))$

```r
# Longitud de cada serie
N <- length_series  # Esto es igual a 1000 en nuestro caso

# Calcular sqrt(N)
sqrt_N <- sqrt(N)

# Calcular sqrt(N) * ln(X_{N}(r,d)) para cada serie y cada valor de d
sqrt_N_ln_X_N <- sqrt_N * log(X_N_results)

# Ver los resultados
head(sqrt_N_ln_X_N)
```

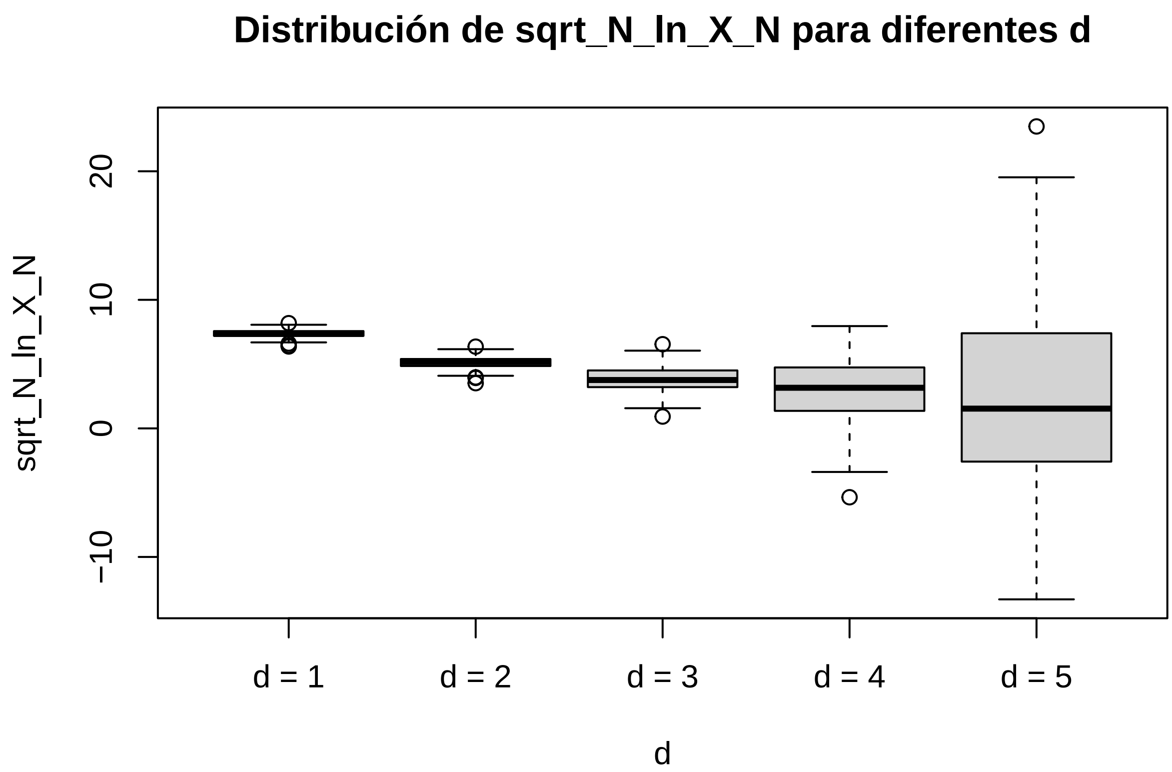
```
##         d = 1    d = 2    d = 3    d = 4      d = 5
## [1,] 6.371243 5.265337 3.104732 2.578005 -0.7824502
## [2,] 7.258767 4.887232 3.290546 5.981618  8.3393556
## [3,] 7.282348 4.979675 4.084542 7.639830  4.5164602
## [4,] 7.528457 5.379257 3.960200 1.428351 -1.2945924
## [5,] 7.520241 5.164286 4.841632 1.267848  6.9700535
## [6,] 7.544310 4.514072 3.374993 4.664206 -2.3251098
```

y vemos como se comporta:

```r
# Resumen de los resultados
summary(sqrt_N_ln_X_N)
```

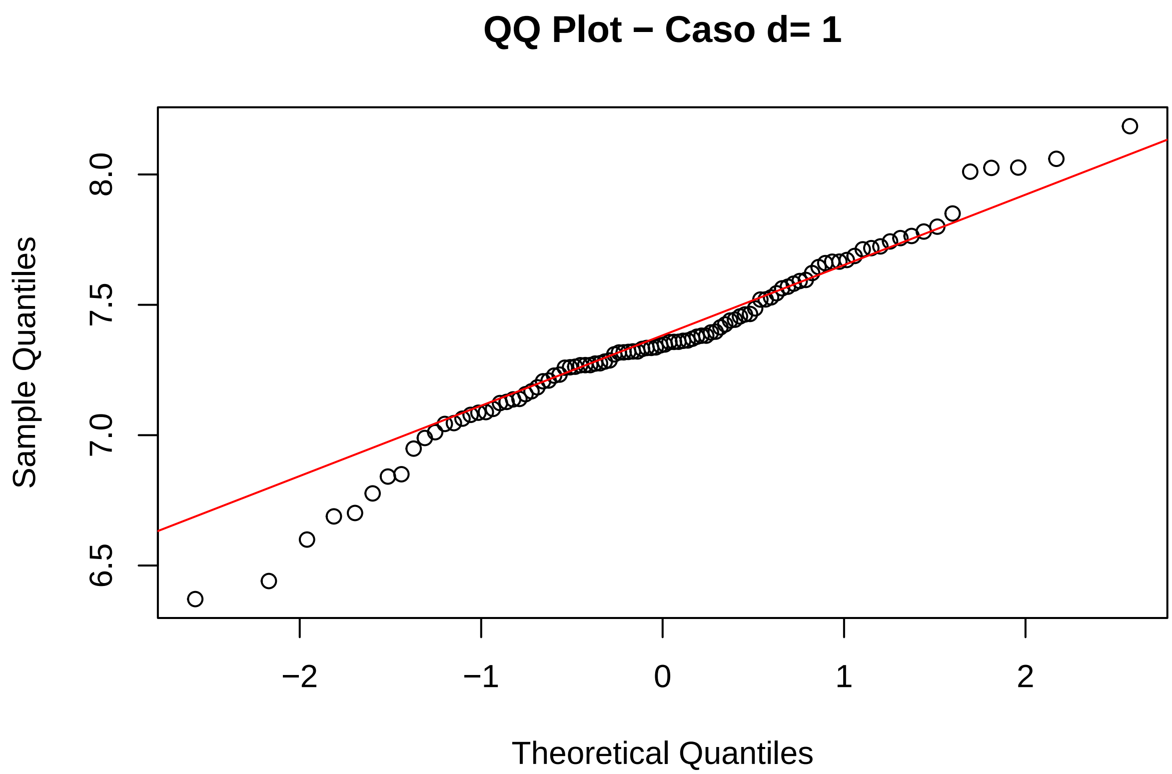
```
##      d = 1            d = 2            d = 3              d = 4       
##  Min.   :6.371   Min.   :3.517   Min.   :0.9206   Min.   :-5.356  
##  1st Qu.:7.201   1st Qu.:4.850   1st Qu.:3.2115   1st Qu.: 1.396  
##  Median :7.346   Median :5.166   Median :3.7610   Median : 3.159  
##  Mean   :7.358   Mean   :5.129   Mean   :3.8030   Mean   : 3.122  
##  3rd Qu.:7.565   3rd Qu.:5.395   3rd Qu.:4.4752   3rd Qu.: 4.704  
##  Max.   :8.185   Max.   :6.357   Max.   :6.5477   Max.   : 7.957  
##      d = 5        
##  Min.   :-13.298  
##  1st Qu.: -2.567  
##  Median :  1.536  
##  Mean   :  2.351  
##  3rd Qu.:  7.243  
##  Max.   : 23.485
```

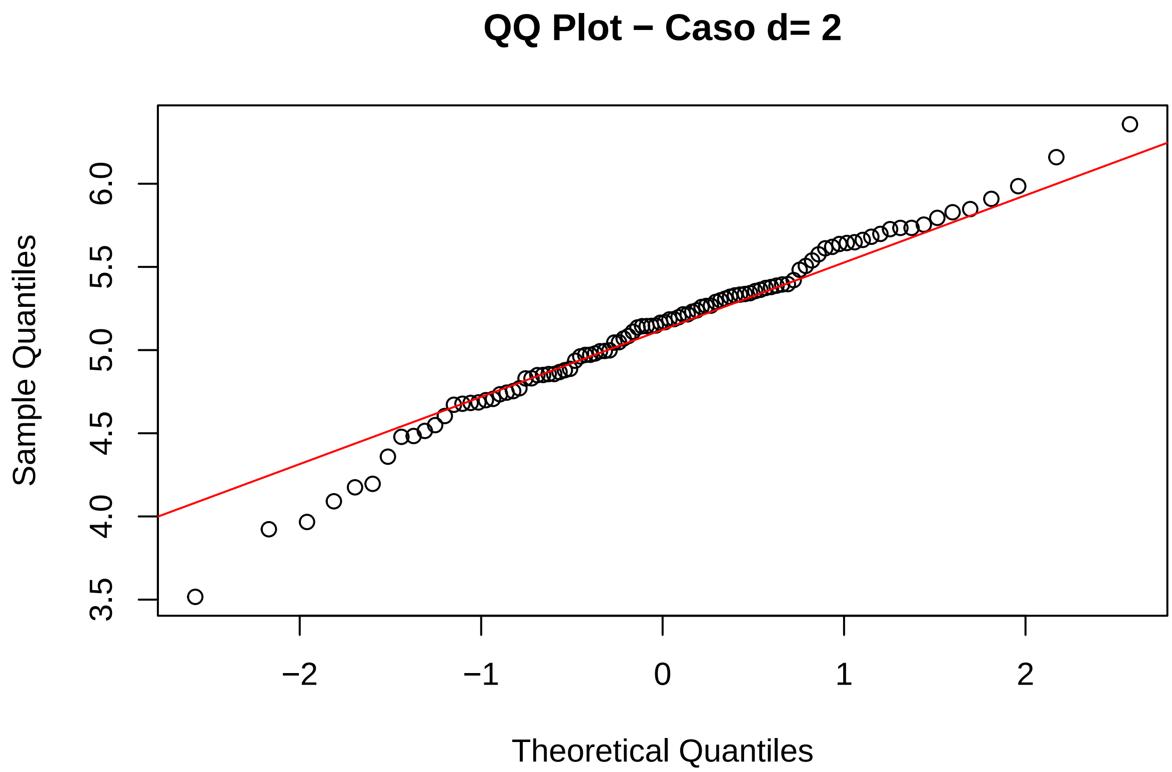
```r
# Graficar los resultados
boxplot(sqrt_N_ln_X_N, main="Distribución de sqrt_N_ln_X_N para diferentes d", ylab="sqrt_N_ln_X_N", xla
```

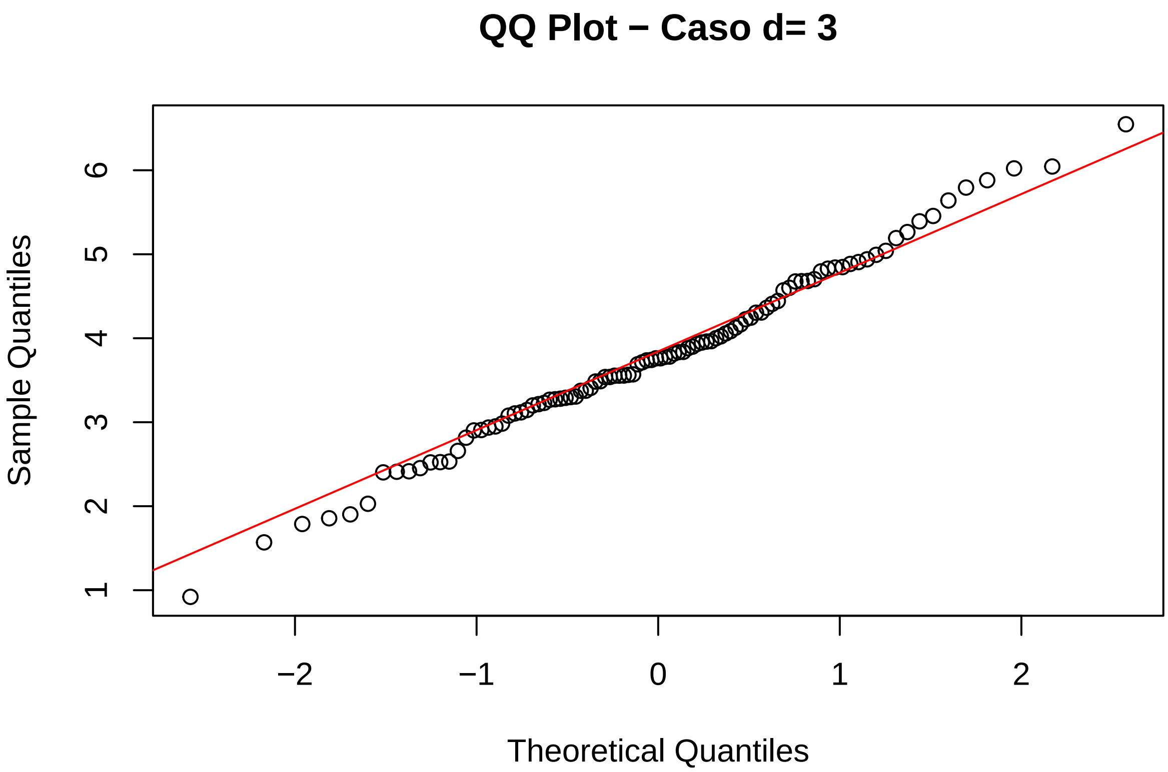


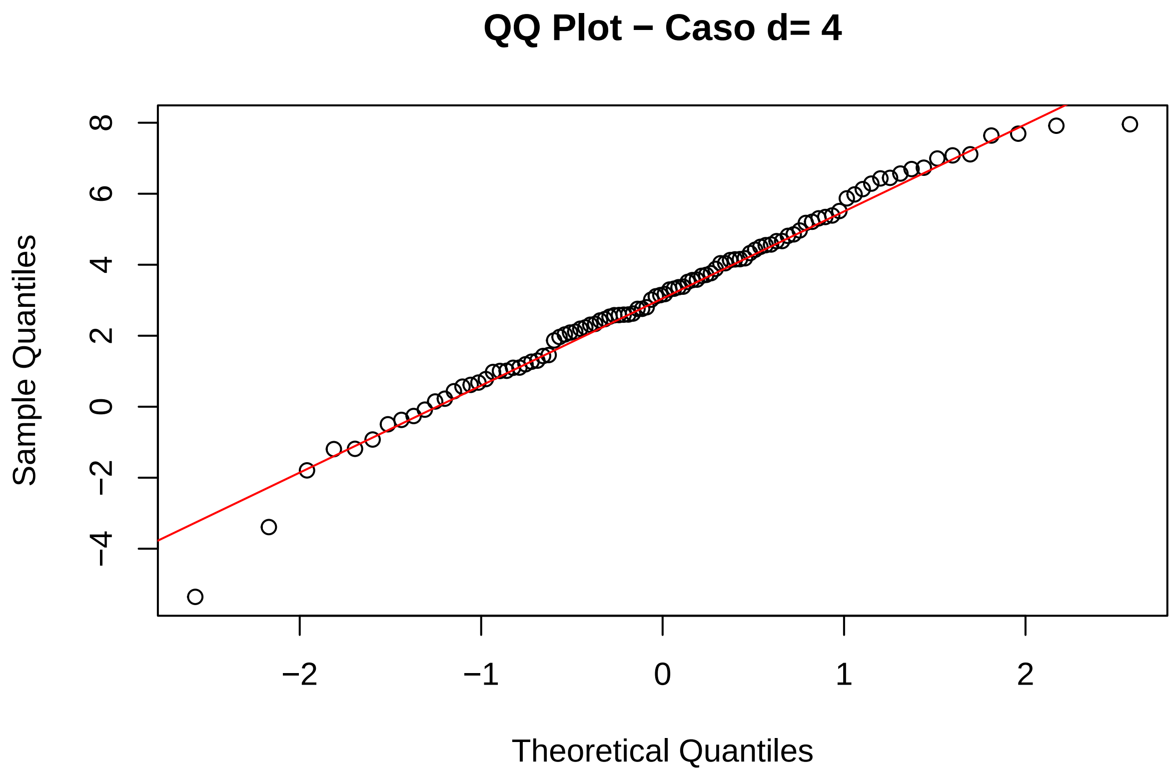

```r
# Crear el QQ plot

for (i in 1:ncol(sqrt_N_ln_X_N)) {
```

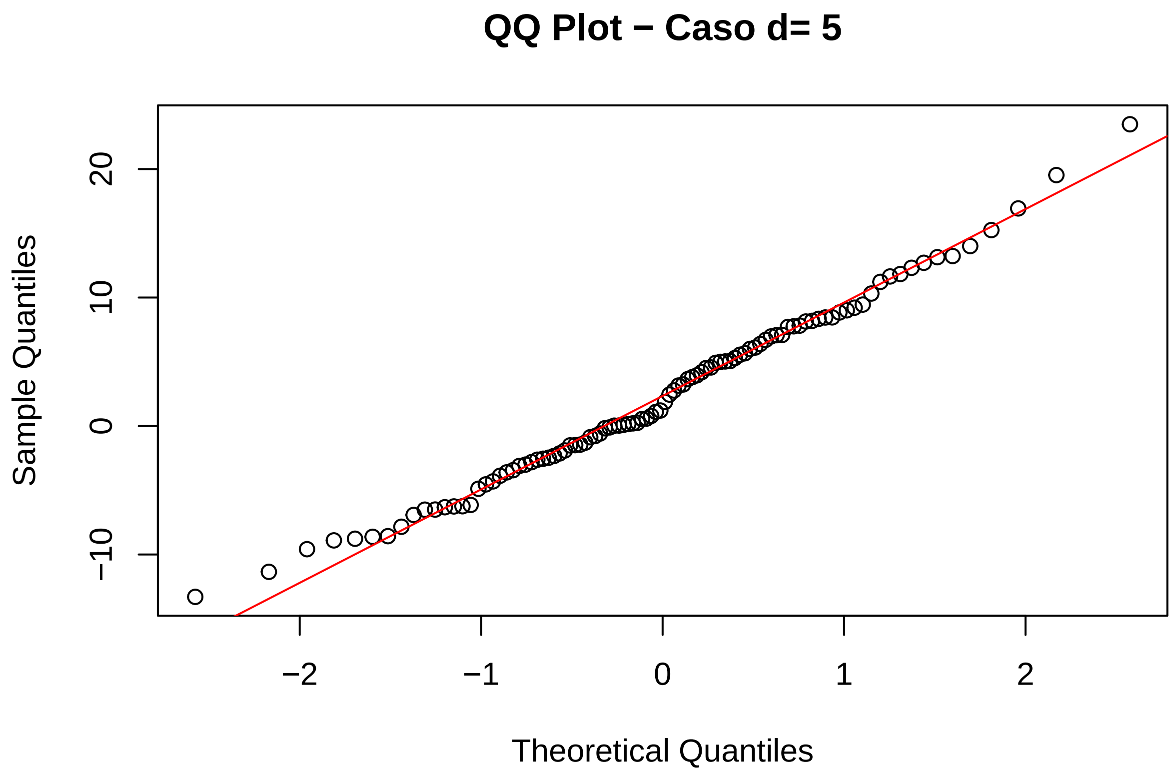
```r
  qqnorm(sqrt_N_ln_X_N[, i], main = paste("QQ Plot - Caso d=", i))
  qqline(sqrt_N_ln_X_N[, i], col = "red")
}
```




QQ Plot − Caso d= 2
Sample Quantiles
Theoretical Quantiles
3.5
4.0
4.5
5.0
5.5
6.0
−2
−1
0
1
2


QQ Plot − Caso d= 3
Sample Quantiles
Theoretical Quantiles
1
2
3
4
5
6
−2
−1
0
1
2


QQ Plot − Caso d= 4
Sample Quantiles
Theoretical Quantiles
−4
−2
0
2
4
6
8
−2
−1
0
1
2




```r
# Calcular la desviación estándar para cada valor de d
std_dev_values <- apply(sqrt_N_ln_X_N, 2, sd)

# Ver los resultados
std_dev_values
```

```
##     d = 1     d = 2     d = 3     d = 4     d = 5
## 0.3281378 0.4913199 1.0465048 2.5170095 7.0994656
```

```r
library(MASS)
```

Distintos test por aplicar:

Falta investigar mas

```r
# Función Shapiro-Wilk test
shapiro_test <- function(series) {
  # Shapiro-Wilk test
  shapiro.test(series)
}

# Aplicar el Shapiro-Wilk test a cada columna
shapiro_results <- apply(sqrt_N_ln_X_N, 2, function(column) {
  # Quitar valores NA
  column <- na.omit(column)
```

```
  shapiro_test(column)
})

# p-values
shapiro_p_values <- sapply(shapiro_results, function(x) x$p.value)


shapiro_p_values
```

```
##      d = 1      d = 2      d = 3      d = 4      d = 5
## 0.06428564 0.22741688 0.89073499 0.28818178 0.67514583
```

```
# Función del Kolmogorov-Smirnov test
ks_test <- function(series) {
  # K-S test comparando la serie a una distribución normal
  ks.test(series, "pnorm", mean = mean(series), sd = sd(series))
}

# Aplicar el K-S test a cada columna
ks_results <- apply(sqrt_N_ln_X_N, 2, function(column) {
  # Remove NA values if they exist
  column <- na.omit(column)
  ks_test(column)
})

# p-values
ks_p_values <- sapply(ks_results, function(x) x$p.value)


ks_p_values
```

```
##     d = 1     d = 2     d = 3     d = 4     d = 5
## 0.3700889 0.7834010 0.8835901 0.9925641 0.7360365
```

```
# Función para el test chi-square para la varianza
std_dev_test <- function(series, sigma_0) {
  n <- length(series)
  s2 <- var(series)  # varianza de muestra

  # Chi-square test statistic
  chi_square_stat <- (n - 1) * s2 / sigma_0^2

  # Grados de libertad
  df <- n - 1

  # p-value
  p_value <- 2 * min(pchisq(chi_square_stat, df), 1 - pchisq(chi_square_stat, df))

  return(list(chi_square_stat = chi_square_stat, p_value = p_value))
}
```

```r
sigma_0 <- 1  # valor según la hipothesis

# Aplicar el test a cada columna
std_dev_results <- apply(sqrt_N_ln_X_N, 2, function(column) {
  # Quitar valores NA values
  column <- na.omit(column)
  std_dev_test(column, sigma_0)
})

# p-values y chi-square statistics
std_dev_p_values <- sapply(std_dev_results, function(x) x$p_value)
chi_square_stats <- sapply(std_dev_results, function(x) x$chi_square_stat)

# p-values
std_dev_p_values
```

```
##        d = 1        d = 2        d = 3        d = 4        d = 5
## 2.369397e-30 8.386996e-16 4.861178e-01 0.000000e+00 0.000000e+00
```

```r
# chi-square statistics
chi_square_stats
```

```
##      d = 1      d = 2      d = 3      d = 4      d = 5
##   10.65977   23.89812  108.42205  627.19835 4989.83877
```

pruebas con caso AR(1)

```r
set.seed(125)  # Fijamos la semilla

# Definimos parámetros
n_series <- 100   # Número de series a generar
n_obs <- 1000      # Número de observaciones por serie
phi <- 0.5         # Coeficiente AR(1)
sigma <- 1         # Desviación estándar del ruido

# Matriz para almacenar las series AR(1)
ar1_series <- matrix(0, nrow = n_obs, ncol = n_series)

# Generamos las series AR(1)
for (i in 1:n_series) {
  ar1_series[, i] <- arima.sim(n = n_obs, list(ar = phi), sd = sigma)
}

# Muestra de las primeras series generadas
head(ar1_series)
```

```
##            [,1]        [,2]       [,3]        [,4]       [,5]       [,6]
## [1,] -0.2091233 -0.17063687 -1.2258896 -0.45526569  0.9079325 -0.4429243
## [2,]  0.3739186  0.03110145  0.8746700 -1.02173656  0.4134092 -0.5498061
## [3,]  0.3831298  1.18009408  0.3305717 -0.96445361 -0.2132346  0.4843161
## [4,]  0.9063922  0.76698036 -0.5899311  1.30209796 -1.5852599  1.7183221
## [5,] -0.5069408  0.72777322  0.2941022  0.44826795  1.1952671  0.6669806
```

```
## [6,]  0.4172486 -0.14367589  0.4075103 -0.03191619  0.4566448 -0.3097206
##           [,7]       [,8]       [,9]     [,10]     [,11]      [,12]      [,13]
## [1,] 3.0427954 -1.2304599  1.3767421 0.2456514 -1.357532 -0.1307202 -0.3964127
## [2,] 1.2858722 -1.1984556  1.4845380 1.1168842 -1.156248 -0.9933014  0.1152657
## [3,] 0.2942972  1.1354962  0.2087438 1.0710801 -2.882519 -0.6041184  2.1299008
## [4,] 2.0381811  0.9250773 -1.6556835 3.3619432 -2.396855 -1.9128842 -0.8223462
## [5,] 0.9969263  1.0844884  0.3648659 1.1823687 -1.047329 -2.7224954  0.3603472
## [6,] 1.0329803  2.0703900  1.7157311 0.5166340 -1.014947 -1.9335793 -0.8587419
##           [,14]      [,15]       [,16]      [,17]      [,18]      [,19]
## [1,] 0.65191012  1.5966359 -0.53186974 -1.0794339 -0.5582870 -0.4743621
## [2,] 0.84376505  2.0005094  0.54287952 -1.0771154 -0.7166428 -1.0935741
## [3,] 0.78536475 -0.3000885 -0.08062948  0.2480565 -0.7008241  1.0590337
## [4,] 0.06399855  0.4934499  0.26334067  0.6070008  0.8261381 -0.4349728
## [5,] 0.19978836  0.7604136 -0.88914838  1.0595916  1.1700860  1.0517623
## [6,] 1.77371090 -0.5649012 -2.38513664  1.6958282  1.0291534  0.9721291
##           [,20]      [,21]       [,22]      [,23]       [,24]      [,25]
## [1,] -0.7465557 -0.2443373  1.90275428 -2.5215683 -1.91581449  0.7376180
## [2,] -0.8878096  0.7260836  1.65511134 -1.9736147 -0.01196093  2.0997224
## [3,]  0.1479134 -0.3715354 -0.73458247 -0.1459359  0.43150020  2.3496813
## [4,]  0.8202819  0.1557178  0.02339111 -0.1986382  0.77398508  0.8971267
## [5,] -0.4604124 -0.1351759  1.29837146 -2.2622356  1.05255195 -0.1405465
## [6,] -0.2188765 -0.4012654  0.92979366 -3.1210386  1.70646237 -0.4342329
##           [,26]      [,27]      [,28]      [,29]       [,30]      [,31]
## [1,] -0.7903399  0.2410650 -0.6838339  0.7385062  0.36039113 -0.9163753
## [2,] -0.4082389 -1.5081072  0.3281353 -1.3061147 -1.14341350 -0.0457781
## [3,] -0.1755411  0.2522325 -0.1589875 -1.7732785 -0.90953605  0.5553913
## [4,]  0.3086250  0.9278158  0.5690929 -0.2499406  0.25770934  0.6984259
## [5,]  0.7976458  0.3129131 -1.7511309 -1.0440410 -0.07861193  1.1948652
## [6,]  0.2776203  0.1353271 -2.6733357 -0.2384293  0.08776222  2.2700240
##            [,32]      [,33]      [,34]       [,35]      [,36]      [,37]
## [1,] -0.70417179  1.1172616 -1.6674428  0.96247965 -1.1832402 -0.4684257
## [2,] -1.53663987  2.0537260  0.2903906 -0.69396364 -2.0966393 -1.6038558
## [3,] -0.58595140  2.1987311  0.4917341 -0.09157113 -0.2793240 -1.8711957
## [4,] -1.27473132  2.8927822 -1.0711781  1.22139279  0.8345864 -0.5954718
## [5,] -0.04033581  0.3071523 -1.3038552 -0.50379755 -0.4418283  0.7179851
## [6,] -0.01886847 -0.6224561 -0.3734599  0.03645332  1.3748546  1.3608034
##           [,38]       [,39]      [,40]      [,41]       [,42]       [,43]
## [1,] -1.7389334  0.08716761  0.1412764 -1.7463232  1.68095976  1.03287418
## [2,] -0.7609163 -1.00190699 -0.1703354 -1.7580663  0.83742070  0.06992978
## [3,] -0.7143432 -0.01412095  0.4007514 -0.9278941  0.27790607  0.24468317
## [4,] -2.7225630  1.09552015  1.1017723 -1.1797056 -0.19887338 -0.56218782
## [5,] -1.0068318 -0.44580011  1.1693608 -1.4343982 -0.07226169  0.20407844
## [6,] -1.0510516 -2.07503938  0.6604314  0.8072246 -0.56720016  0.25894891
##           [,44]      [,45]      [,46]      [,47]      [,48]      [,49]
## [1,]  0.1994826 0.85886193 -1.0272195 -0.6927835 -0.7249583  0.4837484
## [2,]  0.8503173 1.79610738 -0.5137749 -0.1555725  0.9786089  0.3395037
## [3,] -0.6075540 0.88898019 -1.8429097 -0.2414342 -0.3074160  0.8539705
## [4,] -2.0032724 0.58304548  0.1967549  0.3869047 -0.2489504 -1.0761628
## [5,] -3.1830351 0.32053132  0.1794494  0.9400340 -0.8938197  0.2003923
## [6,] -1.5571751 0.09951078  0.9262770  0.2463446 -1.1828865  0.8784514
##           [,50]       [,51]      [,52]      [,53]      [,54]      [,55]
## [1,] -1.6348254  0.05398376  0.8103526 -1.9644006  1.1116159  1.7717251
## [2,] -0.7676988 -0.09000027  0.5241017 -0.5561611  2.5121682  0.5698865
## [3,] -1.4110001 -0.12161192 -0.5918321  0.4345450  1.4049929  0.1162822
```

```
## [4,] -0.2510978  0.47181744 -3.1087029  0.5546181  0.6874636 -2.7220318
## [5,] -0.3325031  1.42923730 -1.2010310  0.9528205 -0.7267274 -1.8047585
## [6,]  0.5587052 -0.85087043 -0.8105048 -1.9638790 -0.6872734 -1.0146606
##             [,56]      [,57]      [,58]      [,59]        [,60]       [,61]
## [1,] -1.695299480 -0.7375152 -0.9941054 -0.5402266  0.945416988 -0.79979608
## [2,] -1.354694131 -1.9998298 -1.1048535  0.8740058  2.586310179  0.09748811
## [3,]  0.226683006 -1.5976354 -2.8140957 -0.1706973  1.697457705  0.01819052
## [4,]  0.007972794  1.2178193 -0.5695814  0.4632639  0.784868897 -0.36155374
## [5,] -0.295007347  1.1419091 -0.2095500  0.2180394 -0.005280912  1.49376789
## [6,]  0.083926181  1.1615915 -1.8687040 -1.6329345 -1.766995586  1.19781055
##           [,62]      [,63]     [,64]      [,65]      [,66]      [,67]
## [1,] -0.4625005 -3.2807307 -2.406089  2.6118173  1.3951817  0.1798267
## [2,] -0.7782710 -2.0662885 -2.260000  1.4849394  0.8742068 -0.2892613
## [3,]  0.2918426 -0.7177883 -2.083709  0.1768486 -1.2959984  0.4082488
## [4,]  0.4139972 -1.6346258 -1.171169  0.2489814  0.1423521  1.1299672
## [5,] -0.9989728 -1.7369372 -1.710855 -0.7075727  1.2257233  0.4432829
## [6,]  1.1055074 -0.6661461 -1.473087 -0.2675765  1.4674760 -1.1839577
##           [,68]        [,69]      [,70]       [,71]      [,72]       [,73]
## [1,]  0.1730520 -0.250645662 -0.2378786 -1.10501624  0.5383666 -0.26607227
## [2,]  0.9034188 -0.052439264 -1.2723075 -1.00821965  0.3185975  0.37599782
## [3,] -0.8911254 -0.134085343 -0.4445694  0.63030066 -0.2400064 -0.36314346
## [4,] -1.3483863  0.001661953 -1.0543220  0.08929446  1.4070769  0.03934903
## [5,] -0.9871889 -1.939457100 -1.1492760  1.33651950  3.1344317  0.07638156
## [6,] -1.3824993 -0.038115794 -1.4467955  1.07015676  1.2077716  3.00064273
##           [,74]      [,75]       [,76]       [,77]      [,78]      [,79]
## [1,]  0.8938640  0.7598160 -0.45854061 -2.17852093 -0.4905326 -1.3976363
## [2,] -0.5514428  0.2022892 -0.09348687 -1.50307220 -1.1371313  1.3991226
## [3,]  0.4436014 -0.2061349 -0.40619276  0.10666467 -1.2770661 -1.0871002
## [4,]  0.7135716 -0.5643501  0.10298238 -0.72664516 -1.8213049 -0.7127415
## [5,]  1.4401742 -2.1916591 -0.88267368 -0.74273152  0.1133994  0.8278199
## [6,]  1.4040670 -0.7164005  0.50536098  0.01394323  2.3636788 -0.8889834
##           [,80]      [,81]     [,82]        [,83]      [,84]       [,85]
## [1,]  0.3604537  0.8589769 2.7951731  0.437454120 -0.2474357  0.80360187
## [2,] -0.8206660  1.8137948 2.8451761  0.201705752  0.7237407  0.40554350
## [3,]  0.5258691 -0.1408078 0.6625379 -1.233671501  0.7086024  0.03158808
## [4,]  1.9351457 -0.2542828 0.5344799  0.005067087  0.5048241  0.26180640
## [5,]  2.5846371  0.2779521 1.0333876  1.062253326 -0.4302410  0.62604989
## [6,]  2.1189408  0.1454749 1.5293007 -0.365329977 -0.6975062 -0.27899180
##           [,86]      [,87]     [,88]       [,89]       [,90]       [,91]
## [1,] -0.5351086 -1.2773065 1.7320299 -0.13613027  0.82200530 -0.08470171
## [2,]  0.4049783 -0.7575080 0.4672331 -0.07183191  1.38516887 -0.56010849
## [3,] -0.2362536 -2.3282554 1.1585433 -0.47159946  1.65144518 -1.14919290
## [4,] -0.3914002 -1.2507008 3.6670366 -0.88068572 -0.05250206 -1.08600542
## [5,] -1.2124505  0.6959906 1.5249968 -0.99395345  0.90987436 -1.22513887
## [6,] -0.9446564 -1.3418153 2.1492708 -2.23892017  0.69256198 -0.33972569
##           [,92]      [,93]      [,94]      [,95]       [,96]      [,97]
## [1,] -0.4222803 -0.4280227 -2.4191456  0.3461769  1.33211557 -0.8678397
## [2,] -0.4561968 -0.6073220 -0.4809160  1.0649597 -0.12850092 -0.7807724
## [3,]  0.4937071 -2.4613629 -0.8480758 -0.6813204  1.87168883 -0.7255681
## [4,] -0.7291415 -0.6091274 -1.4399499  1.0483092  1.82802368 -0.7026647
## [5,] -0.6679330 -0.1836446 -1.1848595  0.3024109 -0.07854060 -0.9724064
## [6,] -0.6399924  0.7343935 -1.1630764  0.6554546  0.08635907 -1.4821627
##           [,98]      [,99]      [,100]
## [1,] -0.5064586  0.3804720  1.51521726
```

```
## [2,] -0.9350701  0.5523989  1.83580924
## [3,] -0.9148146  1.8895935  2.03752379
## [4,]  0.3329389  1.0391400 -0.33491623
## [5,]  0.1699752 -1.2713465  0.06212393
## [6,] -0.2475097  0.1336139 -1.88003835
```

```r
# Definir parámetros
r <- 0.5  # Umbral para la norma
d <- 2    # Dimensión del espacio fase

# Inicializar vectores para almacenar los resultados
X_N_values_ar <- numeric(n_series)
X_N_values_ar_d_1 <- numeric(n_series)

# Calcular X_{N}(r,d) para cada serie
for (i in 1:n_series) {
  series <- ar1_series[i, ]
  X_N_values_ar[i] <- X_N(series, r, d)
  X_N_values_ar_d_1[i] <- X_N_d_1(series, r)
}

# Ver los resultados
X_N_values_ar
```

```
##   [1] 1.4153467 1.2260086 1.0160907 1.3528157 1.2779553 1.2325467 1.3973092
##   [8] 1.4600922 1.1811395 1.2690281 1.0590223 0.9913069 1.1451857 1.0855774
##  [15] 1.1731342 1.6000328 1.1905977 1.1027910 1.0720222 1.2895902 1.0364274
##  [22] 1.2128722 1.2933986 1.1874215 1.3315915 1.0752489 1.6407513 0.8854215
##  [29] 1.0588235 1.1805296 1.2076149 1.2611420 0.9992071 1.3258145 1.7433914
##  [36] 1.1368003 1.0037143 1.4903366 1.4100141 1.1181655 1.2039637 0.9723242
##  [43] 1.2854975 1.0635787 1.0809102 1.1787093 1.2426226 1.8060484 1.1171603
##  [50] 1.0434783 1.5148195 1.3462604 1.0874882 1.0428402 1.2756465 1.2221412
##  [57] 0.9957721 1.3046365 1.1181655 1.0728265 0.9014581 1.0416304 1.1867357
##  [64] 1.1139828 0.8682211 1.4094460 1.2603945 1.2641134 0.8670613 1.1675930
##  [71] 1.2854453 1.3361944 1.3941006 1.5576236 1.1422392 1.3937842 1.2371564
##  [78] 1.1537914 1.1268189 1.0833581 1.1576078 1.1529047 1.0504377 1.4028672
##  [85] 0.9981435 1.4682624 0.9366088 1.3324561 1.0767123 1.3051242 1.3768637
##  [92] 1.4469183 1.5757512 0.9641046 1.7894664 1.1079391 1.1012514 1.3480649
##  [99] 1.3690118 1.2296015
```

```r
X_N_values_ar_d_1
```

```
##   [1] 1.186282 1.311329 1.325994 1.097682 1.306253 1.438425 1.236076 1.322725
##   [9] 1.378978 1.287028 1.368197 1.312721 1.301907 1.118150 1.306323 1.349177
##  [17] 1.396945 1.418778 1.241904 1.305478 1.248103 1.245704 1.365661 1.338625
##  [25] 1.227622 1.317061 1.267556 1.397559 1.233067 1.284085 1.312780 1.386321
##  [33] 1.297856 1.384357 1.223446 1.238132 1.196608 1.307539 1.257013 1.292082
##  [41] 1.182380 1.262535 1.217832 1.259117 1.278055 1.198915 1.483973 1.457017
##  [49] 1.226215 1.224257 1.335263 1.448011 1.329375 1.158317 1.369084 1.174375
##  [57] 1.323450 1.149987 1.292082 1.336198 1.311870 1.281038 1.298815 1.234190
##  [65] 1.250632 1.180038 1.254407 1.387051 1.426750 1.293466 1.267301 1.363468
##  [73] 1.368155 1.213098 1.383125 1.250899 1.233161 1.377626 1.124054 1.269956
##  [81] 1.404237 1.366132 1.409956 1.183277 1.267919 1.398224 1.428711 1.278971
```

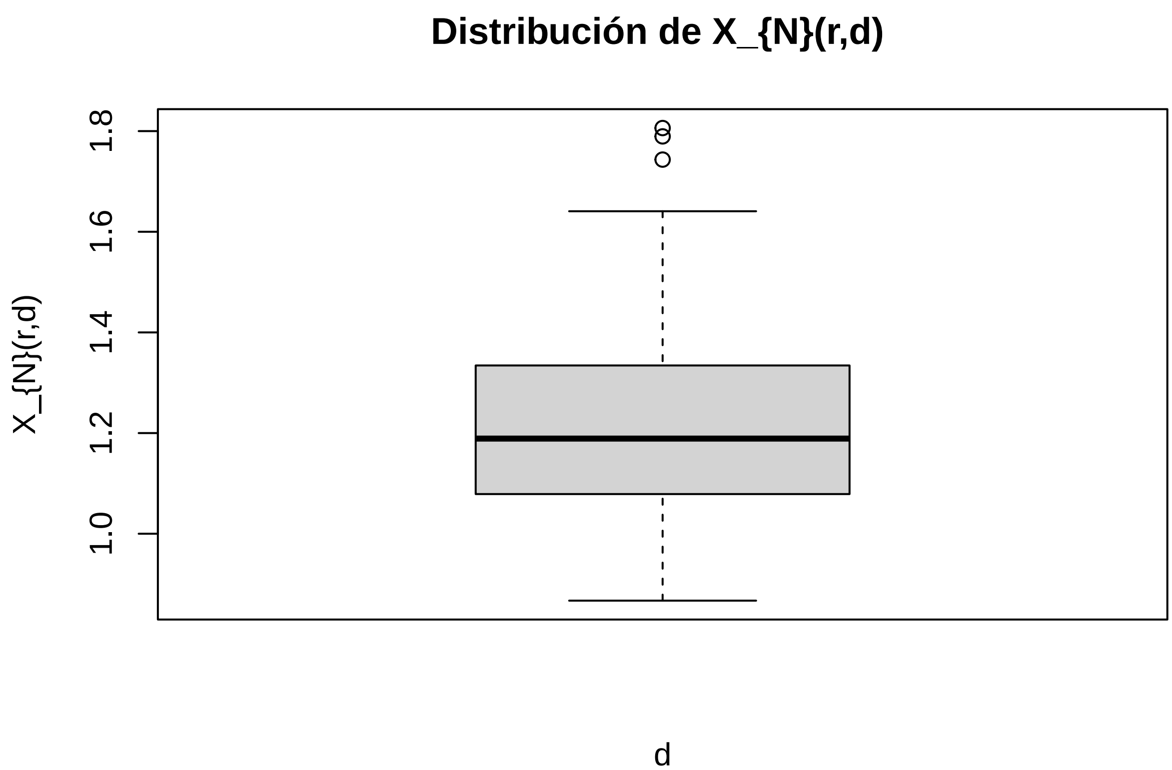
```
## [89] 1.184645 1.348735 1.321172 1.193104 1.430917 1.223530 1.184425 1.214950
## [97] 1.352362 1.367180 1.357012 1.234351
```

Veamos como se comporta X_{N}(r,d) para este caso

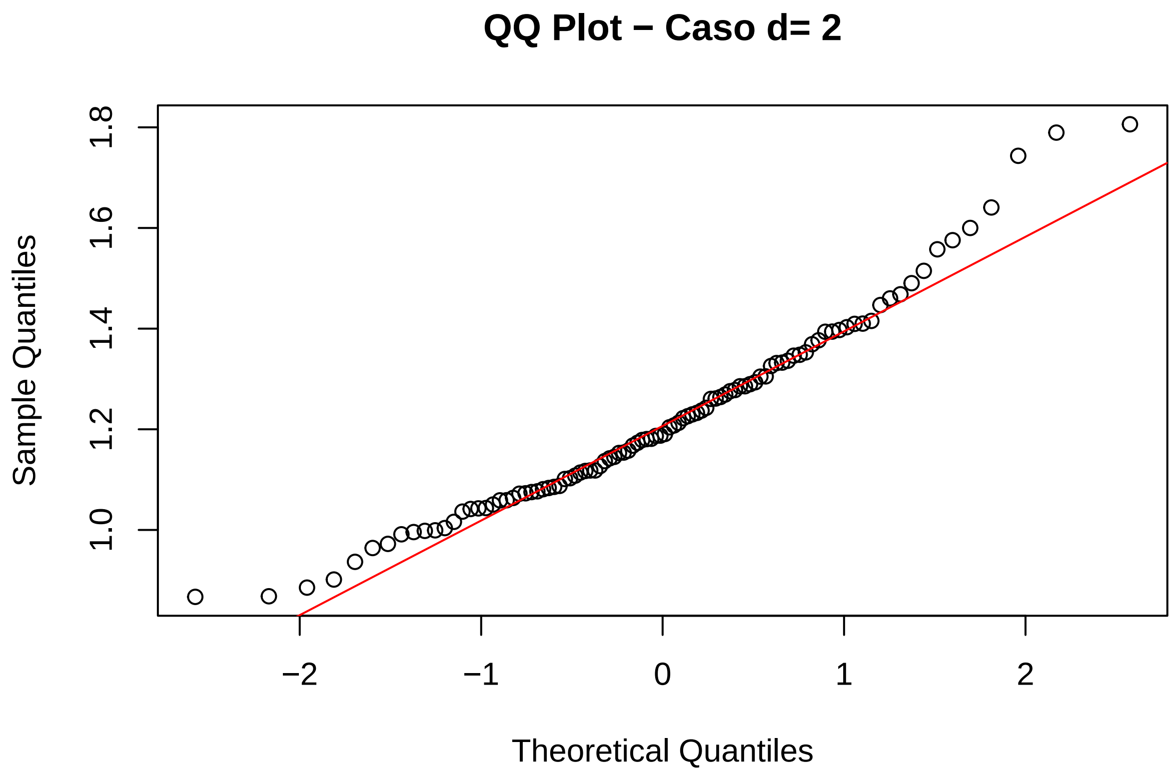
```r
# Resumen de los resultados
summary(X_N_values_ar)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8671  1.0799  1.1890  1.2209  1.3334  1.8060
```

```r
# Graficar los resultados
boxplot(X_N_values_ar, main="Distribución de X_{N}(r,d) ", ylab="X_{N}(r,d)", xlab="d")
```




```r
# Crear el QQ plot


  qqnorm(X_N_values_ar, main = paste("QQ Plot - Caso d=", 2))
  qqline(X_N_values_ar, col = "red")
```



calculamos $\sqrt{N} ln(X_N(r, d))$

```
# Longitud de cada serie
N <- length_series  # Esto es igual a 1000 en nuestro caso

# Calcular sqrt(N)
sqrt_N <- sqrt(N)

# Calcular sqrt(N) * ln(X_{N}(r,d)) para cada serie
sqrt_N_ln_X_N_ar <- sqrt_N * log(X_N_values_ar)

# Ver los resultados
head(sqrt_N_ln_X_N_ar)
```

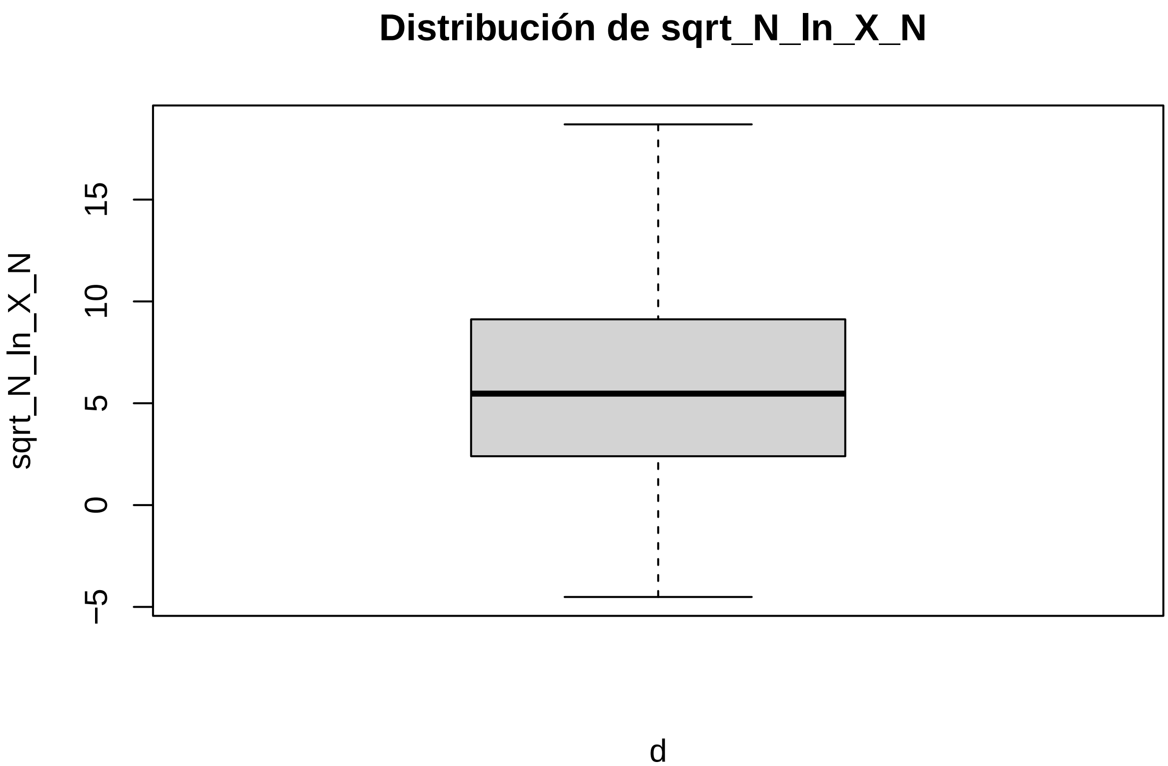
```
## [1] 10.9849478  6.4435784  0.5047827  9.5560272  7.7558451  6.6117686
```

y vemos como se comporta:

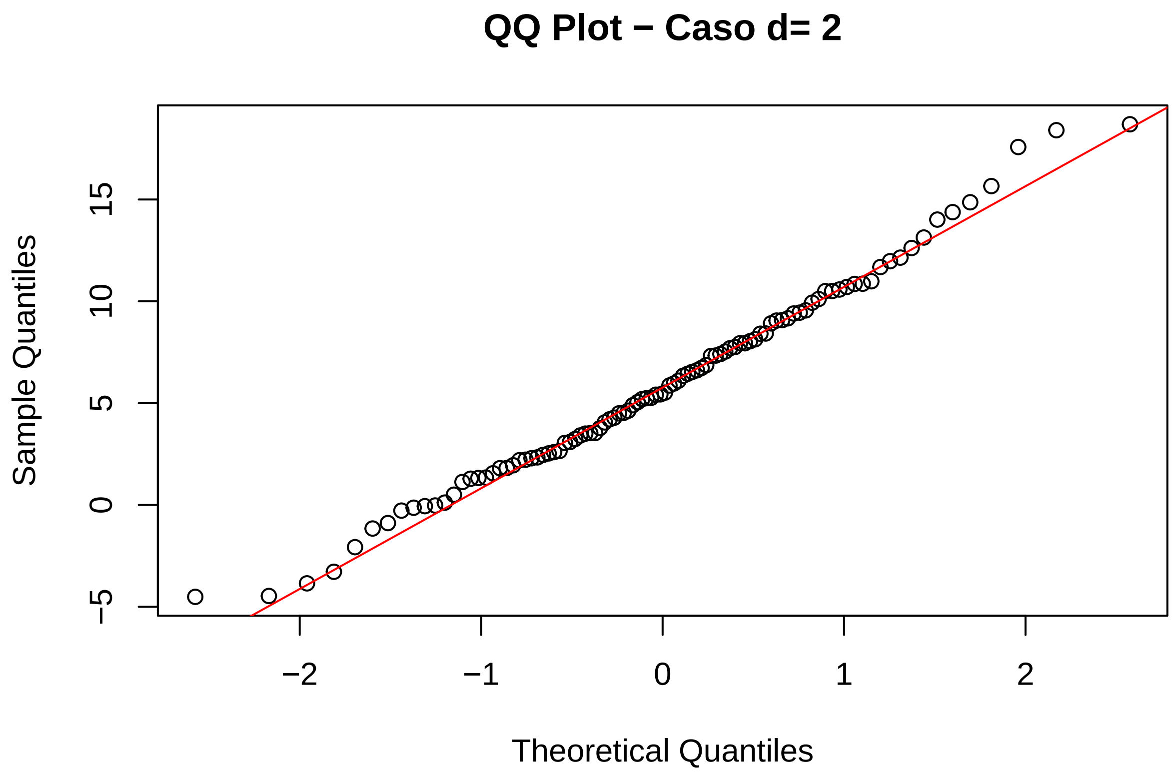
```
# Resumen de los resultados
summary(sqrt_N_ln_X_N_ar)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -4.511   2.430   5.475   5.930   9.099  18.694
```

```r
# Graficar los resultados
boxplot(sqrt_N_ln_X_N_ar, main="Distribución de sqrt_N_ln_X_N ", ylab="sqrt_N_ln_X_N", xlab="d")
```




```r
# Crear el QQ plot

  qqnorm(sqrt_N_ln_X_N_ar, main = paste("QQ Plot - Caso d=", 2))
  qqline(sqrt_N_ln_X_N_ar, col = "red")
```



Pruebas de estimaciones para la varianza, aun con problemas debido a los problemas heredados de las funciones de correlacion integral y X_{N}(r,d), falta por investigar metodos de de estimación.

```
# r constante dada
r <- 1

# Matriz para guardar los resultados
n_series <- nrow(data_matrix)
d_values <- 1:10
correlation_results <- matrix(NA, nrow = n_series, ncol = length(d_values))
colnames(correlation_results) <- paste0("d = ", d_values)

# Aplicar la correlación integral a cada serie por cada d
for (i in 1:n_series) {
  for (d in d_values) {
    correlation_results[i, paste0("d = ", d)] <- correlation_integral(data_matrix[i, ], r, d)
  }
}


head(correlation_results)
```

```
##         d = 1    d = 2    d = 3    d = 4    d = 5    d = 6    d = 7    d = 8
## [1,] 0.522922 0.228284 0.085330 0.028032 0.008380 0.002258 0.000528 0.000128
## [2,] 0.510966 0.212902 0.076502 0.024830 0.007370 0.001966 0.000446 0.000092
## [3,] 0.537422 0.236896 0.090718 0.031068 0.009674 0.002778 0.000738 0.000154
```

```
## [4,] 0.526872 0.224786 0.082322 0.027194 0.008160 0.002272 0.000590 0.000134
## [5,] 0.527422 0.226894 0.084484 0.028182 0.008426 0.002238 0.000526 0.000114
## [6,] 0.537484 0.235350 0.090316 0.031360 0.009812 0.002728 0.000672 0.000154
##        d = 9 d = 10
## [1,] 2.8e-05  4e-06
## [2,] 1.2e-05  2e-06
## [3,] 3.8e-05  1e-05
## [4,] 2.6e-05  6e-06
## [5,] 1.4e-05  0e+00
## [6,] 4.2e-05  6e-06
```

```r
# Calculo de Varianza
calculate_variance <- function(series) {
  return(var(series, na.rm = TRUE))
}

# Función para calcular *sigma_r_d^2 para cada d
calculate_star_sigma <- function(variance_values, correlation_results, r, d, series_index) {
  # Calculos sacados de la matriz anterior de correlaciones integrales
  C_N_2d_minus_2 <- correlation_results[series_index, paste0("d = ", 2*(d-1))]
  C_N_2d <- correlation_results[series_index, paste0("d = ", 2*d)]
  C_N_2d_plus_2 <- correlation_results[series_index, paste0("d = ", 2*(d+1))]

  # Obtener las varianzas según cada d
  sigma_r_d_minus_1_sq <- variance_values[d-1]
  sigma_r_d_sq <- variance_values[d]
  sigma_r_d_plus_1_sq <- variance_values[d+1]

  # Calcular *sigma_r_d^2
  star_sigma_r_d_sq <- (sigma_r_d_minus_1_sq / C_N_2d_minus_2) +
                       (4 * sigma_r_d_sq / C_N_2d) +
                       (sigma_r_d_plus_1_sq / C_N_2d_plus_2)

  return(star_sigma_r_d_sq)
}

# r constante dada
r <- 0.5

# Calcular varianzas para cada d por cada serie en data_matrix
n_series <- nrow(data_matrix)
d_values <- 2:4  # para d = 2, 3, 4

# Matriz para guardar los resultados star_sigma_r_d^2
star_sigma_results <- matrix(NA, nrow = n_series, ncol = length(d_values))
colnames(star_sigma_results) <- paste0("d = ", d_values)

# Loop um cada serie para calcular *sigma_r_d^2
for (i in 1:n_series) {
  # Calcular las varianzas para cada serie
  variance_values <- sapply(1:10, function(d) calculate_variance(data_matrix[i, ]))

  # Calcular *sigma_r_d^2 para cada d entre 2 y 4
  for (d in d_values) {
```

```r
    star_sigma_results[i, paste0("d = ", d)] <- calculate_star_sigma(variance_values, correlation_resul
  }
}


head(star_sigma_results)
```

```
##         d = 2     d = 3    d = 4
## [1,] 578.4638  9432.450 276211.0
## [2,] 704.5862 13523.030 568302.6
## [3,] 455.6538  7363.024 116777.5
## [4,] 570.3348  8925.950 189851.9
## [5,] 565.0788 10092.991      Inf
## [6,] 471.4931  7560.698 182598.8
```

```r
# r constante dada
r <- 0.5

# Matriz para guardar los resultados
n_series <- nrow(data_matrix)
d_values <- 1:10
correlation_results <- matrix(NA, nrow = n_series, ncol = length(d_values))
colnames(correlation_results) <- paste0("d = ", d_values)

# Aplicar la correlación integral a cada serie por cada d
for (i in 1:n_series) {
  for (d in d_values) {
    correlation_results[i, paste0("d = ", d)] <- correlation_integral(data_matrix[i, ], r, d)
  }
}


head(correlation_results)
```

```
##         d = 1    d = 2    d = 3    d = 4    d = 5   d = 6 d = 7 d = 8 d = 9
## [1,] 0.277856 0.063116 0.012138 0.002116 0.000340 5.6e-05 4e-06 0e+00     0
## [2,] 0.270842 0.058310 0.010756 0.001788 0.000246 2.6e-05 2e-06 0e+00     0
## [3,] 0.286776 0.065324 0.012712 0.002174 0.000292 3.4e-05 8e-06 2e-06     0
## [4,] 0.279504 0.061572 0.011442 0.001876 0.000294 4.8e-05 8e-06 0e+00     0
## [5,] 0.280374 0.061972 0.011634 0.001874 0.000290 3.6e-05 0e+00 0e+00     0
## [6,] 0.287238 0.064994 0.012750 0.002248 0.000342 5.6e-05 6e-06 2e-06     0
##      d = 10
## [1,]      0
## [2,]      0
## [3,]      0
## [4,]      0
## [5,]      0
## [6,]      0
```