Code du projet de série temporelles

Ludovic Guyot & Louis Becquart

2023-05-02

Installation librairie

```
library(zoo) #Package permettant de formater les séries
library(fUnitRoots) #Package pour calculer les racines d'un polynome
library(forecast) #Package pour la prévision
```

Paramétrage de l'environnement

```
path <- "/Users/ludovic/Desktop/ENSAE/S2/series/serie_temp" #définition de l'emplacement du projet en l
setwd(path) #définition de l'espace de travail (working directory ou "wd")
```

Import et reraitement des données

lien des données INSEE : https://www.insee.fr/fr/statistiques/serie/010537309#Telechargement

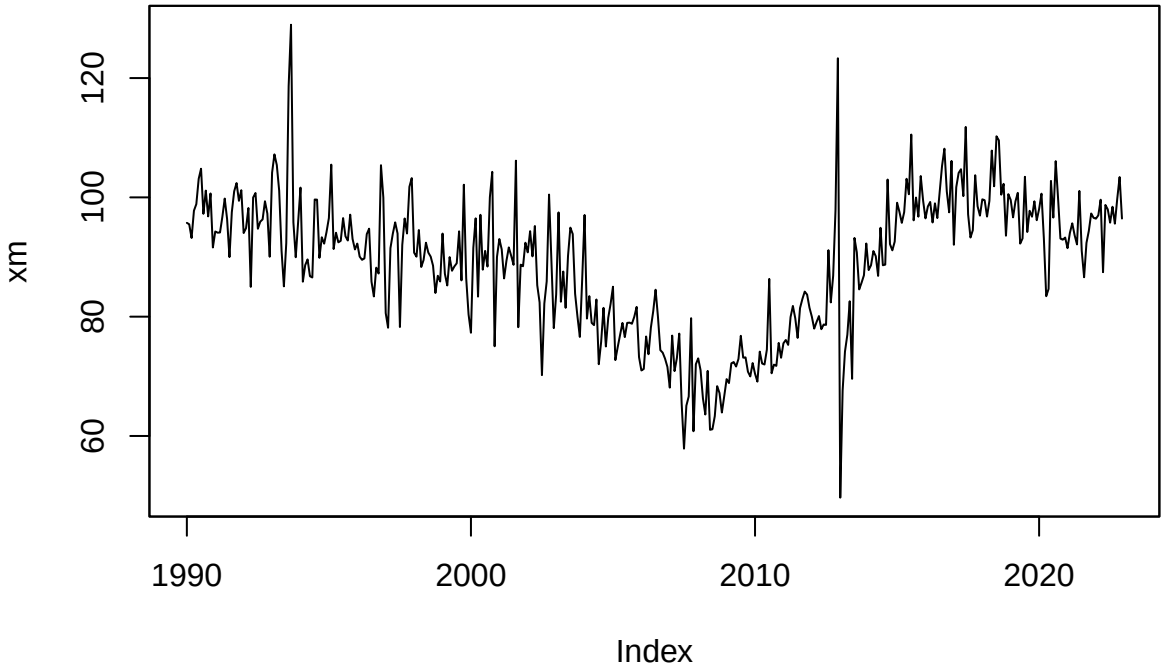
```
# définition du nom du fichier de données dans le wd
datafile <- "input/beer.csv"

# importe un fichier .csv dans un objet de classe data.frame
data <-
  read.csv(datafile, sep = ";")

# inverse l'ordre des données (car elles sont dans l'ordre chronologique inverse à l'import)
data <- data[nrow(data):1,]

# réinitialise l'index (car l'index peut créé des problèmes à cause de la commande précédente)
data <- data.frame(data, row.names = NULL)

#Création d'une liste de type date adapté à l'utilisation de zoo
data$Date <- as.yearmon(seq(from = 1990+0/12, to = 2023+1/12 , by=1/12))

# On stock les données sources avant d'enlever les 2 dernières valeurs
data.source <- data

# On stock la longeur de la série dans T
T <- length(data$Date)

# On enlève les 2 dernière valeurs en vue de la prévision out-of-sample
data <- data[-c(T-1, T),]
```

Formatage en zoo

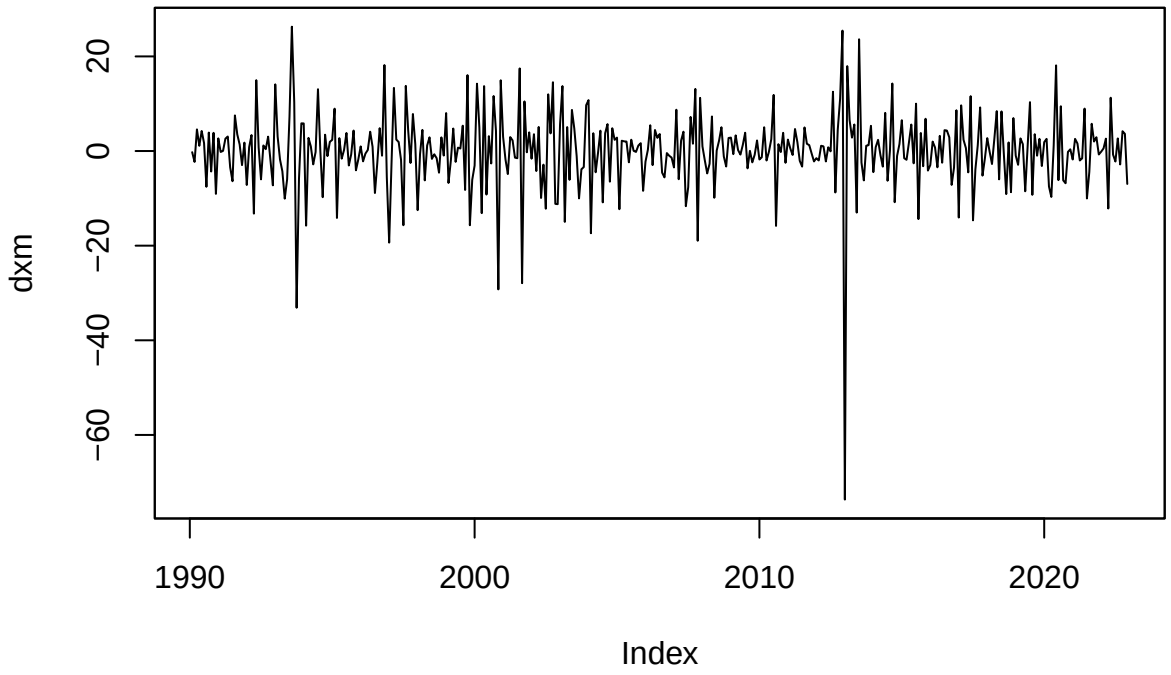
```
# converti les valeurs de data.source en série temporelle de type "zoo" en les indiçant par la colonne
xm.source<-
  zoo(data.source$value, order.by = data.source$Date)

# converti les valeurs de data en série temporelle de type "zoo" en les indiçant par la colonne Date. x
xm <-
  zoo(data$value, order.by = data$Date)
```

Représentation graphique de la série

```
plot(xm) #graphique  de la série xm
```




```
dxm <- diff(xm, 1) #création de dxm, la série xm en différence première
plot(dxm) #graphique  de la série dxm
```



Test de stationnarité

Nous régressons xm sur le temps afin de choisir le modèle adapté dans le test ADF :

```
index <- as.numeric(rownames(data)) #conversion de l'index en vecteur numérique
summary(lm(xm ~ index)) #régréssion de xm sur l'index, c'est à dire sur le temps
```

```
##
## Call:
## lm(formula = xm ~ index)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -39.151  -8.852   2.219   8.106  39.873
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 89.116615   1.157584  76.985   <2e-16 ***
## index       -0.001103   0.005054  -0.218    0.827
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.5 on 394 degrees of freedom
## Multiple R-squared:  0.0001208,  Adjusted R-squared:  -0.002417
## F-statistic: 0.0476 on 1 and 394 DF,  p-value: 0.8274
```

Nous choisissons le modèle avec constante sans tendance temporelle d'après la significativité des coefficients.

```
adf <- adfTest(xm, lag = 0, type = "c") # Test ADF avec constante sans tendance temporelle avec 0 lag
```

```
## Warning in adfTest(xm, lag = 0, type = "c"): p-value smaller than printed
## p-value
```

```
adf #résultat du test
```

```
##
## Title:
##  Augmented Dickey-Fuller Test
##
## Test Results:
##   PARAMETER:
##     Lag Order: 0
##   STATISTIC:
##     Dickey-Fuller: -7.583
##   P VALUE:
##     0.01
##
## Description:
##  Tue May  2 17:42:34 2023 by user:
```

L'hypothèse de non stationarité est réjetée. La série semble donc être stationnaire. Cependant ce test est biaisé si les résidus de la spécification du test ADF sont autocorrélés et il faut alors ajouter des lag dans la specification du test ADF. On teste, avec la fonction Qtests, l'hypothèse de non autoccorélation des résidus avec un test Ljung-Box pour notre test ADF avec 0 lag :

```
Qtests <- function(series, k, fitdf = 0) {
  pvals <- apply(
    matrix(1:k),
    1,
    FUN = function(l) {
      pval <- if (l <= fitdf)
        NA
      else
        Box.test(series,
                 lag = l,
                 type = "Ljung-Box",
                 fitdf = fitdf)$p.value #Calcul la p_value du test Ljung-Box pour chaque résidus avec u
      return(c("lag" = l, "pval" = pval))
    }
  )
  return(t(pvals))
}

Qtests(adf@test$lm$residuals, 20, fitdf = length(adf@test$lm$coefficients))
```

```
##       lag         pval
##  [1,]   1           NA
##  [2,]   2           NA
##  [3,]   3 1.140056e-04
##  [4,]   4 2.912963e-06
##  [5,]   5 1.723650e-06
##  [6,]   6 3.144223e-06
##  [7,]   7 1.068910e-07
##  [8,]   8 6.293536e-08
##  [9,]   9 1.653414e-07
## [10,]  10 7.830100e-09
## [11,]  11 5.418045e-10
## [12,]  12 6.211314e-10
```

```
## [13,]  13 1.555426e-09
## [14,]  14 1.454755e-10
## [15,]  15 3.572365e-11
## [16,]  16 1.861888e-11
## [17,]  17 1.995326e-11
## [18,]  18 1.268097e-12
## [19,]  19 1.561307e-12
## [20,]  20 3.274381e-12
```

On inclue le paramètre 20 pour voir l'autocorrélation des résidus sur les 20 premiers lags. Les deux premiers ne sont pas calculés car il y a 2 degrés de liberté en moins à cause du nombre de coefficients de la spécification du test ADF.

Tous les résidus du test ADF avec 0 lag sont autocorrélés. Il faut ajouter des lags.

Pour trouver le nombre de lag optimal à inclure dans le test ADF, il faut itérer ce test en changeant le nombre de lag du test ADF. Il suffit d'arrêter l'itération quand toutes les p valeurs de Qtests sont supérieur à 0.05. Cela signifie en effet qu'aucun résidus n'est corrélé dans la régression car il y a assez de lag dans spécification pour prendre en compte l'effet du passé sur le présent.

```
adfTest_valid <- function(series, kmax, adftype) {
  k <- 0
  noautocorr <- 0
  while (noautocorr == 0) {
    cat(paste0("ADF with ", k, " lags: residuals OK? "))
    adf <- adfTest(series, lags = k, type = adftype)
    pvals <-
      Qtests(adf@test$lm$residuals,
             kmax,
             fitdf = length(adf@test$lm$coefficients))[, 2]
    if (sum(pvals < 0.05, na.rm = T) == 0) { #tant qu'au moins une p-value est inférieur à 0.05, le tes
      noautocorr <- 1
      cat("OK \n")
    } else
      cat("nope \n")
    k <- k + 1
  }
  return(adf)
}

adf <- adfTest_valid(xm, 20, adftype = "c") # On exécute adfTest_valid sur xm avec le paramètre 20 pour
```

```
## ADF with 0 lags: residuals OK?
```

```
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
```

```
## nope
## ADF with 1 lags: residuals OK?
```

```
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
```

```
## nope
## ADF with 2 lags: residuals OK?
```

```
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
```

```
## nope
## ADF with 3 lags: residuals OK? nope
## ADF with 4 lags: residuals OK? nope
## ADF with 5 lags: residuals OK? nope
## ADF with 6 lags: residuals OK? nope
## ADF with 7 lags: residuals OK? nope
## ADF with 8 lags: residuals OK? nope
## ADF with 9 lags: residuals OK? OK
```

Le nombre de lag optimal a inclure dans la spécification du test ADF est 9. En effet avec 9 lags il n'y a plus de problème d'autocorrélation des résidus. On peut donc analyser la p value du test ADF avec 9 lag et une constante pour conclure a la stationarité ou non de la série. adfTest_valid a en fait enregistré le dernier test réalisé, i.e celui avec le nombre de lag optimal:

```
adf
```

```
##
## Title:
##  Augmented Dickey-Fuller Test
##
## Test Results:
##   PARAMETER:
##     Lag Order: 9
##   STATISTIC:
##     Dickey-Fuller: -1.3982
##   P VALUE:
##     0.5372
##
## Description:
##  Tue May  2 17:42:35 2023 by user:
```

On accepte l'hypothèse de non stationarité. On doit donc différencier la série. On réitère la procédure pour s'assurer que la série différenciée est statioinnaire (on enlève le premier élément d'index car en différenciant la série on a perdu une date) :

```
summary(lm(dxm ~ index[-1]))
```

```
##
## Call:
## lm(formula = dxm ~ index[-1])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -73.705  -3.012   0.454   3.755  26.364
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.1085357  0.8322791  -0.130    0.896
## index[-1]    0.0005546  0.0036288   0.153    0.879
##
## Residual standard error: 8.224 on 393 degrees of freedom
## Multiple R-squared:  5.942e-05,  Adjusted R-squared:  -0.002485
## F-statistic: 0.02335 on 1 and 393 DF,  p-value: 0.8786
```

Ni la constante ou la tendance temporelle ne sont significatifs dans cette régression. On choisit donc un modèle sans trend ni constante dans notre test ADF :

```
adf <- adfTest_valid(dxm, 50, "nc")
```

```
## ADF with 0 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 1 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 2 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 3 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 4 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 5 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 6 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 7 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## nope
## ADF with 8 lags: residuals OK?
## Warning in adfTest(series, lags = k, type = adftype): p-value smaller than
## printed p-value
## OK
```

```
adf
```

```
##
## Title:
##  Augmented Dickey-Fuller Test
##
## Test Results:
```

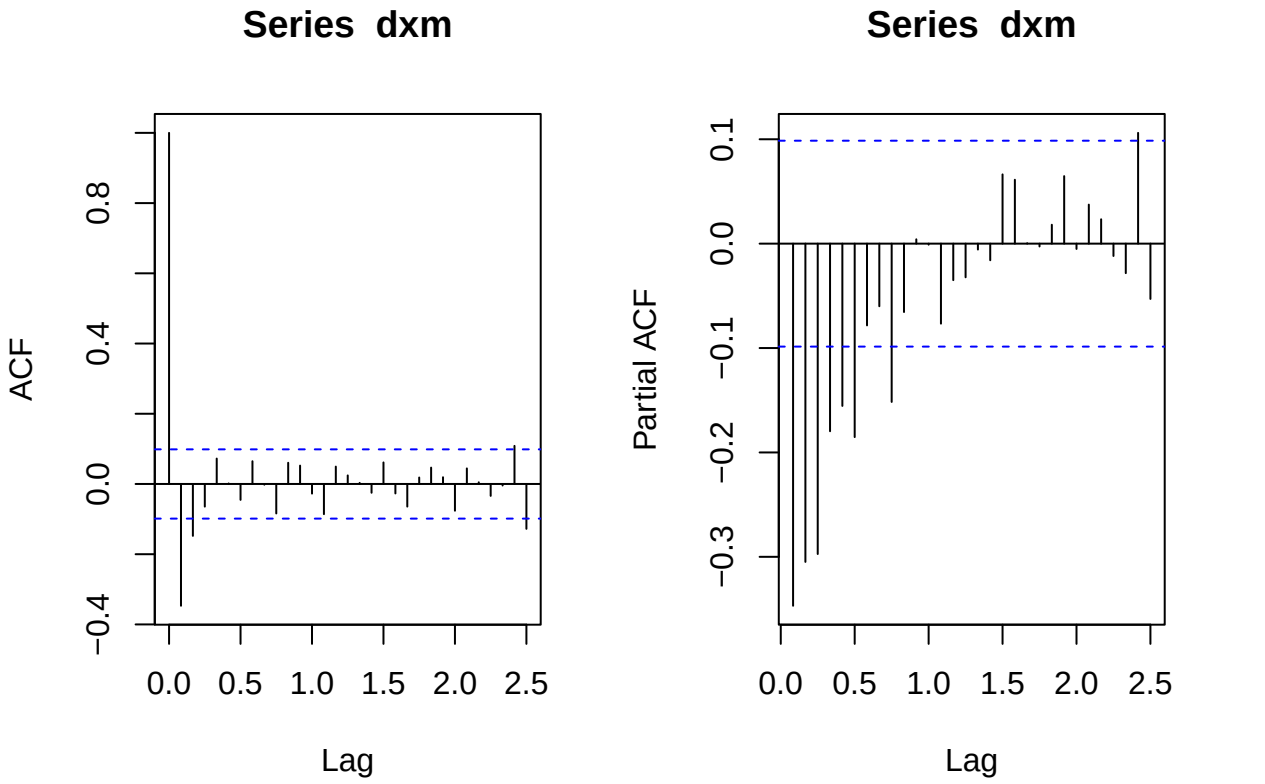
```
##   PARAMETER:
##     Lag Order: 8
##   STATISTIC:
##     Dickey-Fuller: -11.2387
##   P VALUE:
##     0.01
##
## Description:
##  Tue May  2 17:42:35 2023 by user:
```

On rejette l'hypothèse de racine unitaire avec 8 lags. La série est donc bien stationnaire en différence première. Il faut donc travailler avec la série dxm.

Modélisation

Afin de déterminer les ordres maximums vraisemblables de p et q, nous construisons les graphiques d'ACF et PACF :

```
par(mfrow = c(1, 2))
acf(dxm, 30)
pacf(dxm,30)
```



La fonction d'autocorrélation partielle est significative jusqu'à l'ordre 9. La fonction d'autocorrélation est significative jusqu'à l'ordre 2.

```
pmax = 9
qmax = 2
```

Il s'agit maintenant de sélectioner le meilleur modèle ARIMA(p,1,q) avec p et q entre 0 et respectivement pmax et qmax. Pour cela on construit une matrice affichant le critère AIC et BIC de chaque modèle et on choisit les modèles qui minimisent chaque critère :

```
mat <- matrix(NA,nrow=pmax+1,ncol=qmax+1) #matrice vide à remplir
rownames(mat) <- paste0("p=",0:pmax) #renomme les lignes
colnames(mat) <- paste0("q=",0:qmax) #renomme les colonnes
AICs <- mat #matrice des AIC non remplie
BICs <- mat #matrice des BIC non remplie
pqs <- expand.grid(0:pmax,0:qmax) #toutes les combinaisons possibles de p et q
for (row in 1:dim(pqs)[1]){ #boucle pour chaque (p,q)
  p <- pqs[row,1] #récupère p
  q <- pqs[row,2] #récupère q
  estim <- try(arima(xm,c(p,1,q),include.mean = F)) #tente d'estimer l'ARIMA
  AICs[p+1,q+1] <- if (class(estim)=="try-error") NA else estim$aic #assigne l'AIC
  BICs[p+1,q+1] <- if (class(estim)=="try-error") NA else BIC(estim) #assigne le BIC
}
AICs==min(AICs) #cherche le modèle minimisant le critère AIC
```

```
##       q=0   q=1   q=2
## p=0 FALSE FALSE FALSE
## p=1 FALSE FALSE FALSE
## p=2 FALSE FALSE FALSE
## p=3 FALSE FALSE  TRUE
## p=4 FALSE FALSE FALSE
## p=5 FALSE FALSE FALSE
## p=6 FALSE FALSE FALSE
## p=7 FALSE FALSE FALSE
## p=8 FALSE FALSE FALSE
## p=9 FALSE FALSE FALSE
```

```
BICs==min(BICs) #cherche le modèle minimisant le critère BIC
```

```
##       q=0   q=1   q=2
## p=0 FALSE FALSE  TRUE
## p=1 FALSE FALSE FALSE
## p=2 FALSE FALSE FALSE
## p=3 FALSE FALSE FALSE
## p=4 FALSE FALSE FALSE
## p=5 FALSE FALSE FALSE
## p=6 FALSE FALSE FALSE
## p=7 FALSE FALSE FALSE
## p=8 FALSE FALSE FALSE
## p=9 FALSE FALSE FALSE
```

Les modèles ARIMA(3,1,2) et ARIMA(0,1,2) minimisent respectivement le AIC et le BIC. Cependant, nous allons réaliser une étude plus approfondie pour déterminer le meilleur modèle. Nous allons notamment déterminer la significativité des coefficient des AR et des MA, et analyser l'autoccorélatoin des résidus de chaque modèle.

Pour ce faire nous allons définir des fonctions utiles.

La fonction permet de calculer la p_value des coefficients d'un modèle ARMA donné :

```
signif <-
  function(estim) {
    coef <- estim$coef #extrait chaque coefficient
    se <- sqrt(diag(estim$var.coef)) #calcul les écarts types de chaque coefficient
    t <- coef / se #calcul le t de student de chaque coefficient
    pval <- (1 - pnorm(abs(t))) * 2 #calcul la p_value de chaque coefficient
    return(rbind(coef, se, pval))
```

```
  }
```

La fonction suivante estime un arima et verifie l'ajustement et la validité du modèle grâce à signif :

```
modelchoice <- function(p, q, data = dxm, k = 24) {
  estim <-
    try(arima(data, c(p, 0, q), optim.control = list(maxit = 20000)))
  if (class(estim) == "try-error")
    return(c(
      "p" = p,
      "q" = q,
      "arsignif" = NA,
      "masignif" = NA,
      "resnocorr" = NA,
      "ok" = NA
    ))
  arsignif <- if (p == 0) #si p égal 0, il n'y a pas de coefficient AR donc la fonction renvoie NA
    NA
  else
    signif(estim)[3, p] <= 0.05 #si la p-value du coefficient AR, avec l'ordre de retard le plus import
  masignif <- if (q == 0) #si q égal 0, il n'y a pas de coefficient MA donc la fonction renvoie NA
    NA
  else
    signif(estim)[3, p + q] <= 0.05 #si la p-value du coefficient MA, avec l'ordre de retard le plus im
  resnocorr <-
    sum(Qtests(estim$residuals, 30, length(estim$coef) - 1)[, 2] <= 0.05, na.rm =
          T) == 0 #si les résidus ne sont pas autocorrélés, la fonction renvoie 1
  checks <- c(arsignif, masignif, resnocorr)
  ok <-
    as.numeric(sum(checks, na.rm = T) == (3 - sum(is.na(checks)))) #la fonction asssigne 1 à la colonne
  return(
    c(
      "p" = p,
      "q" = q,
      "arsignif" = arsignif,
      "masignif" = masignif,
      "resnocorr" = resnocorr,
      "ok" = ok
    )
  )
}
```

La fonction armamodelchoice estime et éxécute modelchoice sur tous les arima(p,q) avec p<=pmax et q<=max

```
armamodelchoice <- function(pmax, qmax) {
  pqs <- expand.grid(0:pmax, 0:qmax) #crréation de la grille de p et q
  t(apply(matrix(1:dim(pqs)[1]), 1, function(row) { #itération sur la grille
    p <- pqs[row, 1]
    q <- pqs[row, 2]
    cat(paste0("Computing ARMA(", p, ",", q, ") \n"))
    modelchoice(p, q) #éxécute la fonction modelchoice sur le modèle (p,q)
  }))
}
```

On éxécute cette dernière fonction avec pmax et qmax définis précédemment :

```
armamodels <- armamodelchoice(pmax, qmax)
```

```
## Computing ARMA(0,0)
## Computing ARMA(1,0)
## Computing ARMA(2,0)
## Computing ARMA(3,0)
## Computing ARMA(4,0)
## Computing ARMA(5,0)
## Computing ARMA(6,0)
## Computing ARMA(7,0)
## Computing ARMA(8,0)
## Computing ARMA(9,0)
## Computing ARMA(0,1)
## Computing ARMA(1,1)
## Computing ARMA(2,1)
## Computing ARMA(3,1)
## Computing ARMA(4,1)
## Computing ARMA(5,1)
## Computing ARMA(6,1)
## Computing ARMA(7,1)
## Computing ARMA(8,1)
## Computing ARMA(9,1)
## Computing ARMA(0,2)
## Computing ARMA(1,2)
## Computing ARMA(2,2)
## Computing ARMA(3,2)
## Computing ARMA(4,2)
## Computing ARMA(5,2)
## Computing ARMA(6,2)
## Computing ARMA(7,2)
## Computing ARMA(8,2)
## Computing ARMA(9,2)
```

```
armamodels
```

```
##       p q arsignif masignif resnocorr ok
##  [1,] 0 0       NA       NA         0  0
##  [2,] 1 0        1       NA         0  0
##  [3,] 2 0        1       NA         0  0
##  [4,] 3 0        1       NA         0  0
##  [5,] 4 0        1       NA         0  0
##  [6,] 5 0        1       NA         0  0
##  [7,] 6 0        1       NA         0  0
##  [8,] 7 0        0       NA         0  0
##  [9,] 8 0        0       NA         0  0
## [10,] 9 0        1       NA         1  1
## [11,] 0 1       NA        1         0  0
## [12,] 1 1        1        1         0  0
## [13,] 2 1        1        1         1  1
## [14,] 3 1        0        1         1  0
## [15,] 4 1        0        1         1  0
## [16,] 5 1        0        1         1  0
## [17,] 6 1        0        1         1  0
## [18,] 7 1        0        1         1  0
```

```
## [19,] 8 1        0        1         1  0
## [20,] 9 1        0        0         1  0
## [21,] 0 2       NA        1         0  0
## [22,] 1 2        0        0         0  0
## [23,] 2 2        1        1         1  1
## [24,] 3 2        0        1         1  0
## [25,] 4 2        0        0         1  0
## [26,] 5 2        0        1         1  0
## [27,] 6 2        0        0         1  0
## [28,] 7 2        0        0         1  0
## [29,] 8 2        0        1         0  0
## [30,] 9 2        0        0         1  0
```

Il s'agit maintenant de sélectionner uniquement les modèles ou la colonne "ok" est égale à 1, c'est à dire où les coefficients de l'AR et du MA, avec les ordres de retard les plus important, sont significatifs et où les résidus ne sont pas autocorrélés :

```
selec <-
  armamodels[armamodels[, "ok"] == 1 &
               !is.na(armamodels[, "ok"]),]

selec
```

```
##      p q arsignif masignif resnocorr ok
## [1,] 9 0        1       NA         1  1
## [2,] 2 1        1        1         1  1
## [3,] 2 2        1        1         1  1
```

On retient seulement 3 modèles avec cette méthode, les modèles ARIMA(9,1,0), ARIMA(2,1,1) et ARIMA(2,1,2). Les modèles ARIMA(3,1,2) et ARIMA(0,1,2), qui minimisaient respectivement le AIC et le BIC, ne sont pas retenu avec cette méthode car le coefficient de l'AR le plus laggé n'est pas significatif et donc il faut passer au modèle ARIMA(2,1,2), que l'on a retenu. Quant à ARIMA(0,1,2) les résidus sont autocorrélés.

Il faut maintenant choisir le meilleur de ces 3 modèles. On utilise alors la méthode de minimisation des critères AIC et BIC sur ces 3 modèles :

```
pqs <-
  apply(selec, 1, function(row)
    list("p" = as.numeric(row[1]), "q" = as.numeric(row[2]))) #crée une liste des ordres p et q des mod
names(pqs) <-
  paste0("arma(", selec[, 1], ",", selec[, 2], ")") #renomme les éléments de la liste
models <-
  lapply(pqs, function(pq)
    arima(dxm, c(pq[["p"]], 0, pq[["q"]]))) #crée une liste des estimations des modèles candidats
vapply(models, FUN.VALUE = numeric(2), function(m)
  c("AIC" = AIC(m), "BIC" = BIC(m))) #calcule les AIC et BIC des modèles candidats
```

```
##     arma(9,0) arma(2,1) arma(2,2)
## AIC  2631.025  2626.257  2624.127
## BIC  2674.792  2646.152  2648.000
```

ARMA(2,2) minimise l'AIC et ARMA(2,1) minimise le BIC. Le BIC pénalise en effet les modèles avec de grands ordre de retards

On récupère maintenant l'estimation de chacun de ces 2 modèles :

```
arima212 <- arima(xm,c(2,1,2),include.mean=F)
arima211 <- arima(xm,c(2,1,1),include.mean=F)
```

On définit une fonction qui calcule le R2 ajusté de chaque modèle :

```
adj_r2 <- function(model){
  p <- model$arma[1] #récupère le paramètre p du modèle
  q <- model$arma[2] #récupère le paramètre q du modèle
  ss_res <- sum(model$residuals^2) #somme les résidus au carré
  ss_tot <- sum(dxm[-c(1:max(p,q))]^2) #somme les valeurs au carré de dxm en excluant le max(p,q) premi
  n <- model$nobs-max(p,q) #calcul le nombre d'observation de la série avec les max(p,q) premières vale
  adj_r2 <- 1-(ss_res/(n-p-q-1))/(ss_tot/(n-1)) #calcul le R2 ajusté
  return(adj_r2)
}
adj_r2(arima212) #calcule le R2 ajusté d'un arima212
```

```
## [1] 0.3474986
```

```
adj_r2(arima211) #calcule le R2 ajusté d'un arima211
```

```
## [1] 0.3420339
```

On garde le modèleavec le R2 le plus faible, c'est à dire ARIMA(2,1,2). Il donne donc la meilleur prédiction

Étude du modèle choisi

On peut étudier ce modèle plus en détail :

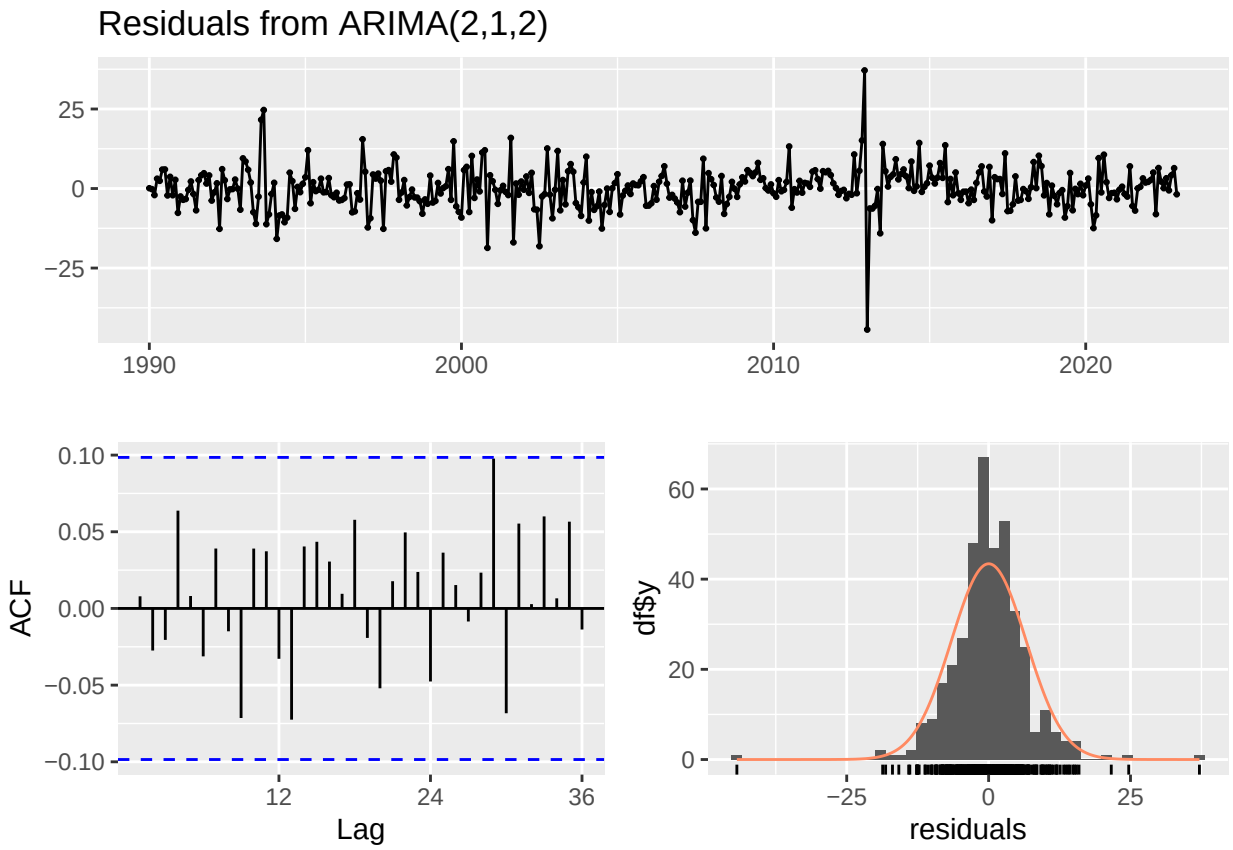
```
signif(arima212)
```

```
##               ar1           ar2       ma1          ma2
## coef 1.002484e+00 -2.418871e-01 -1.713137 7.596126e-01
## se   1.633714e-01  5.767746e-02  0.165049 1.379799e-01
## pval 8.450538e-10  2.743337e-05  0.000000 3.686645e-08
```

Tous les coefficients sont significatifs dans ce modèle.

On peut vérifier également graphiquement la qualité de nos résidus :

```
checkresiduals(arima212)
```



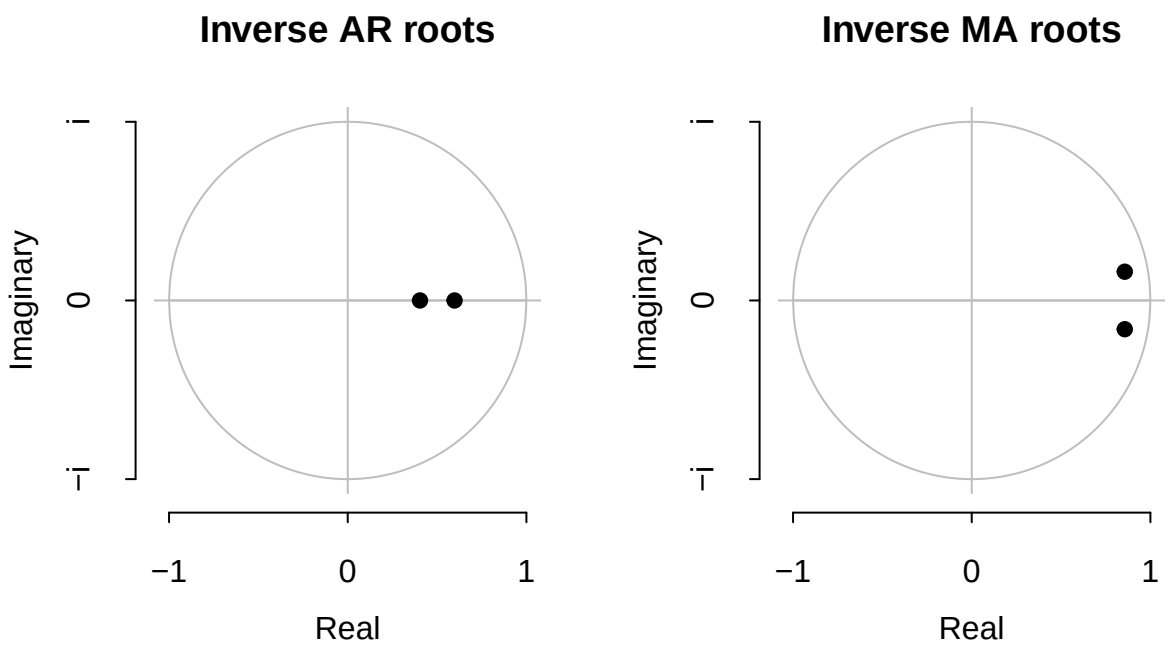

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(2,1,2)
## Q* = 16.067, df = 20, p-value = 0.7125
##
## Model df: 4.   Total lags used: 24
```

On étudie maintenant la causalité du modèle :

```
roots <- polyroot(c(1, -arima212$coef["ar1"], -arima212$coef["ar2"])) #calcul les racines du polynome 1
modulus_roots <- Mod(roots) #récupère les modulo des racines
modulus_roots #les coefficients sont plus grand que 1 donc le modèle est causal
```

```
## [1] 1.672327 2.472101
```

```
plot(Arima(xm,c(2,1,2))) #graphique de l'inverse des racines des polynomes AR et MA
```



Prévision

```
model_pred <- predict(arima212, n.ahead=2) #prédiction des deux valeurs après la dernière valeur de xm
pred <- zoo(model_pred$pred , order.by = as.yearmon(c(2023+0/12,2023+1/12))) #stock ces 2 valeurs dans
```

On crée une fonction qui permet de construire un graphique de la série depuis une date x et des prédictions de janvier et février 2023 :

```
plot_pred <- function(start){
  plot(xm.source, col = 'black', ylab = 'Série', main = 'Prévision des 2 prochaines valeurs de la série
  U = model_pred$pred + 1.96*model_pred$se #calcul la borne supérieure de l'intervalle de confiance
  L = model_pred$pred - 1.96*model_pred$se #calcul la borne inférieure de l'intervalle de confiance
  xx = c(time (U), rev (time (U))) #stock un objet xx pour le construction de l'intervalle de confiance
  yy = c(L, rev(U)) #stock un objet yy pour le construction de l'intervalle de confiance
  polygon(xx, yy, border = 8, col = gray (0.6, alpha=0.2)) #ajoute l'intervalle de confiance sur le gra
  lines(pred, type = "p", col = "red") #ajoute des points sur les valeurs prédites
  lines(pred, type = 'l', col = 'red') #ajoute des lignes entre les valeurs prédites
  link = rbind(xm[length(xm)],pred[1]) #stock un vecteur avec la dernière valeur avant prédiction et la
  lines(link, type = 'l', col = 'red') # trace une ligne entre la dernière valeur avant prédiction et l
  legend("topleft", legend=c("Données réelles", "Prédiction"), col=c("red", "black"), lty=1:2, cex=0.4)
}
```

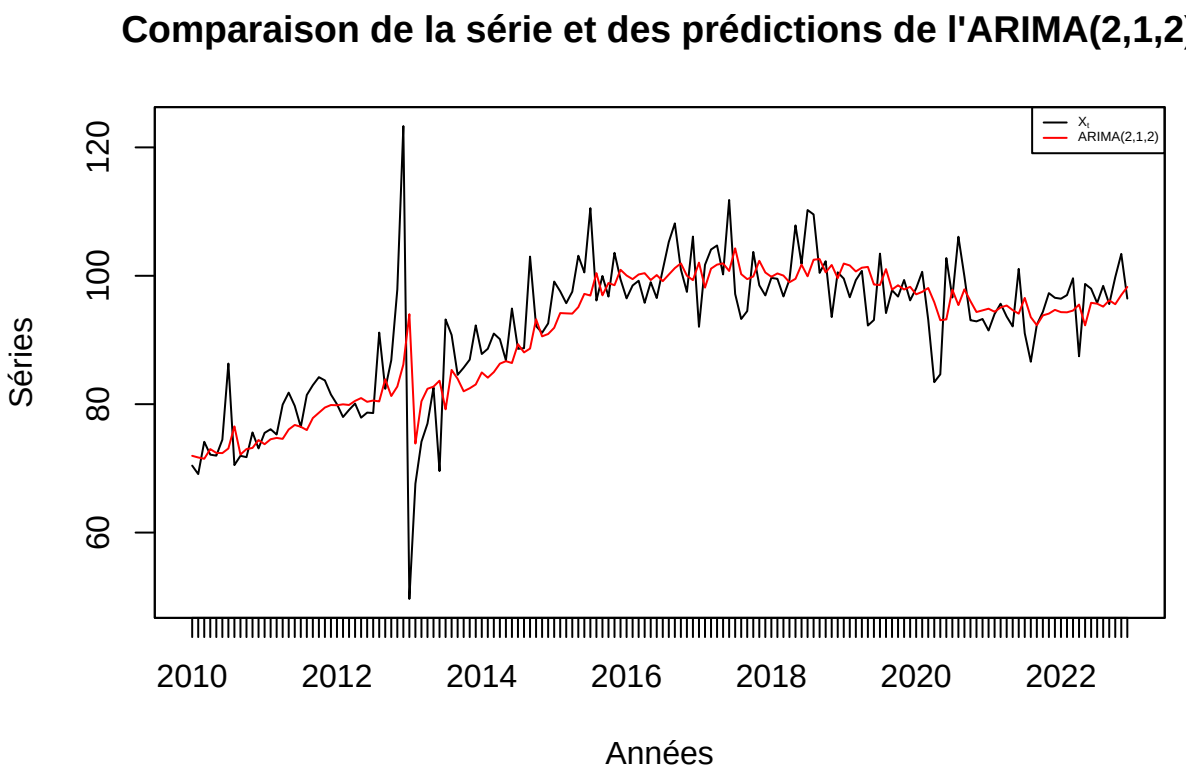
On trace le graphique uniquement depuis 2020 pour mieux voir ce qu'il se passe en prédiction :

```
plot_pred(2020) #graphique de la série depuis 2020 et des prédictions de janvier et février 2023
```

On trace également un graphique avec la série réelle et la prédiction pour chaque période de la série sachant les valeurs réelles précédentes :

```
arma_22 <- function(dxm_1, dxm_2, dxm_arma_1, dxm_arma_2){
  ma_1 <- dxm_1 - dxm_arma_1
  ma_2 <- dxm_2 - dxm_arma_2
  dxm_arima <- dxm_1 * arima212$coef[1]+ dxm_2 * arima212$coef[2] + ma_1 * arima212$coef[3] + ma_2 * ar
  return(dxm_arima)
}

dxm_arma<-c(0, 0)
xm_arima<-c(NA, NA, NA)
for (i in 3:length(dxm)) {
  dxm_arma[i] <- arma_22(as.numeric(dxm[i-1]), as.numeric(dxm[i-2]), as.numeric(dxm_arma[i-1]), as.nume
  xm_arima[i+1] <- as.numeric(dxm_arma[i]) + as.numeric(xm[i])
  }

xm_arima <-
  zoo(xm_arima, order.by = data$Date) #transformation de xm_arima en objet zoo
plot(
  xm[index(xm) >= 2010 + 0 / 12],
  main = "Comparaison de la série et des prédictions de l'ARIMA(2,1,2)",
  xlab = "Années",
  ylab = "Séries",
  col = "black"
) #construction du graphique
lines(xm_arima[index(xm) >= 2010 + 0 / 12], col = "red") #ajout de la série prédite en rouge

legend(
  "topright",
```

```
  legend = expression(X[t], "ARIMA(2,1,2)"),
  col = c("black", "red"),
  lty = 1,
  cex = 0.4
) #ajout de la légende
```

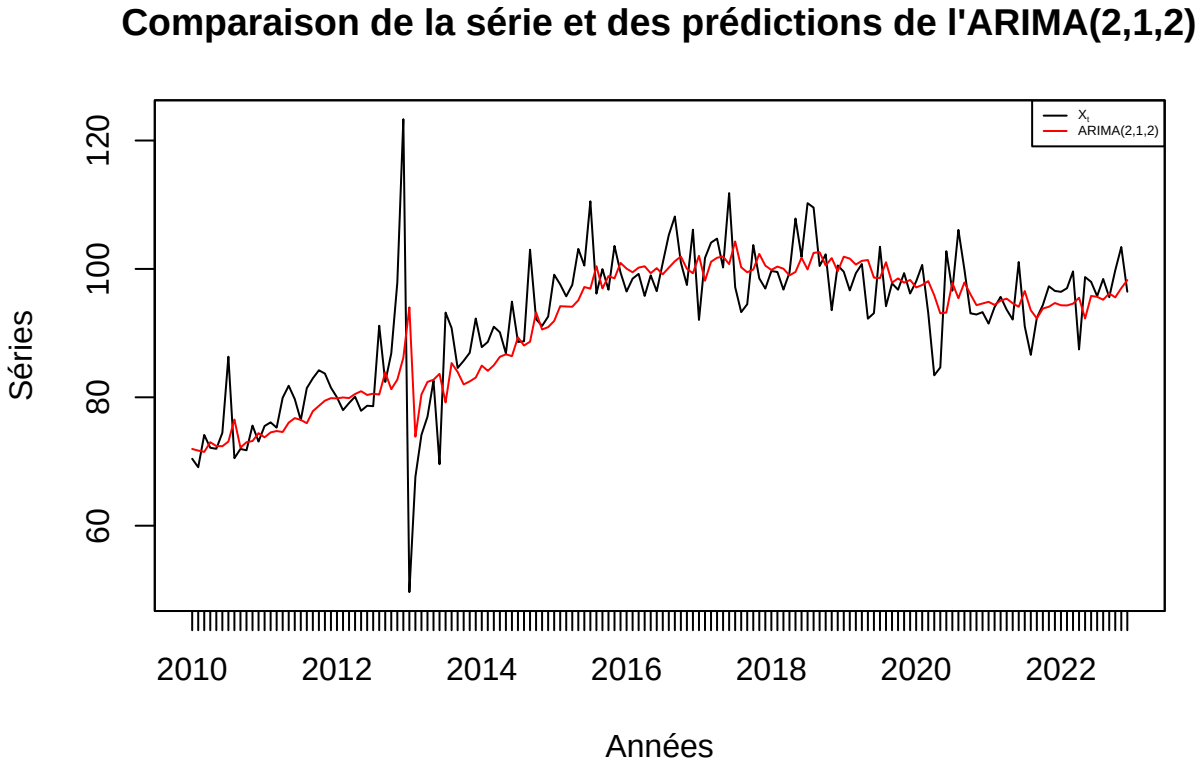


On voit que la modèlisation a du mal a prédire des chocs moyen et a donc tendance à lisser la série.

On a également testé avec une autre méthode qui récupère les valeurs prédites sans reconstruire le modèle :

```
resid <- residuals(arima212) #obtention des résidus de l'estimation

fitted <- data$value - resid #obtention des valeurs prédites

xm_arima <-
  zoo(fitted, order.by = data$Date) #transformation de xm_arima en objet zoo
plot(
  xm[index(xm) >= 2010 + 0 / 12],
  main = "Comparaison de la série et des prédictions de l'ARIMA(2,1,2)",
  xlab = "Années",
  ylab = "Séries",
  col = "black"
) #construction du graphique
lines(xm_arima[index(xm) >= 2010 + 0 / 12], col = "red") #ajout de la série prédite en rouge

legend(
  "topright",
  legend = expression(X[t], "ARIMA(2,1,2)"),
  col = c("black", "red"),
  lty = 1,
```

```
  cex = 0.4
)
```


Comparaison de la série et des prédictions de l'ARIMA(2,1,2)
Séries
60
80
100
120
2010
2012
2014
2016
2018
2020
2022
Années
X_t
ARIMA(2,1,2)