Séries temporelles : TD et Examens

2024-12-11

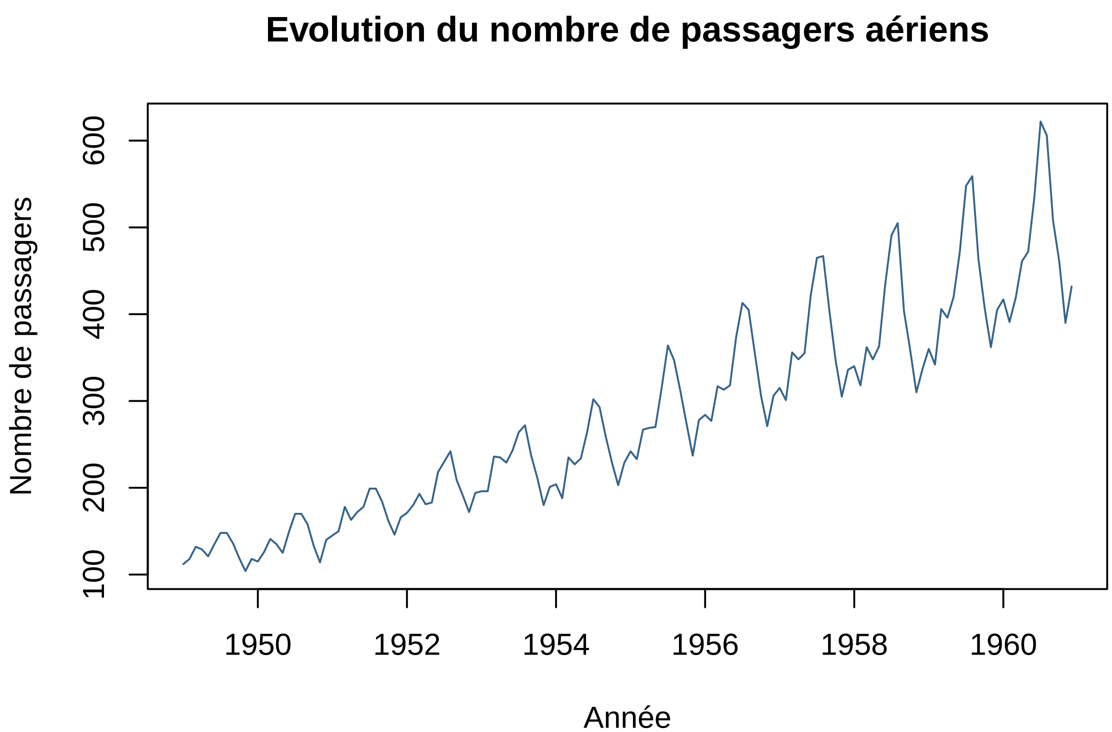
Contents

1 Avants propos, notions cours et codes associés :

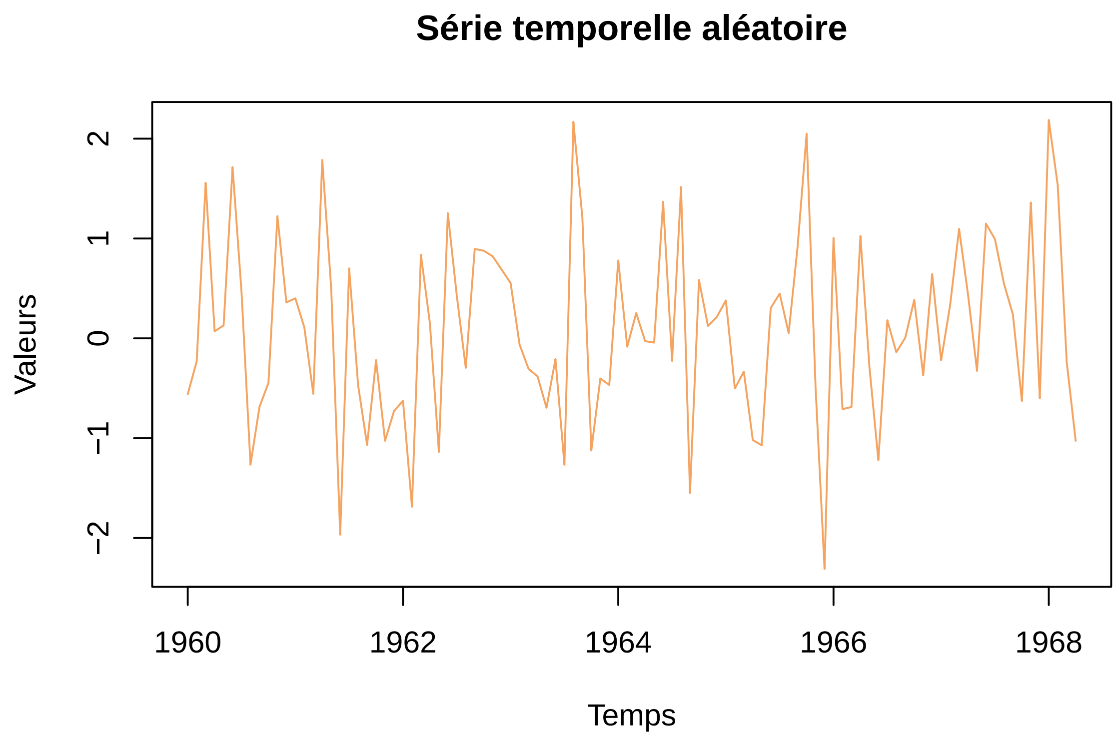
1.1 Le type ts 'time series' :

1.1.1 Exemples et création :

```r
# Exemple de série temporelle :
plot(
  AirPassengers,
  main = "Evolution du nombre de passagers aériens",
  ylab = "Nombre de passagers",
  xlab = "Année",
  type = "l",
  col = palette_couleur[1]
)
```

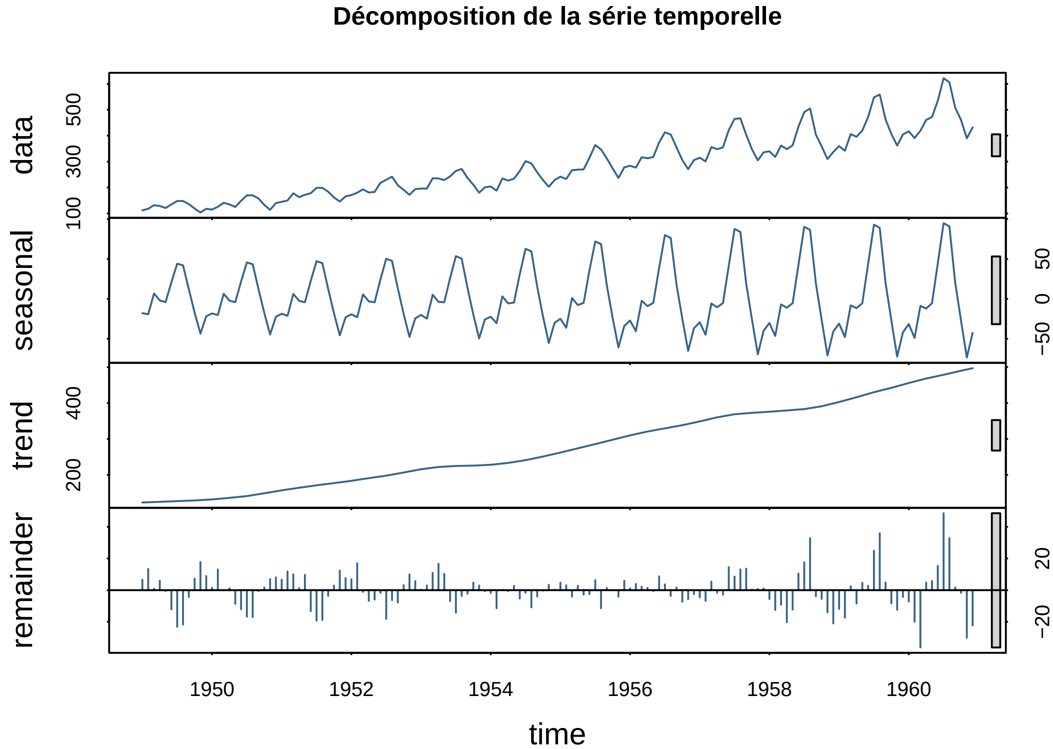



```r
# Création d'un objet time série aléatoire :

serie_temp = ts(rnorm(100), start = c(1960, 1), frequency = 12)
plot(
  serie_temp,
  main = "Série temporelle aléatoire",
  ylab = "Valeurs",
  xlab = "Temps",
  type = "l",
  col = palette_couleur[2]
)
```



1.1.2 Décomposition d'une série temporelle :

```
plot(
  stl(AirPassengers, s.window = 12),
  main = "Décomposition de la série temporelle",
  col = palette_couleur[1],
  lwd = 1
)
```

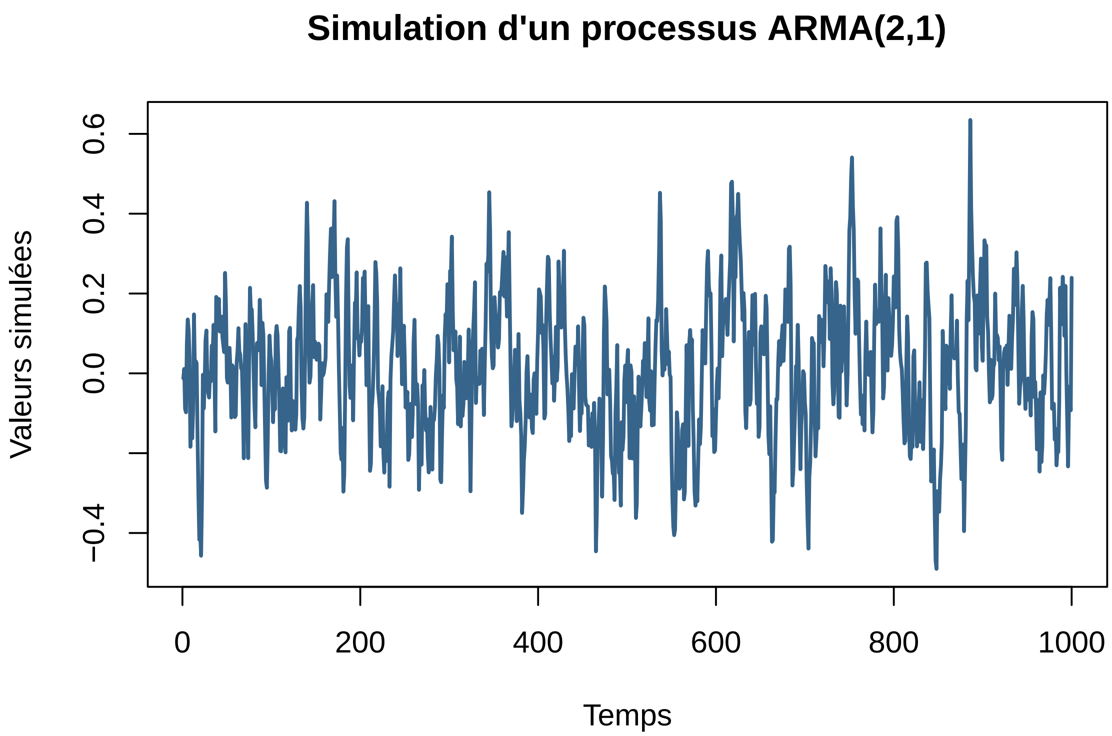



```
# s.window = 12 : période de saisonnalité (nb de mois)
```

1.2 Simulation de processus :

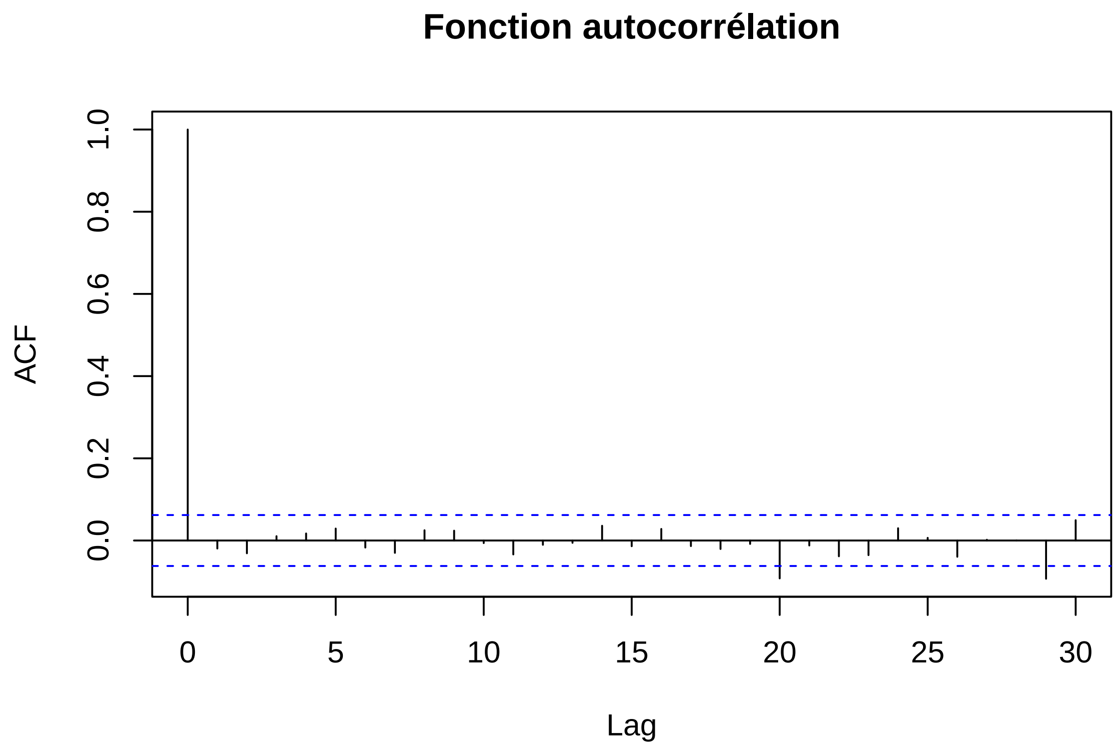
1.2.1 Simulation ARMA(2,1) :

```
alpha = c(.5, .2) #paramètres AR
beta = .5 #paramètres MA
sig = .1 #écart-type du bruit
Time = 1000 #longueur de la série temporelle
x = arima.sim(model = list(ar = alpha, ma = beta),
              sd = sig,
              n = Time) #simulation modèle ARMA
plot(x, type = "l", col = palette_couleur[1], lwd = 2,
     main = "Simulation d'un processus ARMA(2,1)",
     ylab = "Valeurs simulées", xlab = "Temps")
```

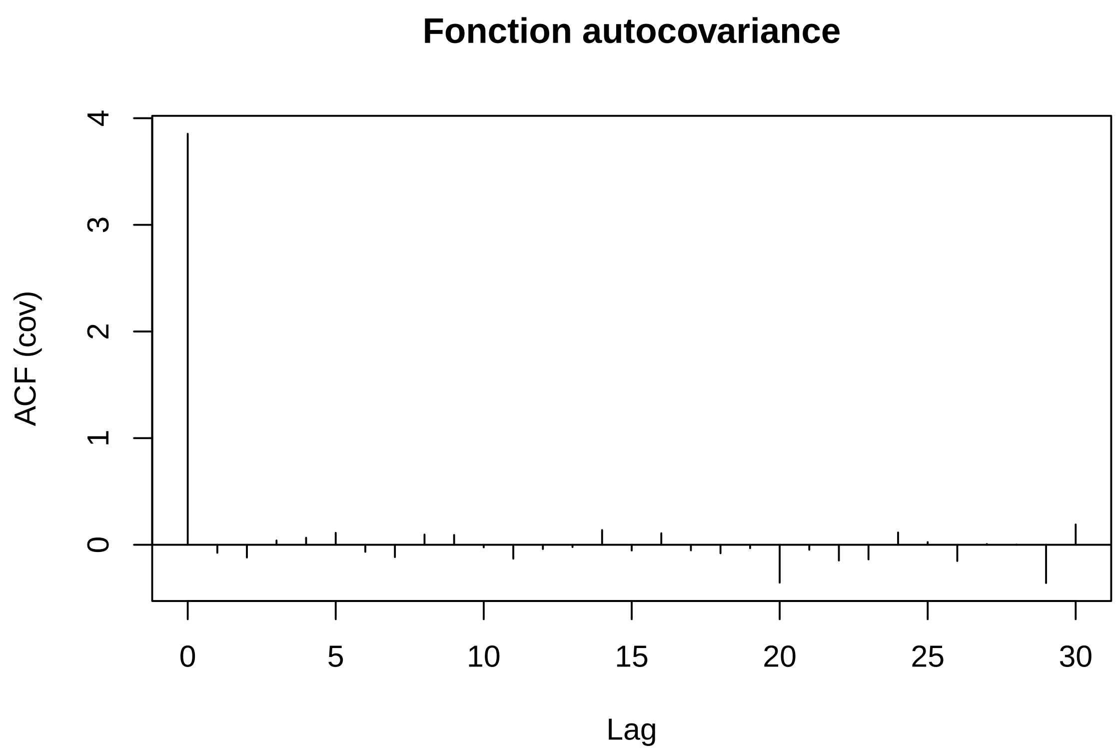


1.3 Fonctions d'autocorrélation et d'autocovariance :

```
Time = 1000
x = 2 * rnorm(Time) #simulation d'un bruit blanc gaussien
acf(x, main = "Fonction autocorrélation")
```



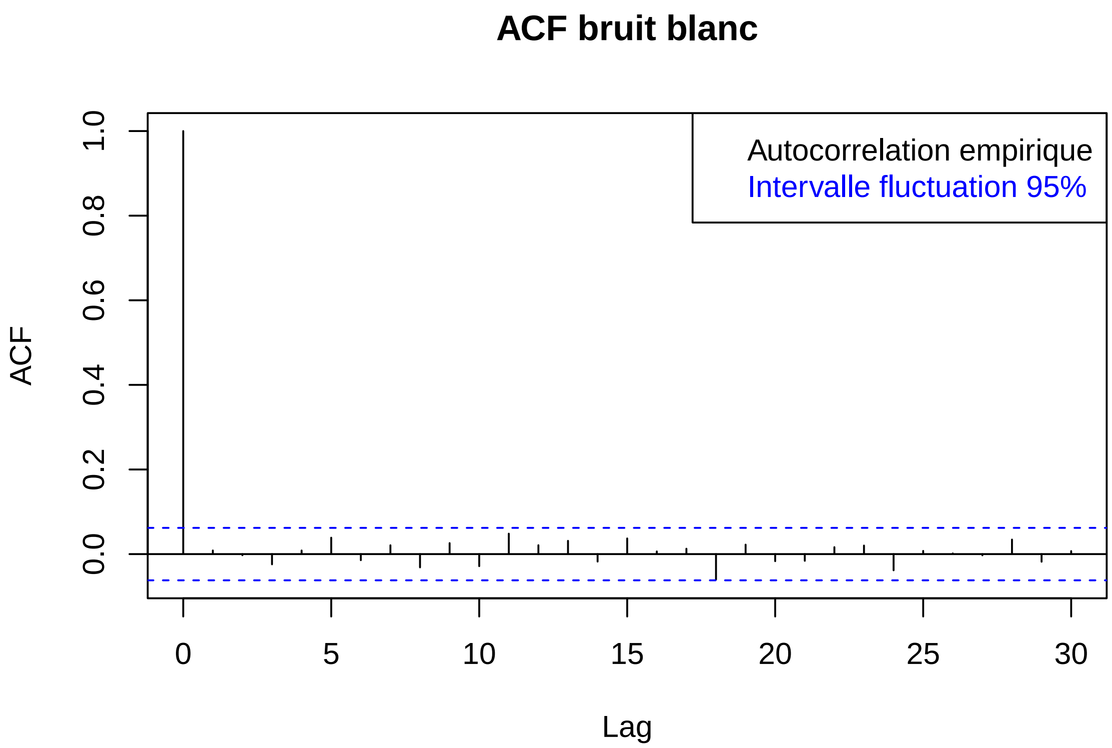

```r
acf(x, type = 'covariance', main = "Fonction autocovariance")
```



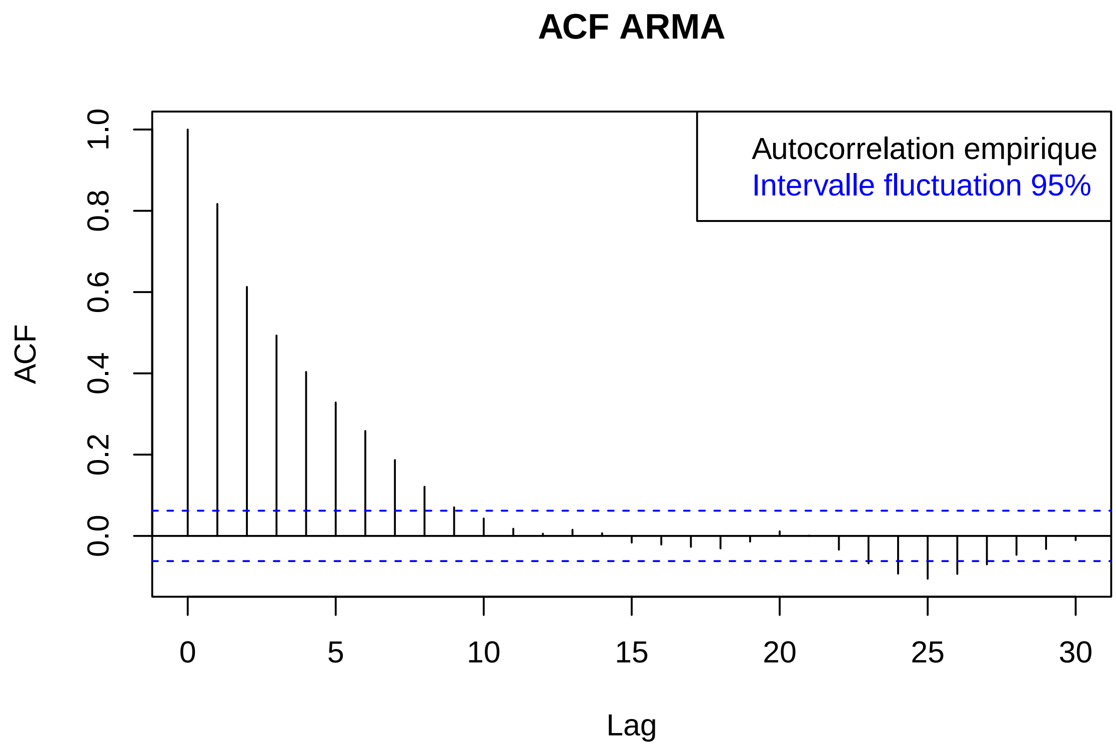
1.4 Etude d'un bruit blanc :

1.4.1 Fonctions d'autocorrélation et d'autocovariance :

```
# Simulation bruit blanc gaussien :
Time = 1000
bbg = rnorm(Time)

acf(bbg, main = 'ACF bruit blanc')
legend(
  'topright',
  c('Autocorrelation empirique', 'Intervalle fluctuation 95%'),
  text.col = c('black', 'blue'))
```




```
x = arima.sim(model = list(ar = c(.5, .2), ma = .5),
              sd = .1,
              n = Time)
acf(x, main = 'ACF ARMA')
legend(
  'topright',
  c('Autocorrelation empirique', 'Intervalle fluctuation 95%'),
  text.col = c('black', 'blue'))
```



1.4.2 Tests statistiques sur le bruit blanc :

Le test de Box-Pierce permet de tester l'hypothèse nulle d'indépendance des observations d'une série temporelle.

Le test de Ljung-Box est une généralisation du test de Box-Pierce qui permet de tester l'indépendance des observations d'une série temporelle sur plusieurs retards.

$$\begin{cases} H_0 : \rho(1) = 0, \text{X est un bruit blanc} \\ H_1 : \rho(1) \neq 0, \text{X n'est pas un bruit blanc} \end{cases}$$

```r
Time = 1000
x = rnorm(Time) #simulation d'un bruit blanc gaussien
ar21 = arima.sim(model = list(ar = c(.5, .2), ma = .5),
              sd = .1, n = Time)

p1 = Box.test(x) #(test uniquement sur rho(1))
p2 = Box.test(x, lag = 10) #(test sur rho(1),...,rho(10))
p3 = Box.test(x, lag = 10, type = 'Ljung-Box') #(test de Ljung-Box)
p4 = Box.test(ar21) #(test uniquement sur rho(1))

p_value = round(c(p1$p.value, p2$p.value, p3$p.value, p4$p.value),4)

data.frame(
  Test = c(
    "BB Box rho(1)",
    "BB Box rho(1:10)",
```

```r
    "BB Lunj-Box rho(1:10)",
    "Arma21 rho(1)"),
  p_value = p_value,
  Bruit_Blanc = p_value > .05
  )
```

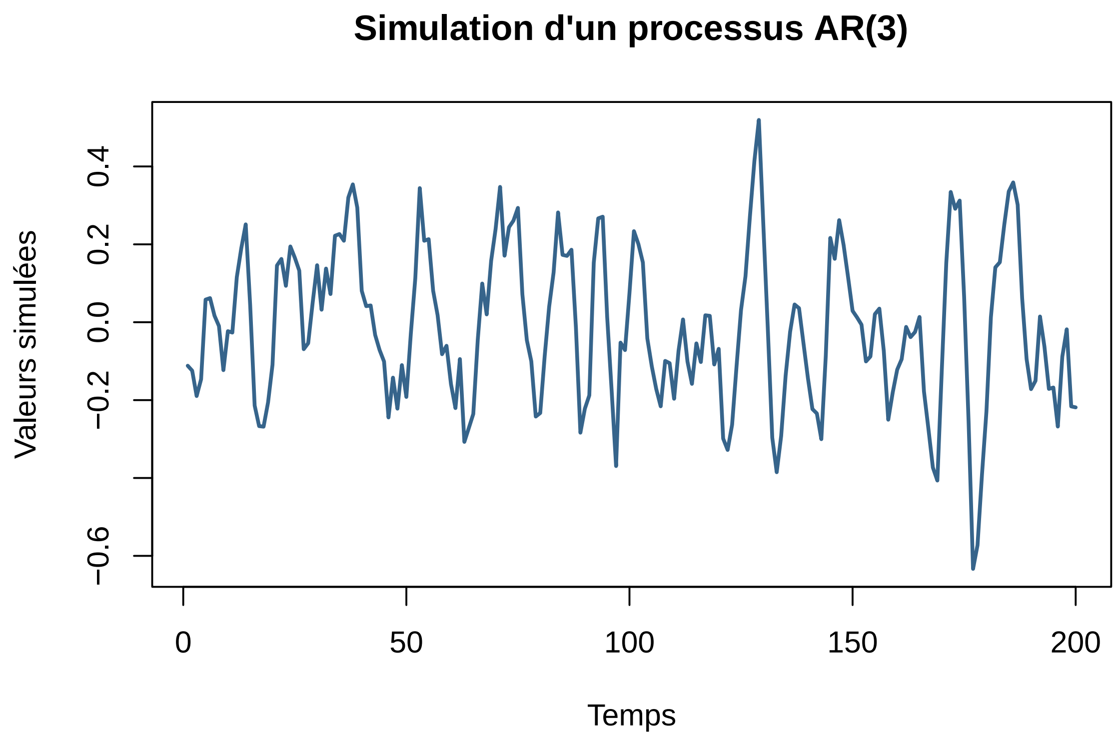
```
                   Test p_value Bruit_Blanc
1         BB Box rho(1)  0.2375        TRUE
2      BB Box rho(1:10)  0.3597        TRUE
3 BB Lunj-Box rho(1:10)  0.3544        TRUE
4         Arma21 rho(1)  0.0000       FALSE
```

1.5 Simulation d'un modèle AR(p) :

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \alpha_3 X_{t-3} + \epsilon_t \text{ avec } \epsilon_t \sim \mathcal{N}(0,1)$$

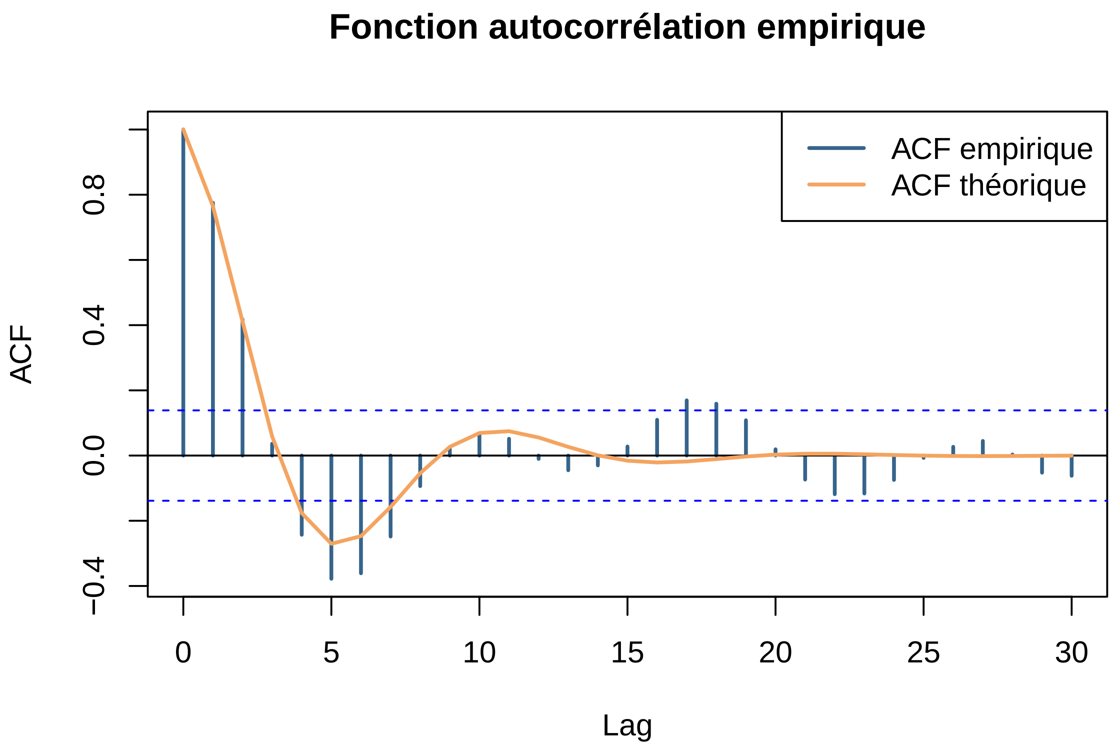
1.5.1 Simulation d'un processus AR(3) :

```r
alpha = c(1,-.2,-.2)
sig = .1
p = 3
x = arima.sim(model = list(ar = alpha),
              sd = sig, n = 200)
plot(
  x,
  type = "l",
  col = palette_couleur[1],
  lwd = 2,
  main = "Simulation d'un processus AR(3)",
  ylab = "Valeurs simulées",
  xlab = "Temps"
)
```



1.5.2 Calcul de la fonction d'autocorrélation théorique :

```
lmax = 30
acfth = ARMAacf(ar = alpha, lag.max = lmax) #fonction autocorrélation théorique
acfemp = acf(
  x,
  lag.max = lmax,
  main = "Fonction autocorrélation empirique",
  col = palette_couleur[1],
  lwd = 2
) #fonction autocorrélation empirique
lines(acfemp$lag, acfth, col = palette_couleur[2],
      lwd = 2) #comparaison

legend(
  'topright',
  c('ACF empirique', 'ACF théorique'),
  col = palette_couleur[1:2],
  lty = 1,
  lwd = 2
)
```



1.5.3 Ajustement d'un modèle AR(p) avec ou sans connaissance de p :

```
# Ajustement d'un modèle AR(p) avec connaissance de p :
y = ar(x, aic = FALSE, order.max = 3, demean = FALSE)
cat("Paramètres estimés en connaissant p : \n")
```

```
Paramètres estimés en connaissant p :
```

```
round(y$ar, 4)
```

```
[1]  1.0153 -0.1722 -0.2527
```

```
# Ajustement d'un modèle AR(p) sans connaissance de p :
y2 = ar(x)
cat("Paramètres estimés sans connaître p : \n")
```

```
Paramètres estimés sans connaître p :
```

```
round(y2$ar, 4) # Selection de l'ordre optimal
```

```
[1]  1.0142 -0.1711 -0.2547
```

1.5.4 Simulation d'un processus AR(3) non-centré :

```
alpha = c(1,-.2,-.2)
sig = .1
p = 3
x = arima.sim(model = list(ar = alpha),
              sd = sig, n = 200)
```

```r
y = x + 1
fit = ar(y, aic = FALSE, order.max = 3, demean = TRUE)
# Demean = TRUE pour centrer la série temporelle

cat('Estimation de la moyenne : ', fit$x.mean, '\n')
```

```
Estimation de la moyenne :  1.019322
```

1.5.5 Ajustement d'un modele AR(p) avec la fonction arima :

Commentaire :

Par défaut la fonction arima centre la série temporelle.

Inclure la moyenne dans le modèle comme intercept permet de centrer la série temporelle. (option demean = TRUE) dans la fonction ar et inclure.mean = TRUE dans la fonction arima.

```

Call:
arima(x = y, order = c(3, 0, 0), include.mean = TRUE)

Coefficients:
         ar1      ar2      ar3  intercept
      0.9315  -0.0868  -0.1523     1.0184
s.e.  0.0696   0.0959   0.0701     0.0207

sigma^2 estimated as 0.008143:  log likelihood = 196.71,  aic = -383.42


Call:
arima(x = y, order = c(3, 0, 0), include.mean = FALSE)

Coefficients:
         ar1      ar2     ar3
      1.0966  -0.1198  0.0175
s.e.  0.0707   0.1048  0.0707

sigma^2 estimated as 0.009753:  log likelihood = 176.94,  aic = -345.88
```

1.5.6 Intervalle de prédiction du modèle AR(3) :

```r
confint(fit, level = .95) #intervalle de confiance
```

```
               2.5 %      97.5 %
ar1        0.7949985  1.06793797
ar2       -0.2746723  0.10113101
ar3       -0.2896533 -0.01502755
intercept  0.9779636  1.05892772
```

1.5.7 Prédiction d'un modèle AR(3) :

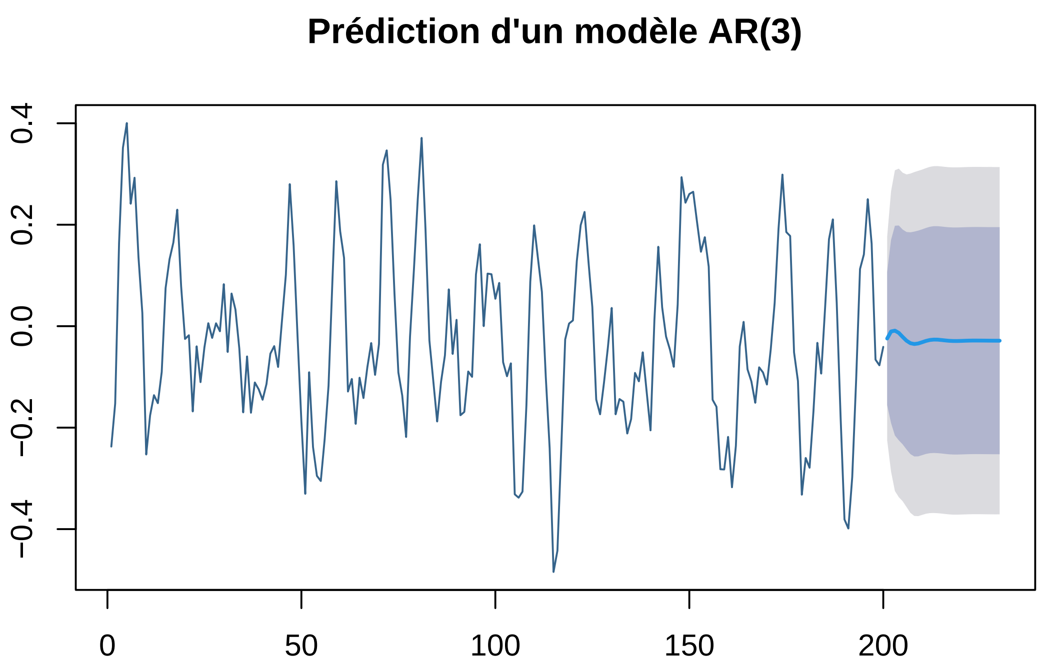
```r
x = arima.sim(model = list(ar = c(1, -.2, -.2)),
              sd = .1, n = 200)
fit = ar(x, aic = FALSE, order.max = 3, demean = TRUE)
pred = predict(fit, n.ahead = 3)
pred
```

```
$pred
Time Series:
Start = 201
End = 203
Frequency = 1
[1] -0.024318171 -0.010415514 -0.008772933

$se
Time Series:
Start = 201
End = 203
Frequency = 1
[1] 0.1024022 0.1408759 0.1613118
```

1.5.7.1 Représentation graphique de la prédiction : Commentaire :

L'intervalle de prédiction est compris entre 80% et 95% de confiance.

```r
library(forecast)
pm <- forecast(fit, h = 30)
plot(pm, main = "Prédiction d'un modèle AR(3)", col = palette_couleur[1])
```



1.6 Modélisation ARMA(p,q) :

1.6.1 Ajustement d'un modèle ARMA(p,q) quand p et q sont connus :

```r
set.seed(123)
p = 2 # ordre AR
```

```r
q = 2 # ordre MA
alpha = c(0.9,-0.2) # coefficients AR
beta = c(0.8,0.5) # coefficients MA
sigma = 1 # écart-type du bruit
n = 10^3 # nombre de simulations
x = arima.sim(model = list(ar = alpha, ma = beta), sd = sigma, n = n) # simulation
mod = arima(x, order = c(p,0,q))

dt = data.frame(
  alpha_reel = alpha,
  alpha_estim = mod$coef[1:p],
  beta_reel = beta,
  beta_estim = mod$coef[(p+1):(p+q)])
rownames(dt) = c('lag1', 'lag2')
round(dt, 4)
```

```
     alpha_reel alpha_estim beta_reel beta_estim
lag1        0.9      0.8322       0.8     0.8462
lag2       -0.2     -0.1636       0.5     0.5273
```

Commentaire :

Les paramètres estimés sont proches des paramètres réels. Une augmentation du nombre de simulations permet d'améliorer la précision des estimations.

1.6.2 Ajustement d'un modèle ARMA(p,q) quand p et q sont inconnus :

Commentaire : quand les paramètres sont inconnus on peut utiliser des critères d'informations tels que (AIC, BIC) pour sélectionner le meilleur modèle, on doit minimiser ces critères.

```r
SELECTION_ARMA <- function(x,pmax, qmax, print_result = TRUE) {
  AIC = matrix(0, nrow = pmax + 1, ncol = qmax + 1) #tableau pour stocker les valeurs du AIC
  for (p in 0:pmax) {
    for (q in 0:qmax) {
      AIC[p + 1, q + 1] = arima(x, order = c(p, 0, q))$aic #calcul du AIC
    }
  }

  colnames(AIC) <- paste("q =", 0:qmax)
  rownames(AIC) <- paste("p =", 0:pmax)
  #AIC

  # On renvoie l'indice du minimun de AIC :
  p = which(AIC == min(AIC), arr.ind = TRUE)[1] - 1
  q = which(AIC == min(AIC), arr.ind = TRUE)[2] - 1

  if(print_result == TRUE){
    cat("Meilleur modèle ARMA(p,q) : p = ", p, " q = ", q,
        "Valeur de l'AIC : ", min(AIC), "\n", "\n")
    cat("Tableau des AIC : \n")
    #AIC
  }
  return(list(AIC, p, q))
}
```

```r
# Fonction de sélection :
SELECTION_ARMA(x, pmax = 5, qmax = 5)
```

1.6.2.1 Méthode de sélection par boucle :

```
Meilleur modèle ARMA(p,q) : p =  2  q =  2 Valeur de l'AIC :  2854.911

Tableau des AIC :

[[1]]
         q = 0    q = 1    q = 2    q = 3    q = 4    q = 5
p = 0 5038.096 3993.394 3264.579 2974.258 2884.715 2864.898
p = 1 3560.485 3118.934 2861.834 2856.077 2856.491 2858.490
p = 2 3008.832 2948.911 2854.911 2856.711 2858.547 2861.214
p = 3 2918.651 2919.119 2856.651 2858.609 2860.586 2862.460
p = 4 2917.311 2893.297 2858.495 2860.460 2862.461 2864.433
p = 5 2883.806 2871.007 2860.461 2862.465 2864.442 2866.471

[[2]]
[1] 2

[[3]]
[1] 2
```

```r
library(forecast)
auto.arima(x, stationary = TRUE, seasonal = FALSE)
```

1.6.2.2 Méthode de sélection avec la fonction auto.arima :

```
Series: x
ARIMA(1,0,4) with zero mean

Coefficients:
         ar1     ma1     ma2     ma3     ma4
      0.5351  1.1427  0.8673  0.2508  0.0667
s.e.  0.0732  0.0783  0.1246  0.1129  0.0529

sigma^2 = 1.007:  log likelihood = -1421.46
AIC=2854.92   AICc=2855   BIC=2884.36
```

```r
# Paramétrages de la fonction :
auto.arima(x,stationary = TRUE,seasonal = FALSE,stepwise=FALSE,max.order=10,approximation = FALSE)
```

```
Series: x
ARIMA(2,0,2) with zero mean

Coefficients:
         ar1      ar2     ma1     ma2
      0.8323  -0.1632  0.8465  0.5274
s.e.  0.0575   0.0536  0.0512  0.0375

sigma^2 = 1.006:  log likelihood = -1421.67
AIC=2853.33   AICc=2853.39   BIC=2877.87
```

```r
# Sélection par la méthode du BIC
auto.arima(x,stationary = TRUE,seasonal = FALSE,ic="bic",max.order=10)
```

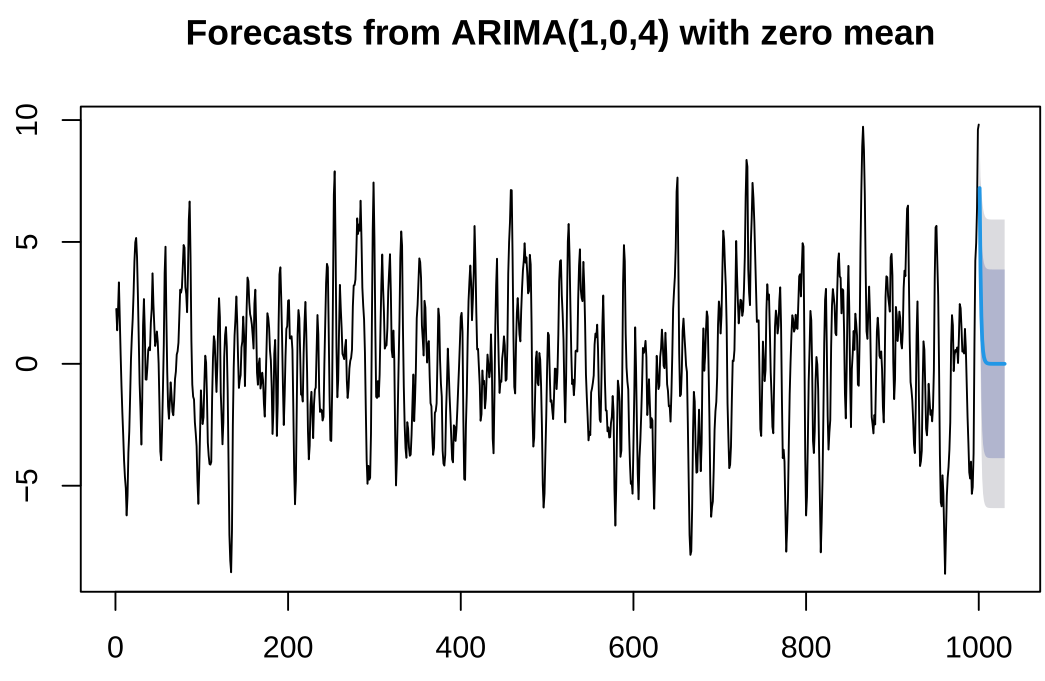
```
Series: x
ARIMA(1,0,3) with zero mean

Coefficients:
         ar1     ma1     ma2     ma3
      0.6043  1.0705  0.7373  0.1238
s.e.  0.0403  0.0473  0.0604  0.0437

sigma^2 = 1.007:  log likelihood = -1422.24
AIC=2854.48   AICc=2854.54   BIC=2879.02
```

Commentaire :

La méthode BIC a tendance à sélectionner moins de paramètres que la méthode AIC. Elle discrimine mieux les modèles complexes.

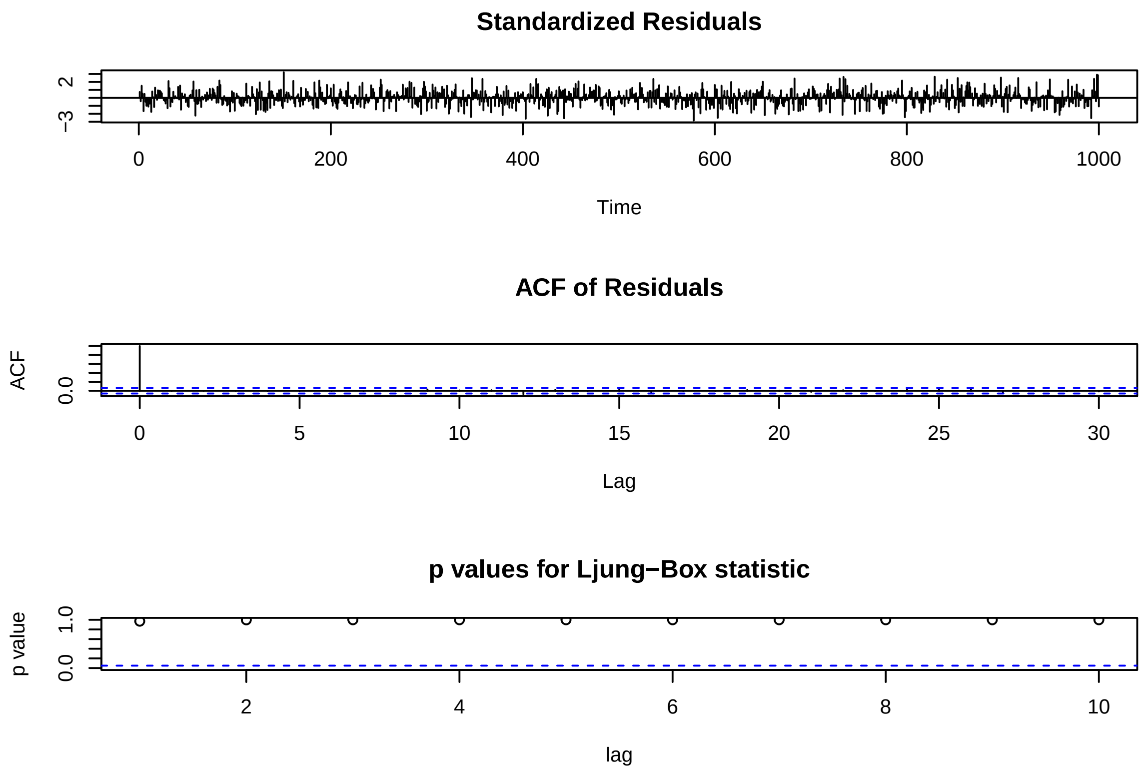
1.6.3 La prédiction des modélisations ARMA(p,q) :

```r
fit = auto.arima(x, stationary = TRUE, seasonal = FALSE)
#ajustement du modèle arma, inclut la sélection de modèle
pm <- forecast(fit, h = 30) #représentation graphique
plot(pm) # intervalles prédiction à 80% et 95%
```



1.6.4 Vérification de la stationnarité d'un processus ARMA(p,q) :

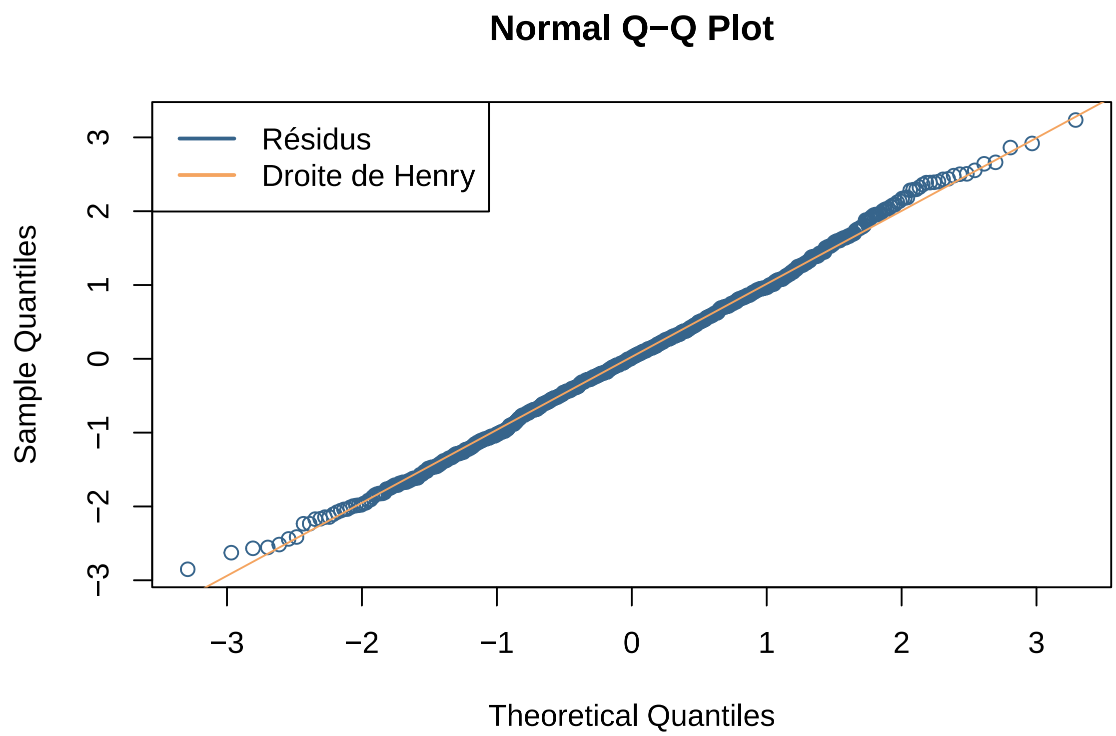
```
# Bruit blanc ?
tsdiag(fit)
```




```
# Test de gaussienneté des résidus :
shapiro.test(fit$residuals)
```

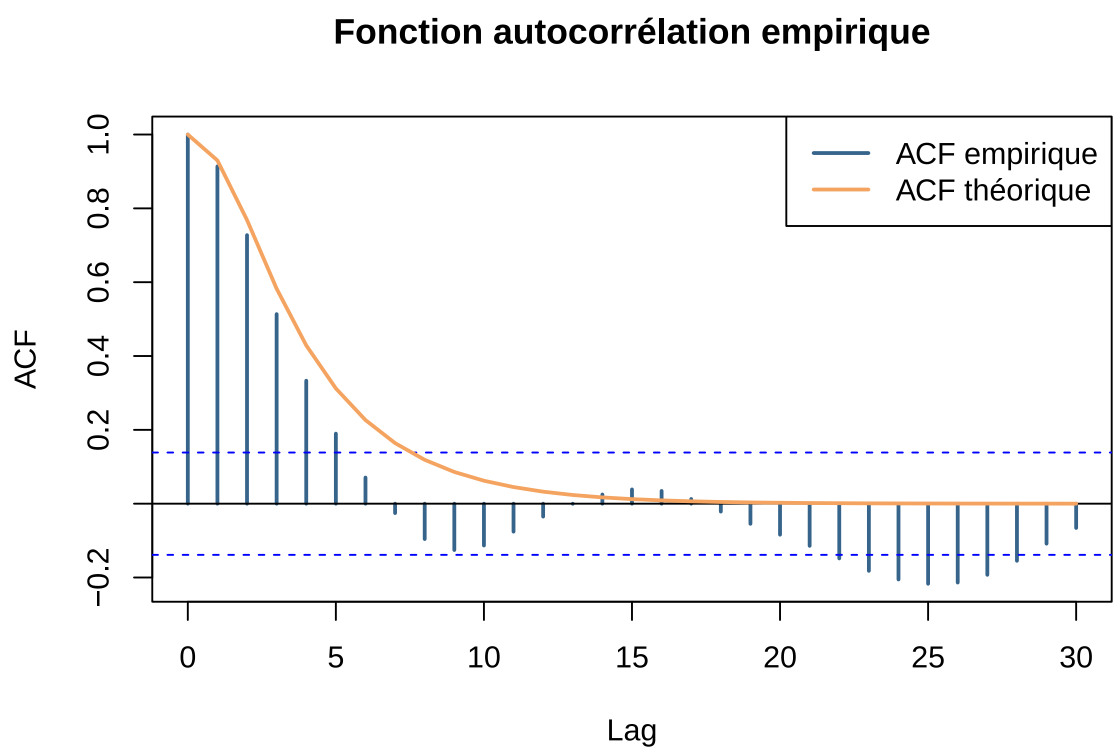
```
    Shapiro-Wilk normality test

data:  fit$residuals
W = 0.9981, p-value = 0.3252
```

```
qqnorm(fit$residuals, col = palette_couleur[1])
qqline(fit$residuals, col = palette_couleur[2])
legend('topleft', c('Résidus', 'Droite de Henry'), col = palette_couleur[1:2], lwd =2)
```



1.6.5 Fonction auto-corrélation d'un processus ARMA(p,q) :

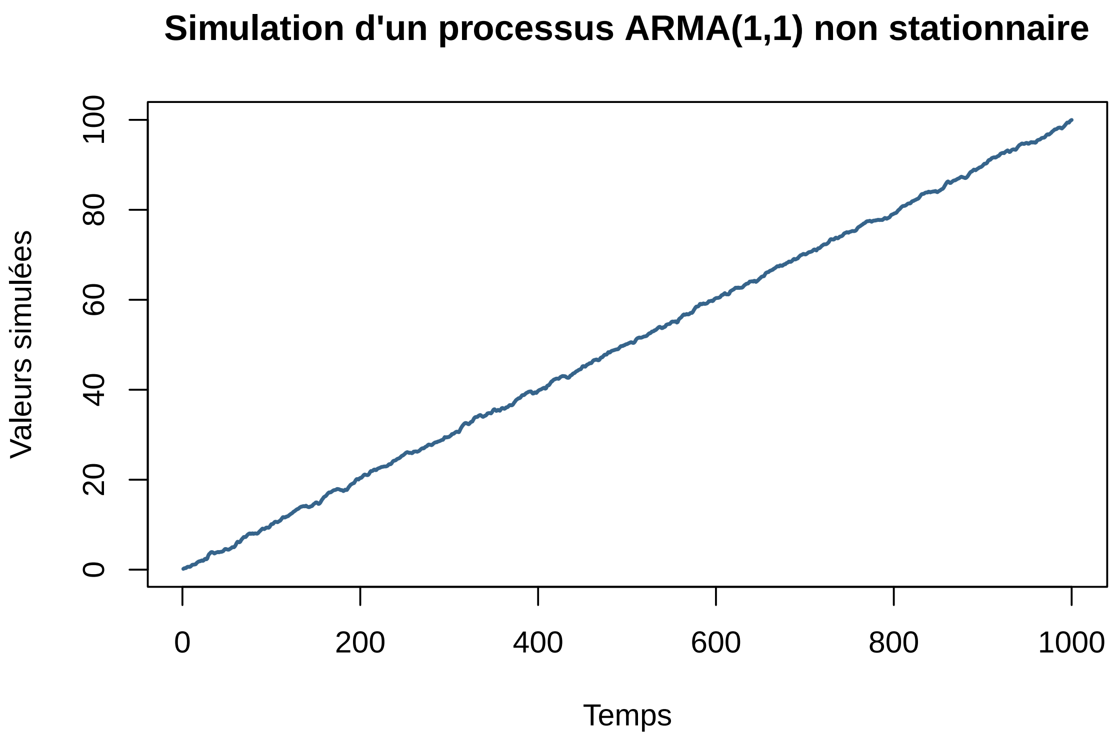
```
lmax = 30
alpha = c(1, -0.2) #partie AR
beta = c(1, 1) #partie MA
x = arima.sim(model = list(ar = alpha, ma = beta),
              sd = 1,
              n = 200)
acfth = ARMAacf(ar = alpha,
                ma = beta,
                lag.max = lmax) #fonction autocorrélation théorique
acfemp = acf(x, lag.max = lmax,
             main = "Fonction autocorrélation empirique",
             col = palette_couleur[1],
             lwd = 2) #fonction autocorrélation empirique
lines(acfemp$lag, acfth, col = palette_couleur[2],
      lwd = 2) #comparaison
legend(
  'topright',
  c('ACF empirique', 'ACF théorique'),
  col = palette_couleur[1:2],
  lty = 1,
  lwd = 2
)
```



1.7 Modélisation de composantes non stationnaires :

1.7.1 Simulation ARMA non stationnaires

```
p = 1 #valeur de p
q = 1 #valeur de q
alpha = c(.9) #alpha
beta = c(.8) #beta
sig = .1 #sigma
time = 10^3
x = arima.sim(model = list(ar = alpha, ma = beta),
              sd = sig,
              n = time) #simulation modèle ARMA(1,1)
y = x + .1 * (1:time) #ajout tendance linaire
plot(
  y,
  main = "Simulation d'un processus ARMA(1,1) non stationnaire",
  ylab = "Valeurs simulées",
  xlab = "Temps",
  type = "l",
  col = palette_couleur[1],
  lwd = 2
)
```

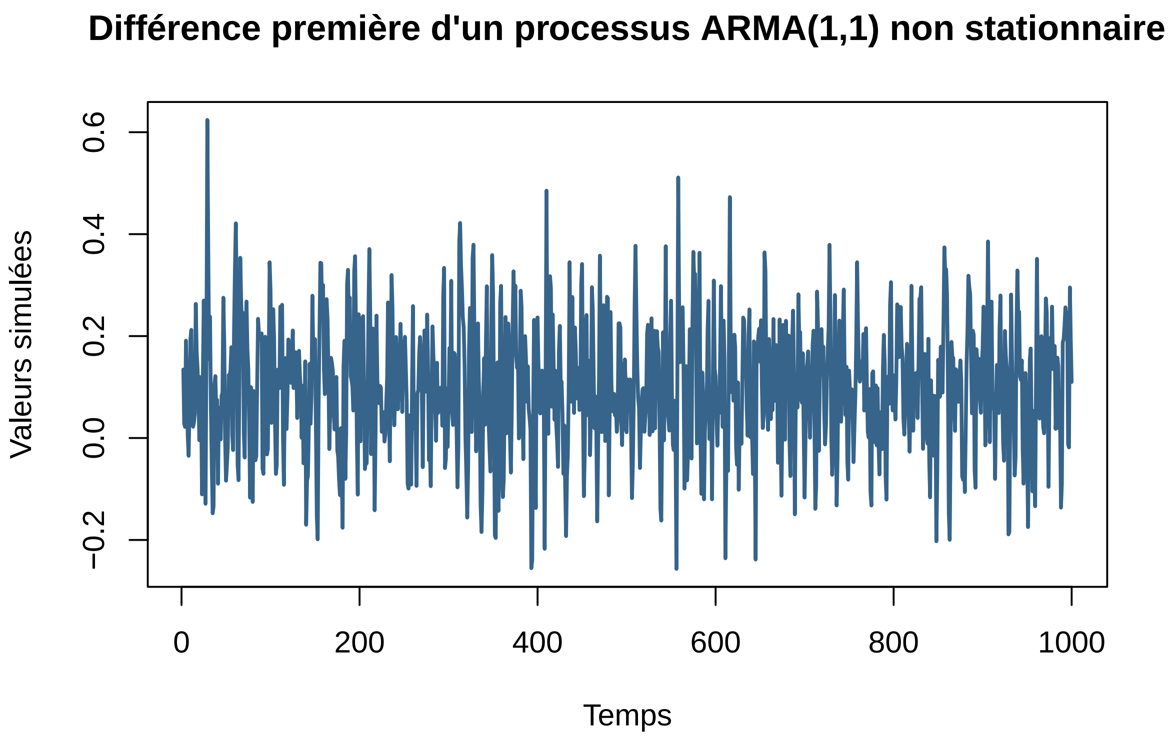


Commentaire :

On voit que le processus n'est pas stationnaire, il y a une tendance linéaire.

1.7.2 Stationnarisation possible, la différence première :

```
plot(diff(y),
     main = "Différence première d'un processus ARMA(1,1) non stationnaire",
     ylab = "Valeurs simulées",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```




```
arma = arima(diff(y), order = c(p, 0, q+1))
arma_centrée = arima(diff(y)-mean(diff(y)), order = c(p, 0, q+1))
arma
```

1.7.2.1 Ajustement du modèle ARMA(p, q+1) pour la différence première :

```

Call:
arima(x = diff(y), order = c(p, 0, q + 1))

Coefficients:
         ar1      ma1      ma2  intercept
      0.8877  -0.1937  -0.8062     0.0997
s.e.  0.0148   0.0195   0.0192     0.0002

sigma^2 estimated as 0.009666:  log likelihood = 897.21,  aic = -1784.41
```

```
arma_centrée
```

```

Call:
arima(x = diff(y) - mean(diff(y)), order = c(p, 0, q + 1))

Coefficients:
         ar1      ma1      ma2  intercept
      0.8877  -0.1937  -0.8062     -2e-04
s.e.  0.0148   0.0195   0.0192      2e-04
```

```
sigma^2 estimated as 0.009666:  log likelihood = 897.21,  aic = -1784.41
```

Commentaire :

La série diff(y) n'est pas de moyenne nulle pourtant la fonction arima considère que la série est centrée.

```r
arma2 = arima(y, order = c(p, 0, q+1))
arma2
```

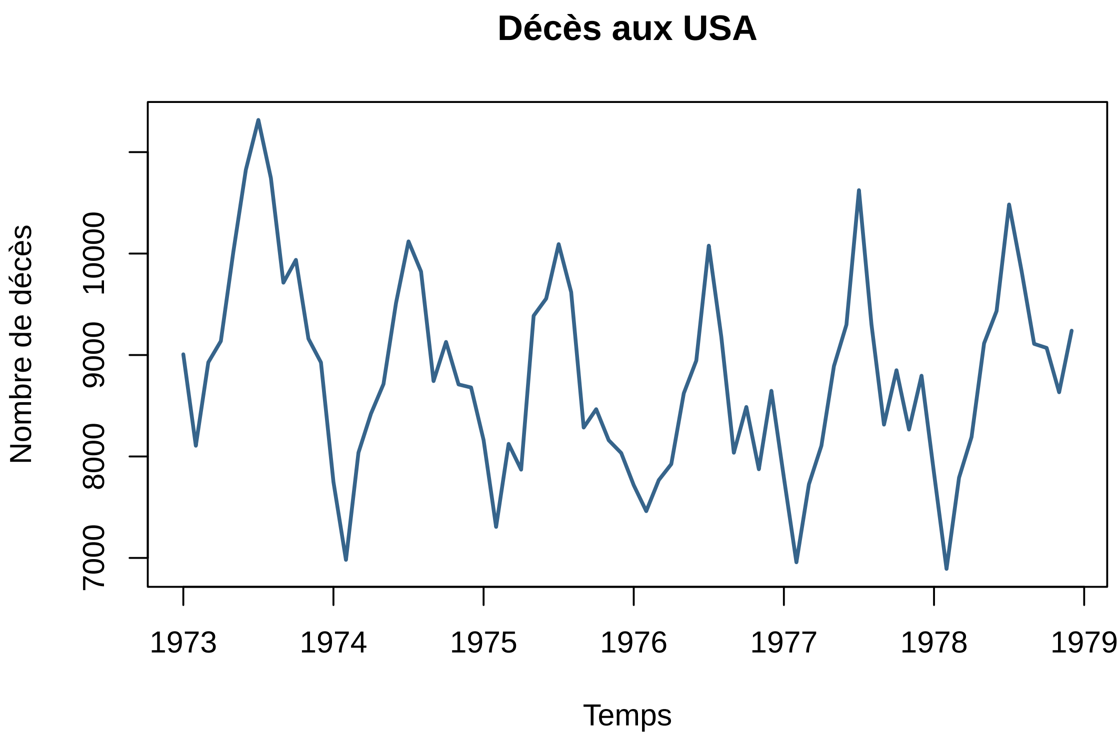
1.7.2.2 Ajustement du modèle ARMA(p,q+1) sur la série temporelle initiale :

```
Call:
arima(x = y, order = c(p, 0, q + 1))

Coefficients:
         ar1     ma1     ma2  intercept
      0.9999  0.9444  0.1327    59.6200
s.e.     NaN  0.0291  0.0282     7.5712

sigma^2 estimated as 0.01291:  log likelihood = 754.78,  aic = -1499.55
```

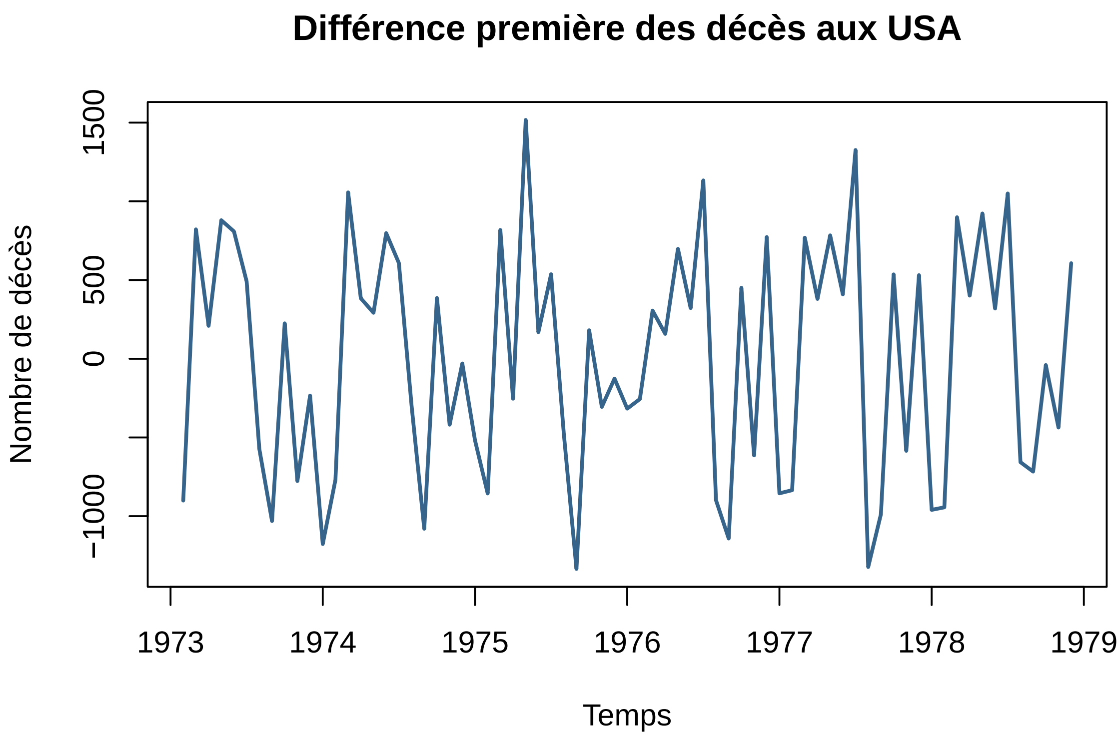
1.8 Illustration des séries temporelles avec de la saisonnalité :

1.8.1 Série temporelle avec saisonnalité et sans tendance :

```r
plot(USAccDeaths,
     main = "Décès aux USA",
     ylab = "Nombre de décès",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```

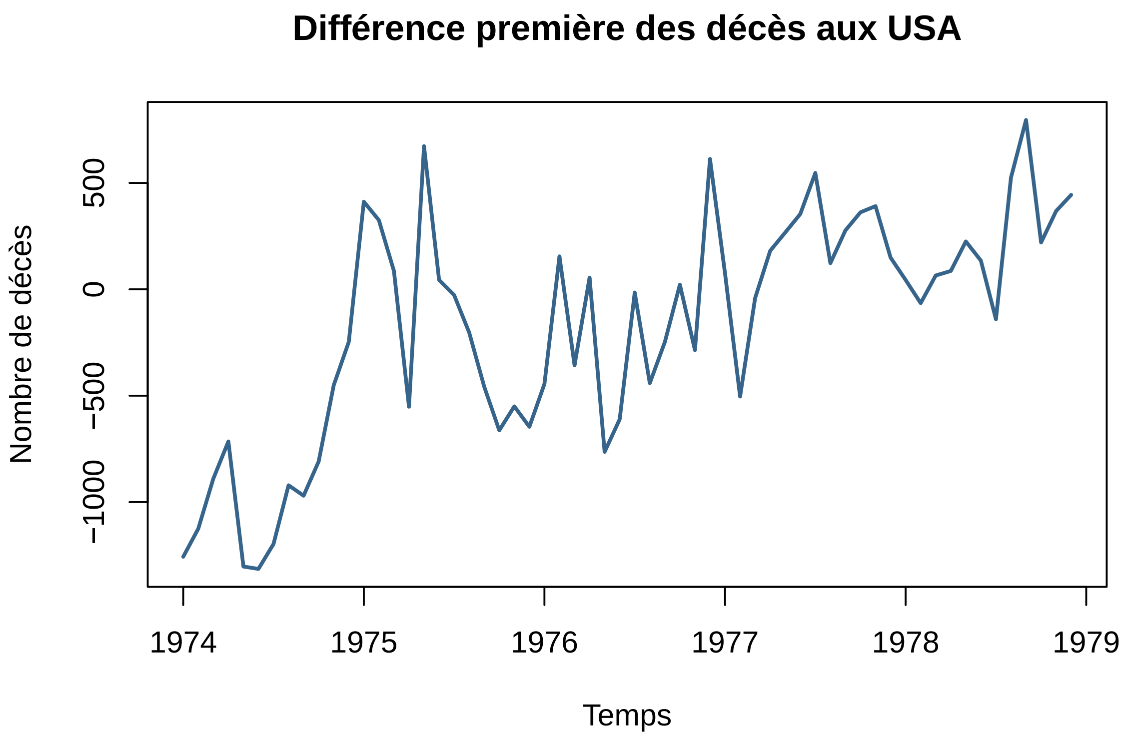



```r
plot(diff(USAccDeaths),
     main = "Différence première des décès aux USA",
     ylab = "Nombre de décès",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```



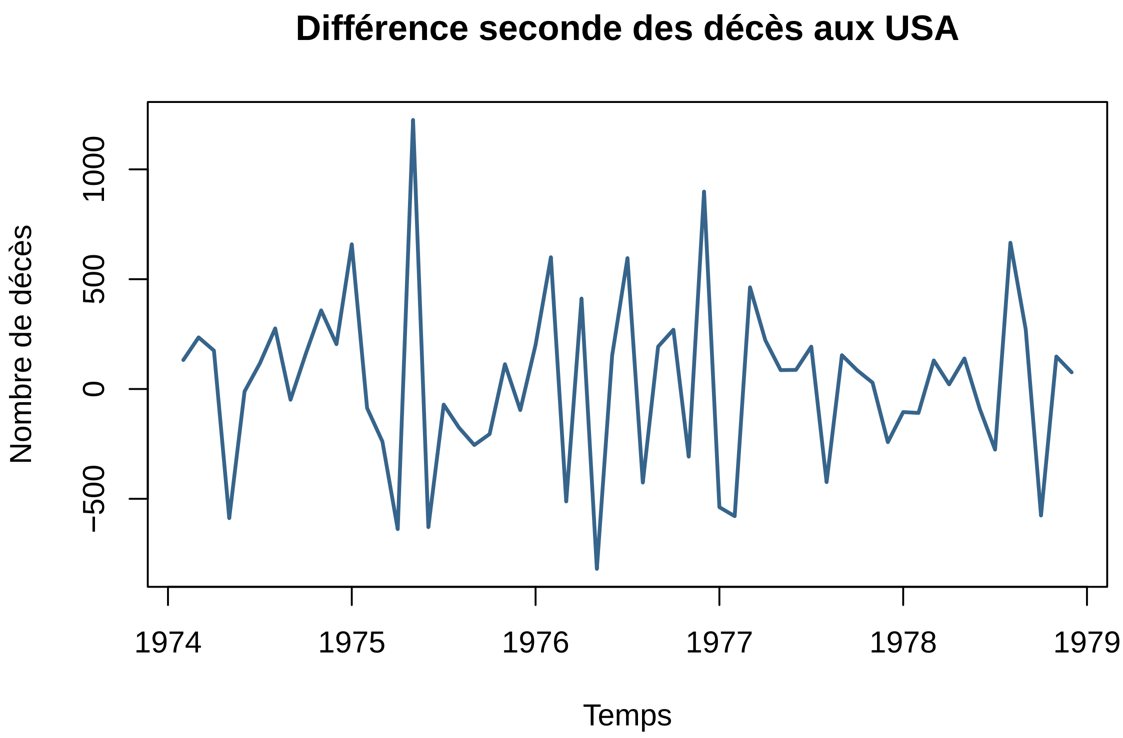
1.8.2 Série temporelle avec saisonnalité et tendance :

```
plot(diff(USAccDeaths, lag = 12),
     main = "Différence première des décès aux USA",
     ylab = "Nombre de décès",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```



1.8.3 Série sans saisonnalité et sans tendance :

```
plot(diff(diff(USAccDeaths, lag = 12)),
     main = "Différence seconde des décès aux USA",
     ylab = "Nombre de décès",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```



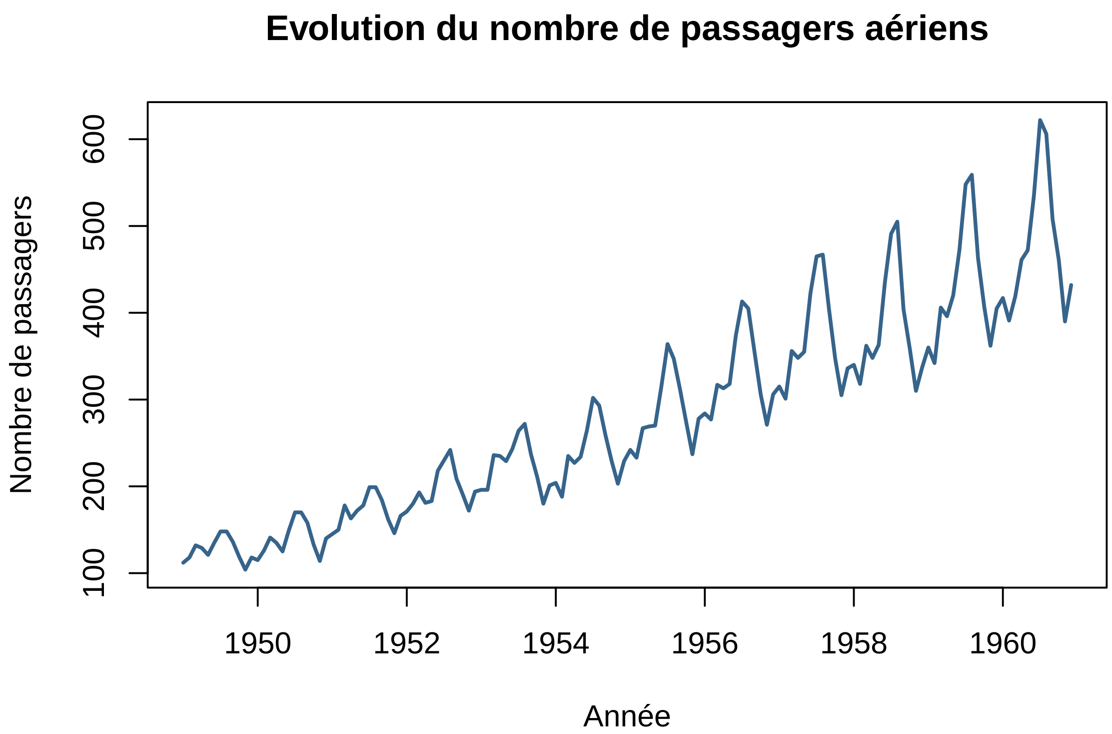

```
dt = data.frame(
  Longueur = c(length(USAccDeaths), length(diff(USAccDeaths)), length(diff(USAccDeaths, lag = 12)), leng
  Différence = c("Aucune", "Première", "Première + lag 12", "première de (première + lag 12)"))
dt
```

1.8.3.1 Diminution de la taille de la série :

```
  Longueur                      Différence
1       72                          Aucune
2       71                        Première
3       60               Première + lag 12
4       59 première de (première + lag 12)
```

1.9 Modéles paramètriques saisonniers :

```
plot(AirPassengers,
     main = "Evolution du nombre de passagers aériens",
     ylab = "Nombre de passagers",
     xlab = "Année",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```

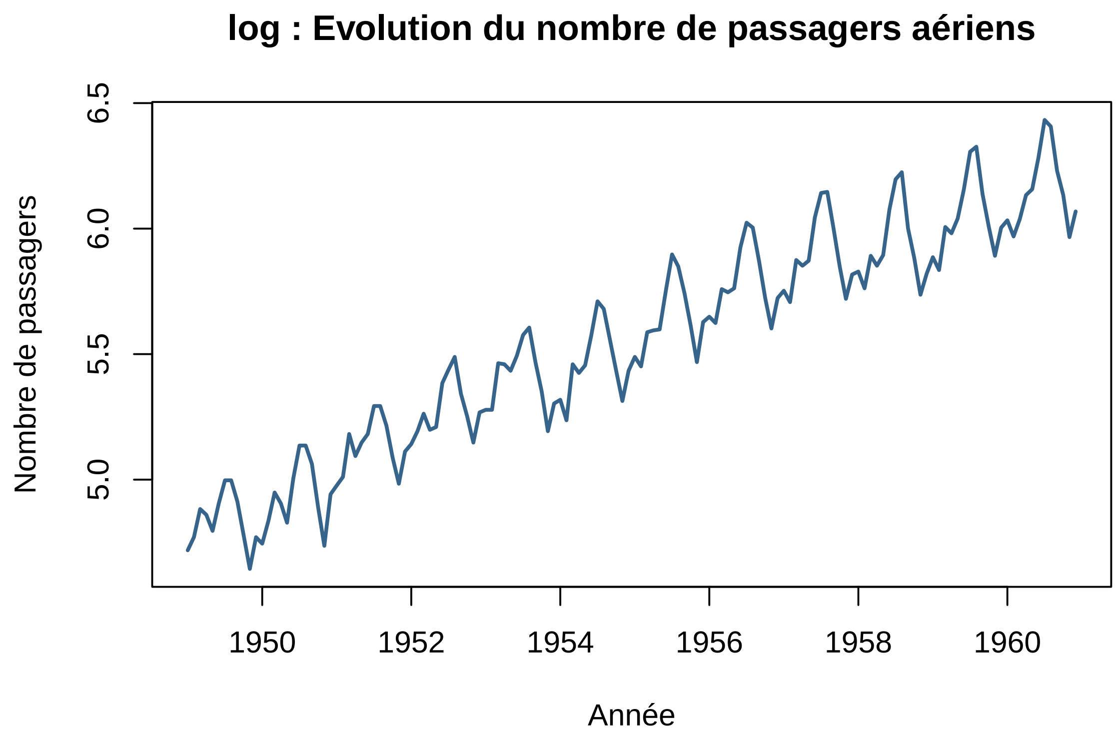


Commentaire :

Augmentation de l'amplitude du cycle saisonnier et la tendance au cours du temps.

1.9.1 Stabilisation du cycle saisonnier par application du log :

```
plot(log(AirPassengers),
     main = "log : Evolution du nombre de passagers aériens",
     ylab = "Nombre de passagers",
     xlab = "Année",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```



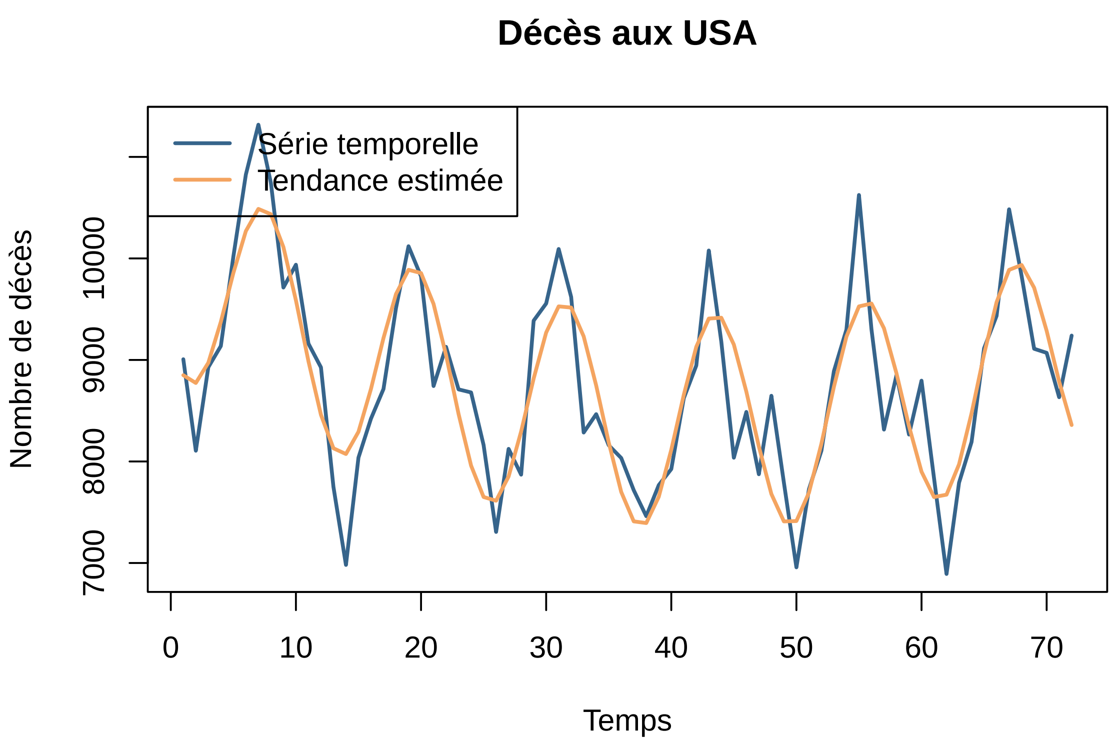
Commentaire :

Il est préférable de prendre un modèle multiplicatif pour les séries temporelles avec une tendance exponentielle. A l'inverse les séries temporelles avec une tendance linéaire peuvent être modélisées par un modèle additif.

1.9.2 Estimation des paramètres de la saisonnalité :

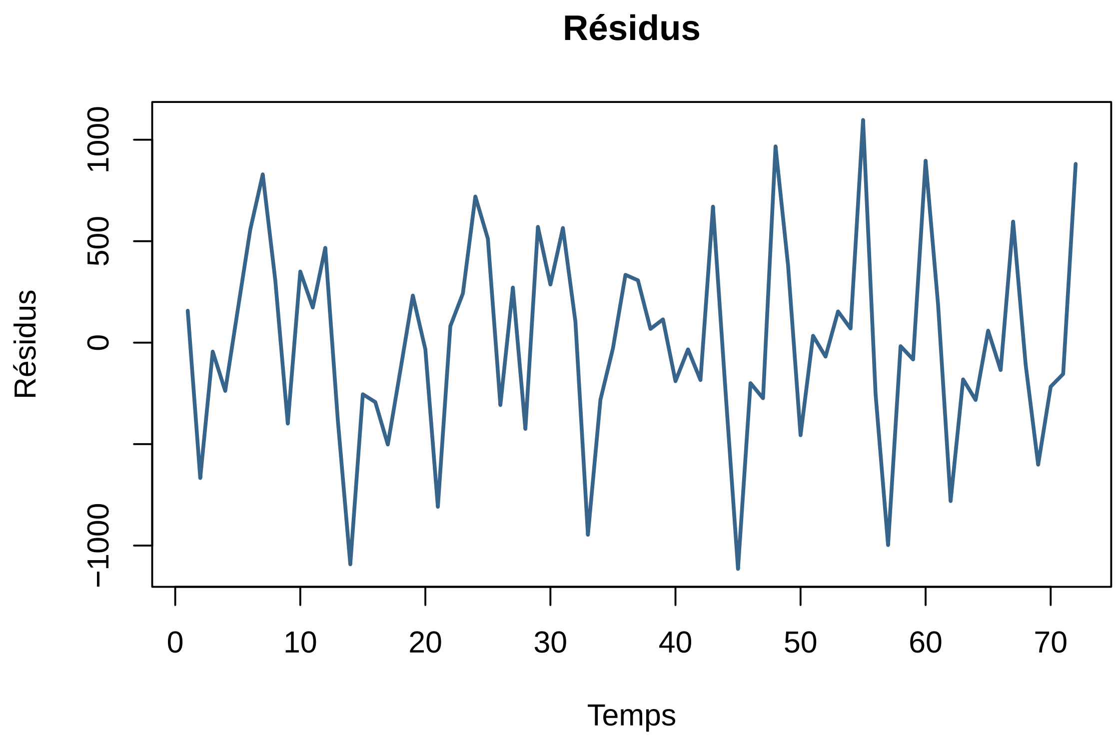
```
Time = length(USAccDeaths)
data = data.frame(
  USAccDeaths,
  t = 1:Time,
  t2 = (1:Time) ^ 2,
  cos = cos(2 * pi * (1:Time) / 12),
  sin = sin(2 * pi * (1:Time) / 12)
)
fit = lm(USAccDeaths ~ ., data = data)
plot(
  1:Time,
  USAccDeaths,
  main = "Décès aux USA",
  ylab = "Nombre de décès",
  xlab = "Temps",
  type = "l",
  col = palette_couleur[1],
  lwd = 2
)
lines(1:Time, fit$fitted.values, col = palette_couleur[2], lwd = 2)
```

```
legend(
  'topleft',
  c('Série temporelle', 'Tendance estimée'),
  col = palette_couleur[1:2],
  lwd = 2
)
```

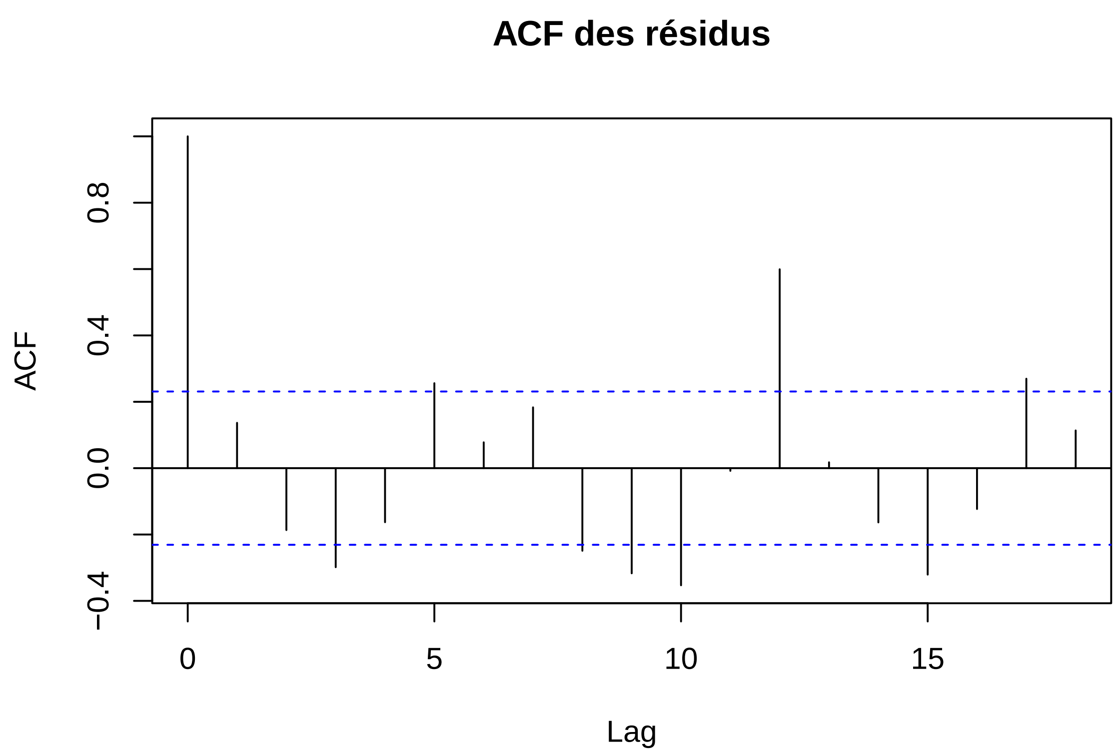


1.9.3 Vérification du choix d'estimation de la composante saisonnière :

```
plot(fit$residuals,
     main = "Résidus",
     ylab = "Résidus",
     xlab = "Temps",
     type = "l", col = palette_couleur[1], lwd = 2)
```




```
acf(fit$residuals, main = "ACF des résidus")
```

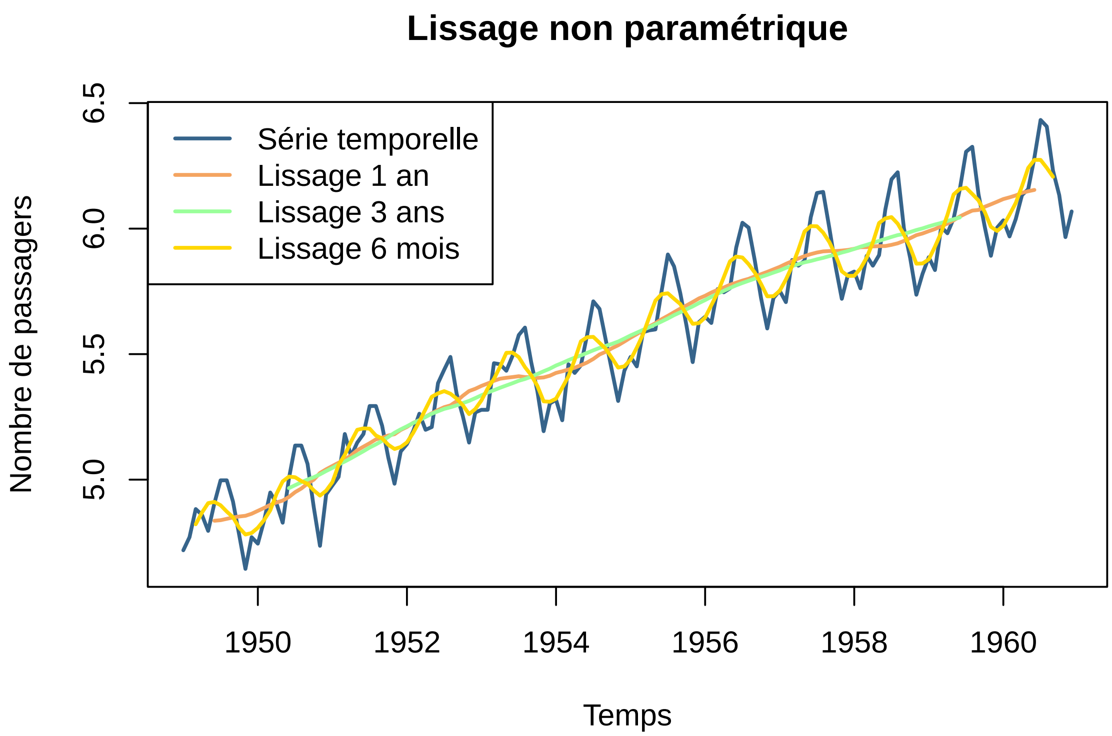


Commentaire :

La saisonnalité n'est pas bien modélisée, on remarque qu'il reste un tendance dans les résidus notamment tous les 12 mois.

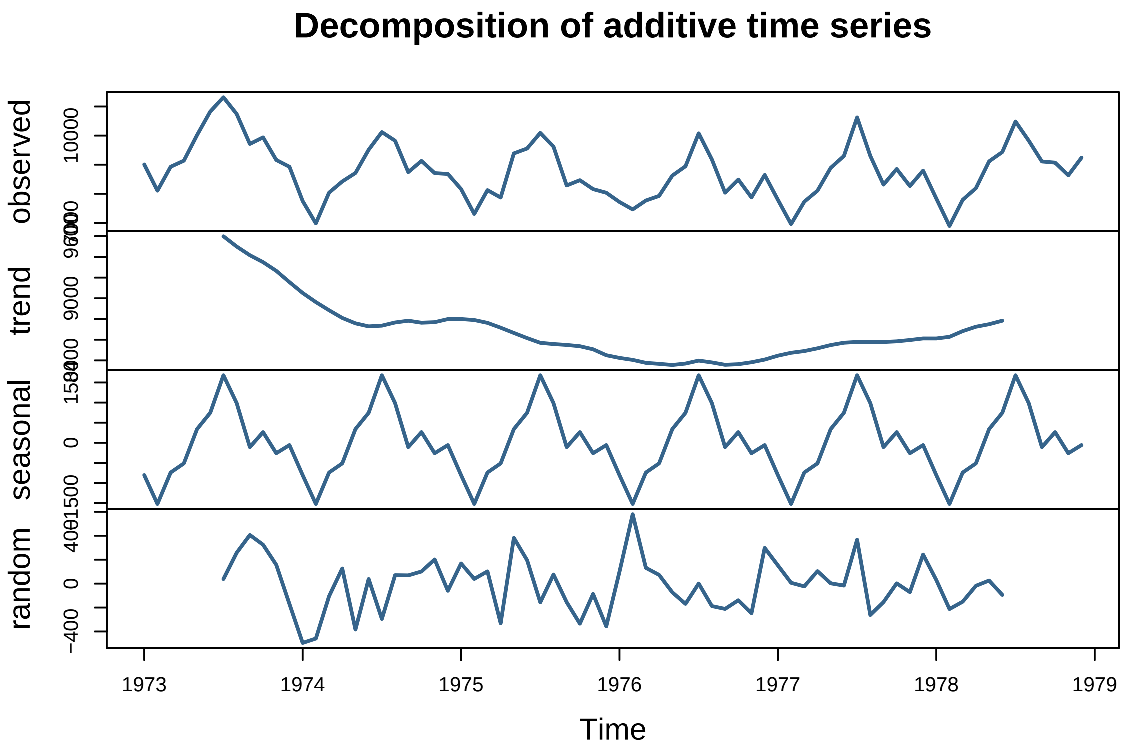
1.10 Lissage non paramétrique :

```
# Estimation de la tendance par moyenne mobile :
plot(log(AirPassengers),
     main = "Lissage non paramétrique",
     ylab = "Nombre de passagers",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
# Fenètre de largeur 1 an :
L = 12
hatx = stats::filter(log(AirPassengers), filter = rep(1 / (L), L))
lines(hatx,
      col = palette_couleur[2],
      lwd = 2) #estimateur non paramétrique de la tendance

# Fenètre de largeur 3 ans :
L = 36
hatx = stats::filter(log(AirPassengers), filter = rep(1 / (L), L))
lines(hatx,
      col = palette_couleur[3],
      lwd = 2) #lissage plus important
# Fenètre de largeur 6 mois :
L = 6
hatx = stats::filter(log(AirPassengers), filter = rep(1 / (L), L))
lines(hatx,
      col = palette_couleur[4],
      lwd = 2) #composante saisonnière toujours présente
legend(
  'topleft',
  c('Série temporelle', 'Lissage 1 an', 'Lissage 3 ans', 'Lissage 6 mois'),
  col = palette_couleur[1:4],
  lwd = 2
)
```

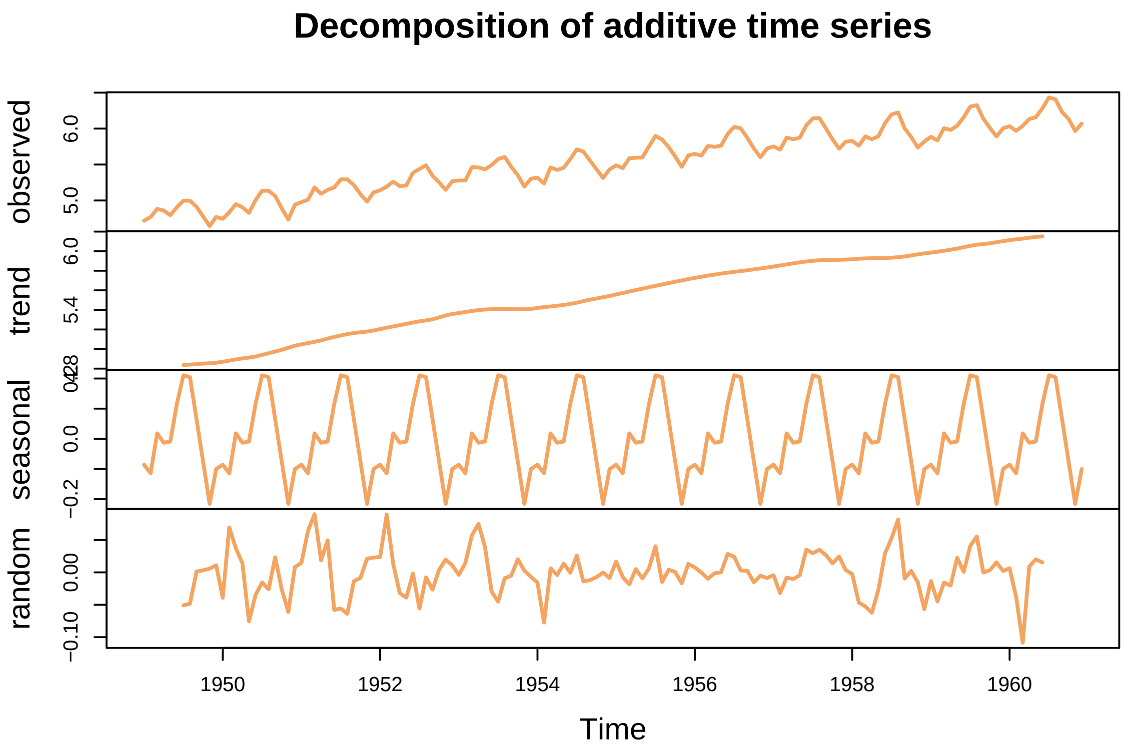


1.10.1 Décomposition de la série temporelle, méthode des moyennes mobiles :

```
plot(decompose(USAccDeaths), lwd = 2,
     col = palette_couleur[1])
```




```r
plot(decompose(log(AirPassengers)), lwd = 2,
     col = palette_couleur[2])
```

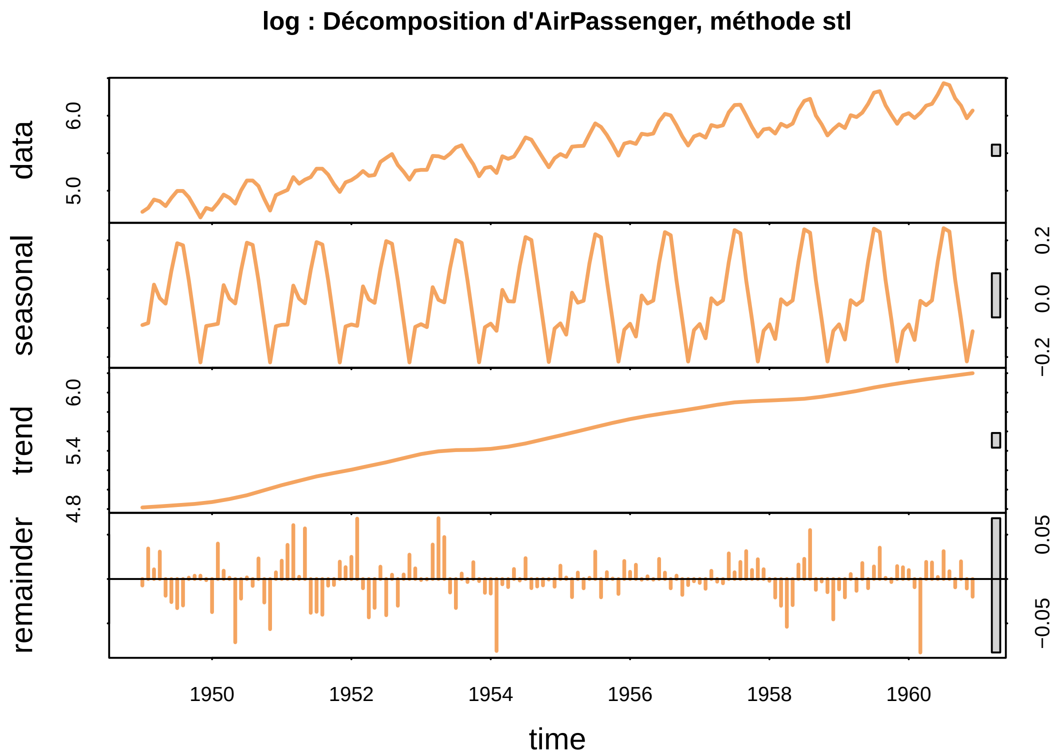


1.10.2 Décomposition de la série temporelle, méthode STL :

Commentaire :

La fonction stl décompose la série temporelle en trois composantes : saisonnalité, tendance et résidus. Elle n'utilise pas la méthodes des moyennes mobiles, mais une méthode de lissage basée sur la régression polynomiale.

```
plot(stl(log(AirPassengers), s.window = 12),
     main = "log : Décomposition d'AirPassenger, méthode stl",
     lwd = 2,
     col = palette_couleur[2])
```



log : Décomposition d'AirPassenger, méthode stl
data
6.0
5.0
seasonal
0.2
0.0
−0.2
trend
6.0
5.4
4.8
remainder
0.05
−0.05
1950
1952
1954
1956
1958
1960
time

2 Travaux dirigés

2.1 Exercice 1: Simulation de différents procesus :

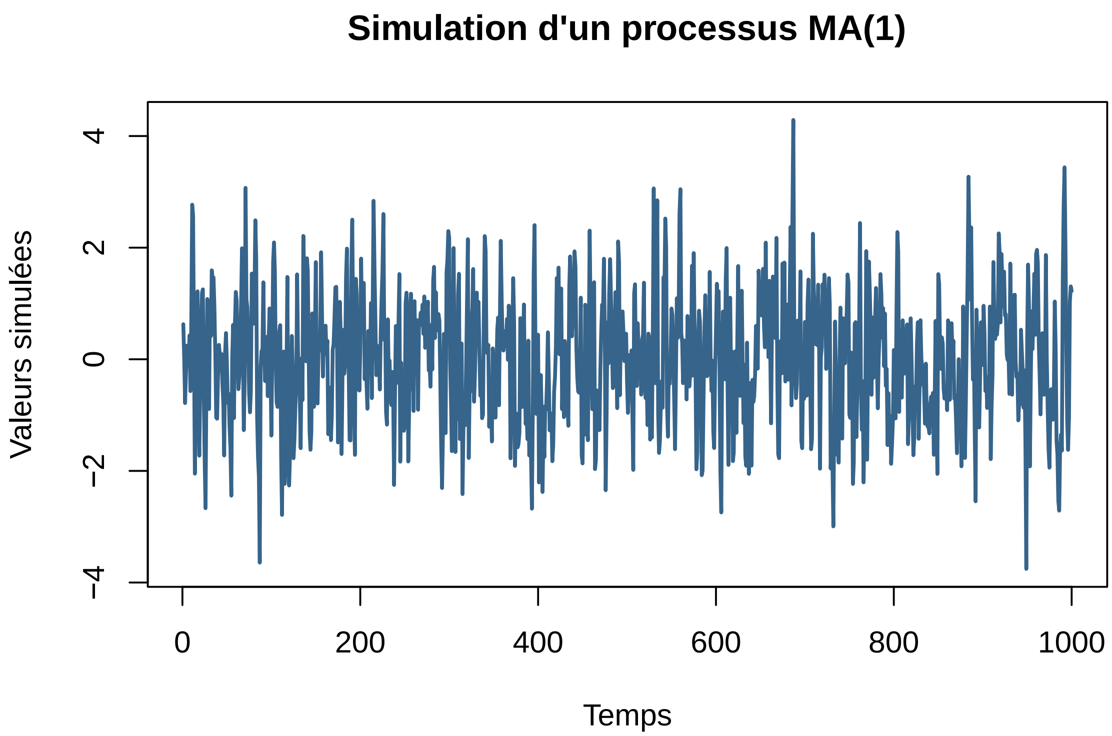
2.1.1 Simulation MA(1) :

$$A_t = \epsilon_t + \beta\epsilon_{t-1} \text{ avec } \epsilon_t \sim \mathcal{N}(0, 1)$$

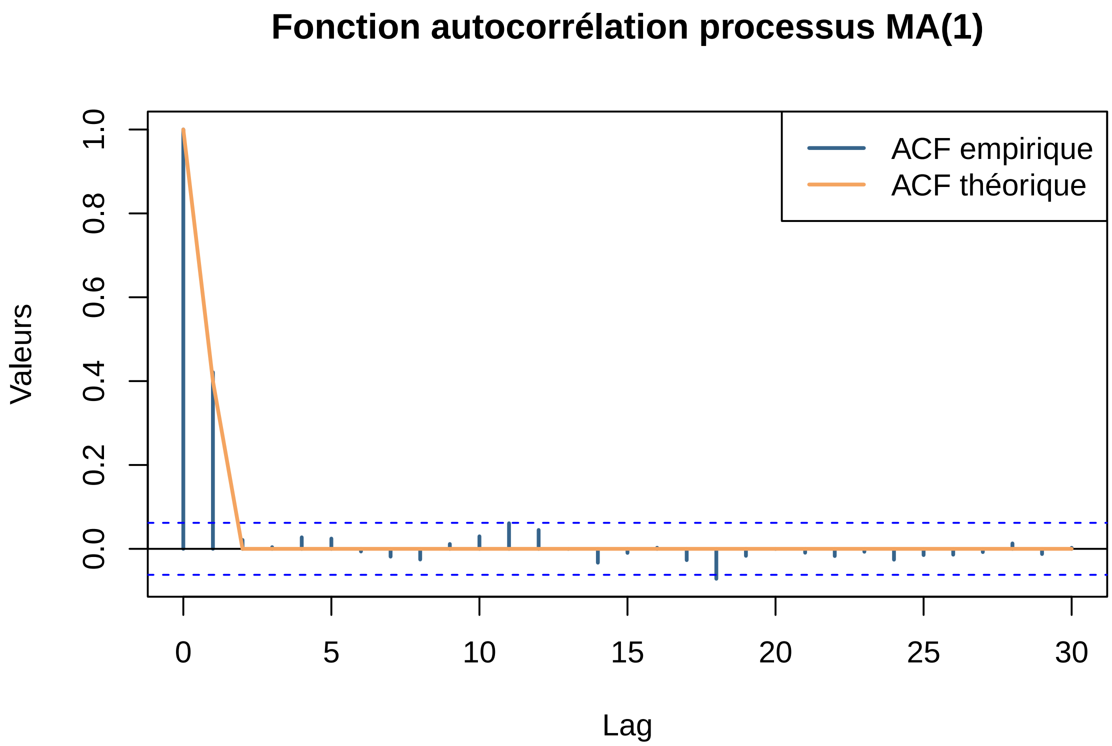
```
# Paramètres de la simulation :
Time = 1000
sigma = 1
beta = 0.5

# Simulation par formule :
esp = rnorm(Time +1, sd = sigma)
A = esp[2:(Time+1)] + beta * esp[1:Time]

# Simulation par fonction intégrée :
A2 = arima.sim(model = list(ma = beta), n = Time, sd = sigma)

# Acf Théorique d'un' processus MA(1) :
ARMAacf(ma = beta, lag.max = 20)
```

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
1.0 0.4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 20
0.0
```

```
# Représentation graphique :
plot(
  A,
  type = 'l',
  main = 'Simulation d\'un processus MA(1)',
  ylab = 'Valeurs simulées',
  xlab = 'Temps',
  col = palette_couleur[1],
  lwd = 2
)
```




```r
ac = acf(A,
         main = "Fonction autocorrélation processus MA(1)",
         ylab = "Valeurs",
         col = palette_couleur[1],
         lwd = 2)  #acf empirique
lines(ac$lag, c(1, beta / (1 + beta ^ 2), rep(0, length(ac$lag) - 2)),
      col = palette_couleur[2],
      lwd = 2)  #acf théorique
#Alternative :
# lines(ac$lag,
#       ARMAacf(ma = beta, lag.max = ac$lag[length(ac$lag)]),
#       col = palette_couleur[3],
#       lwd = 2)  #acf alternative
legend(
  'topright',
  c('ACF empirique', 'ACF théorique'),
  col = palette_couleur[1:2],
  lty = 1,
  lwd = 2)
```



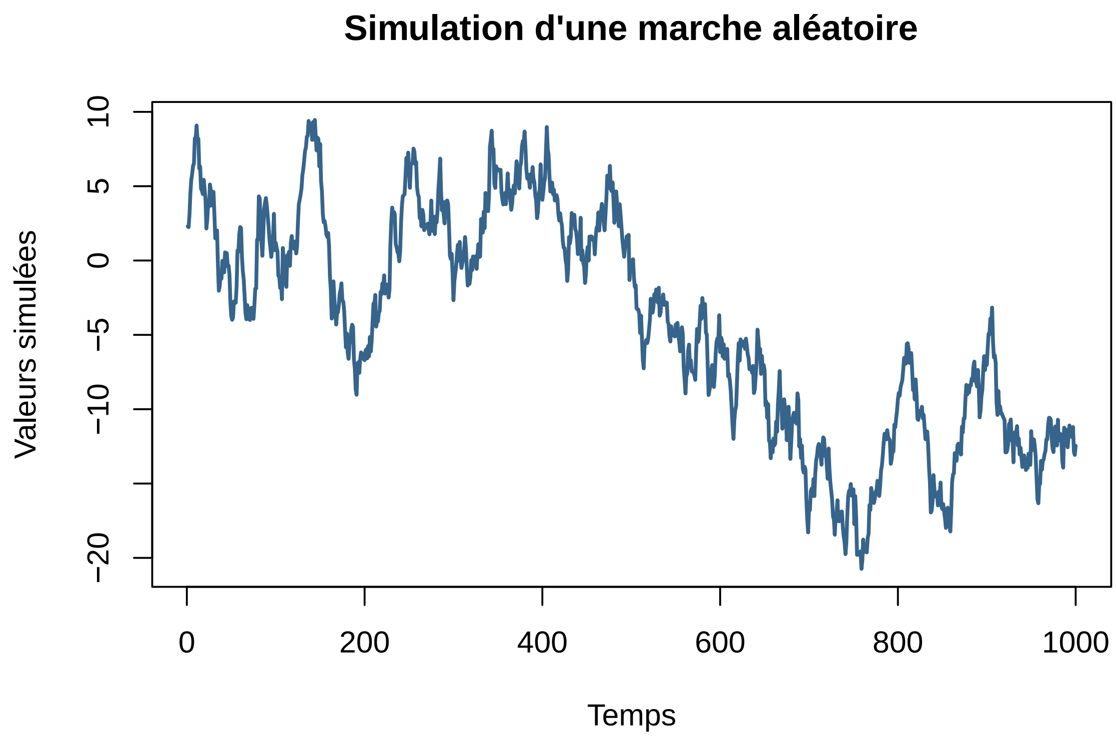
Commentaires :

L'acf (acf=fonction d'autocorrélation) empirique doit être proche de l'acf théorique si T est grand (estimateur consistant) L'acf montre l'existence d'une dépendance entre les valeurs successives, X_t et X_{t+h} sont non corrélées si $h > 1$. Les bornes de l'intervalle bleu sont égales à +-1.96/$T^{0.5}$ Pour un bruit blanc, on doit avoir l'acp empirique dans l'intervalle bleu avec un proba de 95%. On retrouve que $\hat{\rho}(1)$ est (significativement) plus grand que ce qu'on attend pour un bruit blanc.

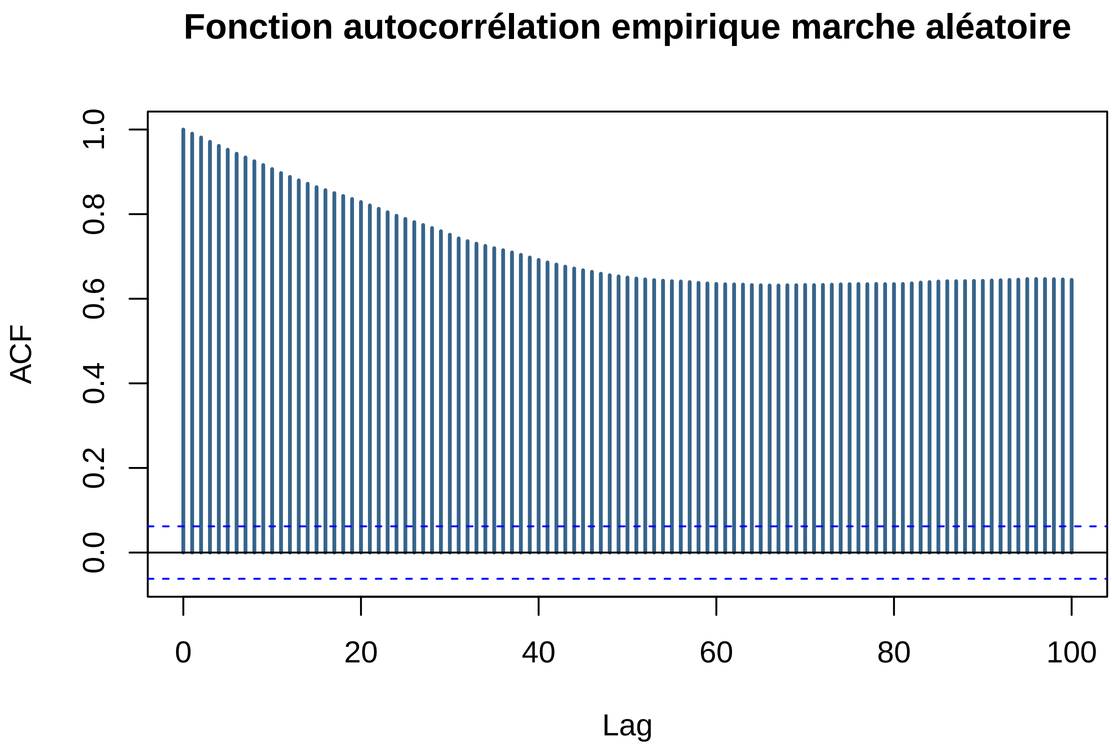
2.1.2 Simulation d'une marche aléatoire :

$$B_t = B_{t-1} + \epsilon_t \text{ avec } \epsilon_t \sim \mathcal{N}(0, 1)$$

```
sig = 1
tau = 1
Time = 1000
eps = rnorm(Time, sd = sig) #bruit blanc
B = rnorm(1, sd = tau) + cumsum(eps[1:Time])  #alternative : boucle
plot(
  B,
  type = 'l',
  main = 'Simulation d\'une marche aléatoire',
  ylab = 'Valeurs simulées',
  xlab = 'Temps',
  col = palette_couleur[1],
  lwd = 2
)
```




```r
acf(
  B,
  lag.max = 100,
  main = "Fonction autocorrélation empirique marche aléatoire",
  col = palette_couleur[1],
  lwd = 2
)
```

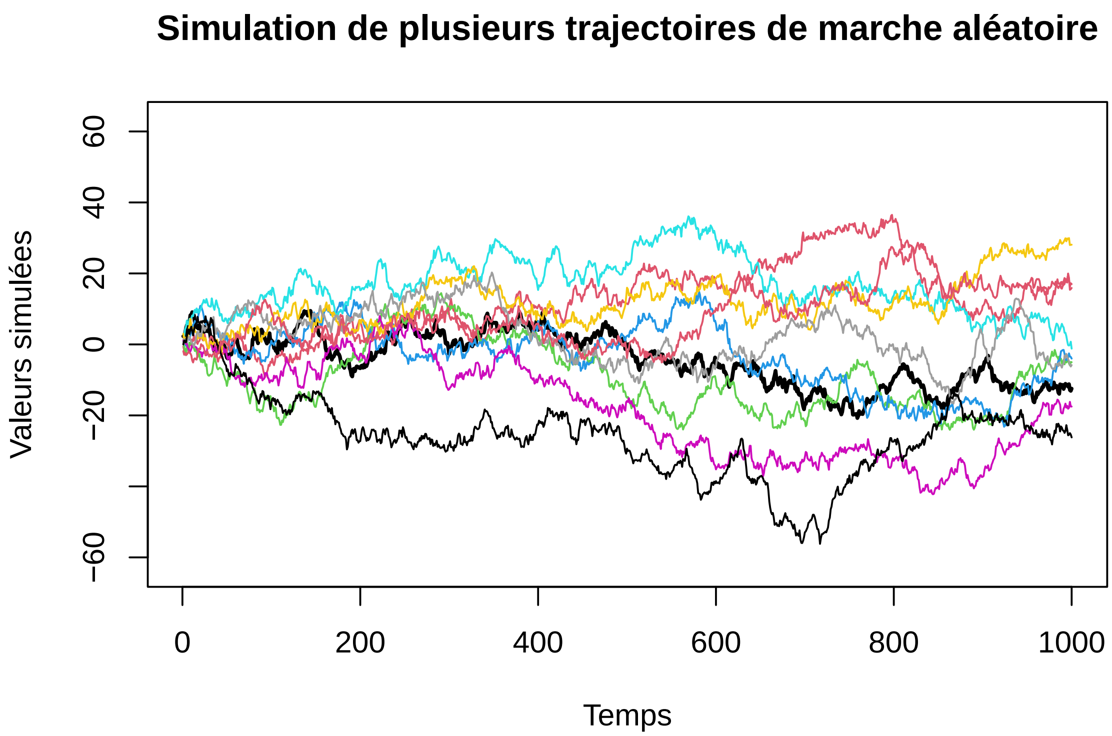


Commentaires :

Pprocessus non-stationnaire, ACF théorique pas définie, décroissance lente vers 0 de l'ACP empirique.

```
Time = 1000
plot(
  B,
  type = 'l',
  main = 'Simulation de plusieurs trajectoires de marche aléatoire',
  ylab = 'Valeurs simulées',
  xlab = 'Temps',
  col = 1,
  ylim = c(-2 * sig * sqrt(Time), 2 * sig * sqrt(Time)),
  lwd = 2
)
for (i in 2:10) {
  eps = rnorm(Time + 1, sd = sig)
  B = rnorm(1, sd = tau) + cumsum(eps[1:Time]) #on prend tau=sigma
  lines(B, col = i)
}
```

2.1.2.1 Simulation de plusieurs trajectoires de marche aléatoire :



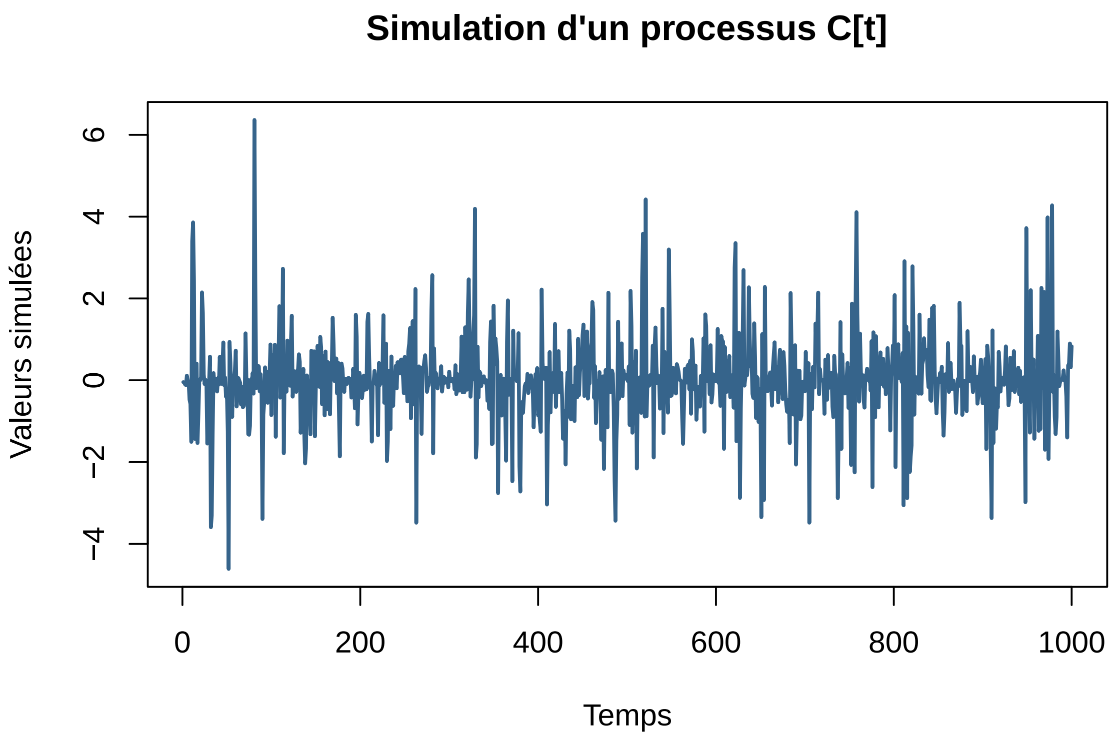
Commentaire :

On retrouve que la variance augmente avec le temps, processus non-stationnaire

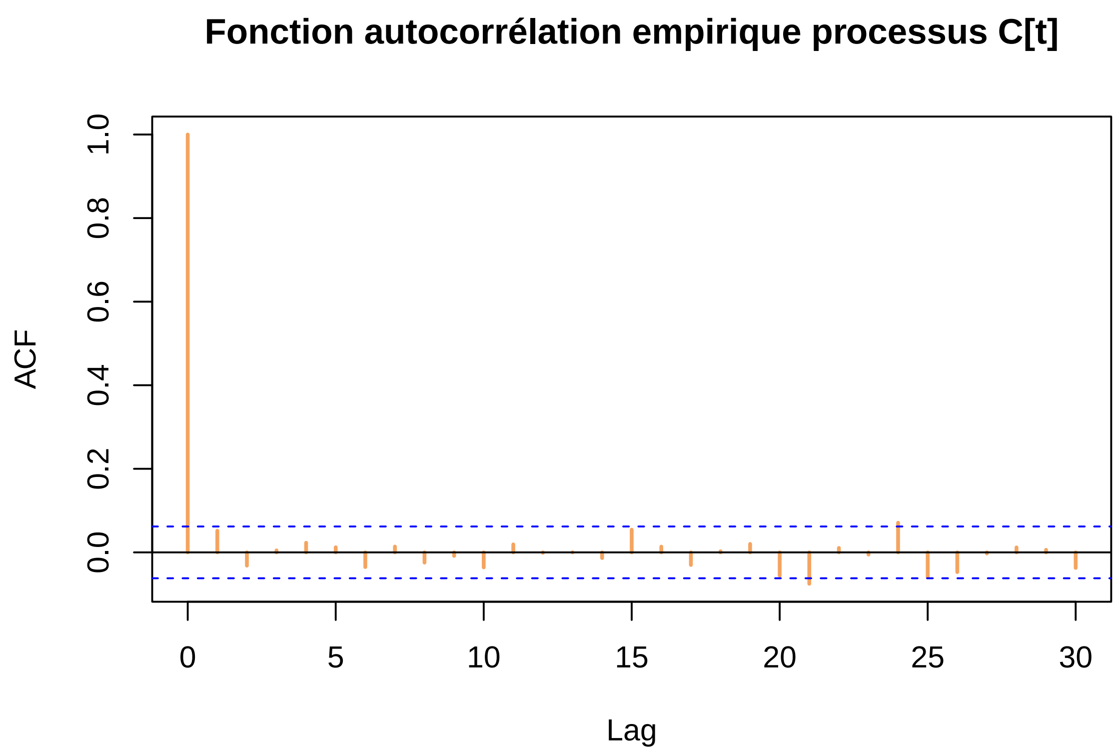
2.1.3 Simulation du troisième processus :

$$C_t = \epsilon_{t-1} \times \epsilon_t \text{ avec } \epsilon_t \sim \mathcal{N}(0,1)$$

```
Time = 1000
sigma = 1
mu = 0
esp = rnorm(Time + 1, sd = sigma , mean = mu)
C = esp[1:Time] * esp[2:(Time + 1)]

plot(C,
     main = "Simulation d'un processus C[t]",
     ylab = "Valeurs simulées",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```




```
acf(C,
    main = "Fonction autocorrélation empirique processus C[t]",
    col = palette_couleur[2],
    lwd = 2)
```



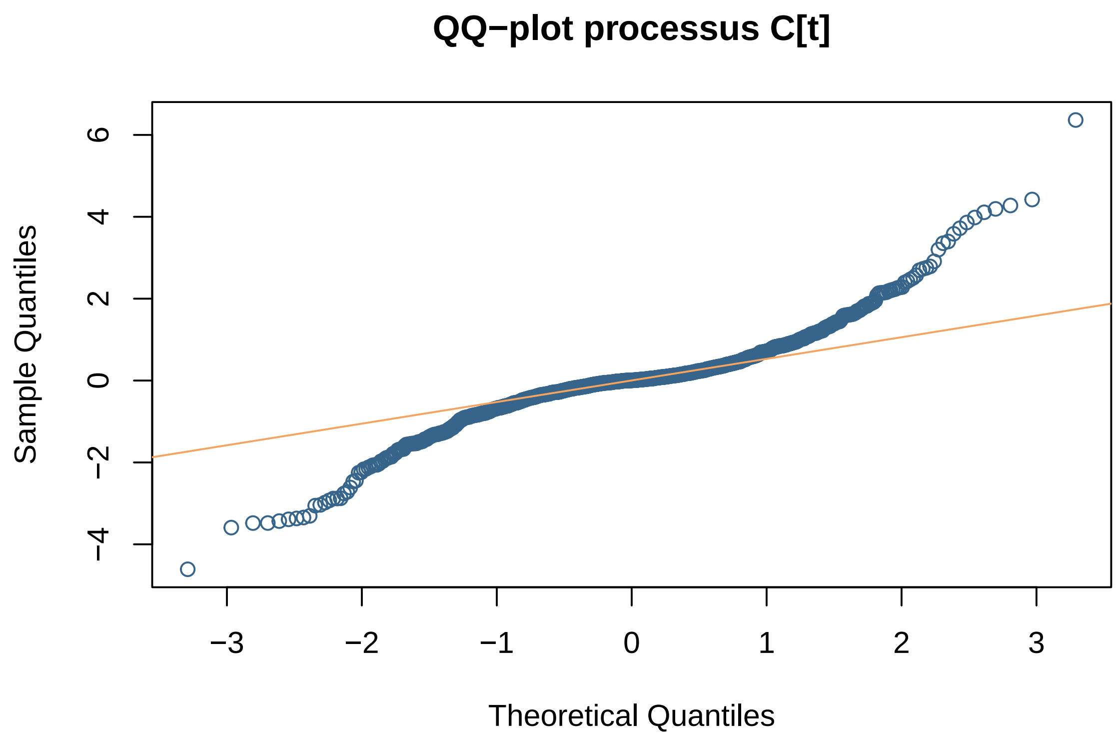
2.1.3.1 Test de normalité des résidus : On peut tester la normalité des résidus avec le test de Shapiro-Wilk.

Le test de shapiro test l'existence de normalité des résidus.

$$\begin{cases} H_0 : \text{Les résidus suivent une loi normale} \\ H_1 : \text{Les résidus ne suivent pas une loi normale} \end{cases}$$

On peut aussi tracer un QQ-plot pour vérifier la normalité des résidus.

```
#Etude de la distribution des valeurs simulées :
qqnorm(C,
       main = "QQ-plot processus C[t]",
       col = palette_couleur[1])
qqline(C, col = palette_couleur[2])
```



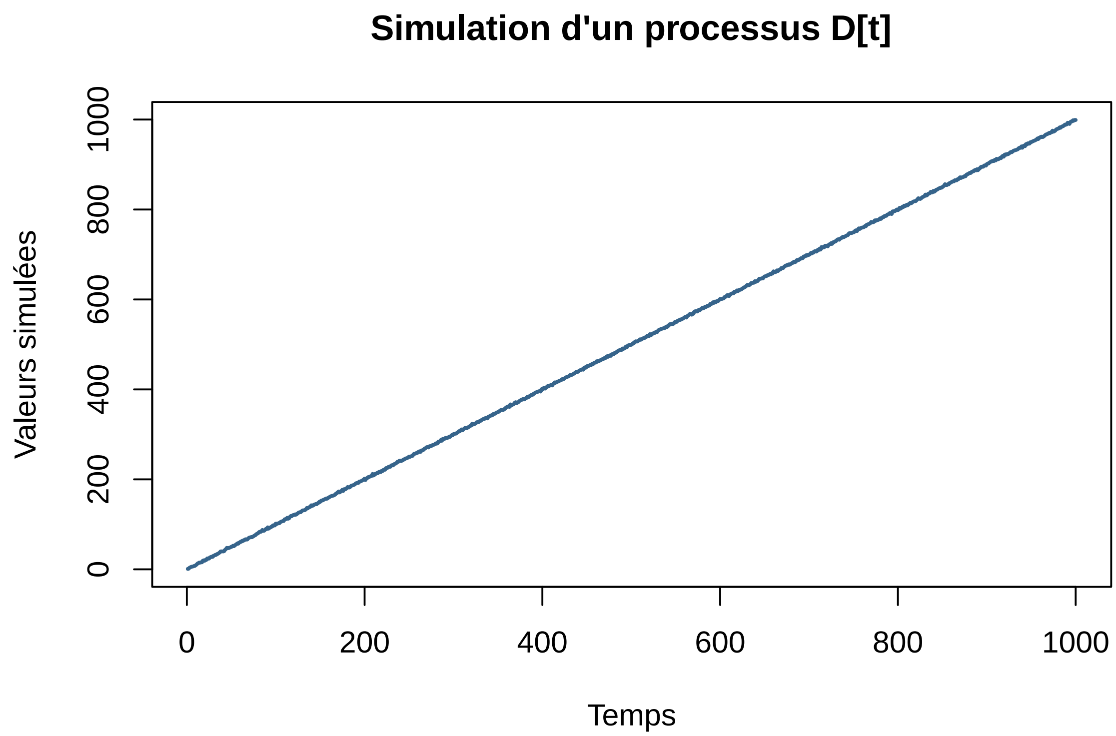

```
shapiro.test(C) #test de normalité des résidus
```

```
    Shapiro-Wilk normality test

data:  C
W = 0.90951, p-value < 2.2e-16
```

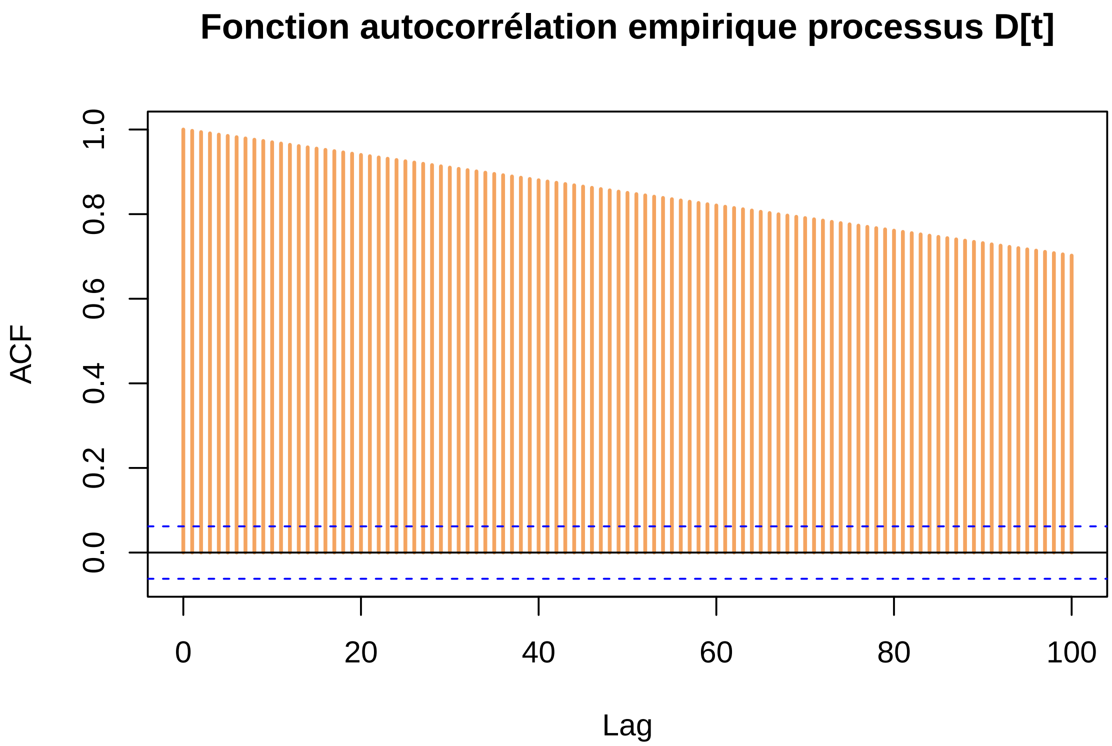
2.1.4 Simulation du processus D :

$$D_t = t + \epsilon_t \text{ avec } \epsilon_t \sim \mathcal{N}(0,1)$$

```
sigma = 1
Time = 1000
esp = rnorm(Time, sd = sigma)

D = 1:Time + esp[1:Time]

plot(D,
     main = "Simulation d'un processus D[t]",
     ylab = "Valeurs simulées",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```




```r
acf(D,
    main = "Fonction autocorrélation empirique processus D[t]",
    col = palette_couleur[2],
    lwd = 2,
    lag.max = 100)
```



Commentaires :

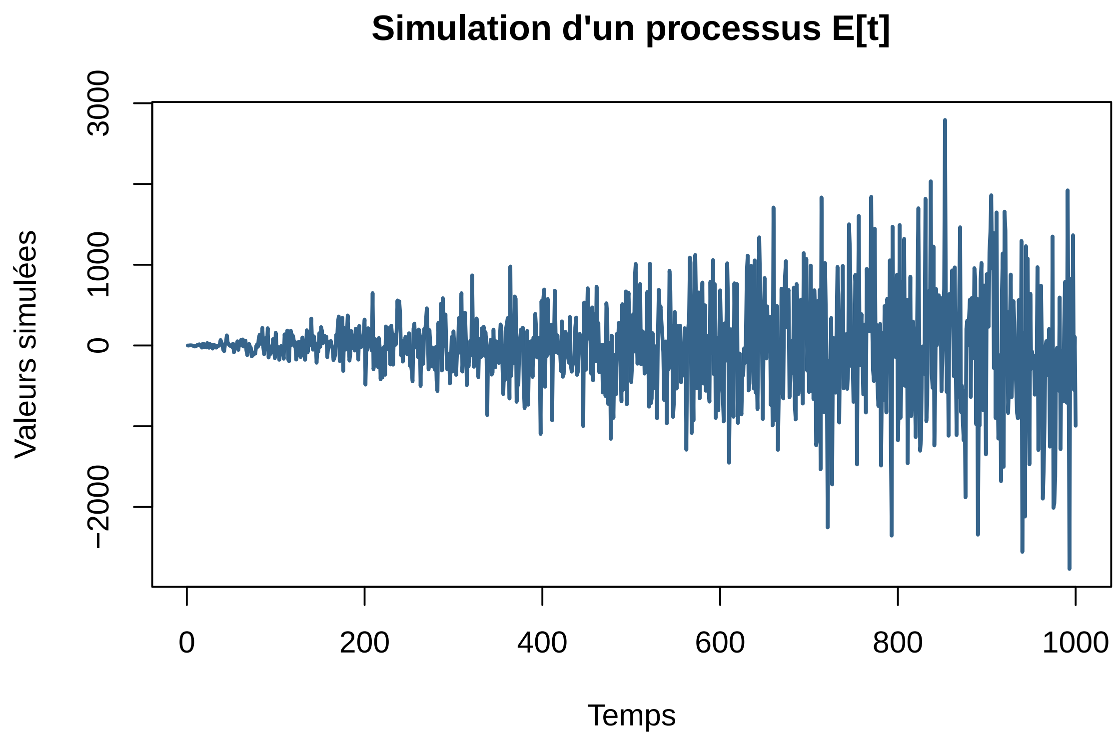
Le proccesus n'est pas stationnaire, il y a une tendance dans la variance.

En considérant $t \in R$ fixé on peut dire que le processus est gaussien.

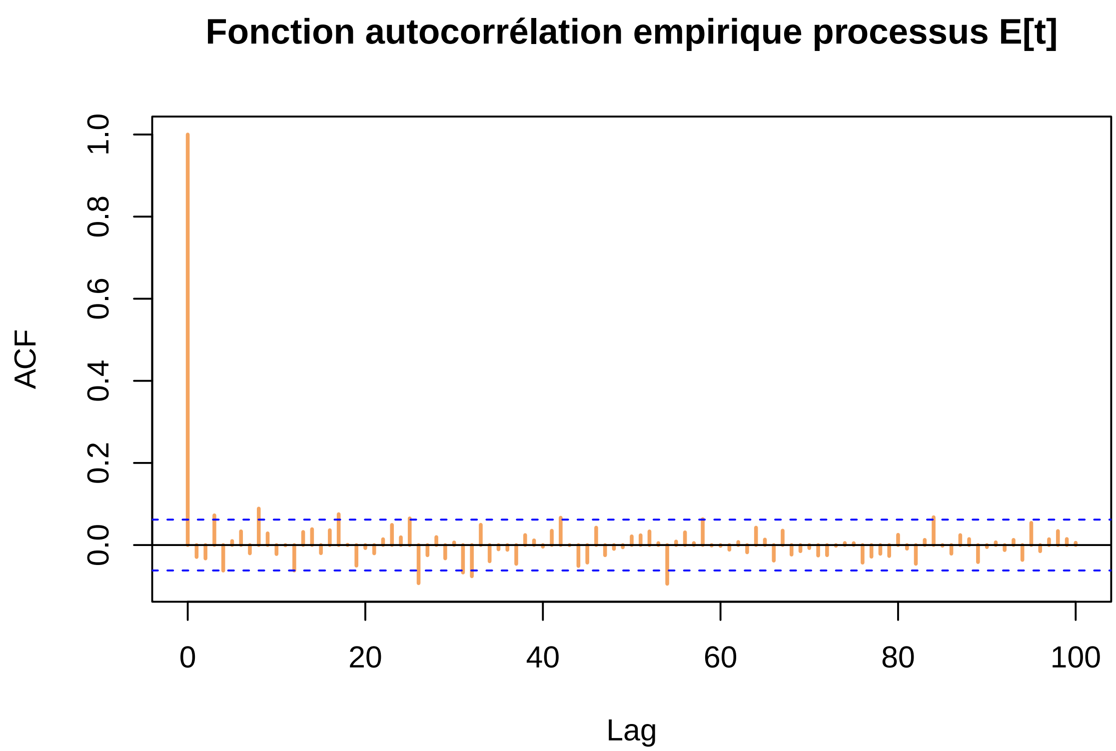
2.1.5 Simulation du processus E :

$$E_t = t \times \epsilon_t \text{ avec } \epsilon_t \sim \mathcal{N}(0, 1)$$

```
E = (1:Time) * esp[1:Time]

plot(E,
     main = "Simulation d'un processus E[t]",
     ylab = "Valeurs simulées",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```




```r
acf(E,
    main = "Fonction autocorrélation empirique processus E[t]",
    col = palette_couleur[2],
    lwd = 2,
    lag.max = 100)
```



Commentaire :

Le processus n'est pas stationnaire il y a une tendance dans la variance.

2.2 Exercice 2 :

$$X_t = \epsilon_t + \beta_1 * \epsilon_{t-1} + \beta_2 * \epsilon_{t-2} + \beta_3 * \epsilon_{t-3} \text{ avec } \epsilon_t \sim \mathcal{N}(0,1)\beta_1 = 1, \beta_2 = 0.5, \beta_3 = -0.5$$

2.2.1 Fonction autocorrélation théorique :

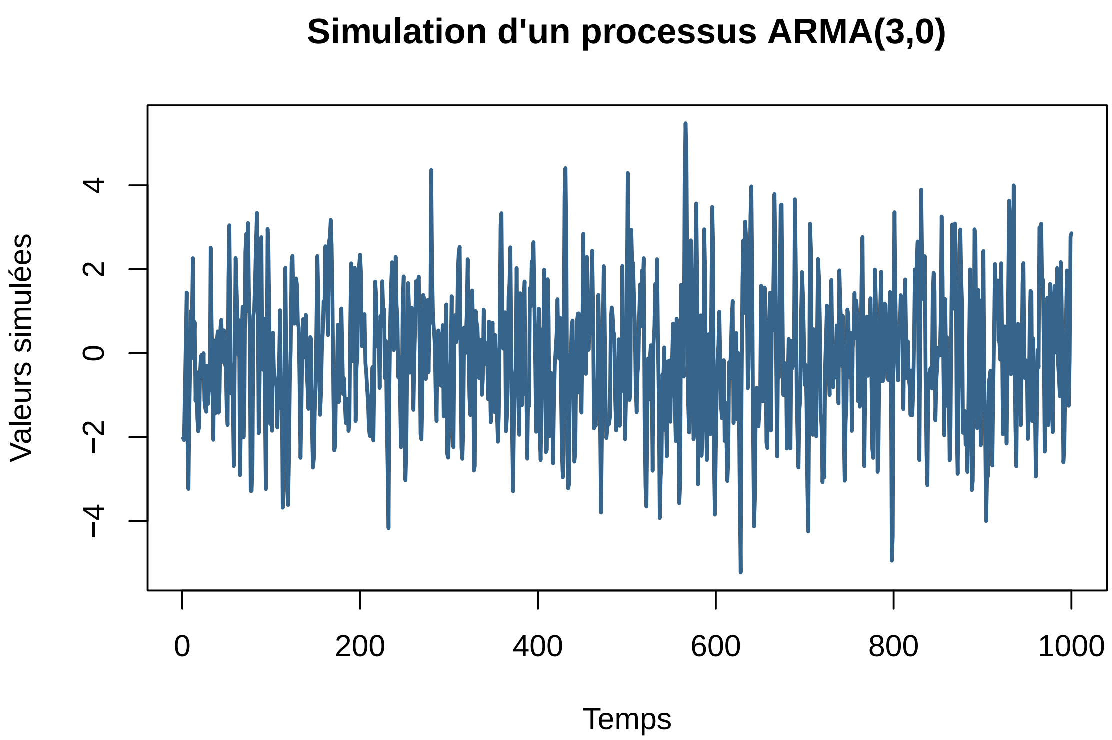
```
# Paramètres du modèle :
beta = c(1, .5, -.5)
rho = ARMAacf(ma = beta)
# Formule théoriques :
rho_1 <- (beta[1]+beta[1]*beta[2]+beta[2]*beta[3])/(1+sum(beta^2))
rho_2 <- (beta[2]+beta[1]*beta[3])/(1+sum(beta^2))
rho_3 <- (beta[3])/(1+sum(beta^2))

# Affichage des résultats :

data.frame(
  Lag = 1:3,
  Formule = c(rho_1, rho_2, rho_3),
  Fonction = rho[2:4]
)
```

```
  Lag Formule Fonction
1   1     0.5      0.5
```

```
2   2     0.0      0.0
3   3    -0.2     -0.2
```

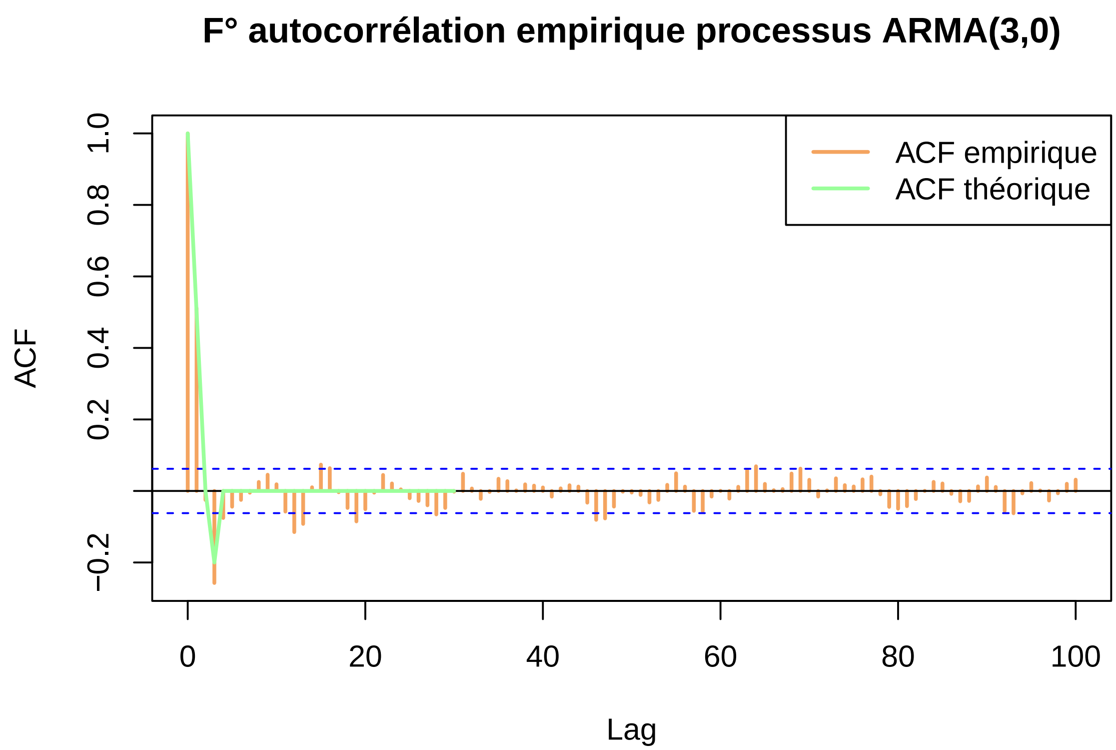
2.2.2 Simulation du processus en utilisant la fonction intégrée :

```
x = arima.sim(model = list(ma = beta), n = 1000)
plot(x,
     main = "Simulation d'un processus ARMA(3,0)",
     ylab = "Valeurs simulées",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2)
```




```
acf(x,
    main = "F° autocorrélation empirique processus ARMA(3,0)",
    col = palette_couleur[2],
    lwd = 2,
    lag.max = 100)

rho = ARMAacf(ma = c(1, .5, -.5), lag.max = 30)
lines(0:30, rho, col = palette_couleur[3], lwd = 2)
legend(
  'topright',
  c('ACF empirique', 'ACF théorique'),
  col = palette_couleur[2:3],
  lty = 1,
  lwd = 2
```

```
)
```



A retenir :

Pour un MA(q), on a rho(h)=0 pour h>q.

2.3 Exercice 3 :

$$X_t = \alpha X_{t-1} + \epsilon_t \text{ avec } \epsilon_t \sim \mathcal{N}(0,1)$$

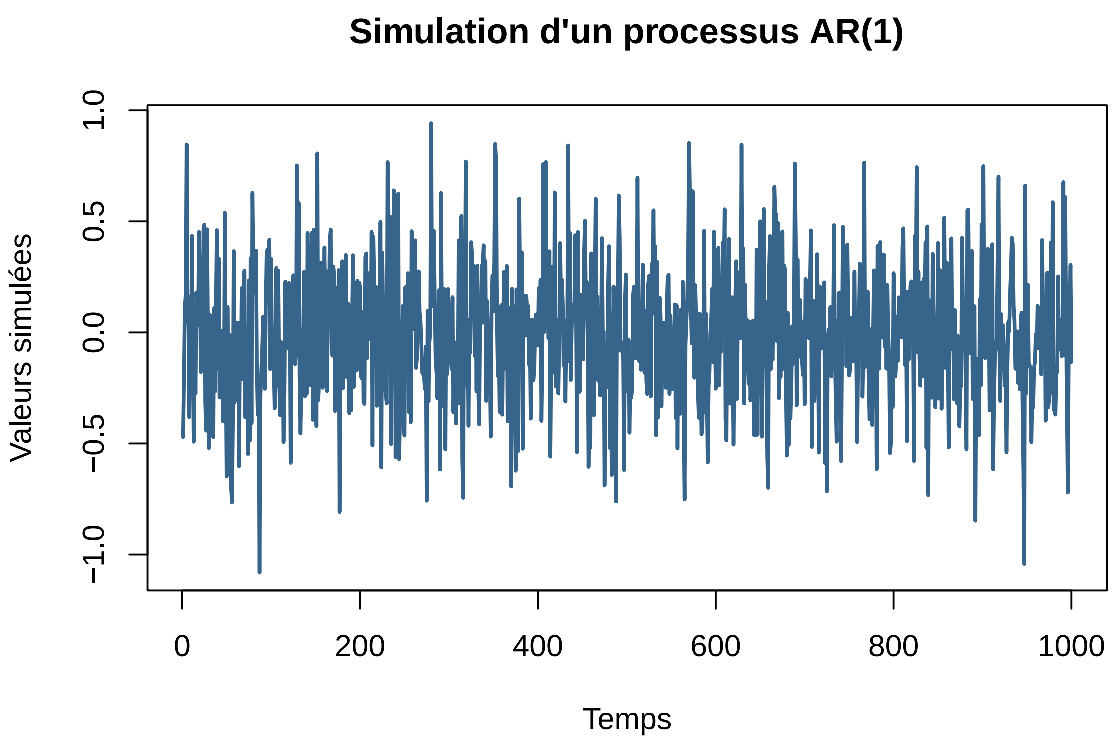
2.3.1 Fonction de simulation d'un processus Ar(1):

```
ar1 = function(alpha, sigma = 1, Time= 1000) {
  x = numeric(Time)
  x[1] = rnorm(1, sd = sqrt(sigma ^ 2 / (1 - alpha ^ 2)))
  for (t in 2:Time) {
    x[t] = alpha * x[Time- 1] + rnorm(1, sd = sigma)
  }
  return(x)
}
```

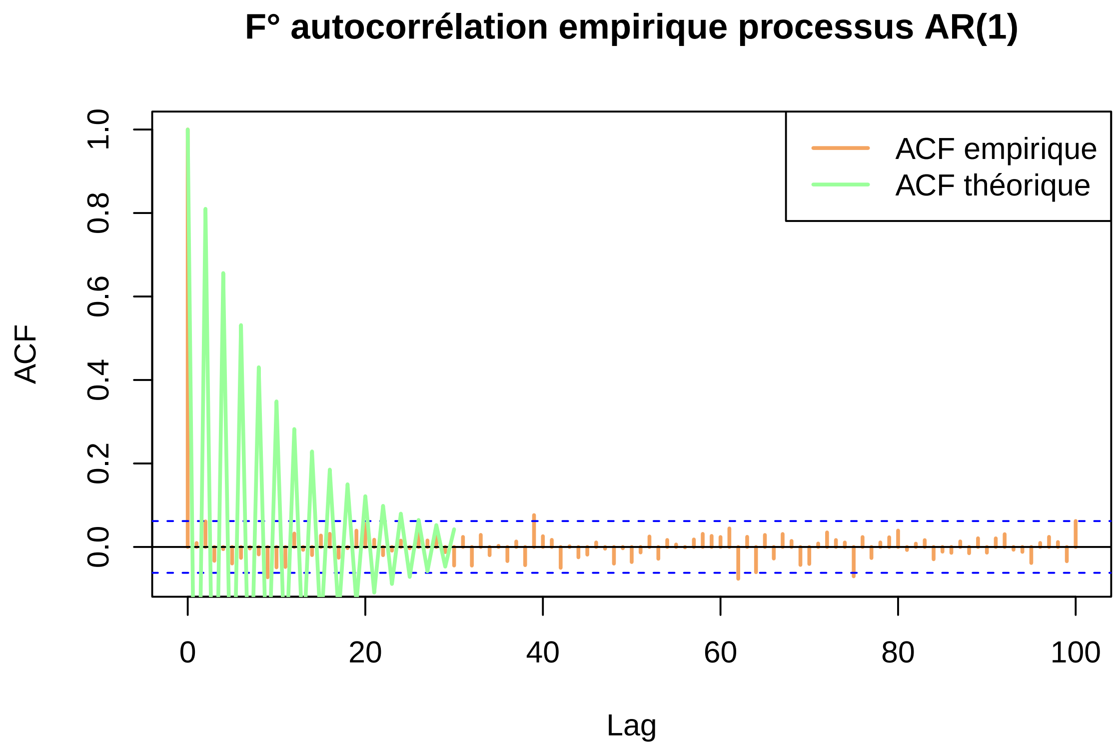
2.3.2 Simulation du processus :

```
alpha = -0.9
# alpha = 0.9
sigma = sqrt(0.1)
Time = 1000
x = ar1(alpha, sigma, Time)
```

```
plot(x,
     main = "Simulation d'un processus AR(1)",
     ylab = "Valeurs simulées",
     xlab = "Temps",
     type = "l",
     col = palette_couleur[1],
     lwd = 2,
     xlim = c(1, Time))
```




```
acf(x,
    main = "F° autocorrélation empirique processus AR(1)",
    col = palette_couleur[2],
    lwd = 2,
    lag.max = 100)
tab = 0:30
rho = alpha ^ tab
lines(tab, rho, col = palette_couleur[3], lwd = 2)
legend(
  'topright',
  c('ACF empirique', 'ACF théorique'),
  col = palette_couleur[2:3],
  lty = 1,
  lwd = 2
)
```



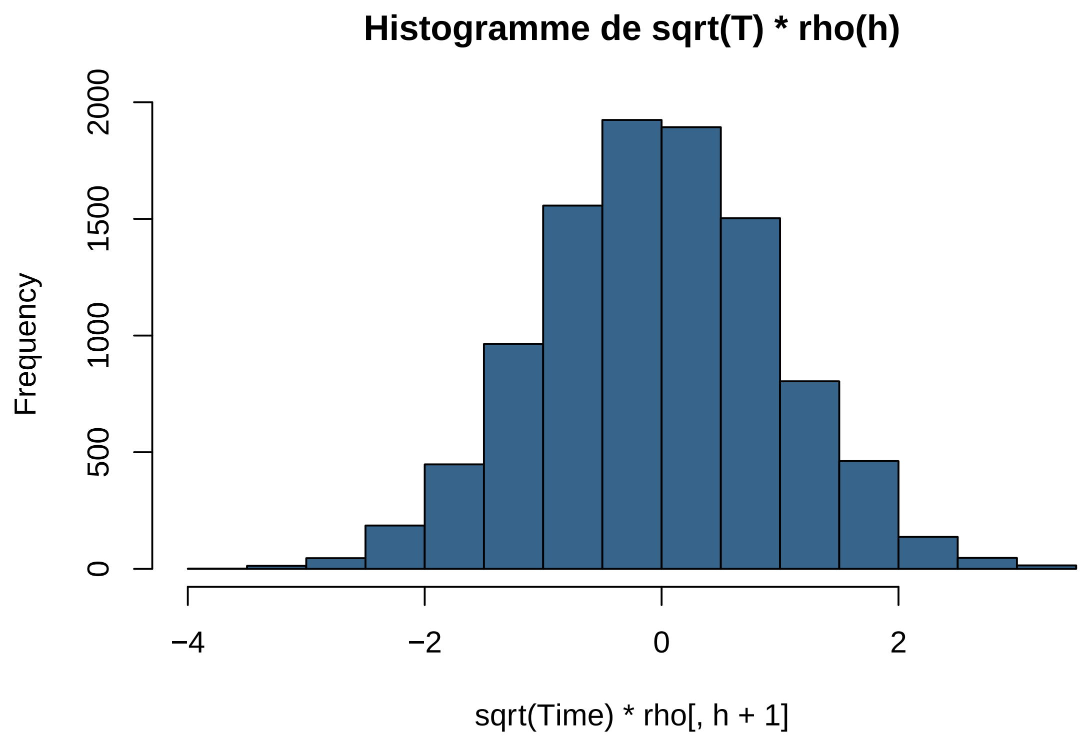
2.4 Exercice 5 :

2.4.1 Simulation du bruit blanc :

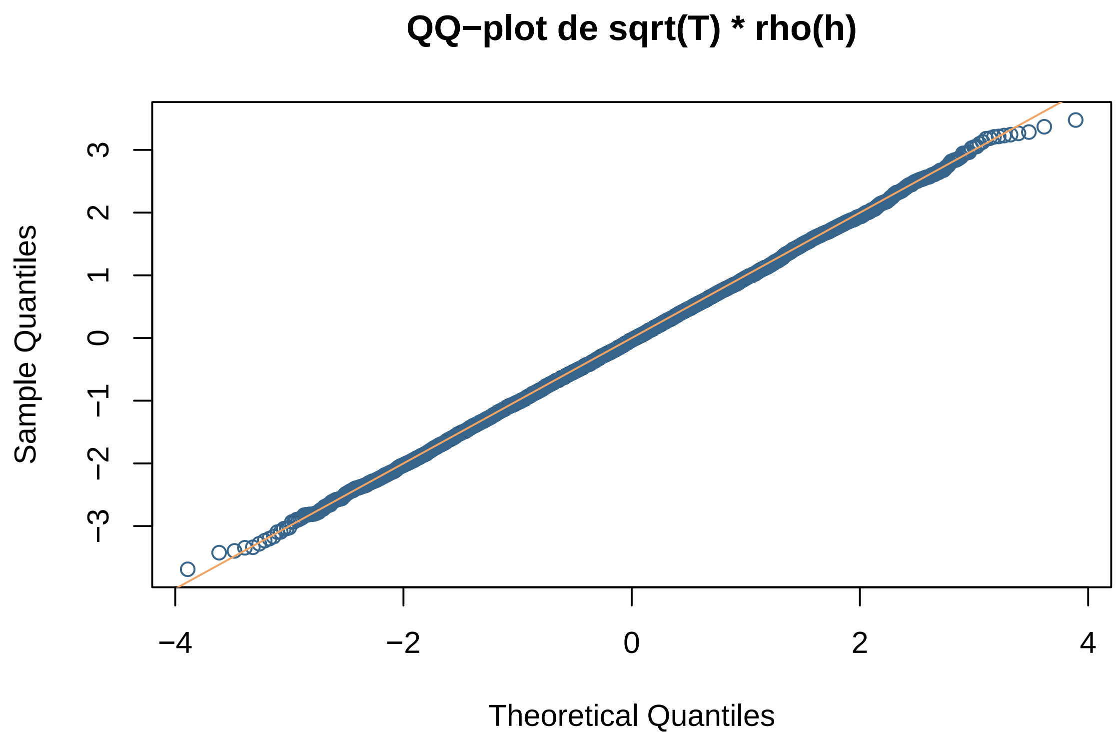
```
N = 10 ^ 4  #nombre de simulations
Time = 1000  #longueur des séries temporelles
rho = NULL
pv = NULL
for (n in 1:N) {
  #on répète l'expérience N fois
  x = rnorm(Time)  #simulation d'un bruit blanc gaussien
  rho = rbind(rho, acf(x, plot = FALSE)$acf)  #calcul et stockage ACF (sans tracer le graphique)
  pv = c(pv, Box.test(x, lag = 5)$p.value) #calcul et stockage p-value du test de Portmenteau
}
```

2.4.2 Vérification que $\sqrt(T) \times \hat{\rho}(h) \sim \mathcal{N}(0,1)$:

```
h = 1
hist(sqrt(Time) * rho[, h+1], # La fonction d'autocorrélation commence à 0.
     main = "Histogramme de sqrt(T) * rho(h)",
     col = palette_couleur[1])
```




```
# Test de normalité :
qqplot = qqnorm(sqrt(Time) * rho[, h+1],
                main = "QQ-plot de sqrt(T) * rho(h)",
                col = palette_couleur[1])
abline(0, 1, col = palette_couleur[2])
```



Commentaire :

On peut retrouver des résultats similaires sur les autres valeurs de h.

2.4.3 L'intervalle de confiance à 95% des coefficients d'auto-corrélation :

```
nb_value <-
  sum(rho[, h + 1] > -1.96 / sqrt(Time) &
        rho[, h + 1] < 1.96 / sqrt(Time)) / N
nb_value # Très proche de 0.95 (prendre N assez grand)
```

```
[1] 0.9516
```

```
alpha = 0.05
sum(pv > 0.05) / N
```

```
[1] 0.9533
```

```
# H0 = c'est un bruit blanc ,  acceptés dans + de 95% des cas
```

2.4.4 Vérification sur une simulation MA(1) :

```
beta = 0.9
sigma = 0.1
rho = NULL ; pv = NULL
N = 10 ^ 4
for (i in 1:N){
  eps = sigma * rnorm(Time + 1)
```

```
  y = eps[2:(Time + 1)] + beta * eps[1:Time]
  rho = rbind(rho, acf(y, plot = FALSE)$acf)
  pv = c(pv, Box.test(y, lag = 5)$p.value)
  }
```

2.4.4.1 Simulation :

2.4.5 Vérifier que $\hat{\rho}(1)$ n'est plus dans l'intervalle pour un MA(1) :

```
h = 1
nb_value = sum(rho[,h+1] > -1.96/sqrt(Time) &
      rho[,h+1] < 1.96/sqrt(Time))/N
nb_value
```

```
[1] 0
```

```
# Test du bruit blanc :
alpha = 0.05
sum(pv > alpha) / N
```

```
[1] 0
```

Commentaire :

L'intervalle de confiance n'est plus respecté pour un MA(1) (on rejette l'hypothèse de bruit blanc).

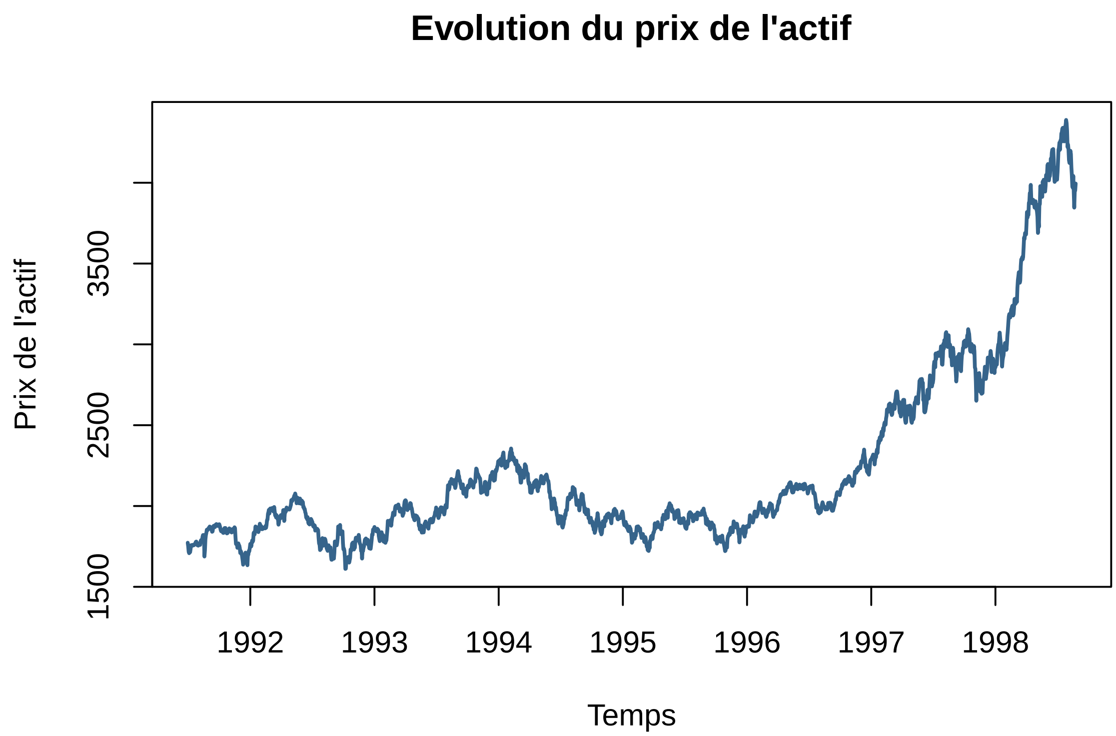
On rejette l'hypothèse de bruit blanc pour un MA(1) (on a une dépendance entre les valeurs successives).

2.4.6 Test de normalité des rendements dans la modélisation de Black-Scholes :

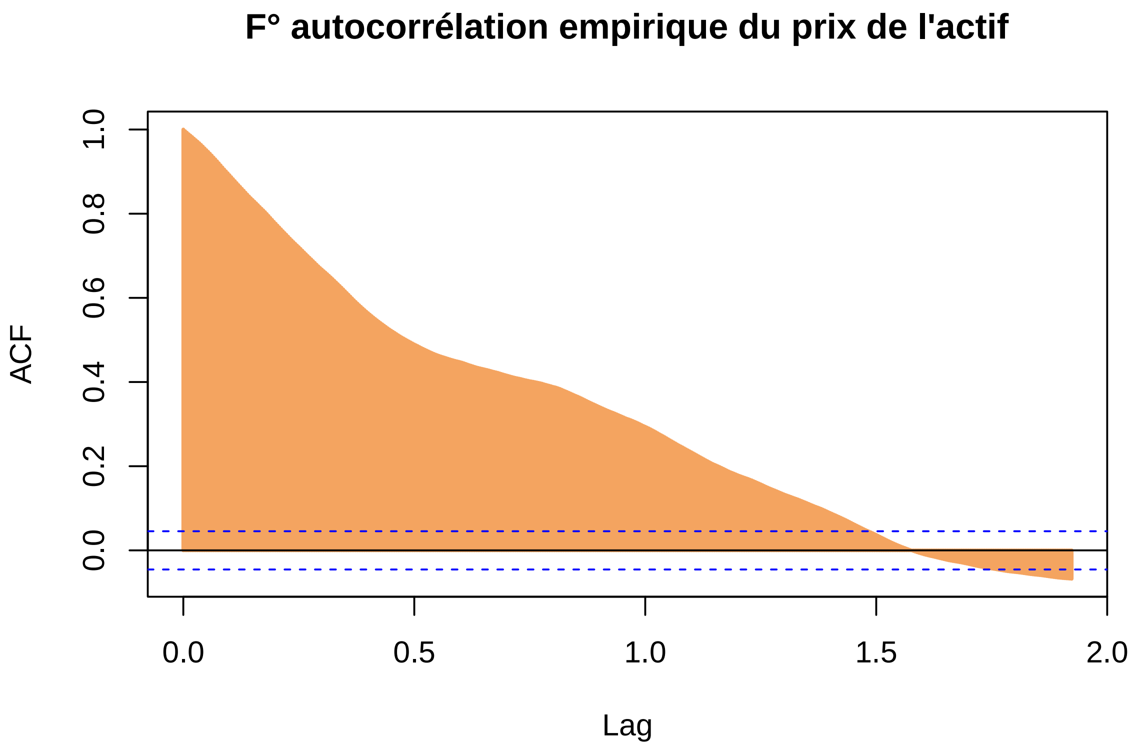
$$X_t = ln(\frac{S_t}{S_{t-1}}) = ln(S_t) - ln(S_{t-1}) S_t : \text{Prix de l'actif à l'instant t} Hypothse : X_t \sim \mathcal{N}(\mu, \sigma^2) iid$$

```
x = EuStockMarkets[, 3]
plot(
  x,
  main = "Evolution du prix de l'actif",
  ylab = "Prix de l'actif",
  xlab = "Temps",
  type = "l",
  col = palette_couleur[1],
  lwd = 2
)
```

2.4.6.1 Importation et représentation graphique :

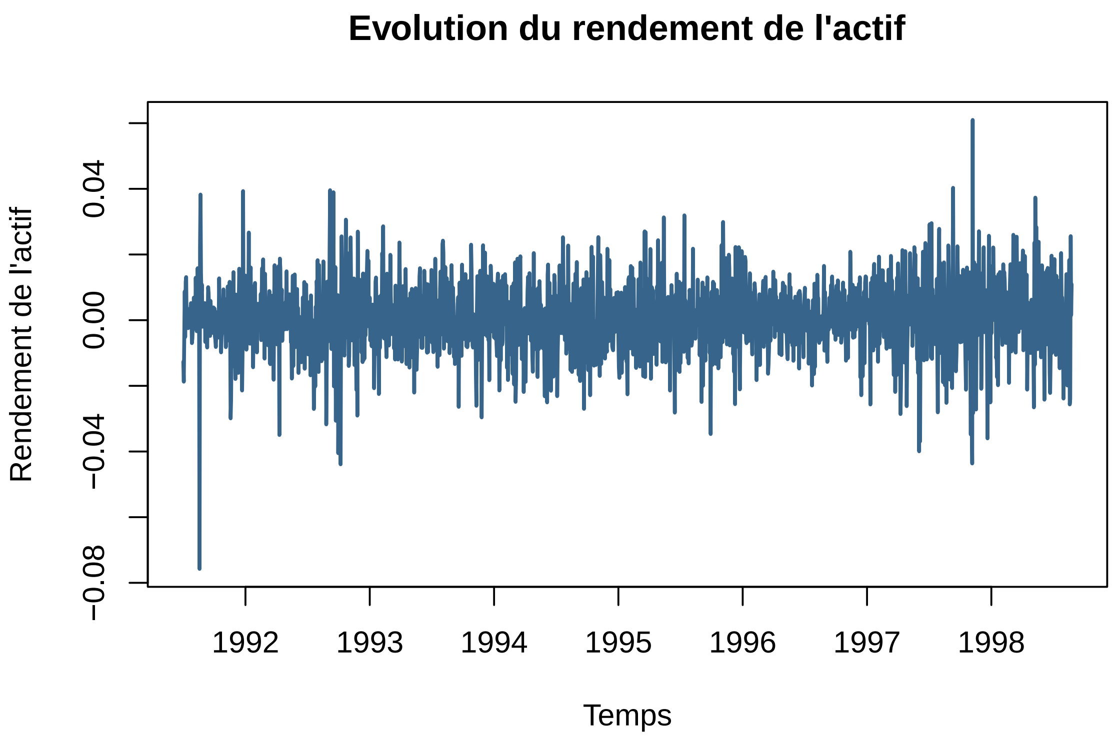



```r
acf(
  x,
  main = "F° autocorrélation empirique du prix de l'actif",
  col = palette_couleur[2],
  lwd = 2,
  lag.max = 500
)
```

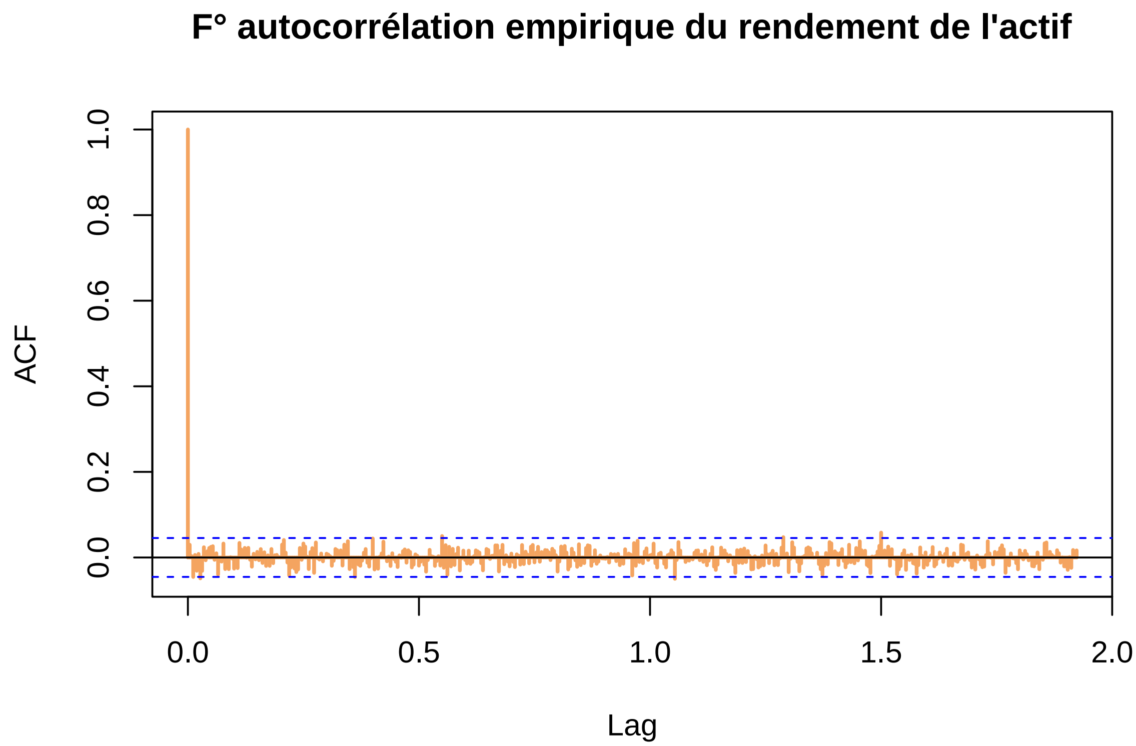



```
y = diff(log(x))
plot (
  y,
  main = "Evolution du rendement de l'actif",
  ylab = "Rendement de l'actif",
  xlab = "Temps",
  type = "l",
  col = palette_couleur[1],
  lwd = 2
)
```

2.4.6.2 Log différenciation de l'actif :




```r
acf(
  y,
  main = "F° autocorrélation empirique du rendement de l'actif",
  col = palette_couleur[2],
  lwd = 2,
  lag.max = 500
)
```

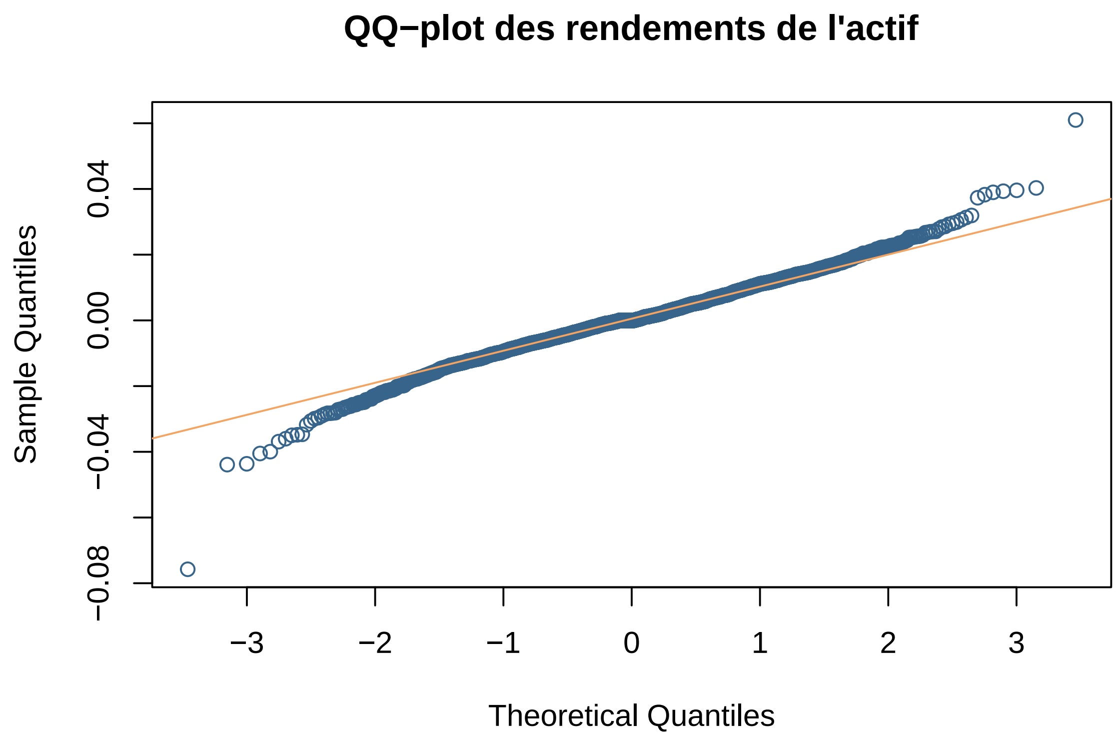



```r
Box.test(y, lag = 5) #test de Portmanteau (BB)
```

2.4.6.3 Test sur le BB et la normalité des rendements :

```
	Box-Pierce test

data:  y
X-squared = 7.3488, df = 5, p-value = 0.196
```

```r
qqnorm(y,
       main = "QQ-plot des rendements de l'actif",
       col = palette_couleur[1])
qqline(y, col = palette_couleur[2])
```




```r
shapiro.test(y) #test de normalité des rendements
```

```
    Shapiro-Wilk normality test

data:  y
W = 0.98203, p-value = 1.574e-14
```

Rappel :

Le test de Shapiro permet de tester l'hypothèse nulle de normalité des résidus.

$$\begin{cases} H_0 : \text{Les résidus suivent une loi normale} \\ H_1 : \text{Les résidus ne suivent pas une loi normale} \end{cases}$$

Le test de Box-Pierce permet de tester l'hypothèse nulle d'indépendance des observations d'une série temporelle.

$$\begin{cases} H_0 : \rho(1) = 0, \text{X est un bruit blanc} \\ H_1 : \rho(1) \neq 0, \text{X n'est pas un bruit blanc} \end{cases}$$

Dans notre cas, les log-rendements de la série temporelle ne suivent pas une loi normale mais sont des bruits blancs. C'est-à-dire que les rendements sont indépendants et identiquement distribués.

2.5 Exercice 6 :

On se base sur une modèle AR(1) :

$$X_t = \alpha X_{t-1} + \epsilon_t \text{ avec } \epsilon_t \sim \mathcal{N}(0,1)$$

2.5.1 Créer une fonction d'estimation du maximun de vraissemblance associé à la série temporelle :

```
ar1.estim = function(x) {
  n = length(x)
  x1 = x[1:(n - 1)]
  x2 = x[2:n]
  alpha = sum(x1 * x2) / sum(x1 ^ 2)
  sigma = sqrt(sum((x2 - alpha * x1) ^ 2) / n)
  return(c(alpha, sigma))
}

# Création d'une série temporelle :
alpha = 0.9 ; sigma = 0.1 ; Time = 1000
x = arima.sim(model = list(ar = alpha), n = Time, sd = sigma)

# Fonction créée :
yf = ar1.estim(x)

# Estimation R :
y = ar(x, aic = FALSE, order.max = 1, demean = FALSE, method = "ols")

# Résultats :
data.frame(
  Explication = c("Vrai valeur", "Estimation F°", "Estimation R"),
  alpha = round(c(alpha,yf[1], y$ar[1]),4),
  sigma = round(c(sigma, yf[2], y$var.pred[1]),4))
```

```
    Explication  alpha  sigma
1   Vrai valeur 0.9000 0.1000
2 Estimation F° 0.9106 0.0962
3  Estimation R 0.9106 0.0093
```

2.5.2 Etude sur les estimateurs par simulation :

```
Nsimu = 10 ^ 2 ; alpha = 0.9 ; sigma = 0.1
tabT = c(20,100,1000,10000) # Différentes longueur de série

# Matrice de stockage des estimations :
alphaols = matrix(0, Nsimu, length(tabT))
sigmaols = matrix(0, Nsimu, length(tabT))
alphamle = matrix(0, Nsimu, length(tabT))
sigmamle = matrix(0, Nsimu, length(tabT))

for (i in 1:length(tabT)) {
  T = tabT[i]
  for (j in 1:Nsimu) {
    x = arima.sim(model = list(ar = alpha), sd = sigma, n = T)
    fit = ar(x, aic = FALSE, order.max = 1, demean = FALSE, method = 'ols')
    alphaols[j, i] = fit$ar
    sigmaols[j, i] = fit$var.pred
```

```r
    fit = ar(x, aic = FALSE, order.max = 1, demean = FALSE, method = 'mle')
    alphamle[j, i] = fit$ar
    sigmamle[j, i] = fit$var.pred
  }
}
```

```r
# Comparaison du biais des estimateurs :
data.frame (
  Duration = tabT,
  MLE_mean = round(apply(alphamle, 2, mean),4),
  OLS_mean = round(apply(alphaols, 2, mean),4)
)
```

2.5.2.1 Comparaison en moyenne :

```
  Duration MLE_mean OLS_mean
1       20   0.8621   0.8602
2      100   0.8831   0.8805
3     1000   0.8971   0.8971
4    10000   0.9007   0.9007
```

Commentaire :

Au regard des moyennes l'estimateur est asymptotiquement sans biais (biais tend vers 0 lorsque T grandit)

```r
data.frame(
  duration = tabT,
  MLE = round(apply(alphamle, 2, var),4),
  OLS = round(apply(alphaols, 2, var),4))
```

2.5.2.2 Comparaison de la variance des estimateurs :

```
  duration    MLE    OLS
1       20 0.0115 0.0140
2      100 0.0025 0.0024
3     1000 0.0002 0.0002
4    10000 0.0000 0.0000
```

Commentaire :

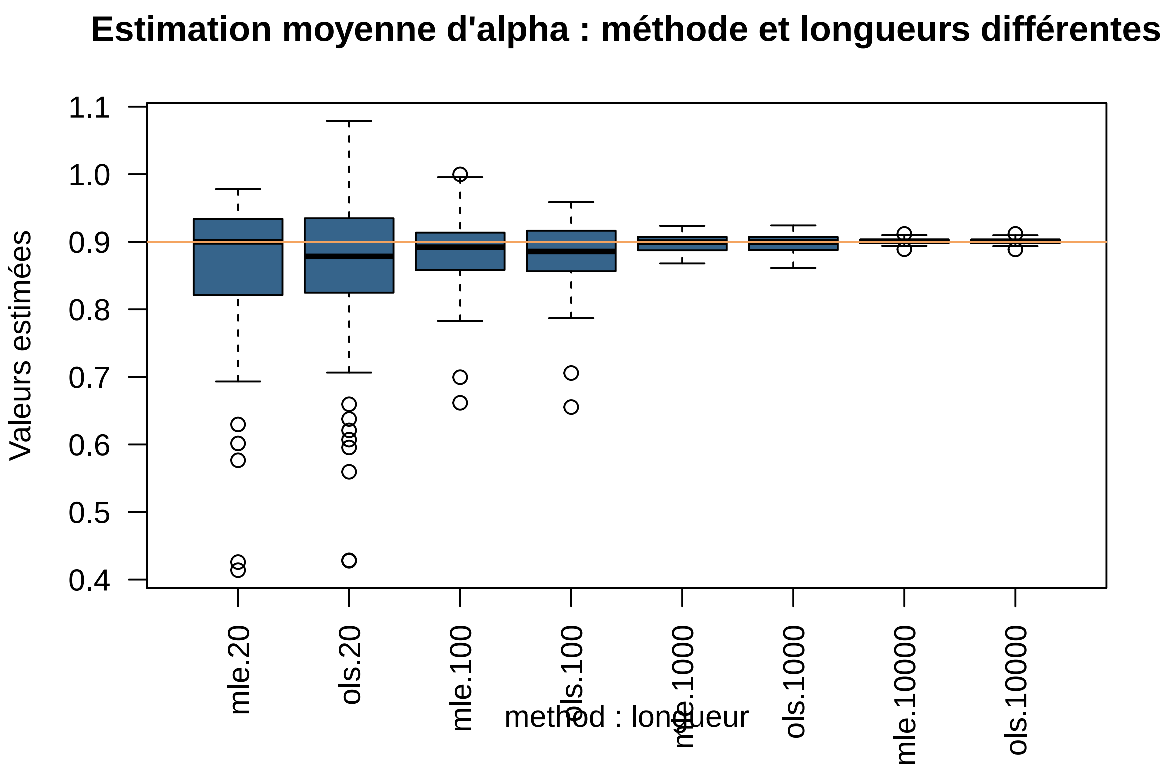
La variance des estimateurs tend vers 0 lorsque T grandit (convergence ordre 2)

```r
alpha = cbind(alphaols, alphamle)
sigma = cbind(sigmaols, sigmamle)

method = cbind(matrix('ols', Nsimu, length(tabT)), matrix('mle', Nsimu, length(tabT)))

longueur = cbind(matrix(rep(tabT, Nsimu), Nsimu, length(tabT), byrow = TRUE),
                 matrix(rep(tabT, Nsimu), Nsimu, length(tabT), byrow = TRUE))
```

2.5.2.3 Représentation graphique boxplot :

```r
boxplot(alpha ~ method + longueur, las = 2, col = palette_couleur[1],
        main = "Estimation moyenne d'alpha : méthode et longueurs différentes",
        ylab = "Valeurs estimées")

abline(h = 0.9, col = palette_couleur[2])
```

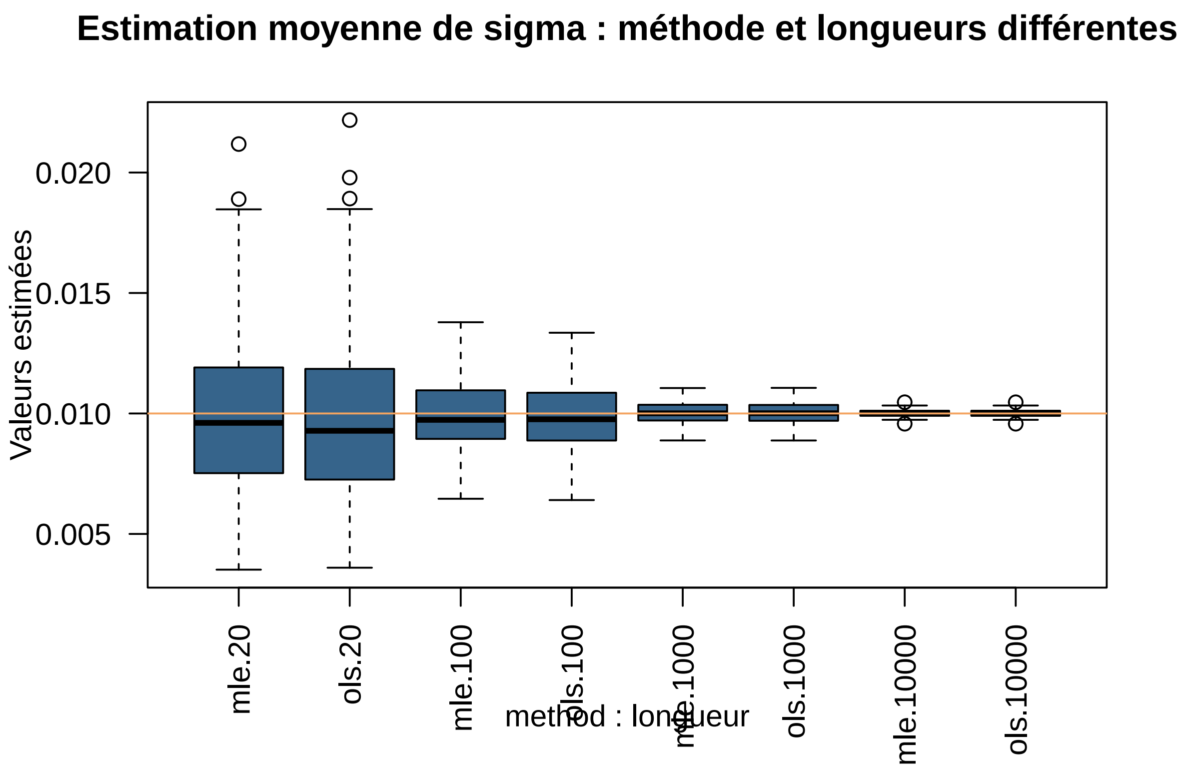
2.5.2.3.1 Estimation moyenne des alpha :




```r
# 0.9 Correspond à la vraie valeur de alpha
```

```r
boxplot(sigma ~ method + longueur, las = 2, col = palette_couleur[1],
        main = "Estimation moyenne de sigma : méthode et longueurs différentes",
        ylab = "Valeurs estimées")
abline(h = 0.1 ^ 2, col = palette_couleur[2])
```

2.5.2.4 Estimation moyenne de sigma :



Commentaire :

Les deux méthodes (mle et ols) donnent des résultats proches. La méthode mle semble donner des résultats légèrement meilleurs pour les petits échantillons. On observe une sous-estimation pour les petites séries temporelles (T=20). On observe une convergence lorsque T tend vers l'infini.

2.5.3 Création d'une fonction de prédiction pour le modèle AR(1) :

Calcul de l'écart type et de la moyenne de la prédiction :

Prédiction de X_t à partir de $(X_1, ..., X_{t-h})$:

```
predictionAR1 = function(x, a, sig, hmax) {
  return(list(moy = x * a ^ (1:hmax),
              sig = sqrt(sig ^ 2 * (1 - a ^ (2 * (1:hmax))) / (1 - a ^ (2)))))
}
```

2.5.4 Simulation et prédiction : (A reprendre simplification possible dans le code)

```
alpha = 0.9 ; sig = 0.1 ; Time = 10 ^ 5
X = arima.sim(model = list(ar = alpha),sd = sig, n= Time)

moypred = NULL
sigpred = NULL
h = 1  #horizon de prévision
for (t in (h + 1):Time) {
  pred = predictionAR1(X[t - h], alpha, sig, h)
  moypred[t] = pred$moy[h]
  sigpred[t] = pred$sig[h]
```

```
}

IC_pr_value = sum(X > moypred - 1.96 * sigpred &
      X < moypred + 1.96 * sigpred, na.rm = TRUE) / (Time - h + 1)

cat("Pourcentage de valeur dans l'intervalle = ", round(IC_pr_value,4), "\n")
```

```
Pourcentage de valeur dans l'intervalle =  0.9509
```

2.6 Exercice 7 :

2.6.1 Trouver les racines d'un polynôme caractéristique :

```
cat("Racines du polynôme : ", polyroot(c(1, -.9, .3)), "\n")
```

```
Racines du polynôme :  1.5+1.040833i 1.5-1.040833i
```

```
cat("Module des racines : ", Mod(polyroot(c(1, -.9, .3))), "\n")
```

```
Module des racines :  1.825742 1.825742
```

Commentaire :

Proposition si le module des racines est supérieur à 1 alors le processus est stationnaire.

2.6.2 Calcul de l'ACF avec Yule-Walker à l'aide du système d'équations :

```
alpha1 = 0.1
alpha2 = -0.3
sigma = 1

mat = cbind(
  c(1, -alpha1, -alpha2, 0, 0, 0),
  c(-alpha1, 1 - alpha2, -alpha1, -alpha2, 0, 0),
  c(-alpha2, 0, 1, -alpha1, -alpha2, 0),
  c(0, 0, 0, 1, -alpha1, -alpha2),
  c(0, 0, 0, 0, 1, -alpha1),
  c(0, 0, 0, 0, 0, 1)
)
b = c(sigma ^ 2, 0, 0, 0, 0, 0)

gamma = solve(mat, b)  #Fonction_autocovariance
f_autocor = gamma / gamma[1]  # Fonction_autocorrélation
```

2.6.3 Simulation et comparaison sur la fonction d'autocorrélation :

```
x = arima.sim(model = list(ar = c(alpha1, alpha2)),
              sd = sigma,
              n = 10 ^ 4)

auto_cor_empi <- acf(x, lag.max = 5, plot = FALSE)$acf
auto_cor_theo <- ARMAacf(ar = c(alpha1, alpha2), lag.max = 5)
auto_cov_empi <- acf(x, lag.max = 5, type='covariance', plot=FALSE)$acf

# Affichage des résultats :
```

```r
data.frame(
  Lag = 1:5,
  Yule_walker = round(f_autocor[2:6],4),
  Empirique = round(auto_cor_empi[2:6],4),
  Théorique = round(auto_cor_theo[2:6],4))
```

```
  Lag Yule_walker Empirique Théorique
1   1      0.0769    0.0798    0.0769
2   2     -0.2923   -0.2963   -0.2923
3   3     -0.0523   -0.0582   -0.0523
4   4      0.0825    0.0759    0.0825
5   5      0.0239    0.0206    0.0239
```

2.6.4 Création d'une fonction d'estimation Yule-Walker pour un modèle AR(2) :

On se base uniquement sur le système d'équations preant en compte les 3 premières valeurs de l'ACF.

```r
fitYWAR2 = function(x) {
  # Afc empirique :
  gammac = acf(x, lag.max = 2, type = 'covariance',
               plot = FALSE)$acf

  # Matrice A et vecteur b :
  A = matrix(c(gammac[2], gammac[1], gammac[2], gammac[3], gammac[2],
               gammac[1], 1, 0, 0), nrow = 3)
  b = c(gammac[1], gammac[2], gammac[3])
  thetamom = solve(A, b)
  return(thetamom)
}

# Vérification :
x = arima.sim(model = list(ar = c(alpha1, alpha2)),
              sd = sigma,
              n = 10 ^ 4)
resultat_r = ar(x, AIC= FALSE, order.max = 2,method = "yule-walker")
resultat_f = fitYWAR2(x)

# Affichage des résultats :
data.frame(
  Explication = c("Vrai valeur", "Estimation F°", "Estimation R"),
  alpha1 = round(c(alpha1, resultat_f[1], resultat_r$ar[1]), 4),
  alpha2 = round(c(alpha2, resultat_f[2], resultat_r$ar[2]), 4),
  sigma = round(c(sigma, resultat_f[3], resultat_r$var.pred[1]), 4))
```

```
    Explication alpha1  alpha2  sigma
1   Vrai valeur 0.1000 -0.3000 1.0000
2 Estimation F° 0.0967 -0.2992 0.9962
3  Estimation R 0.0967 -0.2992 0.9965
```

2.6.4.1 A reprendre #### :

2.7 Exercice 8 : Modélisation de taux de Vasicek :

Le modèle de Vasicek est un modèle de taux d'intérêt qui est souvent utilisé pour modéliser les taux d'intérêt à court terme. Le modèle est donné par l'équation différentielle stochastique suivante :

$$dr_t = -\alpha(r_t - \mu)dt + \sigma dW_t$$

où r_t est le taux d'intérêt à l'instant t, $_alpha$ est la vitesse de réversion, μ est le niveau de long terme, σ est la volatilité et W_t est un mouvement brownien.

On peut montrer que si R_t est solution de l'équation différentielle stochastique ci-dessus, alors R_t est une solution de l'équation différentielle stochastique suivante :

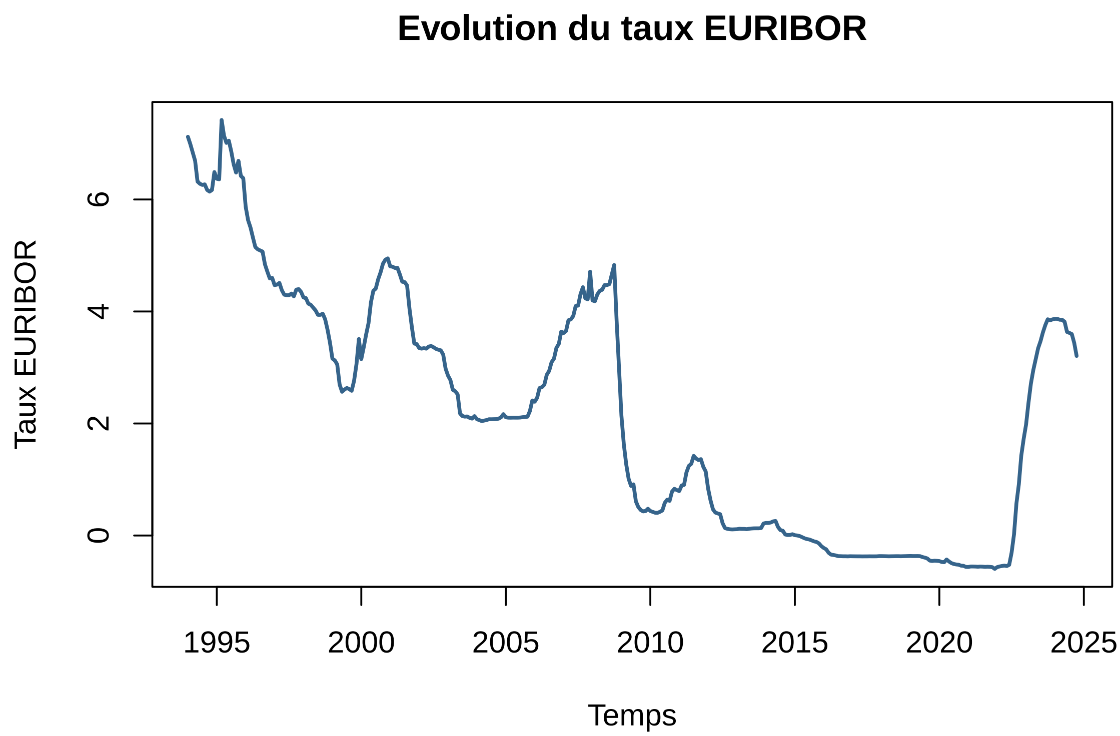
$$R_t = R_{t-1}e^{-\alpha} + \mu(1 - e^{-\alpha}) + \sigma \int_{t-1}^{t} e^{-\alpha(t-s)} dW_s$$

Avec en particulier :

$$\int_{t-1}^{t} e^{-\alpha(t-s)} dW_s \approx \mathcal{N}(0, \frac{\sigma^2}{2\alpha}(1 - e^{-2\alpha}))$$

2.7.1 Création d'un objet ts avec les données EURIBOR :

```
z <- read.csv("DATA/EURIBOR.csv", header = TRUE)
x = ts(data = z[, 3],
       start = c(1994, 1),
       frequency = 12) #création d'on objet de type ts
# La fréquence est de 12 car les données sont mensuelles

plot(x, col = palette_couleur[1],
     main = "Evolution du taux EURIBOR",
     ylab = "Taux EURIBOR",
     xlab = "Temps",
     lwd = 2)
```

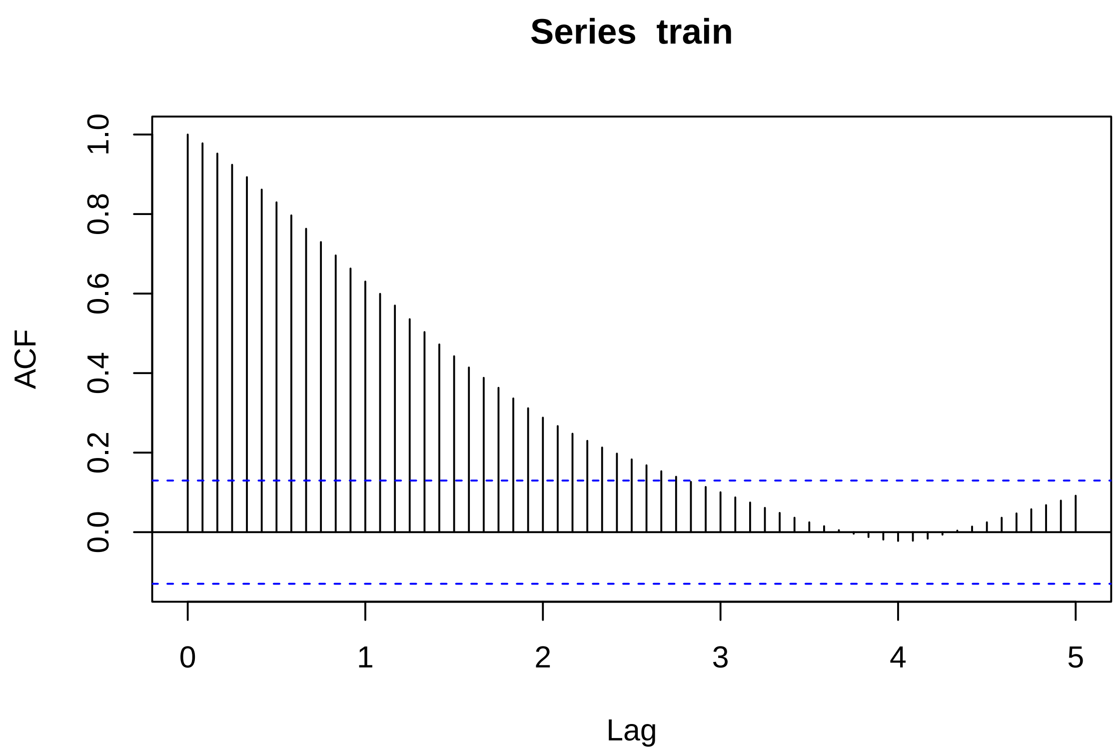


2.7.2 Séparation des données train et test :

```
train = window(x, end = 2012.999) #séquence d'apprentissage
test = window(x, start = 2013) #séquence de test
```

2.7.3 Fonction d'autocorrélation sur la séquence d'apprentissage

```
acf(train, lag.max = 5 * 12)
```



Pour un processus AR(1), la fonction d'autocorrélation décroit à une vitesse exponentielle vers 0

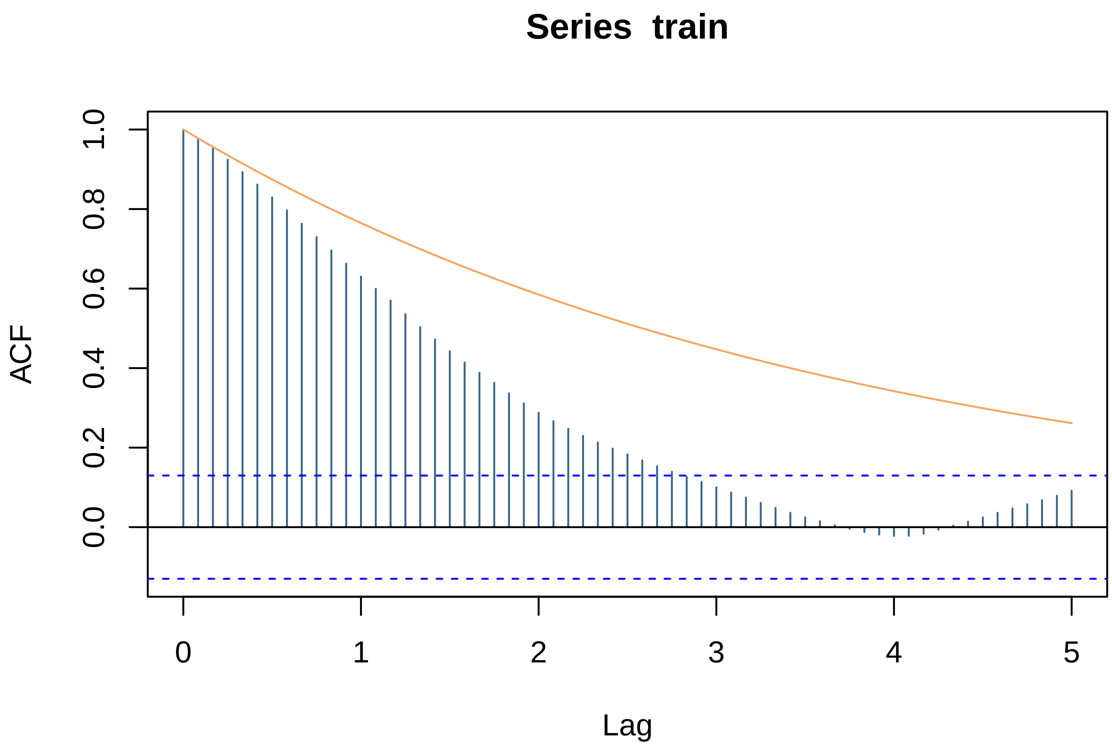
2.7.4 Ajustement d'un modèle AR(1) sur la séquence d'apprentissage :

```
fit <- ar(train, aic = FALSE, order.max = 1)
```

Commentaire : Comme la série n'est pas centrée on utilise pas l'argument dmean = FALSE. La série est alors automatiquement centrée.

```

lmax = 5 * 12 # 5 ans
ACFemp = acf(train, lag.max = lmax, col = palette_couleur[1])
acfAR = ARMAacf(ar = fit$ar, lag.max = lmax)
lines(ACFemp$lag, acfAR, col = palette_couleur[2])
```

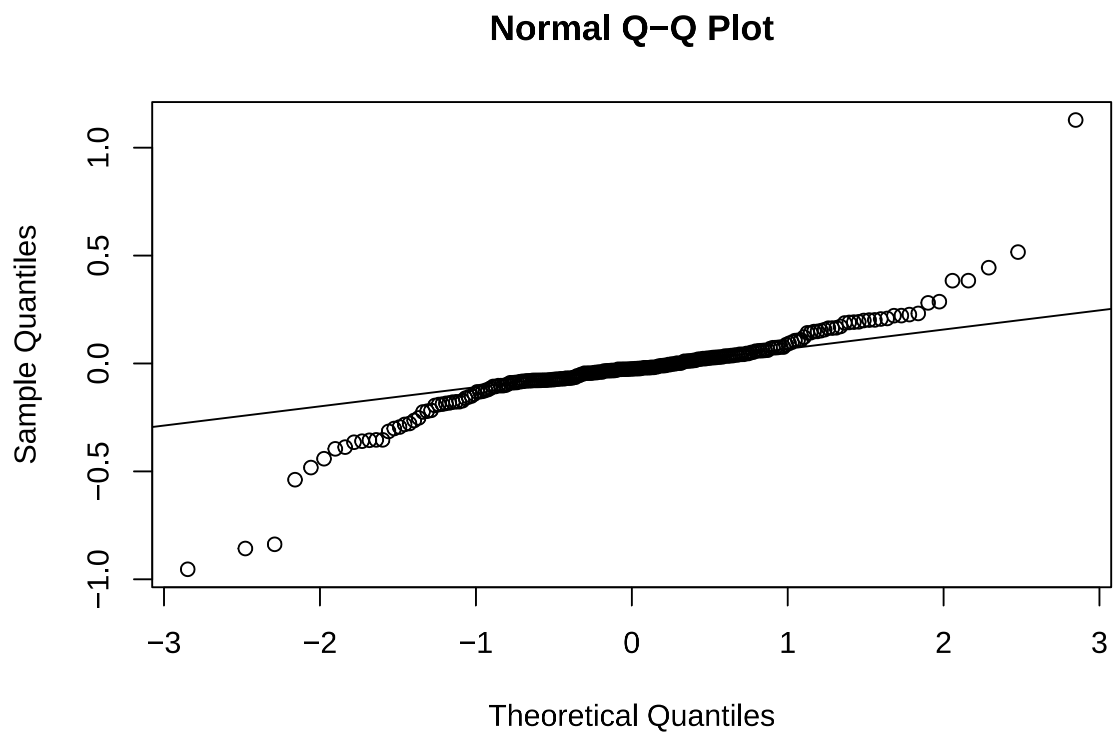
2.7.4.1 Comparaison des ACF :



Commentaire : L'acf du modèle ajusté proche de l'acf empirique seulement pour premiers lags.

2.7.5 Test sur les résidus empiriques :

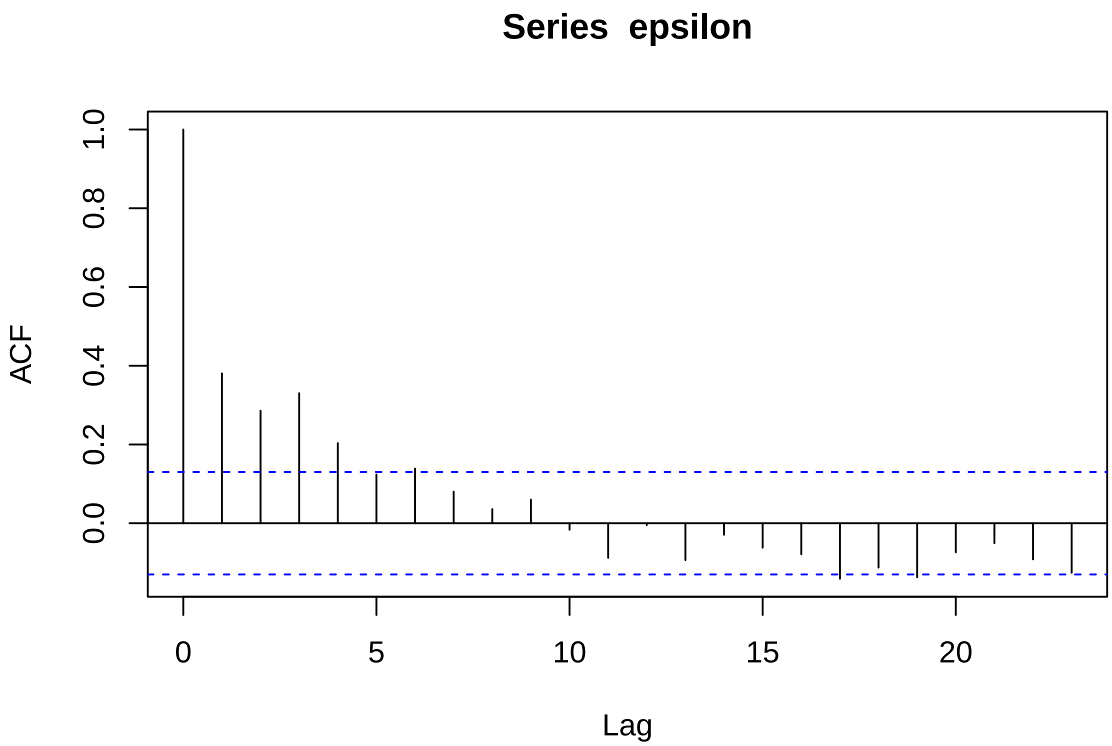
```
epsilon = fit$resid #résidu empirique renvoyé par fonction ar
epsilon = epsilon[!is.na(epsilon)]
qqnorm(epsilon) #le résidu suit-il une loi normale?
qqline(epsilon)
```




```r
shapiro.test(epsilon) #on refuse l'hypothèse de Gaussianité
```

```
    Shapiro-Wilk normality test

data:  epsilon
W = 0.8447, p-value = 2.474e-14
```

```r
acf(epsilon) #le résidu est-il un bruit blanc?
```




```
Box.test(epsilon, lag = 20) #le test de Portementeau refuse l'hypothèse de bruit blanc
```

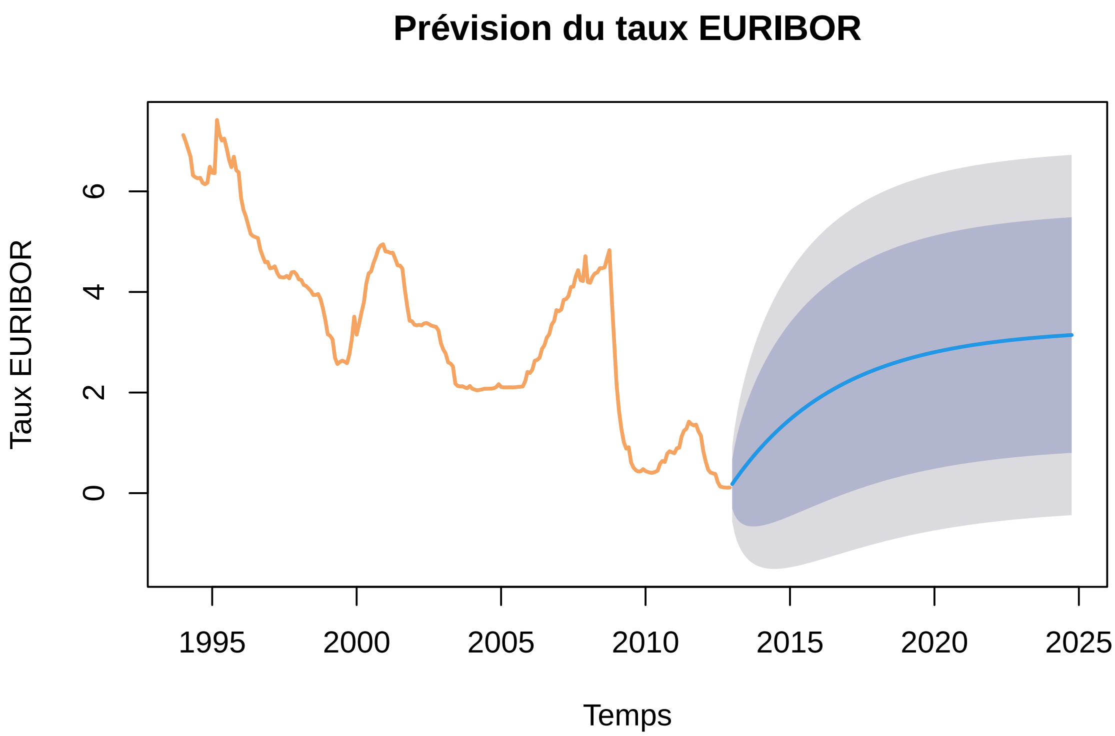
```
	Box-Pierce test

data:  epsilon
X-squared = 115.21, df = 20, p-value = 2.22e-15
```

Commentaires :

La plupart des coefficients de l'ACF empiriques ne sont pas dans l'intervalle de fluctuation en bleu, C'est une indication que le résidu n'est pas un bruit blanc. Le test de Portementeau refuse l'hypothèse de bruit blanc. Le test de Shapiro refuse l'hypothèse de normalité.
Le modèle AR(1) n'est pas valide, on pourrait tester un modèle ARMA plus complexe.

2.7.6 Prévision sur la séquence de test :

```
pm <- forecast(fit, h = length(test)) #représentation graphique
plot(pm,
     main = "Prévision du taux EURIBOR",
     ylab = "Taux EURIBOR",
     xlab = "Temps",
     col = palette_couleur[2],
     lwd = 2)
```



Commentaires : La prévision ponctuelle (courbe bleue) converge vers la moyenne empirique sur la séquence d'apprentissage ('retour à la moyenne'), la largeur de l'intervalle de prédiction augmente avec l'horizon de prédiction, les observations sont bien dans l'intervalle de prédiction à 95%.

2.8 Exerice 9 :

2.8.1 Calcul de l'autocovariance à partir des paramètres :

Dans un premier temps on résoud le système de telle sorte à trouver les valeurs de gamma. On résoud le problème avec seulement deux équations en isolant la valeur de gamma 2 qui dépend des valeurs de gamma 1

```
alpha = 0.8
beta = 0.5
sigma = 1
A = matrix(c(-alpha, 1, 1,-alpha), nrow = 2)
b = c(beta * sigma ^ 2,
      sigma ^ 2 * (1 + beta ^ 2 + alpha * beta))
v_gamma = solve(A, b)
v_gamma = c(v_gamma, alpha * v_gamma[2])

# Fonction auto-corrélation réelle :
dt = data.frame(theorique_r = ARMAacf(ar = alpha, ma = beta)[1:3],
                theorique = v_gamma / v_gamma[1])
round(dt, 4)
```

```
  theorique_r theorique
0      1.0000    1.0000
1      0.8878    0.8878
```

```
2       0.7102    0.7102
```

2.8.2 Calcul de l'autocovariance par simulation :

```
x = arima.sim(model = list(ar = alpha, ma = beta), sd = sigma, n = 10 ^ 5)
acf_sim = acf(x, lag.max = 3, plot = FALSE, type = 'covariance')

dt = data.frame(simulation = acf_sim$acf[1:3],
                theorique = v_gamma )
round(dt,5)
```

```
  simulation theorique
1    5.65015   5.69444
2    5.00915   5.05556
3    3.98906   4.04444
```

2.8.3 Ajustements par la méthode des moments:

On considère que alpha est connu ainsi on peut estimer beta et sigma par la méthode des moments.~ On créer une fonction a optimiser pour obtenir les valeurs de beta et sigma.

```
f_estim_moment = function(theta, r, alpha) {
  beta = theta[1]
  sig = theta[2]
  A = matrix(c(-alpha, 1, 1, - alpha), nrow = 2)
  x = c(r[1], r[2])
  b = c(beta * sig ^ 2, sig ^ 2 * (1 + beta ^ 2 + alpha * beta))
  d = A %*% x - b
  return(sum(d ^ 2))
}
```

2.8.3.1 Fonction d'estimation des moments :

2.8.3.2 Estimation des paramètres : Dans cette fonction, on prend les valeurs simulées des covariances gamma. Ainsi on obtientl la valeur d'alpha. On initialise des paramètres beta et sigma. On optimise la fonction d'estimation des moments pour obtenir les valeurs de beta et sigma.

```
fitARIMA11 = function(x) {
  acfsim = acf(x,
               lagmax = 3,
               plot = FALSE,
               type = 'covariance')  #calcul de l'acf
  r = acfsim$acf[1:3]
  a = r[3] / r[2]  #estimation de alpha
  theta0 = c(0, .5)  #valeur initiale des paramètres
  thetaf = optim(theta0, f_estim_moment, r, a, gr=NULL)$par
  return(list(
    alpha1 = a,
    beta1 = thetaf[1],
    sigma = thetaf[2]
  ))
}

y_theo <- fitARIMA11(x)
```

```r
y = arima(x,order=c(1,0,1))

dt = data.frame(
  théorique = unlist(y_theo),
  estimation = c(y$coef[1:2], y$sigma2)
)
rownames(dt) = c("alpha", "beta", "sigma")
round(dt, 4)
```

```
      théorique estimation
alpha    0.7964     0.7983
beta     0.5125     0.5006
sigma    0.9971     0.9995
```

3 Annales :